Réseaux de neurones : le perceptron multi-couches

Fabrice Rossi

http://apiacoa.org/contact.html.

Université Paris-IX Dauphine

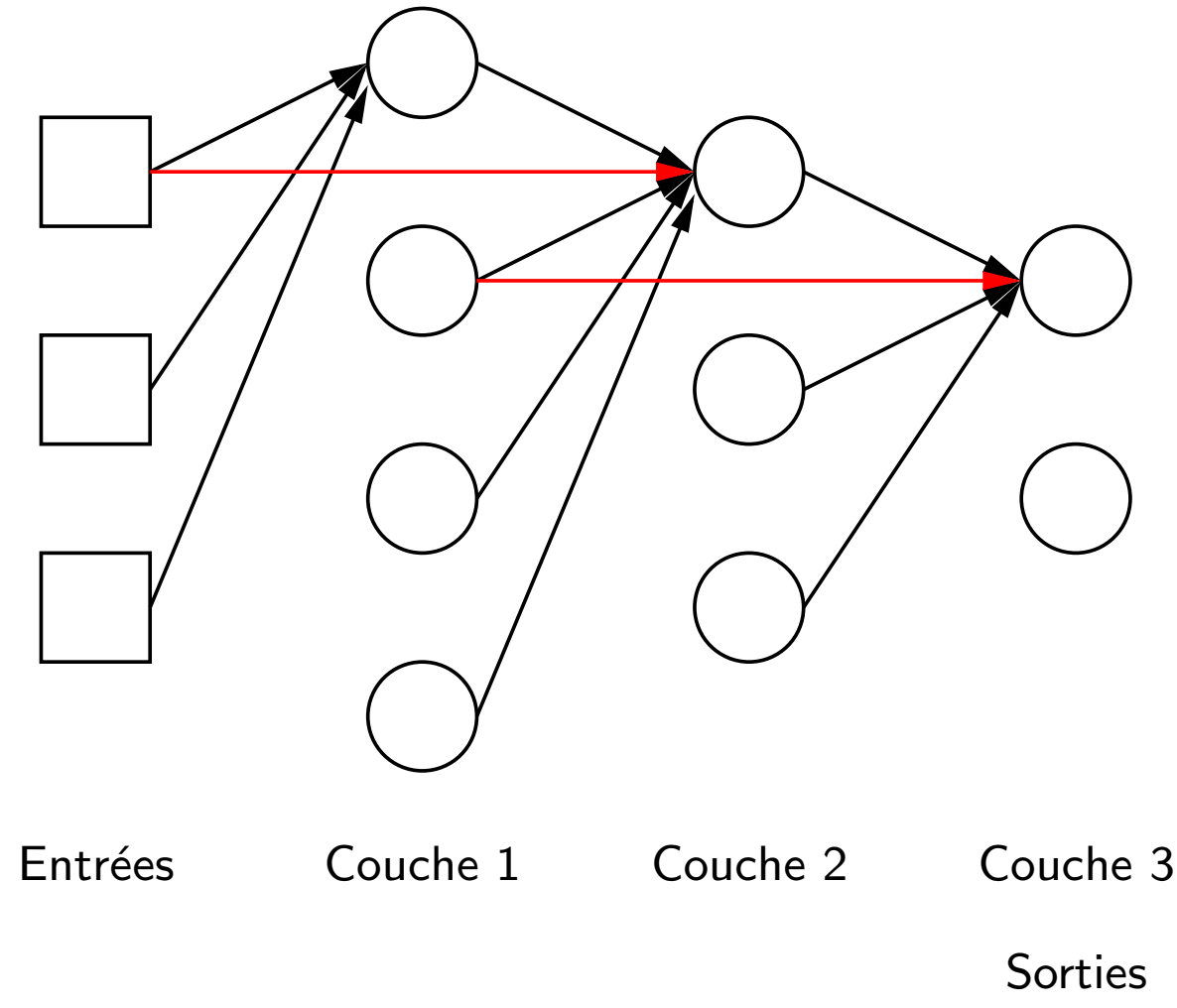
Plan du cours “le perceptron multi-couches”

1. le modèle
2. apprentissage :
   (a) optimisation
   (b) rétro-propagation
   (c) initialisation
   (d) critères d’arrêt
3. en pratique :
   (a) régularisation (*weight decay*, injection de bruit, élagage, etc.)
   (b) sélection de modèle
4. discrimination

Le perceptron multi-couches

- PMC ou MLP (*Multi-Layer Perceptron*)
- réseau organisé en couches :
  - une couche : un groupe de neurones uniformes sans connexion les uns avec les autres
  - réalise une transformation vectorielle :
    - une couche reçoit un vecteur d'entrée et le transforme en vecteur de sortie
    - une couche au moins n'est pas linéaire
    - les dimensions d'entrée et de sortie peuvent être différentes
  - au moins 2 couches (une couche dite "cachée" et une couche de sortie)
  - structure sans cycle (*feed-forward*) : une couche ne peut utiliser que les sorties des couches précédentes

Exemple



Dessin simplifié : il manque des connexions

Neurone élémentaire

Un neurone de PMC est défini par :

1. une fonction d'activation (ou de transfert), $T : \mathbb{R} \to \mathbb{R}$
2. un nombre entier d'entrées

Un neurone à n entrées est associé à $n + 1$ paramètres numériques :

- n poids synaptiques (chaque poids correspond à une flèche dans le dessin)
- un seuil (*threshold*) ou biais (*bias*)

Un neurone à n entrées calcule la fonction suivante :

$$f(x_1, \ldots, x_n) = T\left(\sum_{i=1}^{n} w_i x_i + t\right)$$

Fonction d'activation

Choix pour T :

- fonction non linéaire (au moins dans une des couches)
- propriétés théoriques importantes :
  - continue presque partout (on peut relâcher cette hypothèse)
  - non polynomiale
- propriété pratique importante : dérivable
- choix classiques :
  - $T(x) = \tanh(x)$ tangente hyperbolique (le choix standard)
  - $T(x) = \frac{1}{1+e^{-x}}$ fonction logistique

Le perceptron multi-couches (2)

Mode de calcul :

- la première couche reçoit l'entrée du réseau et produit son résultat :
  - chaque neurone reçoit comme vecteur d'entrée celui du réseau
  - le vecteur de sortie est le regroupement des sorties des neurones
- la deuxième couche reçoit la sortie de la première couche (et éventuellement l'entrée du réseau) et produit son résultat
- etc.

Réseau "classique" :

- connexions complètes entre une couche et sa suivante
- pas d'autres connexions
- tous les neurones d'une même couche ont la même fonction d'activation

Notations vectorielles

PMC à l couches. Pour la couche i :

- $n^{(i)}$: nombre de neurones dans la couche ($n^{(0)}$ nombre d'entrées du réseau)
- $o^{(i)}$: vecteur de sortie de la couche ($o^{(o)}$ vecteur d'entrée du réseau)
- $T^{(i)}$: fonction d'activation vectorielle de la couche
- $W^{(i)}$: matrice des poids synaptiques de la couche
- $t^{(i)}$: vecteur des seuils de la couche

Propagation (pour $i \geq 1$) :

$$\begin{aligned} v^{(i)} &= t^{(i)} + W^{(i)} o^{(i-1)} \\ o^{(i)} &= T^{(i)}(v^{(i)}) \end{aligned}$$

$v^{(i)}$ représente la partie linéaire du traitement de la couche i.

Apprentissage

Un PMC calcule donc une fonction F

- de $\mathbb{R}^{n^{(0)}} \times \mathbb{R}^p$ dans $\mathbb{R}^{n^{(l)}}$
- p, nombre total de paramètres, $p = \sum_{i=1}^{l} n^{(i)}(1 + n^{(i-1)})$
- comme toujours, le but est de trouver w tel que $x \mapsto F(x, w)$ soit un bon modèle des données (les $(x^i, y^i)_{1 \leq i \leq N}$)
- erreur ponctuelle d fonction de $\mathbb{R}^{n^{(l)}} \times \mathbb{R}^{n^{(l)}}$ dans $\mathbb{R}$ (par exemple la distance euclidienne)
- erreur à minimiser

$$E(w) = \sum_{i=1}^{N} d(F(x^i, w), y^i)$$

- cf les cours précédents pour les propriétés théoriques

Approximation universelle

Cas particulier :

- PMC à deux couches
- deuxième couche linéaire ($T^{(2)}(x) = x$)
- $T^{(1)}$ doit vérifier certaines propriétés

Théorème :

- soit G une fonction continue de K (un compact de $\mathbb{R}^n$) dans $\mathbb{R}^q$ et $\epsilon > 0$
- il existe un PMC (cas particulier) et des paramètres numériques pour ce PMC tels que

$$\sup_{x \in K} |F(x, w) - G(x)| < \epsilon$$

$|.|$ est une norme sur $\mathbb{R}^q$ fixée à l'avance.

Résultat non constructif.

Optimisation (Apprentissage)

Problème : comment trouver w pour bien approcher des données ?

- $w \mapsto F(., w)$ n'est pas linéaire
- donc, $w \mapsto E(w)$ n'est pas quadratique
- donc, on ne sait pas minimiser $E(w)$ de façon exacte
- solution : optimisation non linéaire classique
- inconvénients :
  - coûteux (en temps et parfois en mémoire)
  - programmation subtile
  - minimum local, pas global

Algorithmes de descente

Forme générale d'algorithme de minimisation de $E(w)$:

1. on part de w_0 (choisit au hasard)
2. à l'étape i :
   (a) on calcule une direction de descente $d_i \in \mathbb{R}^p$ (dépend de l'algorithme)
   (b) on calcule un pas ϵ_i (dépend de l'algorithme)
   (c) on remplace w_{i-1} par

$$w_i = w_{i-1} + \epsilon_i d_i$$

3. si $E(w_i)$ est satisfaisant, on s'arrête, sinon on retourne en 2

Le gradient simple

L'algorithme le plus mauvais, mais le plus simple :

- d_i est la plus grande pente :

$$d_i = -\nabla E(w_{i-1})$$

- ϵ_i est une "petite" valeur (mais pas trop) et on doit avoir :
  - $\sum_i \epsilon_i = \infty$
  - $\sum_i \epsilon_i^2 < \infty$

  par exemple $\epsilon_i = \frac{1}{i}$
- en théorie, l'algorithme du gradient simple converge
- en pratique, même appliqué à une fonction quadratique, il peut être très lent

Les gradients conjugués

Bien meilleurs (mais plus complexes à programmer) :

- d_i est la plus grande pente corrigée :

$$d_i = -\nabla E(w_{i-1}) + \beta_i d_{i-1}$$

- formule de Fletcher-Reeves :

$$\beta_i = \frac{\nabla E(w_{i-1}).\nabla E(w_{i-1})}{\nabla E(w_{i-2}).\nabla E(w_{i-2})}$$

- formule de Polak-Ribiere (donne souvent de meilleurs résultats) :

$$\beta_i = \frac{\nabla E(w_{i-1}).(\nabla E(w_{i-1}) - \nabla E(w_{i-2}))}{\nabla E(w_{i-2}).\nabla E(w_{i-2})}$$

Les gradients conjugués (2)

- difficulté : le choix de ϵ_i
- il faut que ϵ_i soit une bonne estimation d'un minimiseur de

$$\epsilon \mapsto E(w_{i-1} + \epsilon d_i)$$

- minimisation d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$:
  - algorithme de la section dorée
  - algorithme de Brent
  - etc.
- en théorie, les algorithmes de gradient conjugués convergent
- pour une fonction quadratique sur $\mathbb{R}^p$, ils demandent p itérations
- en pratique, bien plus rapides que le gradient simple

Autres algorithmes

Divers autres algorithmes :

- gradient conjugué formule BFGS : en général meilleur de Polak-Ribiere, mais mise à jour plus complexe
- *scaled conjugate gradient* : amélioration subtile des gradients conjugués, permettant de supprimer la minimisation unidimensionnelle
- quasi-newton (méthode d'ordre deux) BFGS : une des meilleures méthodes, mais coûteuse
- Levemberg-Marquardt : une des meilleures méthodes, mais spécifique à une erreur quadratique

Toujours basé sur le calcul de ∇E.

