

ERPM — Avancement Wrapper Et Nouveaux Effets

Jérémie Chichignoud (CUB'iTECH)

30 octobre 2025

État d'avancement : wrapper `erpm` et statistiques/effets

Objectifs (portée du document)

L'objectif est de concevoir une fonction `erpm()` inspirée de la logique des modèles linéaires généralisés, capable d'estimer des effets statistiques définis sur des **partitions** plutôt que sur des graphes. La syntaxe cible est :

```
erpm(partition ~ effets + options)
```

La partie gauche représente la partition des acteurs en groupes, et la partie droite décrit les effets statistiques à calculer.

La fonction agit comme un **wrapper** autour du moteur `ergm()`. Elle transforme la partition en un **graphe biparti** (acteurs–groupes), sur lequel `ergm()` peut être appliqué directement. Cette traduction permet d'utiliser les outils et algorithmes d'ERGM tout en conservant une lecture centrée sur les groupes.

Un autre rôle clé de `erpm()` est d'assurer la **correspondance entre effets ERPM et ERGM**, selon trois situations :

1. **Effet direct** : déjà présent dans ERGM et utilisable tel quel ;
2. **Effet partiel** : partiellement couvert, nécessitant un traitement complémentaire (agrégation, normalisation, etc.) ;
3. **Effet spécifique** : absent d'ERGM, à créer avec `InitErgmTerm()` et une fonction C de changement (`change_stat`).

L'objectif final est que `erpm()` pilote l'ensemble du processus : de la définition de la partition à l'appel de `ergm()`, en passant par la préparation des effets, des contraintes et des propositions.

Travail réalisé

La structure principale de `erpm()` est **fonctionnelle et stable**. Elle gère correctement la conversion des entrées, le renommage des effets et l'appel au moteur `ergm()`.

- **Structure du wrapper** : prise en charge d'une partition et d'une formule d'effets ; construction de la matrice d'affiliation (acteurs \times groupes) ; création du **network biparti** ; ajout automatique des contraintes et propositions nécessaires (`InitErgmConstraint.b1part`, `InitErgmProposal.B1Part`) ; puis exécution de `ergm()`.
- **Traduction des effets** : les effets connus sont traduits vers leurs équivalents ERGM (par ex. `groups` \rightarrow `b2degrange`). Les effets dyadiques utilisent les opérateurs `Proj1` ou `B` selon le mode d'analyse.
- **Effets implémentés** : un effet spécifique, `squared_sizes`, est pleinement fonctionnel. Il dispose de sa fonction `InitErgmTerm()` et de son code C associé, basé sur les macros d'ERGM (`ergm_changestat_common.do_not_include_directly.h`, `ergm_changestat.h`, etc.).
- **Tests et validation** : les scripts `squared_sizes.R` et `test_groups.R` vérifient la cohérence du pipeline complet : création de partitions simples, conversion bipartie, appel à `erpm()`, et calcul des statistiques. Ces tests sont conçus comme des *selftests* autonomes.

Les résultats montrent que l'intégration avec `ergm()` fonctionne correctement et que la structure du wrapper est fiable.

Travail restant et perspectives

La prochaine étape est la **caractérisation complète des effets ERPM**. Chaque effet doit être identifié comme direct, partiel ou spécifique, et implémenté en conséquence, avec un test unitaire dédié.

- **Finalisation du wrapper** : améliorer la gestion des options `control.ergm`, définir des valeurs par défaut cohérentes, traiter les cas limites (groupes vides, partitions incomplètes), et renforcer la traduction automatique des effets directs `erpm` \rightarrow `ergm`.
- **Validation des effets et tests d'intégration** : ajouter un *selftest* autonome pour chaque nouvel effet, et construire des cas minimaux pour vérifier la stabilité statistique (moyennes, variances, convergence).
- **Effets liés aux attributs dyadiques** : une partition ne comporte pas d'attributs dyadiques, mais il peut être utile de modéliser la probabilité d'appartenance à un groupe à partir d'informations dyadiques externes. Dans ce cas, une matrice d'attributs dyadiques peut être transmise en complément de la partition.
- **Documentation** : compléter les blocs `roxygen2` et rédiger une vignette “`erpm` en dix lignes” illustrant le flux complet : partition \rightarrow graphe biparti \rightarrow estimation \rightarrow résumé.

