Réseaux de neurones : modèle linéaire généralisé

Fabrice Rossi

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

Université Paris-IX Dauphine

Plan du cours “modèle linéaire généralisé”

1. Le modèle linéaire généralisé (expression neuronale)
2. Estimateur des moindres carrés
3. Bases fonctionnelles
4. Problème des grandes dimensions
5. Régularisation
6. Discrimination

Modèle linéaire

Le modèle linéaire (en fait affine, $y = Ax + b$) est doublement linéaire :

1. $x \mapsto y = Ax + b$ est une fonction affine
2. $(A, b) \mapsto (x \mapsto y = Ax + b)$ est une fonction linéaire

Conséquences :

1. le modèle est limité : un lien non linéaire entre les sorties et les entrées ne peut pas être découvert
2. le modèle est facile à optimiser : calcul matriciel

Amélioration : garder la deuxième linéarité en supprimant la première.

Principe de base

Combinaison linéaire de fonctions de base (exemple de $\mathbb{R}^n$ dans $\mathbb{R}$, se généralise à une cible dans $\mathbb{R}^p$ sans difficulté) :

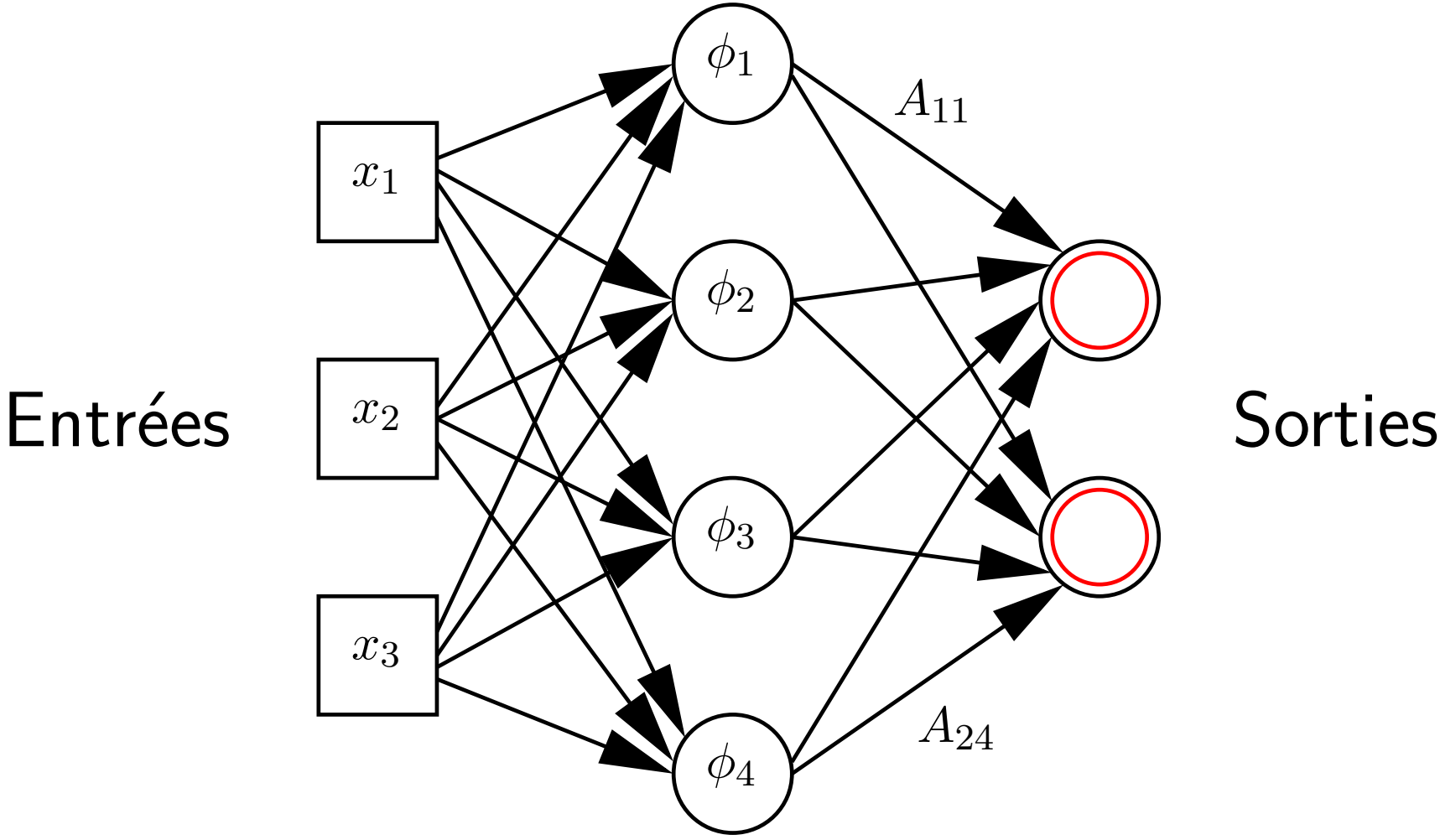
$$y = \sum_{i=1}^{k} a_i \phi_i(x) + b$$