Algorithmes stochastiques

- basés sur la définition de l'erreur $E(w) = \sum_{i=1}^{N} d(F(x^i, w), y^i)$ par une somme
- idée fondatrice : correction de l'erreur exemple par exemple
- algorithme de base

$$w_i = w_{i-1} - \epsilon_i \nabla d(F(x^{j_i}, w), y^{j_i})$$

- choix des j_i : en général au hasard (uniformément) dans $\{1, \ldots, N\}$
- choix de ϵ_i : mêmes propriétés que pour le gradient simple
- amélioration : terme de moment, i.e.

$$w_i = w_{i-1} - \epsilon_i \nabla d(F(x^{j_i}, w), y^{j_i}) + \mu_i \left(w_{i-1} - w_{i-2}\right)$$

- En pratique : **éviter les algorithmes stochastiques**

La rétro-propagation

- algorithme efficace pour calculer ∇E pour un PMC
- **ce n'est pas un algorithme d'apprentissage !**
- principe :
  - calcul exemple par exemple puis somme des résultats (i.e., $\nabla E = \sum_{i=1}^{N} \nabla d(F(x^i, w), y^i)$)
  - application astucieuse de la règle de dérivation de fonctions composées
  - **mauvaise idée** : calculer $\frac{\partial o^{(i)}}{\partial W^{(i)}}$ et propager
  - bonne idée : calculer $\frac{\partial d}{\partial W^{(i)}}$ et rétro-propager (en arrière dans le réseau)
  - coût : $O(p)$ au lieu de $O(p^2)$.
- notation : $\frac{\partial d}{\partial o_k^{(l)}}$ dérivée de d par rapport à la k-ième sortie du réseau

La rétro-propagation (2)

on a

$$\frac{\partial d}{\partial W_{kr}^{(i)}} = \frac{\partial d}{\partial v_k^{(i)}} \frac{\partial v_k^{(i)}}{\partial W_{kr}^{(i)}}$$

$$\frac{\partial d}{\partial t_k^{(i)}} = \frac{\partial d}{\partial v_k^{(i)}} \frac{\partial v_k^{(i)}}{\partial t_k^{(i)}}$$

et donc

$$\frac{\partial d}{\partial W_{kr}^{(i)}} = \frac{\partial d}{\partial v_k^{(i)}} o_r^{(i-1)}$$

$$\frac{\partial d}{\partial t_k^{(i)}} = \frac{\partial d}{\partial v_k^{(i)}}$$

La rétro-propagation (3)

Pour la dernière couche

$$\frac{\partial d}{\partial v_k^{(l)}} = \frac{\partial d}{\partial o_k^{(l)}} T_k^{(l)\prime}(v_k^{(l)})$$

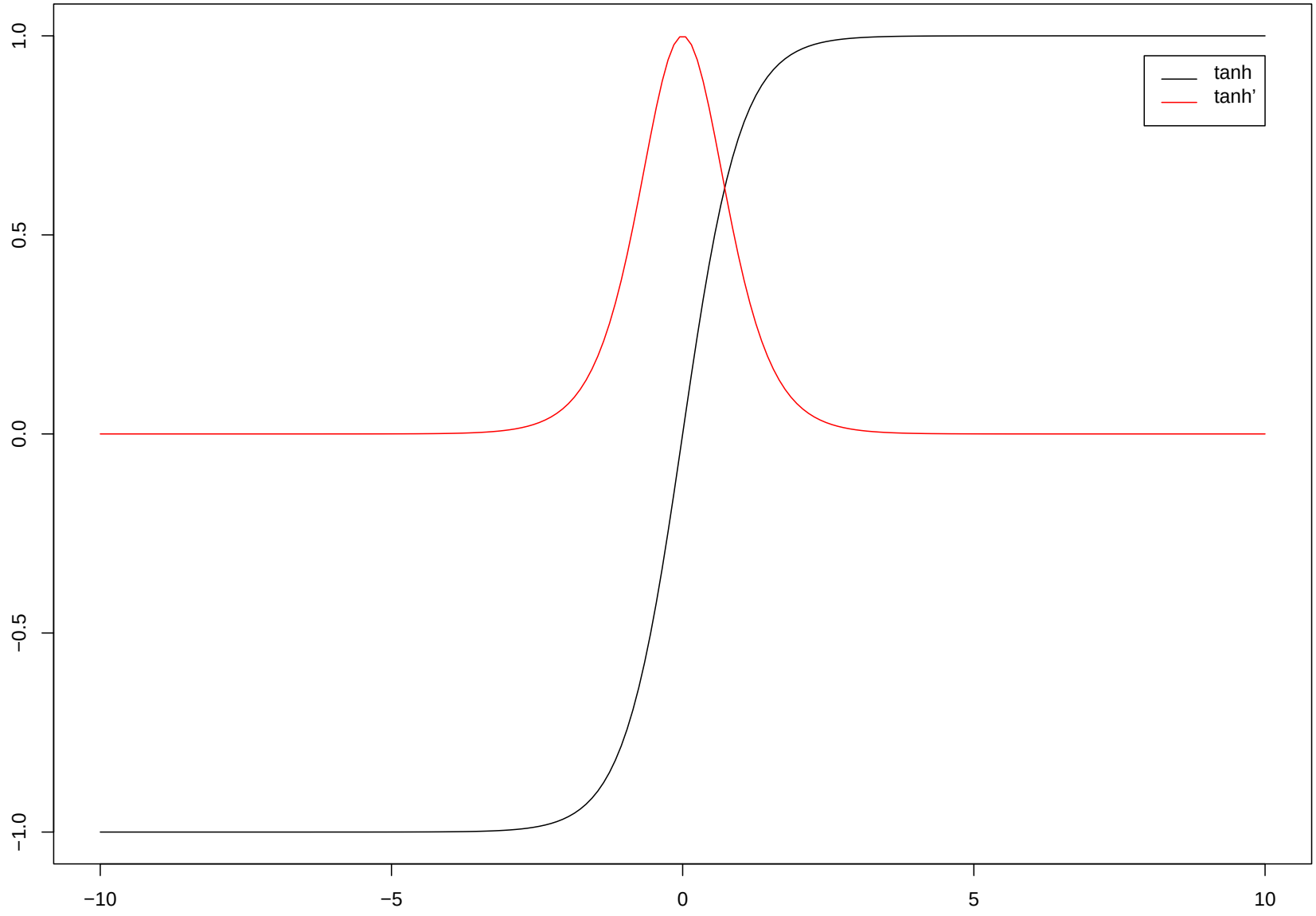
Pour les autres couches

$$\frac{\partial d}{\partial v_k^{(i)}} = \sum_{r=1}^{n(i+1)} \frac{\partial d}{\partial v_r^{(i+1)}} W_{rk}^{(i)} T_k^{(i)\prime}(v_k^{(i)})$$

Initialisation

- comment choisir w_0 ?
  - problème lié à $T^{(1)}$ quand on prend $\tanh$ ou la fonction logistique :
    - dès que x est “grand”, $\tanh'(x)$ est “nul”
    - exemple $\tanh'(10) = 8\ 10^{-9}$
    - donc les dérivées par rapport aux paramètres de la première couche sont nulles $\Rightarrow$ impossible à régler par descente de gradient !
  - si les paramètres sont “nuls”, les dérivées le sont aussi
- donc il faut des paramètres “moyens” :
  - on suppose les entrées centrées-réduites
  - on choisit les paramètres de la couche i selon une distribution gaussienne d’écart-type $\frac{1}{\sqrt{n^{(i-1)}}}$
  - diverses variantes possibles (seuils fixés à zéro, initialisation géométrique, etc.)

Tangente hyperbolique



Arrêt de l'apprentissage

- Critères classiques :
  1. borne sur le temps de calcul (i.e., nombre d'itérations de l'algorithme)
  2. valeur à atteindre (on s'arrête quand l'erreur passe en dessous d'un seuil)
  3. vitesse de progression (on s'arrête quand l'erreur ne diminue plus assez ou quand le vecteur de paramètres ne change plus)
- en général, on combine 1 et 3
- une technique de régularisation : arrêt prématuré (*early stopping*) :
  - éviter le sur-apprentissage en arrêtant l'algorithme avant d'avoir atteint le minimum
  - utilise un ensemble d'exemples dit de validation : on s'arrête quand l'erreur remonte "trop" sur cet ensemble

Remarques pratiques

- les PMC sont plus sensibles que méthodes linéaires : il faut centrer et réduire les données :
  - chaque entrée doit avoir une moyenne nulle
  - chaque entrée doit avoir un écart-type de un
- on peut aussi centrer et réduire les sorties (en régression)
- on peut aussi réduire le nombre d'entrées (et donc le nombre de paramètres) pour accélérer l'apprentissage :
  - analyse en composantes principales
  - autres techniques (modélisation linéaire et test de Fisher par exemple)
- les algorithmes d'optimisation donnent un minimum **local** : on peut essayer plusieurs points de départ pour trouver un minimum global