En résumé, `erpm()` est stable, bien intégré à `ergm()`, et prêt à accueillir progressivement de nouveaux effets ERPM.

Tableau de correspondance des effets

Effet ERPM	Correspondance ERGM	Alias/Termes ERGM	Calcul partiel (si partielle)	Si aucune : InitErgmTerm + C	Commentaire	Status
<code>groups(from,to)</code>	Directe	<code>b2degrange(from,to)</code>	—	—	Taille des groupes = degré (mode 2).	✓
<code>squared_sizes(from,to,pow)</code>	Effet spécifique	—	—	<code>InitErgmTerm.squared_sizes_c_squared_sizes</code>	Somme tailles ^{pow} par fenêtre (mode 2).	✓
<code>cov_match(cov)</code>	Directe via projection	<code>Proj1(~nodematch("cov"))</code>	—	—	Homophilie intra-groupe (projection mode 1).	✗
<code>cov_diff(cov)</code>	Directe via projection	<code>Proj1(~absdiff("cov"))</code>	—	—	Différences intra-groupe (paires).	✗
<code>dyadcov(X)</code>	Directe via projection	<code>Proj1(~edgecov(X))</code>	—	—	Effets dyadiques sur la projection. <i>Optionnel</i> : <code>B(form="nonzero")</code> si nécessaire.	✗
<code>cov_fullmatch(cov)</code>	Partielle	<code>Proj1</code> + comptage	Post-traitement : groupes 100% homogènes.	(optionnel) <code>InitErgmTerm.cov_fullmatch_c_cov_fullmatch</code>	Post-hoc OK ; term C si nécessaire en MCMC.	✗
<code>cliques(k=2)</code>	Partielle	<code>Proj1</code> + stats paires	Selon définition (norm., seuils).	(optionnel) <code>InitErgmTerm.cliques_c_cliques</code>	Préférer projection pour $k = 2$.	✗
<code>cliques_GW(\$\lambda\$)</code>	Pas encore évaluée	—	—	<code>InitErgmTerm.cliques_GW_c_cliques_GW</code>	Pondération géométrique (probable custom).	✗
<code>log_factorial_sizes</code>	Directe (custom)	<code>b2degm1lfactorial ↪ sum_lfactorial_idegreem1</code>	—	<code>InitErgmTerm.b2degm1lfactorial_c_sum_lfactorial_idegreem1</code>	C OK (<code>lgammafn</code>).	✗
<code>inertia_longitudinal</code>	Pas encore évaluée	—	—	<code>InitErgmTerm.inertia_c_inertia</code>	Effet temporel avec auxiliaires ($t-1$).	✗

Effet ERPM	Correspondance ERGM	Alias/Termes ERGM	Calcul partiel (si partielle)	Si aucune : InitErgmTerm + C	Commentaire	Status
<code>dyadcov_intragroup_sum</code>	Partielle	<code>Proj1(~edgecov) + agrégations</code>	Sommes/normalisations par groupe (R).	(<i>optionnel</i>) term dédié si requis	Pour métriques par groupe dans la vraisemblance.	X
<code>range_attribute(attr)</code>	Partielle	<code>Proj1 + absdiff/résumés</code>	Max-min par groupe (post-traitement).	(<i>optionnel</i>) term dédié	Contraintes d'hétérogénéité intra-groupe.	X

Annexe — ERGM et Metropolis–Hastings

A.1 Rappel : qu'est-ce qu'un ERGM ?

Un modèle de graphe aléatoire exponentiel (ERGM, *Exponential Random Graph Model*) définit une loi de probabilité sur l'ensemble des graphes possibles y :

$$\Pr_{\theta}(Y = y) \propto \exp(\theta^T g(y)),$$

où :

- $g(y)$ est le **vecteur de statistiques du graphe** : il regroupe des caractéristiques mesurées sur le réseau, par exemple le *nombre d'arêtes* (statistique dénombrable ou « brute ») et le *degré moyen* (statistique agrégée ou « moyenne »).
- θ est le **vecteur de paramètres du modèle** : chaque composante pondère la propension du réseau à présenter la structure correspondante (par ex. triangles, réciprocité).
- \Pr_{θ} désigne la **distribution de probabilité induite par θ** : c'est la loi selon laquelle un graphe est susceptible d'être généré.

Motivation.