Chaque ϕ_i est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$, “bien choisie”.
Exemple pour $n = 1$, modèle polynomial :

$$y = \sum_{i=1}^{k} a_i x^i + b$$

L’association paramètres vers fonction modèle reste linéaire.

Réseau à deux couches



Attention, la première couche n'est pas "réglable".

Réseau à deux couches (2)

Neurone de la première couche :

- pas de paramètre numérique réglable
- sortie : $\phi(x_1, \ldots, x_n)$

Neurone de la deuxième couche :

- sortie : $f(x_1, \ldots, x_k) = T\left(\sum_{i=1}^{k} a_k x_k + b\right)$
- on règle les paramètres numériques

Matriciellement :

$$f(x) = \mathbb{T}(A\Phi(x) + b),$$

avec

$$\Phi(x) = \begin{pmatrix} \phi_1(x_1, \ldots, x_n) \\ \cdots \\ \phi_k(x_1, \ldots, x_n) \end{pmatrix}$$

Régression (Rappel)

En régression, on cherche à écrire $Y \simeq \mathbb{T}(A\Phi(X) + b)$ et donc à choisir A et b dans ce but. Or, on a vu que :

- on peut approcher $E(Y|X)$ en déterminant A et b qui minimisent l'**erreur quadratique** entre y^l et $\mathbb{T}(A\Phi(x^l) + b)$
- si on suppose que les erreurs d'observation sont gaussiennes avec une variance constante, l'estimation de A et b par moindres carrés correspond au **maximum de vraisemblance**

Moralité : comme pour le modèle linéaire, on cherche un estimateur de A et b correspondant aux moindres carrés.

Moindres carrés

On cherche donc A et b qui minimisent

$$\mathcal{E}(A,b) = \sum_{l=1}^{N} \left\| \mathbb{T}(A\Phi(x^l) + b) - y^l \right\|^2$$

Quand $\mathbb{T} = I$, on procède comme dans le cas linéaire. On définit

$$\begin{aligned} C &= (Ab) \\ Z &= \begin{pmatrix} \Phi(x^1) & \dots & \Phi(x^N) \\ 1 & \dots & 1 \end{pmatrix} \\ Y &= (y^1 \dots y^N) \end{aligned}$$

Moindres carrés (2)

On doit résoudre :

$$ZZ^T C^T = ZY^T$$

Comme dans le cas linéaire, on peut résoudre :

