# ERPM — Avancement Wrapper Et Nouveaux Effets

Jérémie Chichignoud (CUB'iTECH)

21 octobre 2025

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

## Objectifs (portée du document)

L'objectif de ce travail est de concevoir et valider une fonction `erpm` dont la logique d'interface s'aligne sur celle des modèles linéaires généralisés (GLM). En pratique, la fonction doit accepter une syntaxe de type :

`erpm(partition ~ effets + options)`

où la partie gauche correspond à la **partition** (affectation des acteurs à des groupes) et la partie droite décrit les **effets statistiques** à évaluer pour cette partition.

La fonction `erpm` agit comme un *enveloppeur* (*wrapper*) autour du moteur `ergm`. Concrètement, elle transforme la partition initiale en un **graphe biparti** (acteurs–groupes), sur lequel `ergm` peut être appliqué. Cette étape de traduction permet d'utiliser le cadre théorique et algorithmique d'ERGM tout en conservant une lecture « par groupes » propre à ERPM.

Le second rôle fondamental de `erpm` est d'assurer la **correspondance des effets ERPM** dans le formalisme ERGM. Chaque effet défini dans ERPM doit être relié à un ou plusieurs équivalents ERGM, selon trois cas :

1. **Effet direct** : l'effet existe déjà dans ERGM et peut être appelé tel quel.
2. **Effet partiel** : l'effet n'est que partiellement couvert ; une adaptation ou un post-traitement (agrégation, normalisation, etc.) est nécessaire.
3. **Effet spécifique** : l'effet n'existe pas dans ERGM ; il faut alors créer un terme sur mesure via `InitErgmTerm` et un code C associé (fonction de changement `change_stat`).

L'objectif final est que `erpm` encapsule 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 mécanismes de proposition.

—

## Travail réalisé

À ce stade, la fonction `erpm` est **fonctionnelle dans sa structure principale**. Le wrapper gère la transformation des entrées, le renommage automatique des effets, et l'initialisation correcte du moteur ERGM.

- **Structure du wrapper** : la fonction prend en entrée une partition et une formule d'effets, construit la matrice d'affiliation (acteurs × groupes), crée l'objet `network` biparti, puis injecte les contraintes et propositions adaptées (`InitErgmConstraint.b1part`, `InitErgmProposal.B1Part`, etc.) avant d'appeler `ergm()`.
- **Renommage automatique des effets** : certains effets ERPM sont directement traduits vers leurs équivalents ERGM (`groups` → `b2degrange`, etc.). Pour les effets dyadiques, la fonction encapsule également les opérateurs `Proj1` et `B` lorsque c'est nécessaire.
- **Effets implémentés** : deux effets spécifiques ont été amorcés et validés : `b2degm1lfactorial` (lié à `sum_lfactorial_idegreem1`) et `squared_sizes`. Dans les deux cas, les fonctions `InitErgmTerm` et les codes C associés (fonctions `change_stat`) sont correctement intégrés et opérationnels.
- **Tests unitaires** : des tests basiques existent déjà (`test-prop-b1part.R`, `test-b2degm1lfactorial.R`, etc.) , validant la cohérence du pipeline de bout en bout : création du réseau, transformation bipartie, appel à `ergm()` et calcul des statistiques.

Ces résultats montrent que la couche d'intégration avec ERGM est stable, et que la structure logicielle de `erpm` est saine, modulaire et extensible.

—

## Travail restant et perspectives

La partie principale à compléter concerne la **caractérisation et l'intégration des effets ERPM**. Chaque effet doit être analysé pour déterminer s'il relève du cas "direct", "partiel" ou "spécifique", et être adapté en conséquence.

- **Finalisation de la fonction `erpm`** : affiner la gestion des options `control.ergm`, définir des valeurs par défaut homogènes et robustes, et améliorer la tolérance aux cas limites (groupes vides, partitions incomplètes, etc.).
- **Validation des effets existants** : compléter la vérification de `squared_sizes` et de `b2degm1lfactorial`, notamment pour les cas "from" et "to" côté biparti.
- **Exploration des effets intragroupes** : tester et formaliser l'usage de `Proj1` avec des termes comme `nodematch`, `absdiff` ou `edgecov`.
- **Tests d'intégration** : construire une série de tests sur réseaux de petite taille pour valider la cohérence statistique du pipeline (moyennes, variances, convergence).
- **Documentation** : compléter la documentation `roxygen2` et rédiger une vignette "`erpm` en dix lignes" pour illustrer le flux complet (partition → graphe → estimation → résumé).