Application en régression

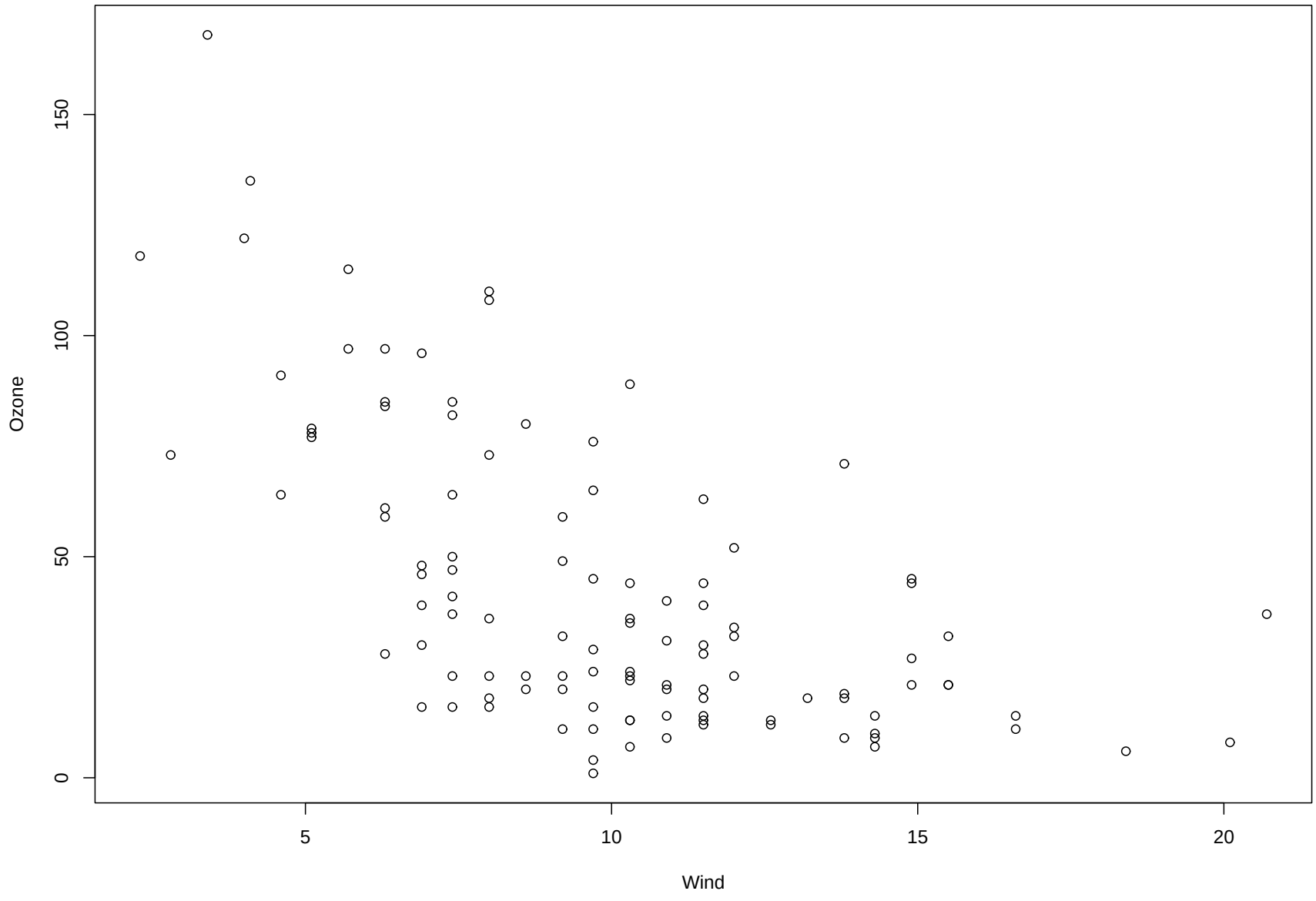
Même principe que les modèles linéaires (généralisés) :

- Critère d'erreur : moindres carrés, i.e.

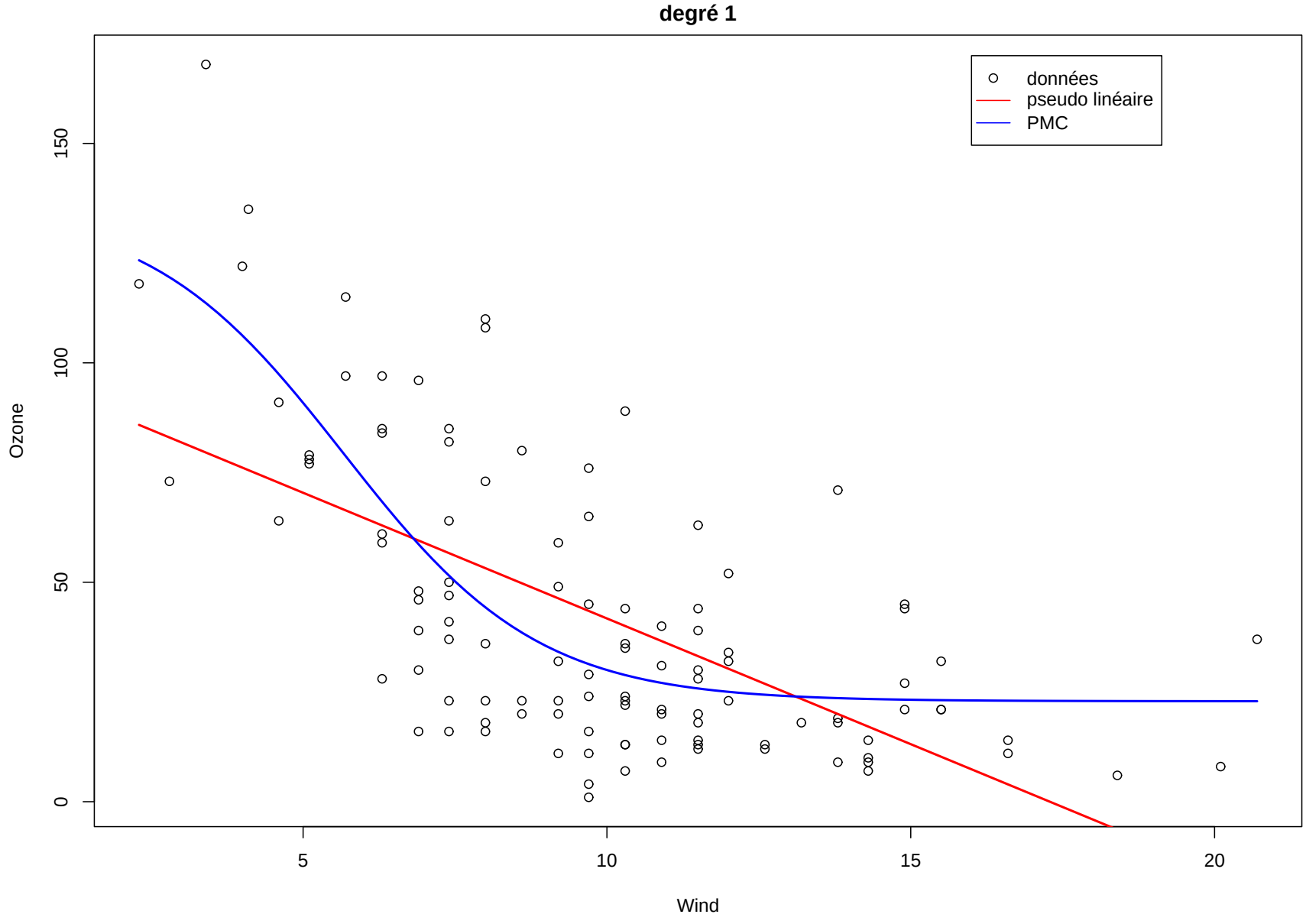
$$E(w) = \sum_{i=1}^{N} \left\| F(x^i, w), y^i \right\|^2$$

- Bonnes propriétés théoriques :
  - convergence des paramètres optimaux empiriques vers les paramètres optimaux théoriques
  - maximum de vraisemblance dans le cas d'une erreur gaussienne
  - estimation de $E(Y|X)$
- Problèmes classiques : évaluation et sélection de modèle (nombre de neurones et nombre de couches)