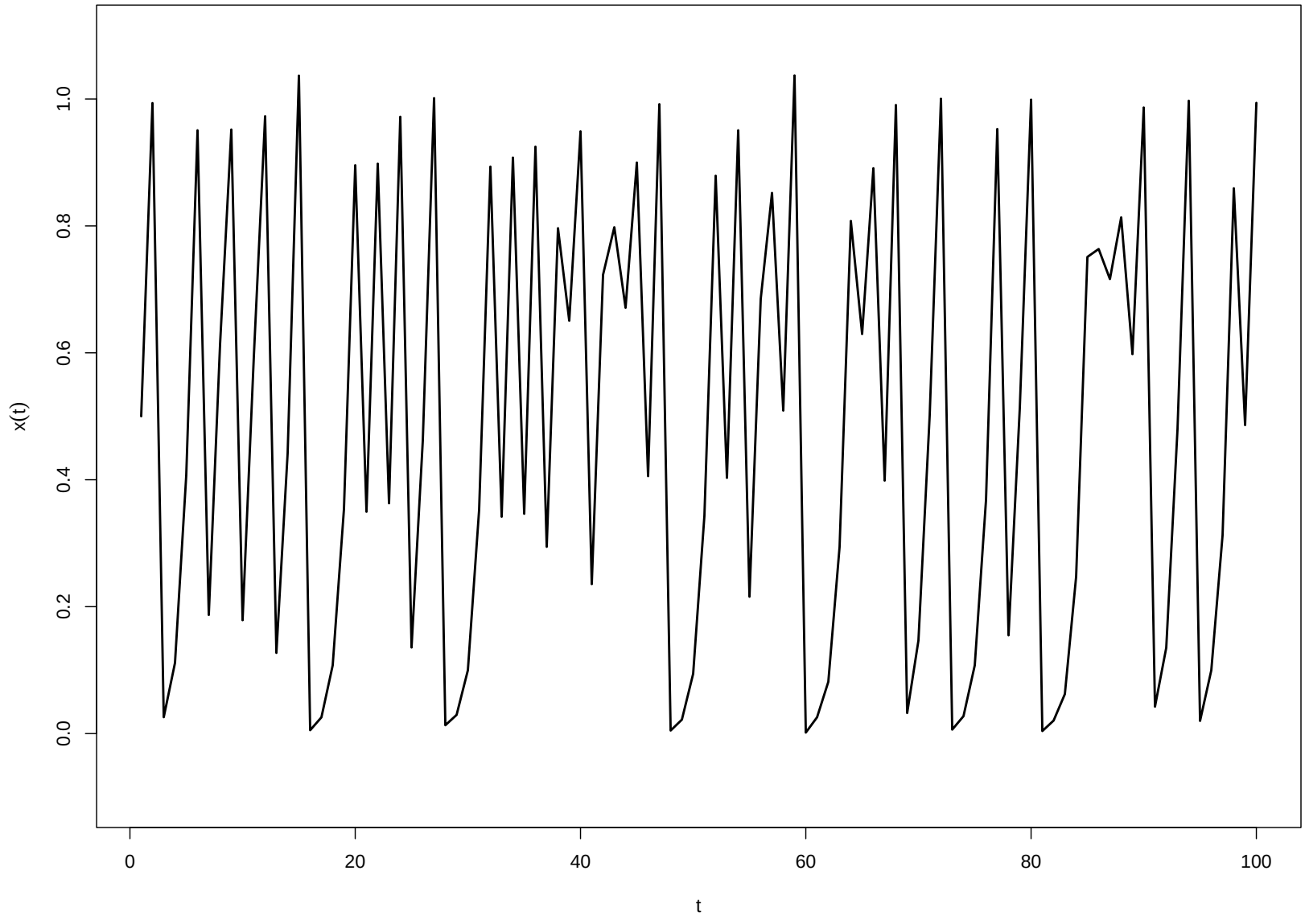
- par inversion
- par SVD (décomposition en valeurs singulières)

L'introduction des fonctions (non linéaires) ϕ_i ne change donc rien à la difficulté du problème de l'apprentissage : il existe toujours une solution optimale qu'on peut calculer efficacement.