En résumé, la base du wrapper est solide et l'interfaçage avec ERGM est validé. Le travail restant concerne essentiellement la formalisation et la validation statistique des effets, ainsi que la documentation et les tests unitaires nécessaires à une diffusion complète.

# Tableau de correspondance des effets

| Effet ERPM | Correspondance ERGM | Alias/Termes ERGM | Calcul partiel (si partielle) | Si aucune : `InitErgmTerm` + C | Commentaire | Status |
|---|---|---|---|---|---|---|
| `groups(from,to)` | **Directe** | `b2degrange(from,to)` | — | — | Taille des groupes = degré (mode 2). | ✓ |
| `squared_sizes(from,to, pow)` | **Directe (custom)** | `squared_sizes(from,to, pow)` | — | `InitErgmTerm.squared_sizes` `c_squared_sizes` | Somme tailles$^{\text{pow}}$ par fenêtre (mode 2). | ➪ |
| `cov_match(cov)` | **Directe via projection** | `Proj1(~nodematch("cov"))` | — | — | Homophilie intra-groupe (projection mode 1). | ✓ |
| `cov_diff(cov)` | **Directe via projection** | `Proj1(~absdiff("cov"))` | — | — | Différences intra-groupe (paires). | ✓ |
| `dyadcov(X)` | **Directe via projection** | `Proj1(~edgecov(X))` | — | — | Effets dyadiques sur la projection. *Optionnel* : `B(form="nonzero")` si nécessaire. | ✓ |
| `cov_fullmatch(cov)` | **Partielle** | `Proj1` + comptage | Post-traitement : groupes 100% homogènes. | (*optionnel*) `InitErgmTerm.cov_fullmatch` `c_cov_fullmatch` | Post-hoc OK ; term C si nécessaire en MCMC. | ➪ |
| `cliques(k=2)` | **Partielle** | `Proj1` + stats paires | Selon définition (norm., seuils). | (*optionnel*) `InitErgmTerm.cliques` `c_cliques` | Préférer projection pour $k = 2$. | ➪ |
| `cliques_GW($\lambda$)` | **Pas encore évaluée** | — | — | `InitErgmTerm.cliques_GW` `c_cliques_GW` | Pondération géométrique (probable custom). | ✗ |
| `log_factorial_sizes` | **Directe (custom)** | `b2degm1lfactorial` ↦ `sum_lfactorial_idegreem1` | — | `InitErgmTerm.b2degm1lfactorial` `c_sum_lfactorial_idegreem1` | C OK (`lgammafn`). | ✓ |
| `inertia_longitudinal` | **Pas encore évaluée** | — | — | `InitErgmTerm.inertia` `c_inertia` | Effet temporel avec auxiliaires ($t-1$). | ✗ |

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

# 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 $y$ :

$$\Pr_\theta(Y = y) \propto \exp\big(\theta^\top g(y)\big),$$

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 »).
- $\theta$ est le **vecteur de paramètres du modèle** : chaque composante pondère la propension du réseau à présenter la structure correspondante (par ex. triangles, réciprocité).
- $\Pr_\theta$ désigne la **distribution de probabilité induite par** $\theta$ : c'est la loi selon laquelle un graphe est susceptible d'être généré.

### Motivation.

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

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

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

### Principe général.

L'estimation de $\theta$ se fait par deux boucles :

- **Boucle interne (MCMC–MH)** : pour un $\theta$ fixé, on échantillonne des graphes selon $\Pr_\theta$ 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] :

  $$\Delta = \theta^\top\big[g(y') - g(y)\big],$$

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

  En répétant ces étapes, on obtient une chaîne de Markov dont la loi stationnaire est précisément $\Pr_\theta$. 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 $\Pr_\theta$.

- **Boucle externe (mise à jour de $\theta$)** : 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 $\alpha$ est un **pas d'apprentissage**. Un pas trop grand peut faire diverger l'algorithme ; trop petit, la convergence devient lente.