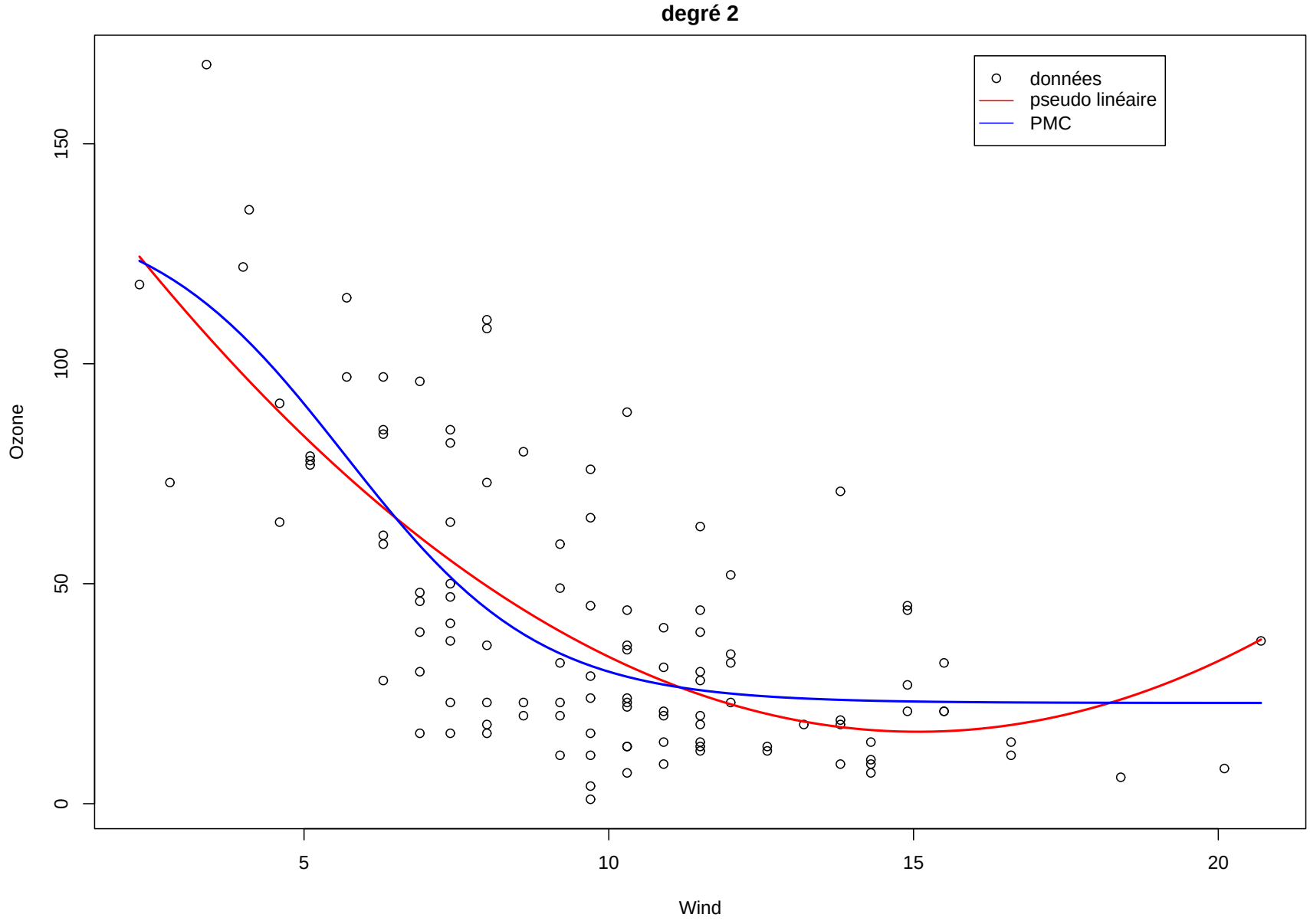
Prédire l'ozone à partir du vent



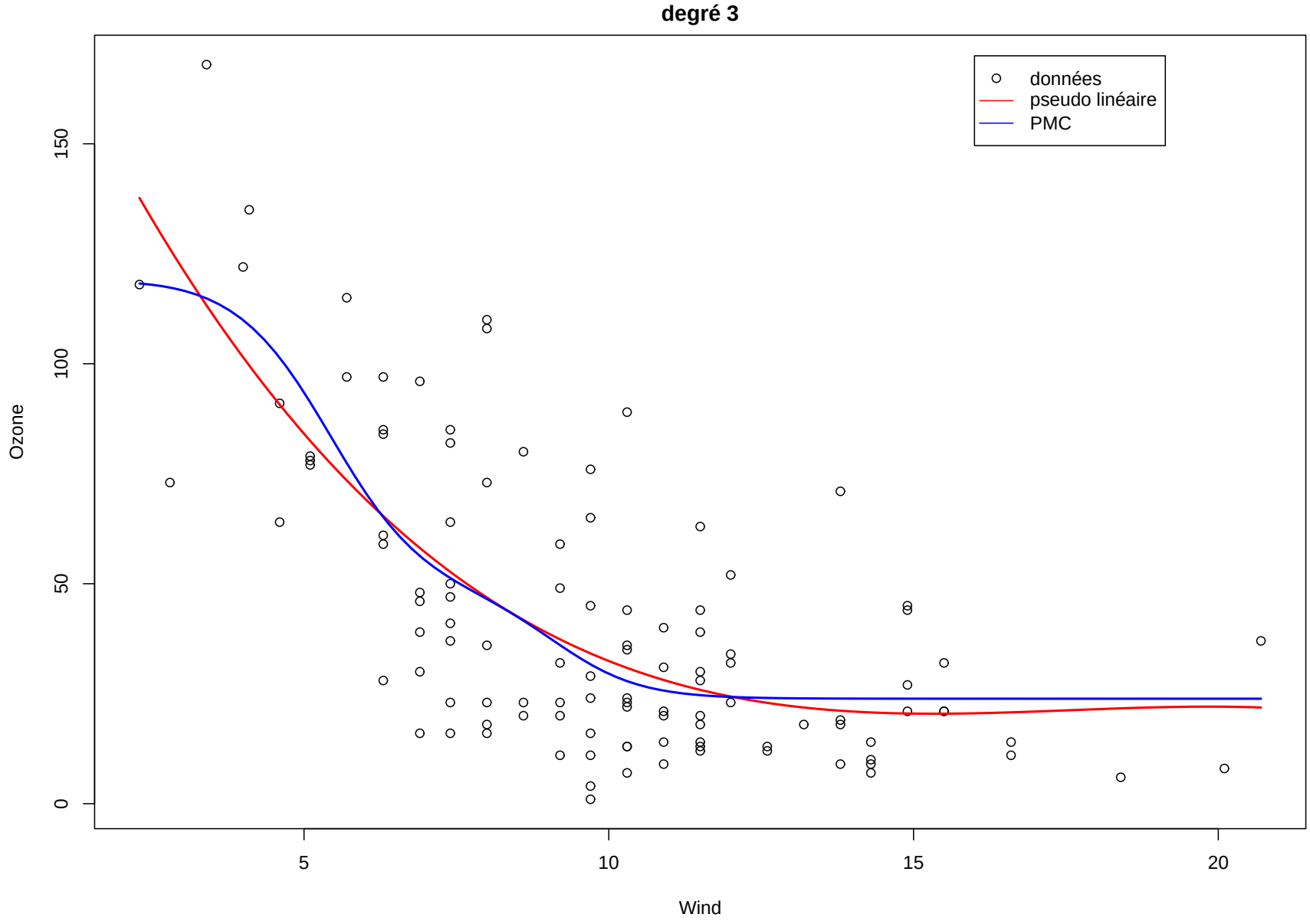
Prédire l'ozone à partir du vent



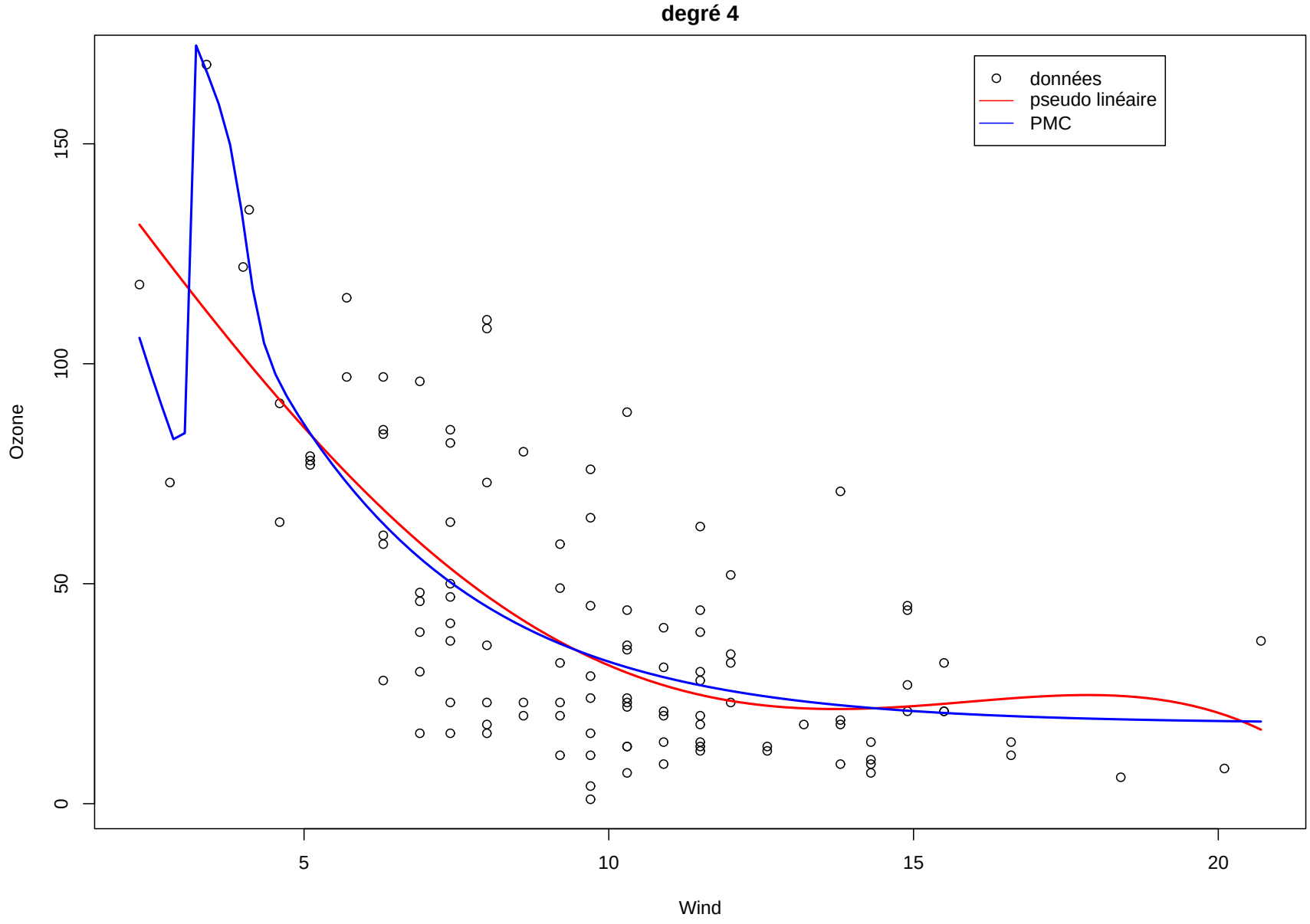
Prédire l'ozone à partir du vent



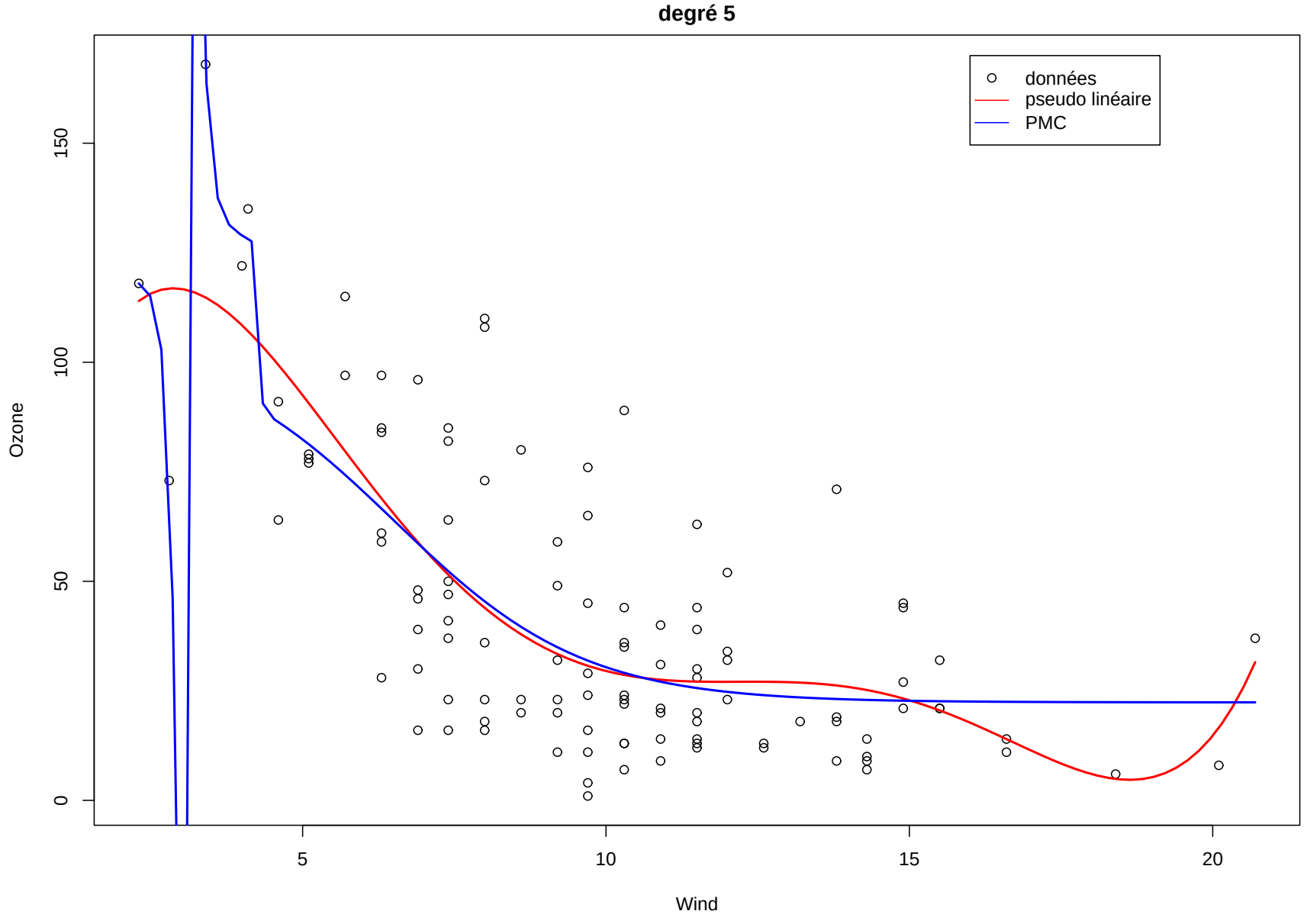
Prédire l'ozone à partir du vent



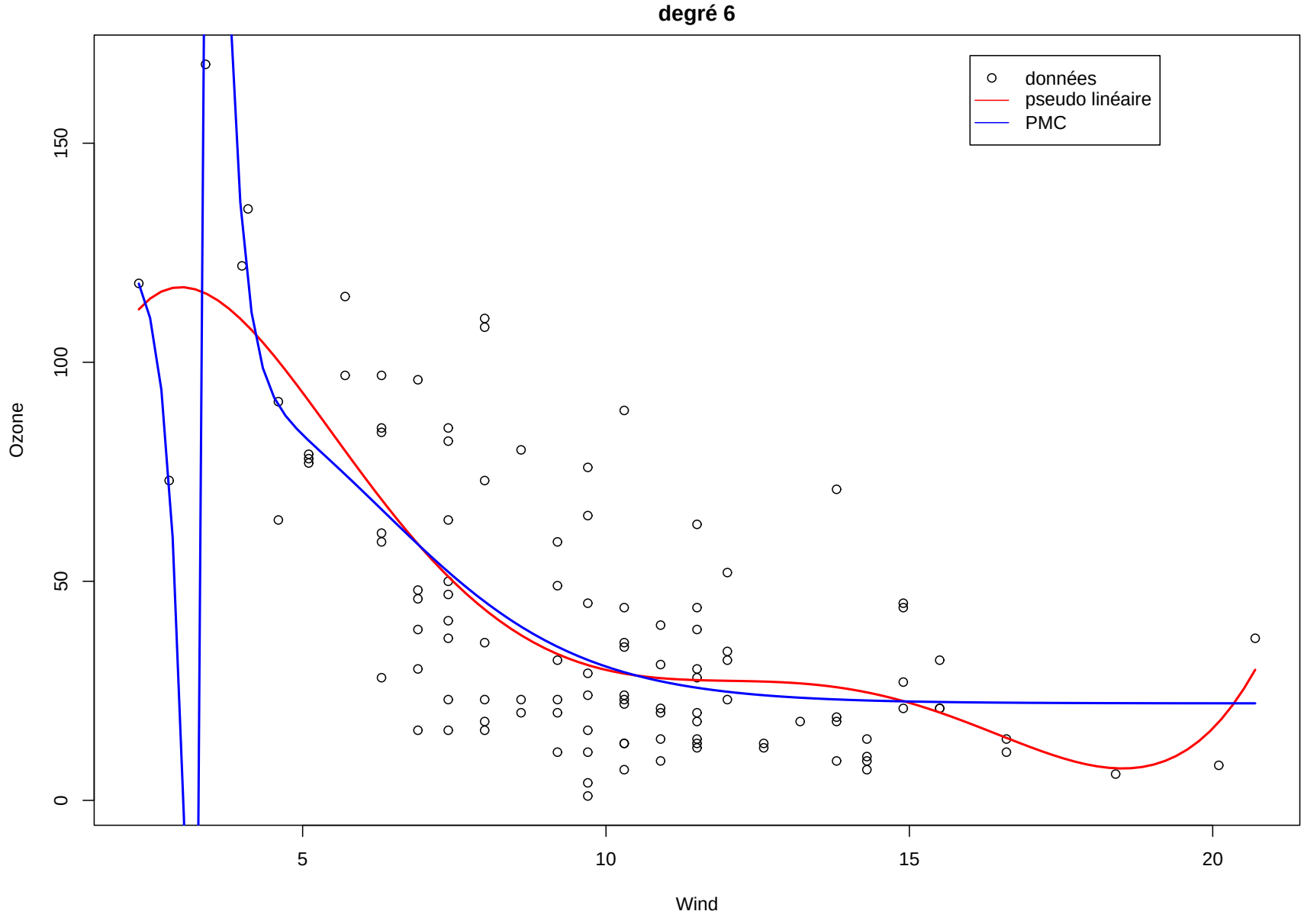
Prédire l'ozone à partir du vent



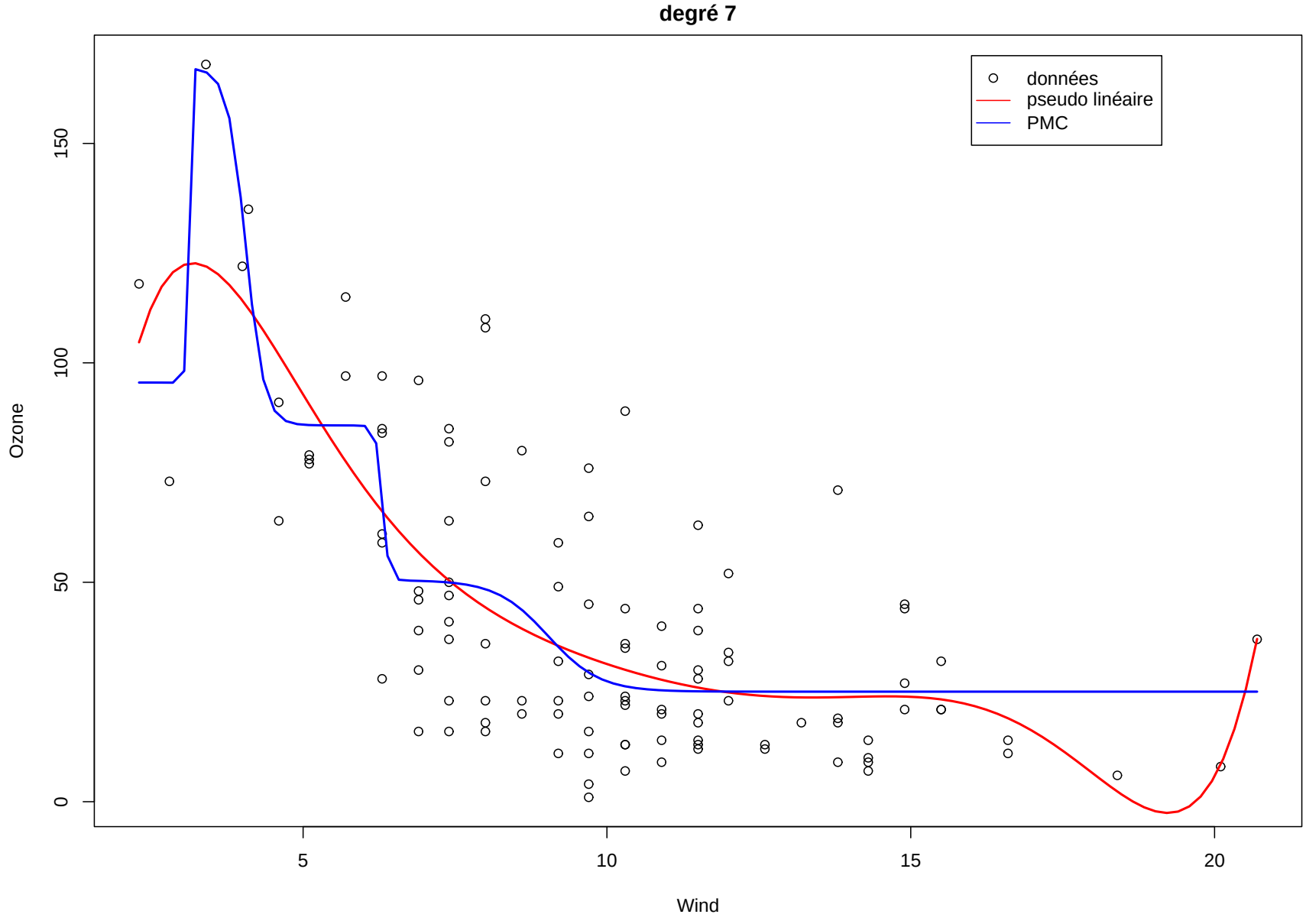
Prédire l'ozone à partir du vent



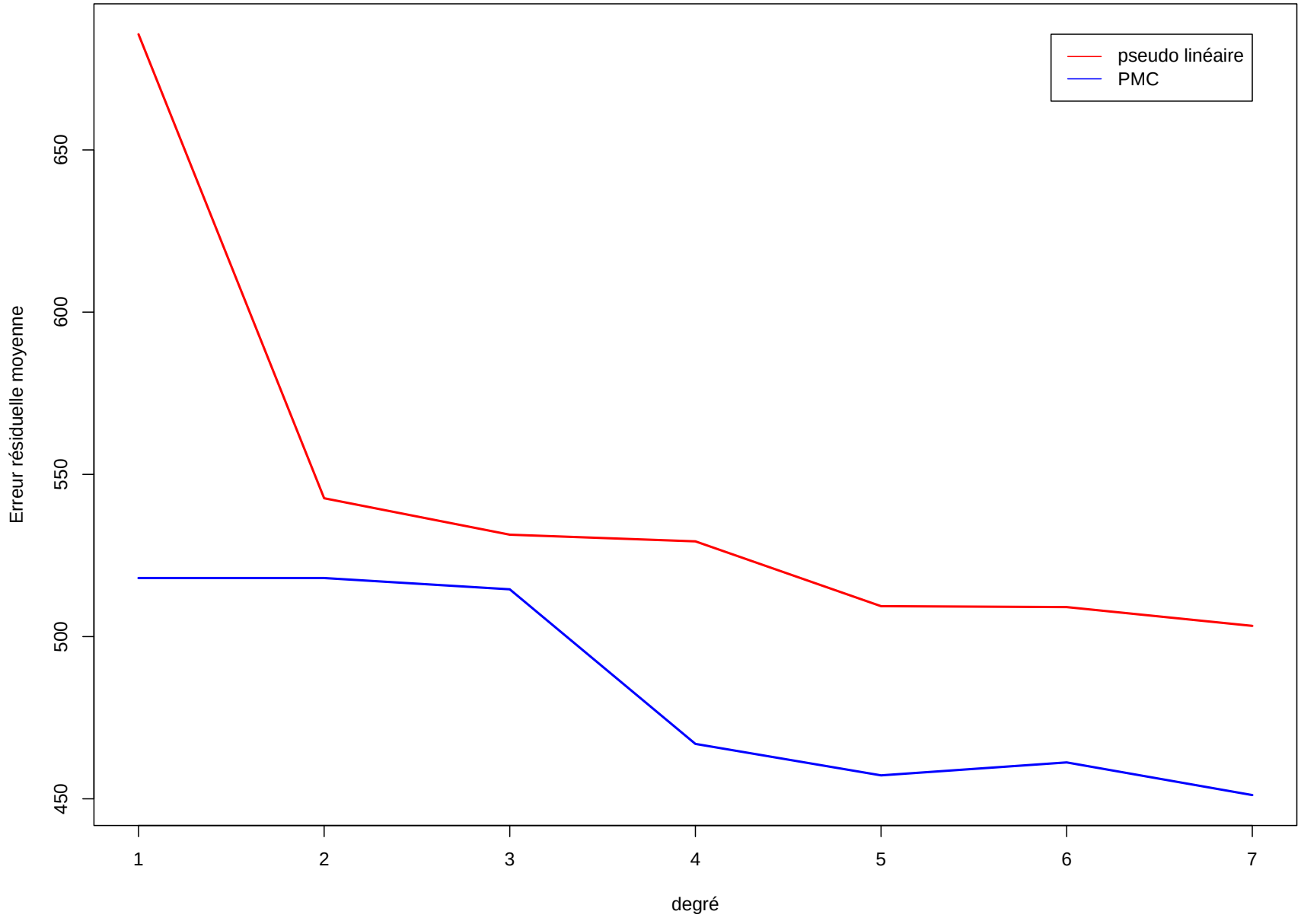
Prédire l'ozone à partir du vent



Prédire l'ozone à partir du vent



Evolution de l'erreur



Contrôle de complexité

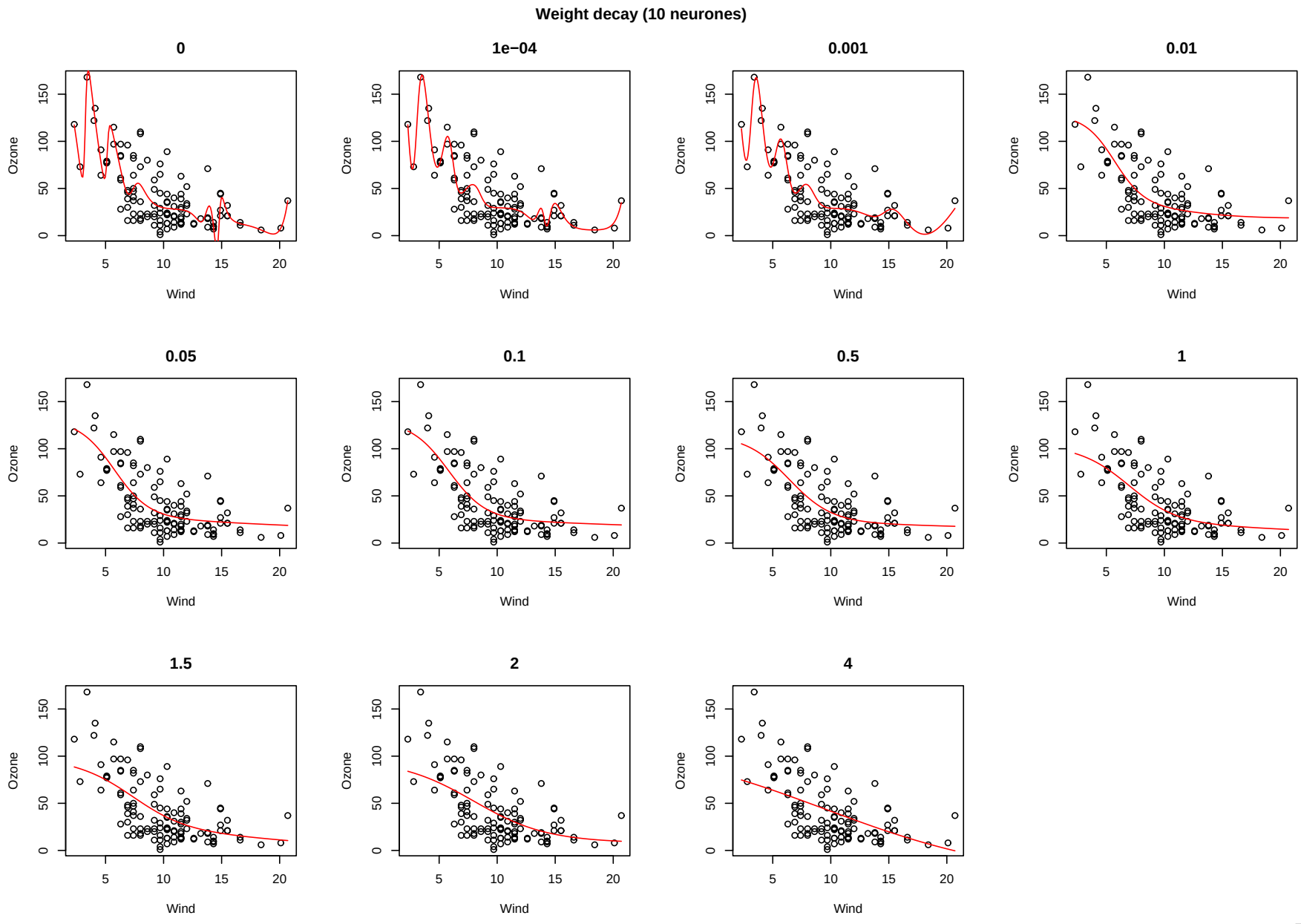
- Solutions classiques par pénalisation :
  - sur les paramètres (*weight decay*) :