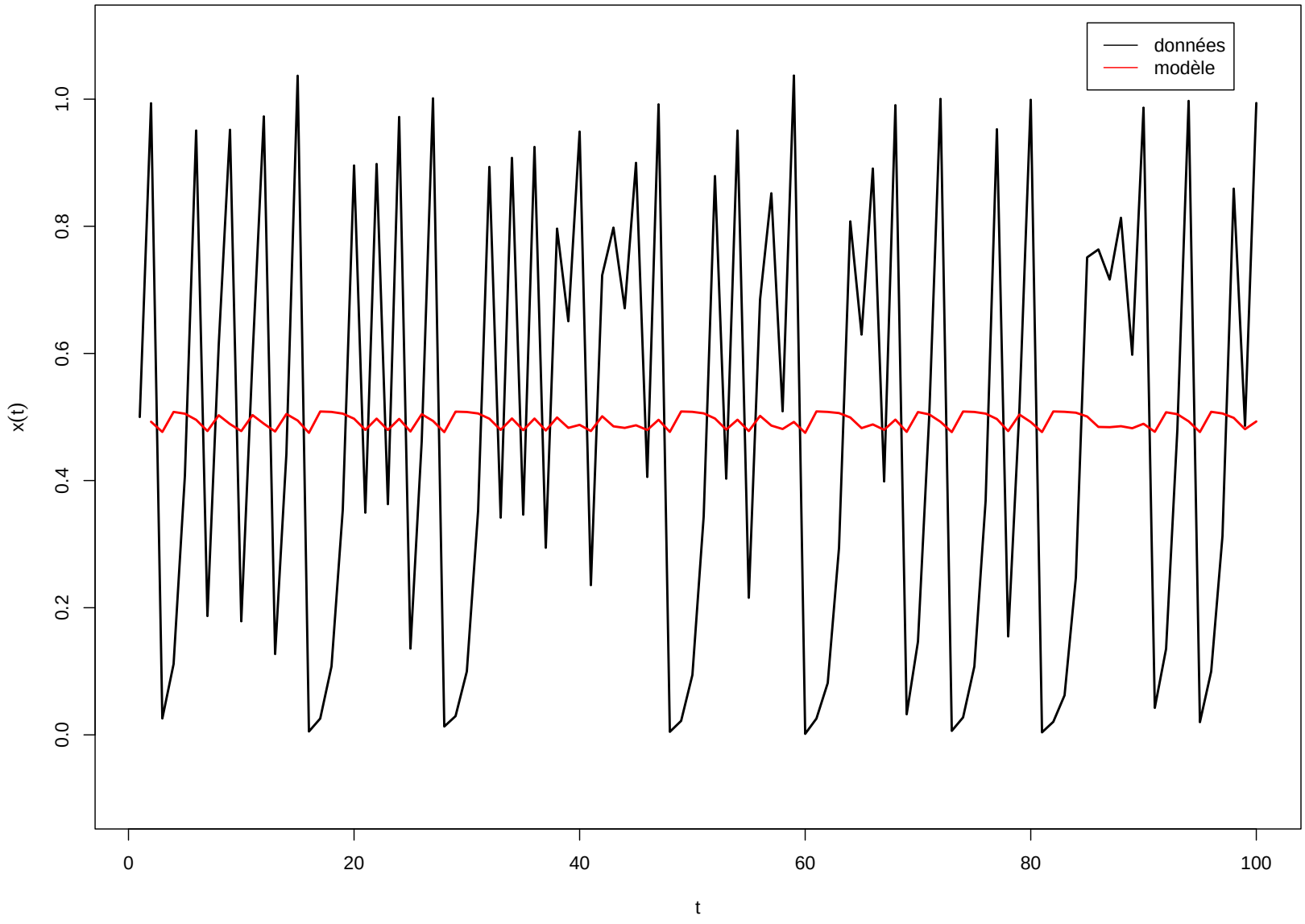
Problème : choix des ϕ_i !

Le plus simple : des monômes.