1. Pour deux graphes $y$ et $y'$ sous un même $\theta$ :

$$\frac{\Pr_\theta(Y = y')}{\Pr_\theta(Y = y)} = \exp\Big\{ \theta^\top [g(y') - g(y)] \Big\} = \exp(\Delta), \quad \Delta = \theta^\top [g(y') - g(y)],$$

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

### Lien avec les algorithmes de gradient.

La mise à jour de $\theta$ 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 $\alpha$ 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 :

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

où $\mathcal{Y}$ est l'ensemble de *tous* les graphes possibles sur le même jeu de nœuds. $Z(\theta)$ **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 $\theta$ donné, à quel point le modèle juge plausible le graphe observé $y_{\text{obs}}$ :

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

Maximiser $L(\theta)$ (ou son logarithme) revient à choisir les paramètres qui rendent $y_{\text{obs}}$ 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/gradients 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 $\chi^2$, 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 :

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

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** $z$ ($z = \hat{\theta}_i/\mathrm{SE}(\hat{\theta}_i)$) 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 **décorréler les échantillons** entre eux (réduire l'autocorrélation sérielle) ; **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 $q(y \to y')$ 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 $\frac{q(y' \to y)}{q(y \to y')}$ **corrige** cette asymétrie dans la probabilité d'acceptation.

**Pourquoi ces corrections sont indispensables.**

Le couple « $\Delta$ du modèle » + « ratio de Hastings » assure la **réversibilité** (détail de balance) de la chaîne et garantit que la **loi stationnaire** visée est bien $\Pr_\theta$. 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)

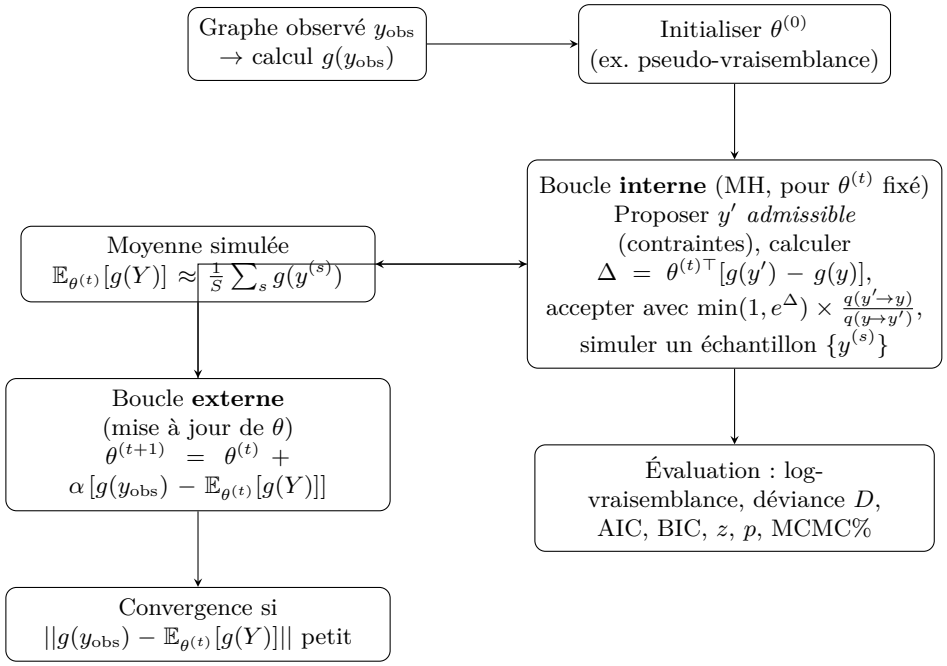
```r
library(ergm)
# Réseau biparti jouet : 4 acteurs -> 2 groupes
m <- matrix(c(1,0,
              1,0,
              0,1,
              0,1), nrow=4, byrow=TRUE)
nw <- network::network(m, bipartite=2, directed=FALSE)

# Modèle 1 : densité (edges)
fit1 <- ergm(nw ~ edges)

# Modèle 2 : densité + distribution de degrés côté groupes (mode 2)
fit2 <- ergm(nw ~ edges + b2degrange(from=2, to=3))

summary(fit1)
summary(fit2)  # comparer log-vraisemblance, AIC, BIC, z, p-values, MCMC%
```

## A.6 Schéma du processus d'estimation




Deux boucles imbriquées : **interne** (MH : échantillonnage pour $\theta$ fixé) et **externe** (mise à jour de $\theta$ jusqu'à convergence).

## 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 **décorréler 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 **écarts** 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.