    - l'approche la plus utilisée
    - nécessite une normalisation des entrées
    - on a intérêt à utiliser un coefficient de pénalisation par couche
  - sur les dérivées : lourd à mettre en place
- Arrêt prématuré : "équivalent" au *weight decay*
- Injection de bruit :
  - consiste à modifier les vecteurs d'entrée en leur ajoutant du bruit (différent à chaque présentation du vecteur)
  - "équivalent" à une régularisation par des dérivées
  - facile à mettre en œuvre
- Élagage : suppression de connexion *a posteriori*
- Dans tous les cas : sélection d'hyper-paramètres !

En pratique

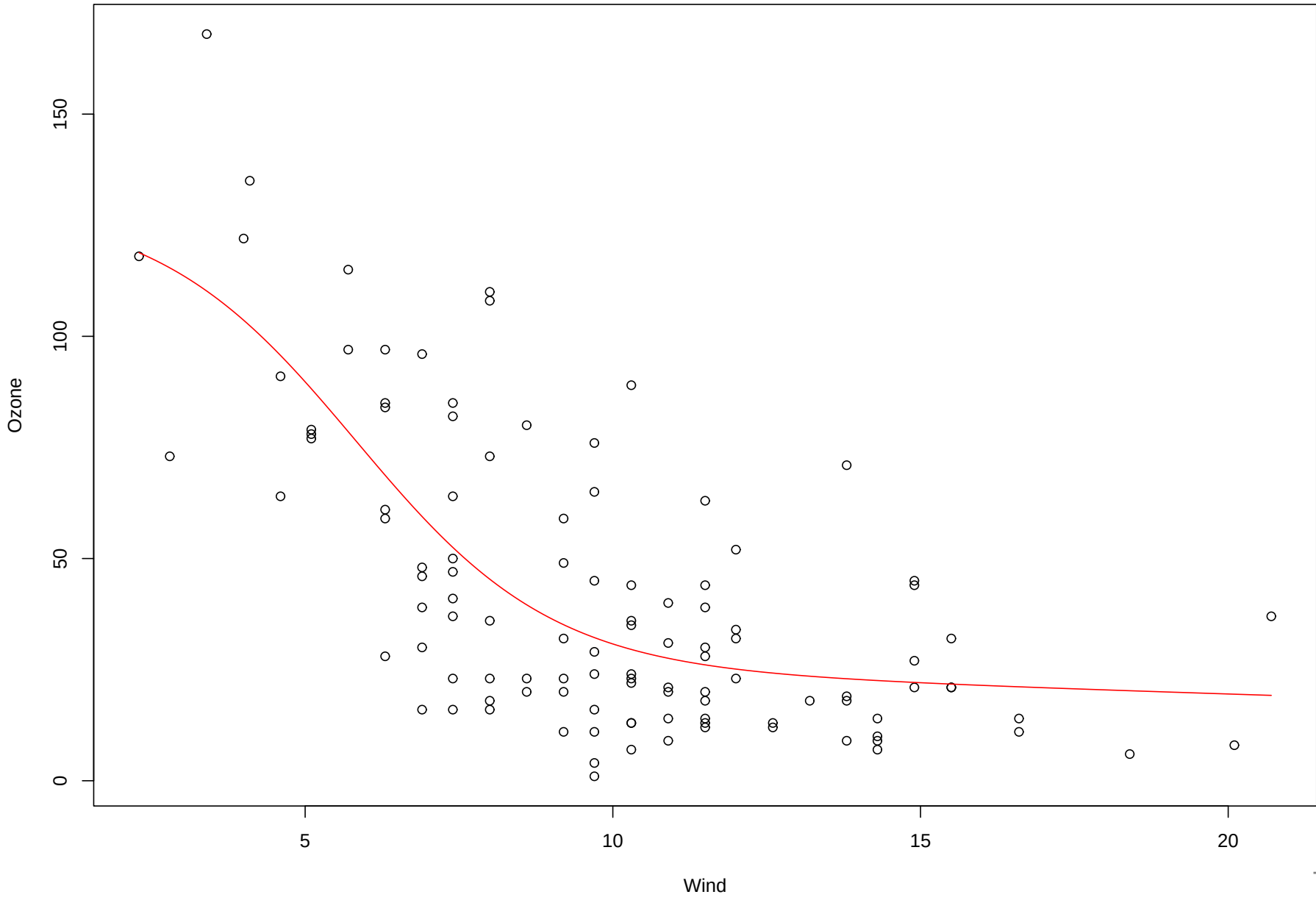
Il faut faire impérativement de la sélection de modèle et de la régularisation :

- Régularisation :