Exemple

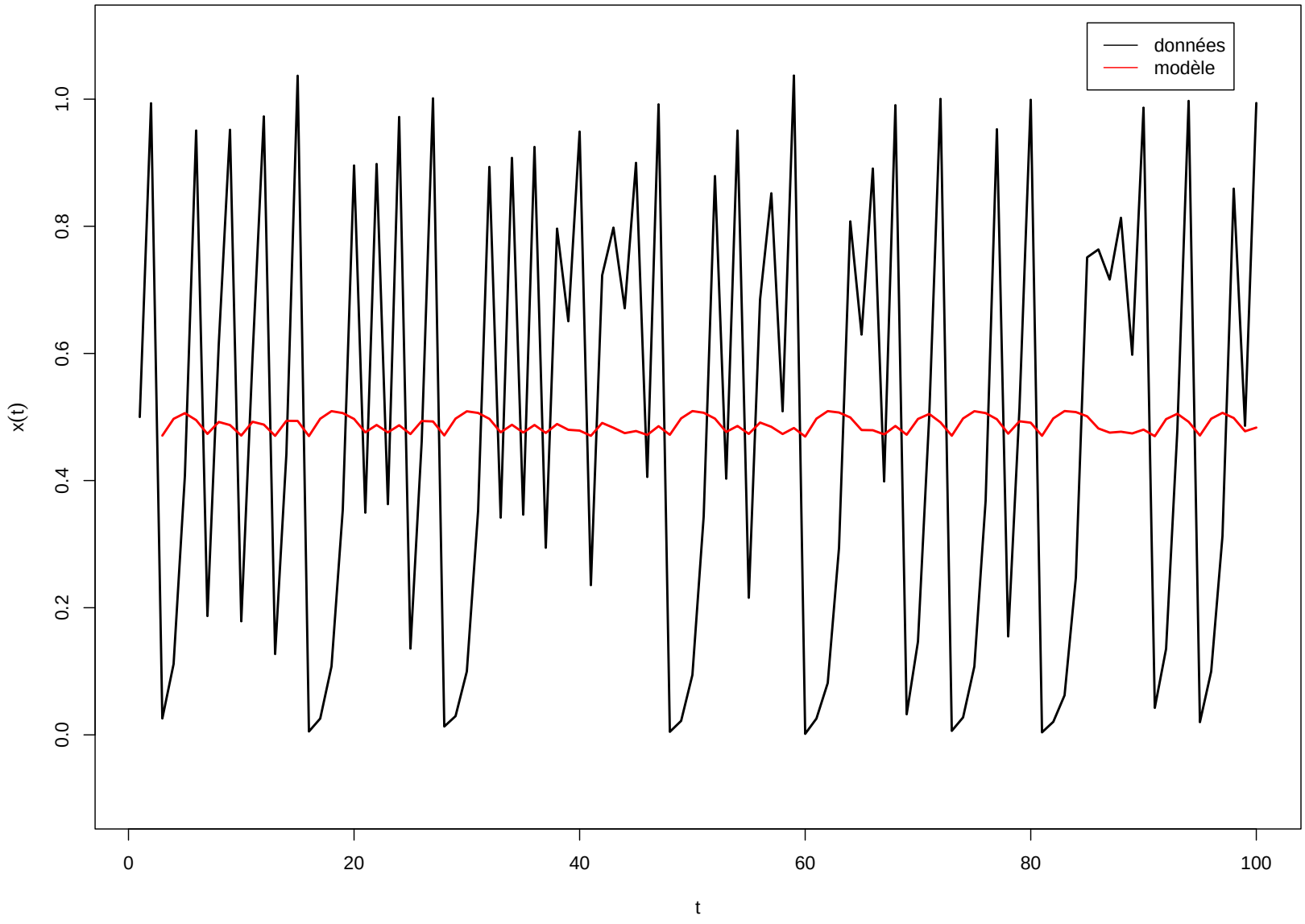


Exemple