On dispose d'un **graphe observé** y_{obs} (données empiriques). Ce graphe est fixe : on calcule ses statistiques $g(y_{\text{obs}})$ une fois pour toutes. L'objectif est de trouver θ tel que les graphes simulés depuis \Pr_{θ} aient, *en moyenne*, les mêmes statistiques que y_{obs} :

$$\mathbb{E}_{\theta}[g(Y)] \approx g(y_{\text{obs}}),$$

afin de reproduire les régularités structurelles du réseau (densité, distribution des degrés, triangles, homophilie, etc.).

-
1. Pour deux graphes y et y' sous un même θ :

$$\frac{\Pr_{\theta}(Y = y')}{\Pr_{\theta}(Y = y)} = \exp\left\{ \theta^T [g(y') - g(y)] \right\} = \exp(\Delta), \quad \Delta = \theta^T [g(y') - g(y)],$$

Si $\Delta > 0$, y' est plus probable que y (toutes choses égales par ailleurs).

Principe général.

L'estimation de θ se fait par deux boucles :

- **Boucle interne (MCMC–MH)** : pour un θ fixé, on échantillonne des graphes selon \Pr_{θ} à l'aide de **Metropolis–Hastings (MH)**. Concrètement :

1. partir d'un graphe initial $y^{(0)}$;
2. proposer une modification locale donnant un candidat y' (ajout/suppression d'une arête) ;
3. calculer la variation de log-probabilité ¹ :

$$\Delta = \theta^T [g(y') - g(y)],$$

4. accepter y' avec probabilité $\alpha = \min(1, \exp(\Delta) \times \frac{q(y' \rightarrow y)}{q(y \rightarrow y')})$, où q est la loi de proposition.

En répétant ces étapes, on obtient une chaîne de Markov dont la loi stationnaire est précisément \Pr_{θ} . L'échantillon $\{y^{(s)}\}$, constitué de plusieurs graphes simulés indépendamment (après burn-in et thinning), sert à approximer les moyennes des statistiques $g(y)$ sous la distribution \Pr_{θ} .

- **Boucle externe (mise à jour de θ)** : après simulation, on compare la moyenne simulée $\mathbb{E}_{\theta^{(t)}}[g(Y)]$ à $g(y_{\text{obs}})$ et on ajuste

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \left(g(y_{\text{obs}}) - \frac{1}{S} \sum_{s=1}^S g(y^{(s)}) \right),$$

où α est un **pas d'apprentissage**. Un pas trop grand peut faire diverger l'algorithme ; trop petit, la convergence devient lente.

Lien avec les algorithmes de gradient.

La mise à jour de θ est analogue à une **montée de gradient** sur la log-vraisemblance

$$\log L(\theta) = \theta^\top g(y_{\text{obs}}) - \log Z(\theta),$$

car $g(y_{\text{obs}}) - \mathbb{E}_\theta[g(Y)]$ joue le rôle d'un **gradient stochastique**. Le pas α influence vitesse et stabilité de convergence. La fonction de vraisemblance peut être **non convexe** dans l'espace des graphes : la convergence dépend du point de départ et de la structure du réseau.

A.2 Log-vraisemblance, déviance, AIC et BIC

Constante de normalisation $Z(\theta)$.

Dans un ERGM, l'écriture exacte de la probabilité de tirage est :

$$\Pr_\theta(Y = y) = \frac{\exp\{\theta^\top g(y)\}}{Z(\theta)}, \quad Z(\theta) = \sum_{y' \in \mathcal{Y}} \exp\{\theta^\top g(y')\},$$

où \mathcal{Y} est l'ensemble de *tous* les graphes possibles sur le même jeu de noeuds. $Z(\theta)$ assure la **normalisation** (les probabilités somment à 1). Comme \mathcal{Y} est gigantesque, $Z(\theta)$ n'est pas calculable exactement ; on l'approxime par MCMC.

Log-vraisemblance.

Idée générale.

La **vraisemblance** $L(\theta)$ mesure, pour un θ donné, à quel point le modèle juge plausible le graphe observé y_{obs} :

$$L(\theta) = \Pr_\theta(Y = y_{\text{obs}}) = \frac{\exp\{\theta^\top g(y_{\text{obs}})\}}{Z(\theta)},$$

Maximiser $L(\theta)$ (ou son logarithme) revient à choisir les paramètres qui rendent y_{obs} le plus probable dans la famille des modèles.

Pourquoi travailler au log.