  - éviter l'arrêt prématuré qui demande un découpage des données
  - utiliser une régularisation comme le *weight decay* ou l'injection de bruit
  - idéalement, utiliser une régularisation basée sur les dérivées
  - on peut aussi élaguer le modèle
- Sélection de modèle :
  - éviter la validation par découpage
  - utiliser au minimum un critère de complexité (type BIC)
  - idéalement, procéder par rééchantillonnage (validation croisée ou *bootstrap*)

Ozone avec *Weight decay*



Meilleur modèle (validation croisée)



Injection de bruit

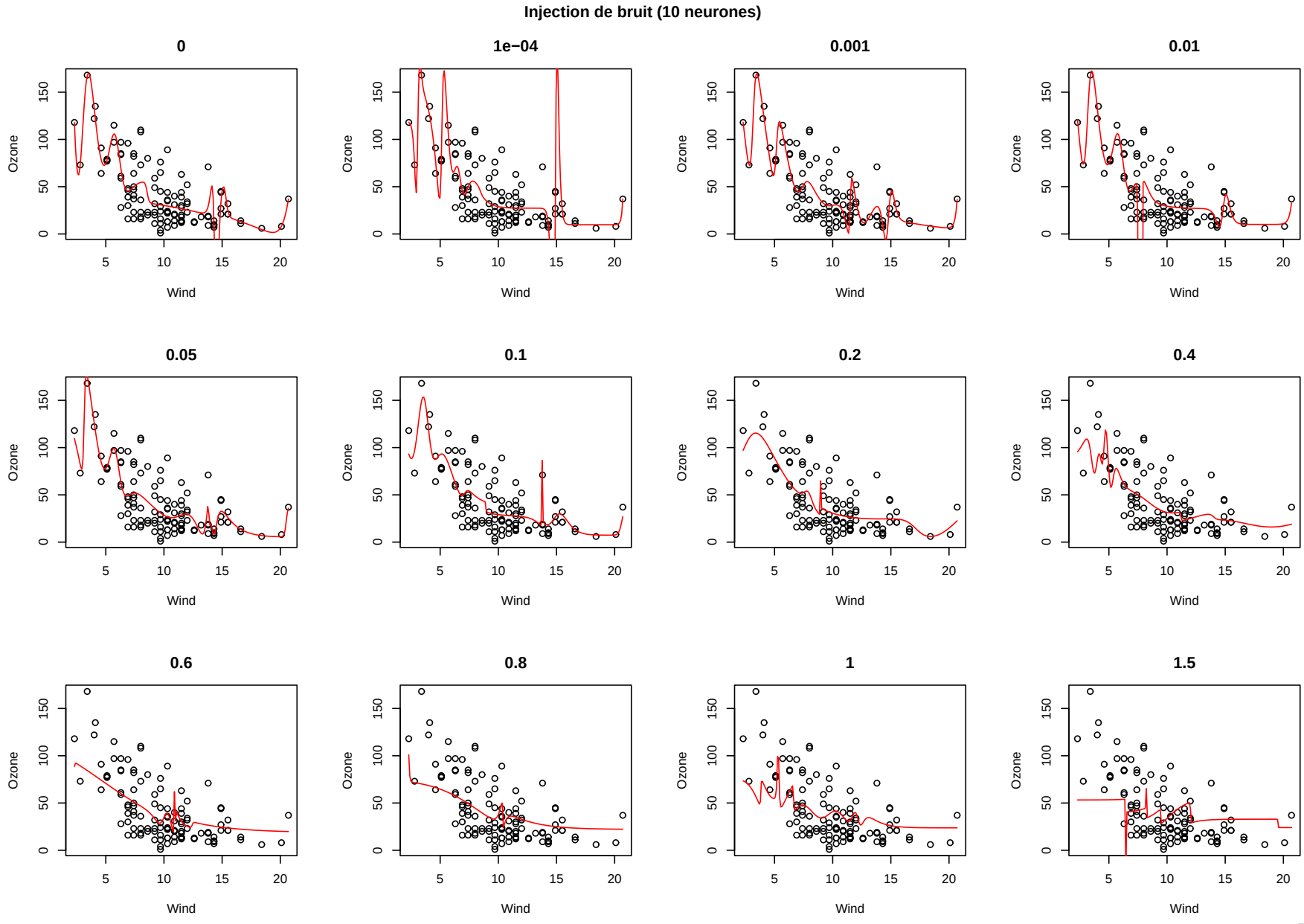
- Pertuber les observations pour rendre la fonction calculée $F(w, x)$ moins sensible à des petites variations de x
- En pratique, on remplace chaque (x^i, y^i) par s répliques :