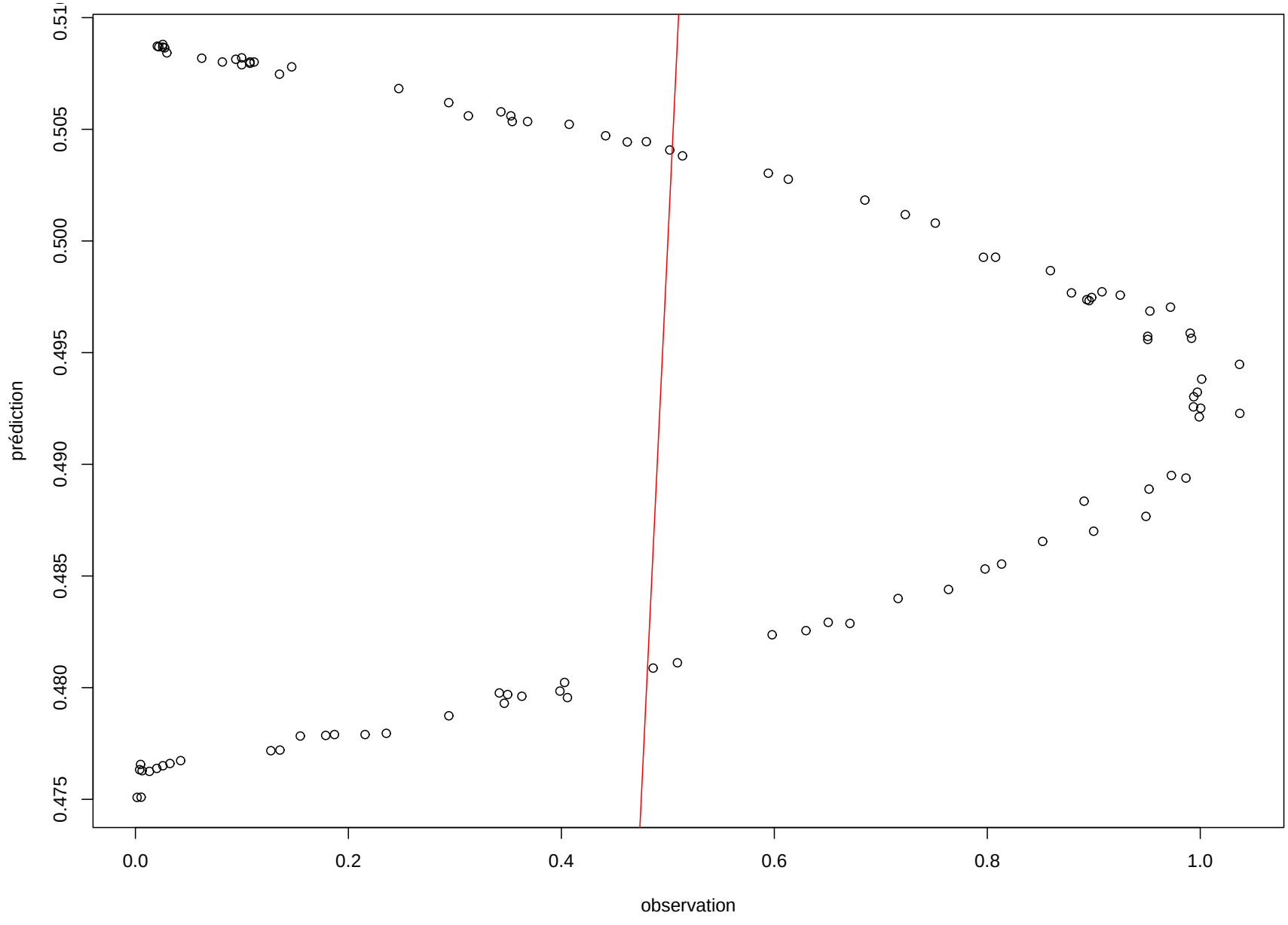
Modèle linéaire classique : $x(t) = a_1 x(t-1) + a_0$