On utilise la **log-vraisemblance**

$$\ell(\theta) = \log L(\theta) = \theta^\top g(y_{\text{obs}}) - \log Z(\theta),$$

car (i) les produits de probabilités deviennent des *sommes* (plus stables numériquement), et (ii) le calcul des dérivées/gradiants est direct.

Déviance et critères d'information.

La **déviance** d'un modèle est définie par

$$D = -2\ell(\hat{\theta}),$$

où $\hat{\theta}$ est l'estimateur (par exemple le maximum de vraisemblance). Sous conditions régulières, les **rapports de vraisemblance** ont une loi asymptotique de type χ^2 , d'où le facteur -2 qui permet d'interpréter les *différences* de déviance dans des tests de comparaison de modèles.

Déviance et AIC/BIC (version unifiée).

Dans un ERGM, $D = -2\ell(\hat{\theta})$ sert d'indicateur global d'ajustement : plus $\ell(\hat{\theta})$ est grand, plus D est petit. La déviance mesure toutefois l'ajustement *pur* : un modèle plus complexe a presque toujours une déviance plus faible. Pour **équilibrer ajustement et complexité**, on utilise les critères d'information AIC/BIC :

$$\text{AIC} = 2k - 2\ell(\hat{\theta}), \quad \text{BIC} = k \log n - 2\ell(\hat{\theta}),$$

où k est le nombre de paramètres et n une taille effective (en réseau, souvent proche du nombre de dyades potentielles). On compare des modèles ajustés sur les mêmes données et on retient en pratique celui dont l'AIC/BIC est **minimal** (meilleur compromis « fidélité/parcimonie »). AIC/BIC permettent en outre de comparer des modèles *non nécessairement emboîtés*, ce que ne permet pas un test basé uniquement sur la différence de déviance.

A.3 Sorties usuelles de `summary(ergm)`

- **Call** : l'appel exact (traçabilité, reproductibilité).
- **Coefficients** : estimés $\hat{\theta}_i$ (un par statistique), **erreur-type** (SE), **statistique** z ($z = \hat{\theta}_i/\text{SE}(\hat{\theta}_i)$) et **p-valeur bilatérale** pour $H_0 : \theta_i = 0$. Un grand $|z|$ suggère un effet éloigné de 0 au regard de l'incertitude.
- **MCMC %** : fraction (en %) de l'erreur-type due au *bruit Monte-Carlo*. *Interprétation et actions* : un MCMC% élevé indique que l'échantillon simulé est peu informatif (forte autocorrélation). **Burn-in** = nombre d'itérations initiales jetées avant d'enregistrer des échantillons (laisser la chaîne se stabiliser); **Thinning** = ne garder qu'une itération sur m pour **décorrérer les échantillons** entre eux (réduire l'autocorrélation serielle); **Taille d'échantillon** = nombre total d'états conservés pour estimer les moyennes. Si la chaîne *mélange mal*, il peut être nécessaire d'ajuster les contraintes ou le mécanisme de proposition.
- **Log-vraisemblance, déviance D, AIC, BIC** : indicateurs globaux (définitions ci-dessus). AIC/BIC plus petits \Rightarrow meilleur compromis.

A.4 Contraintes et mécanisme de proposition (MH) en biparti

Contraintes (restriction de l'espace d'état).

Les **contraintes** (par exemple `b1part`) restreignent l'espace des graphes explorés par MH aux configurations *valides* : en biparti, interdire les arêtes intra-mode, imposer certaines bornes de degrés, etc. Elles ne constituent pas un « assouplissement » : elles **excluent** simplement les états invalides et garantissent que chaque graphe visité respecte la structure voulue.

Mécanisme de proposition.

Le **mécanisme de proposition** (souvent appelé « proposal ») génère un candidat y' à partir de y (par exemple tirer une dyade admissible et toggler l'arête), en *respectant* les contraintes. Il définit la distribution $q(y \rightarrow y')$ qui intervient dans

le **ratio de Hastings**. Si q est asymétrique (par exemple plus de façons de détruire que de créer une structure), le terme $\frac{q(y' \rightarrow y)}{q(y \rightarrow y')}$ corrige cette asymétrie dans la probabilité d'acceptation.

Pourquoi ces corrections sont indispensables.