$$(x^i(t) = x^i + \nu\epsilon^i(t), y^i),$$

où $\epsilon^i(t)$ est gaussien (moyenne nulle, écart type 1)

- Deux hyper-paramètres
  - s : qualité de l'injection de bruit, compromis entre l'insensibilité au tirage aléatoire et le temps de calcul
  - ν : importance de la régularisation
- On peut déterminer ν par validation croisée (très coûteux !)

Ozone avec injection de bruit



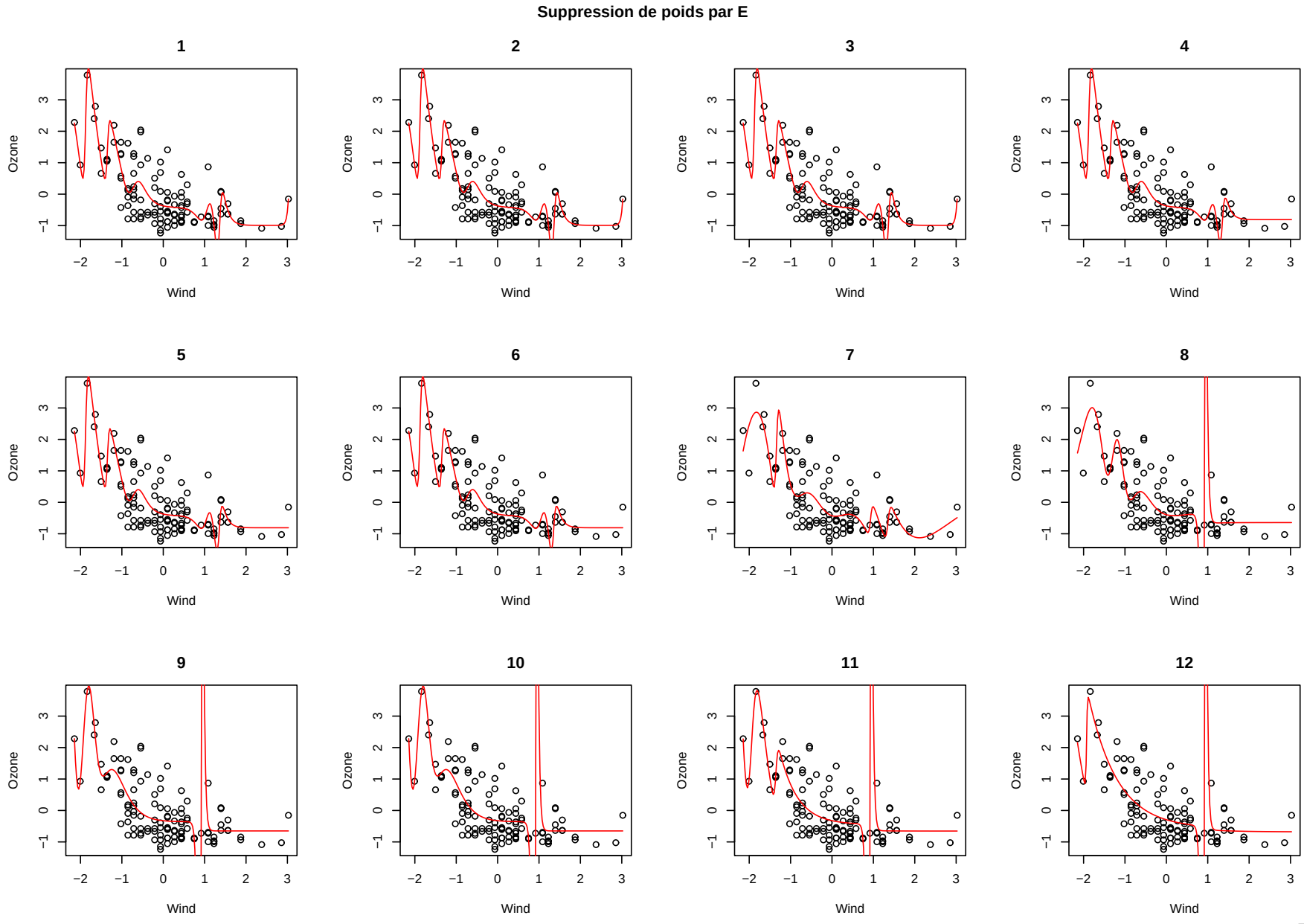
Élagage

- Idée de base : entraîner un “gros” PMC, puis supprimer les connexions (ou les neurones) inutiles
- Algorithme :
  1. entraîner le PMC
  2. supprimer certains poids “inutiles”
  3. retourner en 1 sauf si un critère d’arrêt est vérifié
- Critères de qualité des poids :
  - basique : qualité de w_i donnée par l’accroissement de $E(w)$ quand on remplace w_i par 0
  - *Optimal Brain Damage* : qualité de w_i donnée par

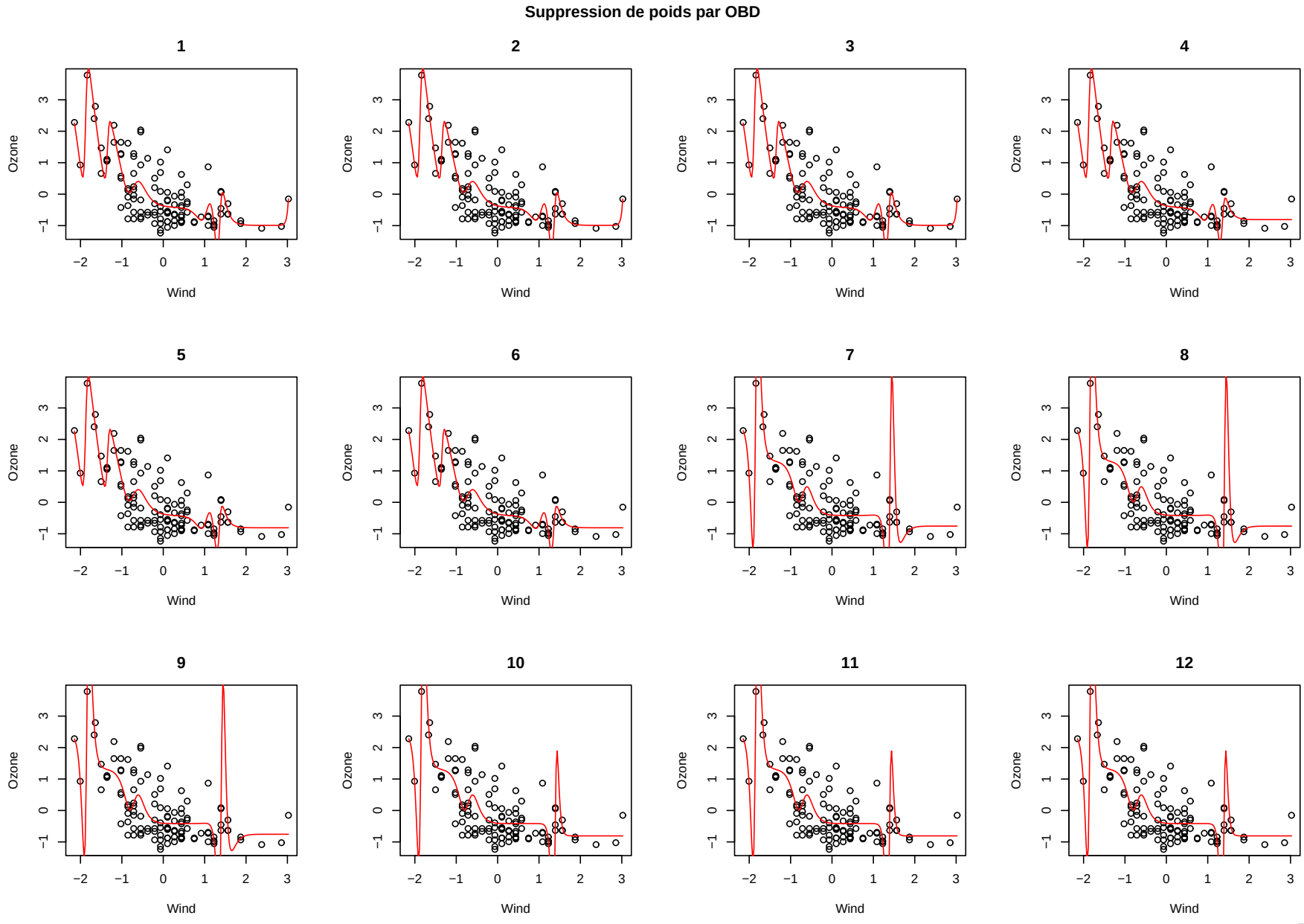
$$\frac{\partial^2 E}{\partial w_i^2} w_i^2$$

Approximation de E au voisinage de son minimum.