Exemple

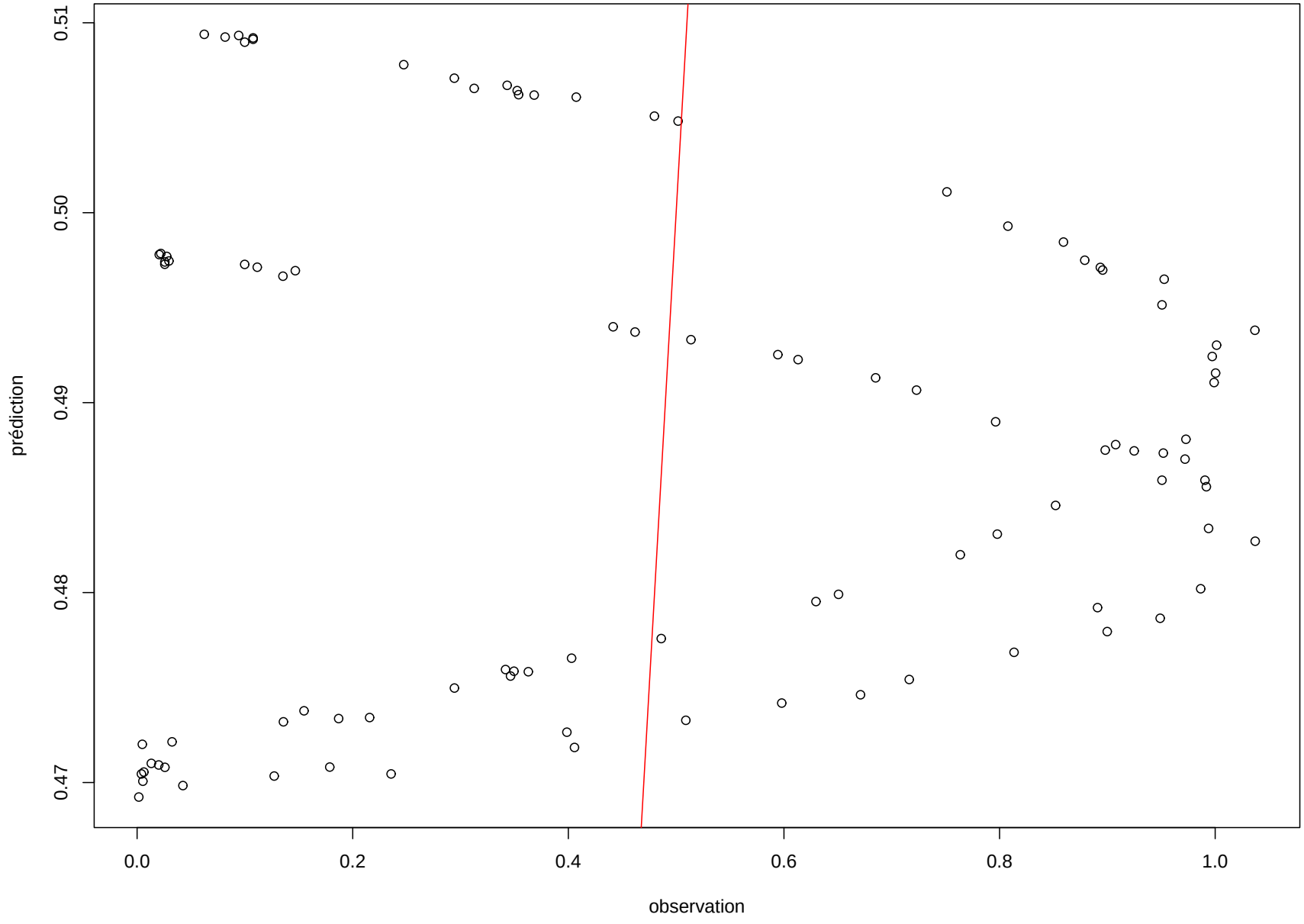


Modèle linéaire classique : $x(t) = a_2 x(t-2) + a_1 x(t-1) + a_0$

Qualité de la prédiction

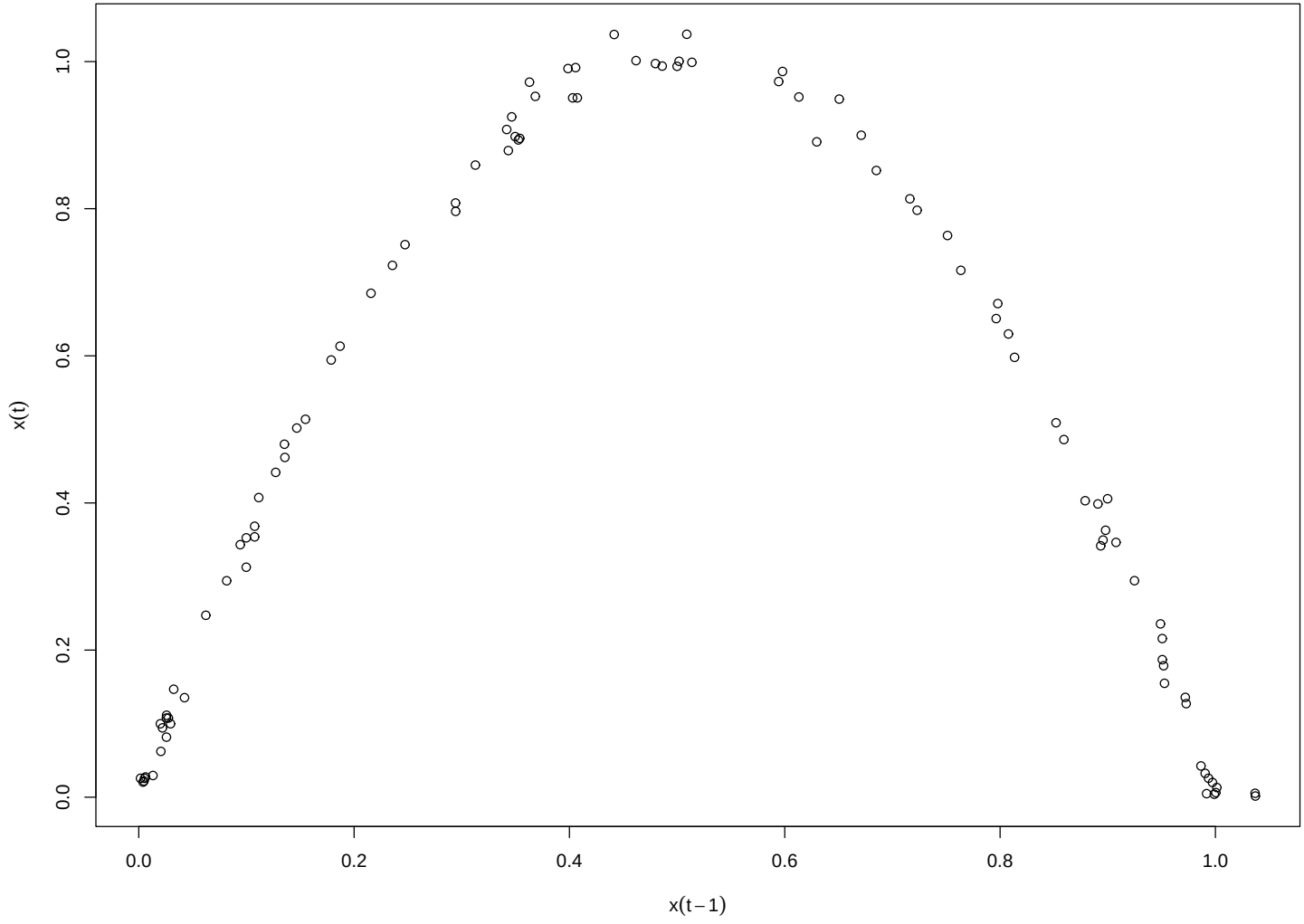