Le couple « Δ du modèle » + « ratio de Hastings » assure la **réversibilité** (détail de balance) de la chaîne et garantit que la **loi stationnaire** visée est bien Pr_θ . Sans cette correction, la chaîne convergerait vers une distribution biaisée (dépendante du mécanisme de proposition) au lieu de la loi ERGM recherchée.

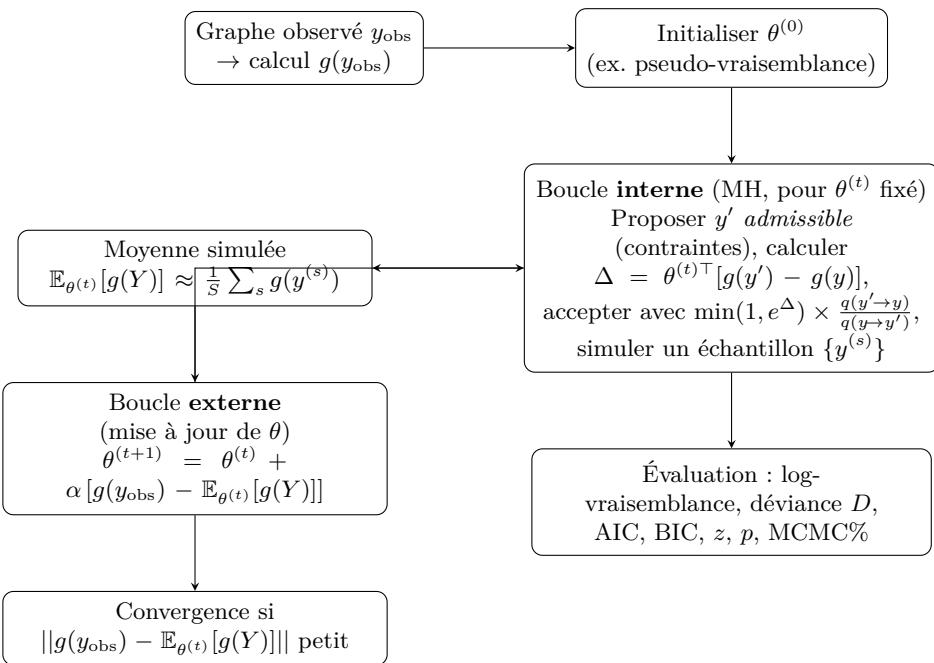
A.5 Exemple minimal (R)

```

1 library(ergm)
2 # Réseau biparti jouet : 4 acteurs -> 2 groupes
3 m <- matrix(c(1,0,
4                 1,0,
5                 0,1,
6                 0,1), nrow=4, byrow=TRUE)
7 nw <- network::network(m, bipartite=2, directed=FALSE)
8
9 # Modèle 1 : densité (edges)
10 fit1 <- ergm(nw ~ edges)
11
12 # Modèle 2 : densité + distribution de degrés côté groupes (
13 #               mode 2)
14 fit2 <- ergm(nw ~ edges + b2degrange(from=2, to=3))
15
16 summary(fit1)
17 summary(fit2) # comparer log-vraisemblance, AIC, BIC, z, p-
values, MCMC%

```

A.6 Schéma du processus d'estimation



A.7 Points d'attention

Choix de $g(y)$.

Les statistiques doivent refléter des mécanismes plausibles (densité, degrés, triangles, homophilie, effets d'attributs, etc.) sans surcharger le modèle : empiler trop de termes peut entraîner instabilités, colinéarités ou non-identifiabilité. Une approche progressive (noyau simple, diagnostics, puis complexification) est souvent utile.

Diagnostics MCMC.

Surveiller l'autocorrélation et la stabilité des moyennes simulées. Un **MCMC%** élevé signale une forte dépendance entre échantillons. Augmenter le **burn-in** (laisser la chaîne se stabiliser avant de collecter), appliquer du **thinning** (garder une itération sur m pour **décorreler les échantillons entre eux**), et/ou accroître la **taille d'échantillon**. Si le mélange reste faible, ajuster contraintes et mécanisme de proposition peut aider.

Comparaison de modèles.

AIC/BIC comparent des modèles ajustés sur les mêmes données en pénalisant la complexité. L'information est dans les **écart**s d'AIC/BIC (pas dans leur niveau absolu). Les conclusions gagnent à être croisées avec une **évaluation de type goodness-of-fit** : comparer, entre réseaux simulés et observé, des distributions de degrés, distances géodésiques, motifs triadiques, etc.