Ozone avec la méthode de base



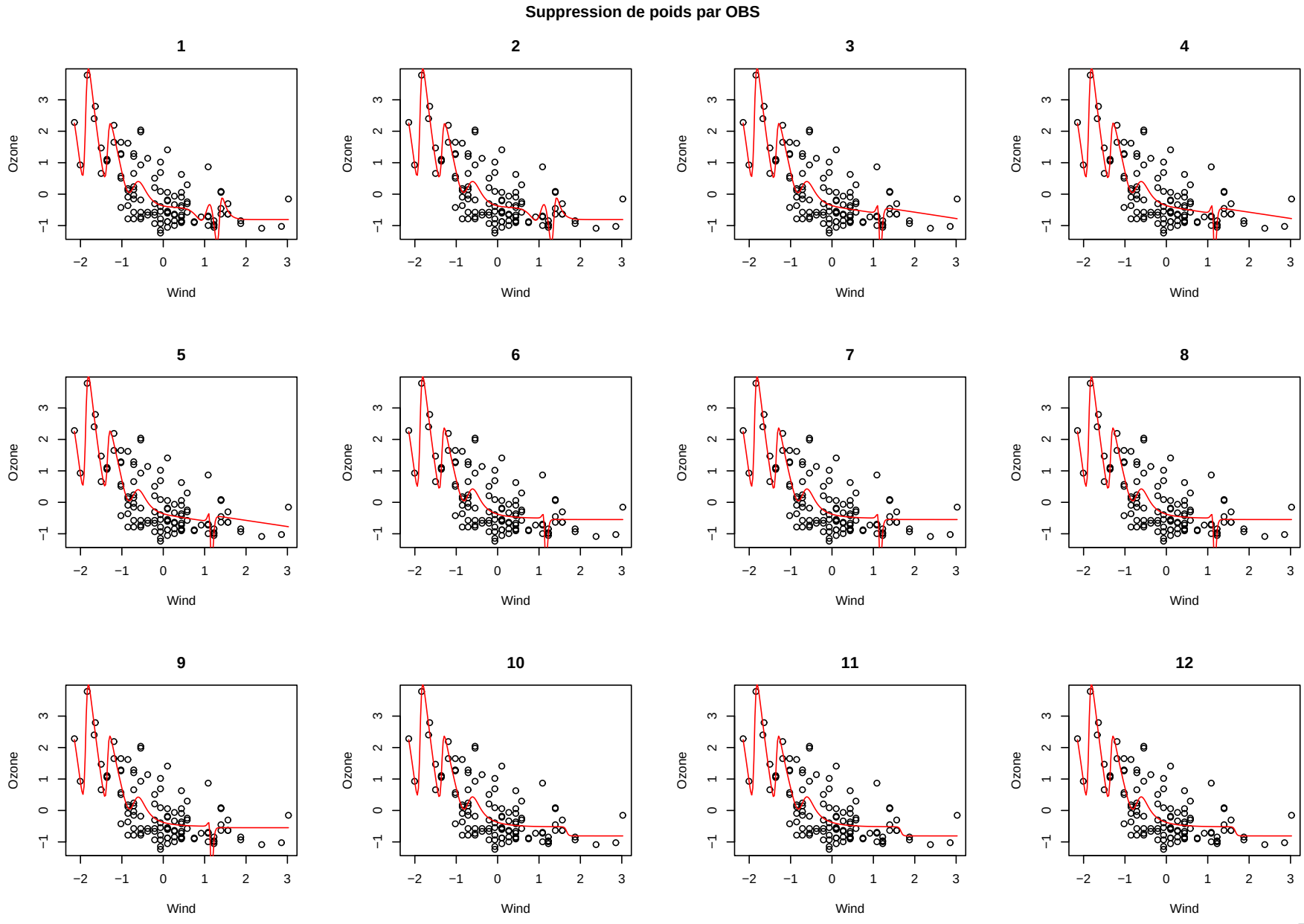
Ozone avec *Optimal Brain Damage*



Élagage (2)

- *Optimal Brain Surgeon* :
  - *Optimal Brain Damage* est basé sur une approximation diagonale du Hessien : en général, cette approximation est très mauvaise
  - idées de *Optimal Brain Surgeon* :
    - ne pas faire d'approximation diagonale
    - tenir compte du réapprentissage (avec $w_i = 0$) pour estimer la qualité d'un poids
  - qualité de w_i donnée par $\frac{w_i^2}{a_{ii}}$, où a_{ii} est le terme de place (i, i) de l'inverse de la Hessienne de E
  - de plus l'algorithme calcule l'inverse grâce à une approximation obtenue itérativement (le temps de calcul reste raisonnable)
  - plus coûteux que *Optimal Brain Damage* mais donne de meilleurs résultats

Ozone avec *Optimal Brain Surgeon*



Élagage (3)

- élagage de neurones :
  - approche basique (effet de la suppression)
  - diverses méthodes par approximation
- méthodes statistiques classiques :
  - basées sur l'étude asymptotique de la distribution des poids (théorème de la limite centrale)
  - quelques difficultés théoriques (problème d'identifiabilité)
- méthodes bayésiennes :
  - estimer la distribution des poids
  - quelques difficultés pratiques et théoriques

Application en discrimination

- Classe représentée par codage disjonctif complet
- Mesure de l'erreur :
  - Erreur quadratique : même approche que pour les modèles linéaires généralisés
  - Entropie croisée, correspondant au maximum de vraisemblance, pour deux classes (codage 0/1) :

$$E(w) = -\sum_{i=1}^{N} \left(y^i \ln \frac{F(x^i, w)}{y^i} + (1 - y^i) \ln \frac{1 - F(x^i, w)}{1 - y^i} \right)$$

On a alors intérêt à utiliser pour $T^{(2)}$ la fonction logistique (sortie comprise entre 0 et 1, et simplification des calculs)

Application en discrimination (2)

- Entropie croisée pour p classes (codage disjonctif complet) :

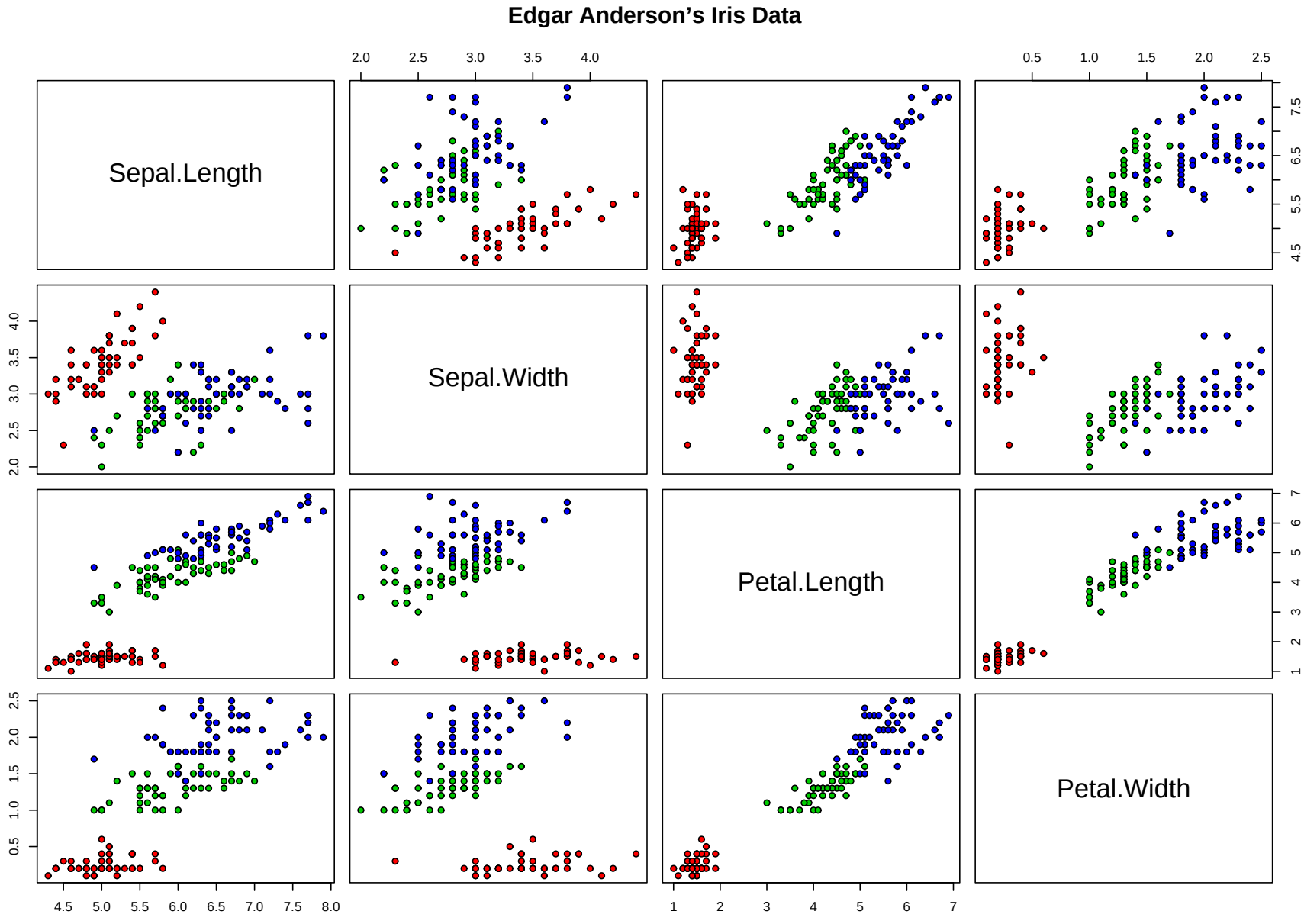
$$E(w) = -\sum_{i=1}^{N}\sum_{j=1}^{q} y_j^i \ln \frac{F(x^i, w)_j}{y_j^i}$$

- On a intérêt à utiliser un *softmax*, c'est-à-dire définir $o^{(2)}$ par :

$$o_j^{(2)} = \frac{e^{v_j^{(2)}}}{\sum_{k=1}^{q} e^{v_k^{(2)}}}$$

- Avantages :
  - $\sum_{j=1}^{q} o_j^{(2)} = 1$
  - $0 \leq o_j^{(2)} \leq 1$
  - simplification des calculs

Les iris d'Anderson



Les iris d'Anderson (2)

Traitement effectué :

- PMC à 10 neurones
- sortie *softmax* (codage disjonctif complet)
- erreur entropie croisée
- régularisation par *weight decay* déterminé par validation croisée (0.05)
- matrice de confusion (sur les données d'origine)

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.96 & 0.02 \\ 0 & 0.04 & 0.98 \end{pmatrix}$$

- il faut d'autres données pour évaluer le modèle

Régression évoluée

- Modèle de régression classique : $E(Y|X=x)$, i.e., une seule valeur pour un x donné
- Modèle évolué : estimer $p(y|X=x)$, la densité conditionnelle de Y sachant $X=x$
  - modèle additif : $p(y|X=x) = \sum_{j=1}^{s} \alpha_j(x)\phi_j(y,x)$ (les ϕ_j sont des noyaux)
  - par exemple :

$$\phi_j(y,x) = \frac{1}{(2\pi)^{\frac{p}{2}}\sigma_j^p(x)} \exp^{-\frac{\|y-\mu_j(x)\|^2}{2\sigma_j^2(x)}}$$

  - trouver $\alpha_j(x)$, $\sigma_j(x)$, $\mu_j(x)$ par un PMC

Régression évoluée (2)

- il faut utiliser un *softmax* pour les $\alpha_j(x)$
- pour les $\sigma_j(x)$ l'idéal est d'utiliser une activation exponentielle (ce qui donne $\sigma_j(x) > 0$)
- pour les $\mu_j(x)$, on utilise une activation linéaire
- on estime le modèle par maximum de vraisemblance, i.e., on minimise

$$E = -\sum_{i=1}^{N} \ln \left(\sum_{j=1}^{s} \alpha_j(x^i) \phi_j(y^i, x^i) \right)$$

- on obtient ∇E par rétro-propagation