Qualité de la prédiction



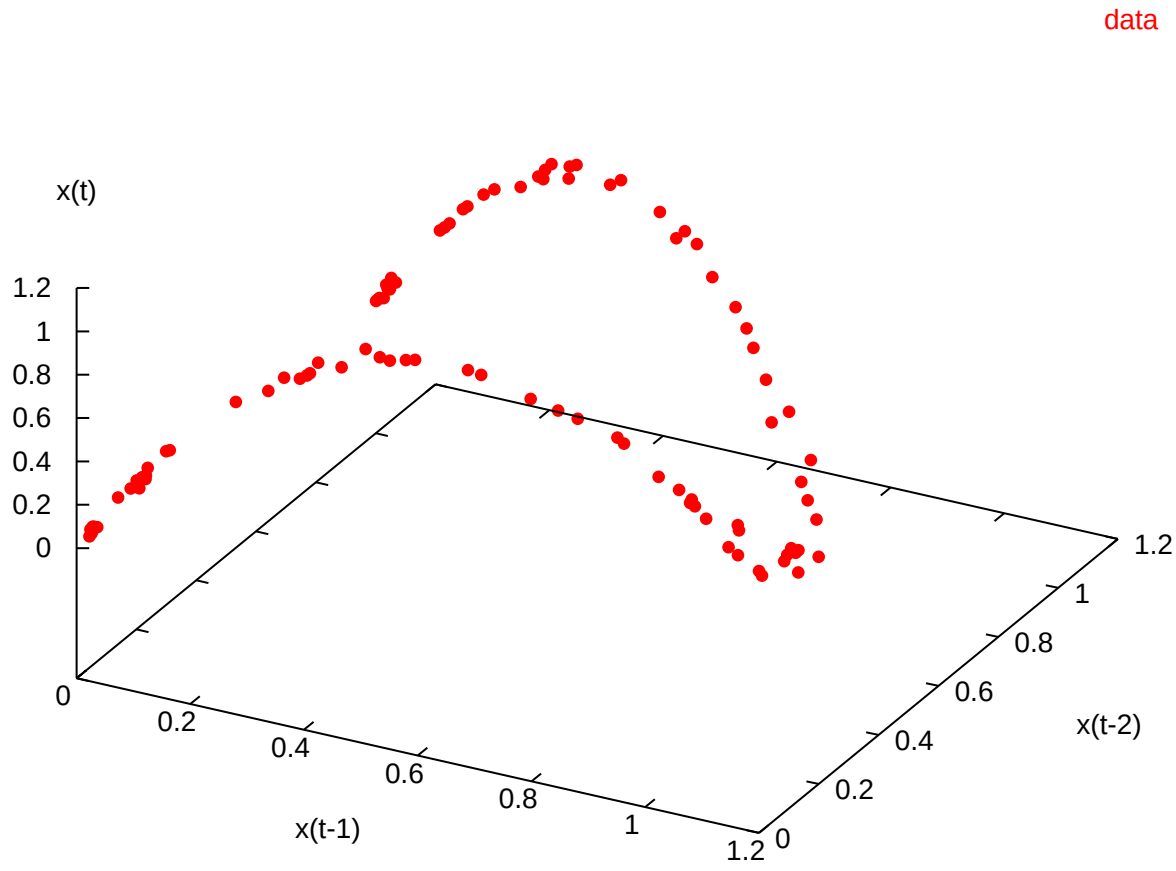
Représentation de $x(t)$

On comprend mieux le problème en traçant $x(t)$ comme fonction de $x(t-1)$ ...

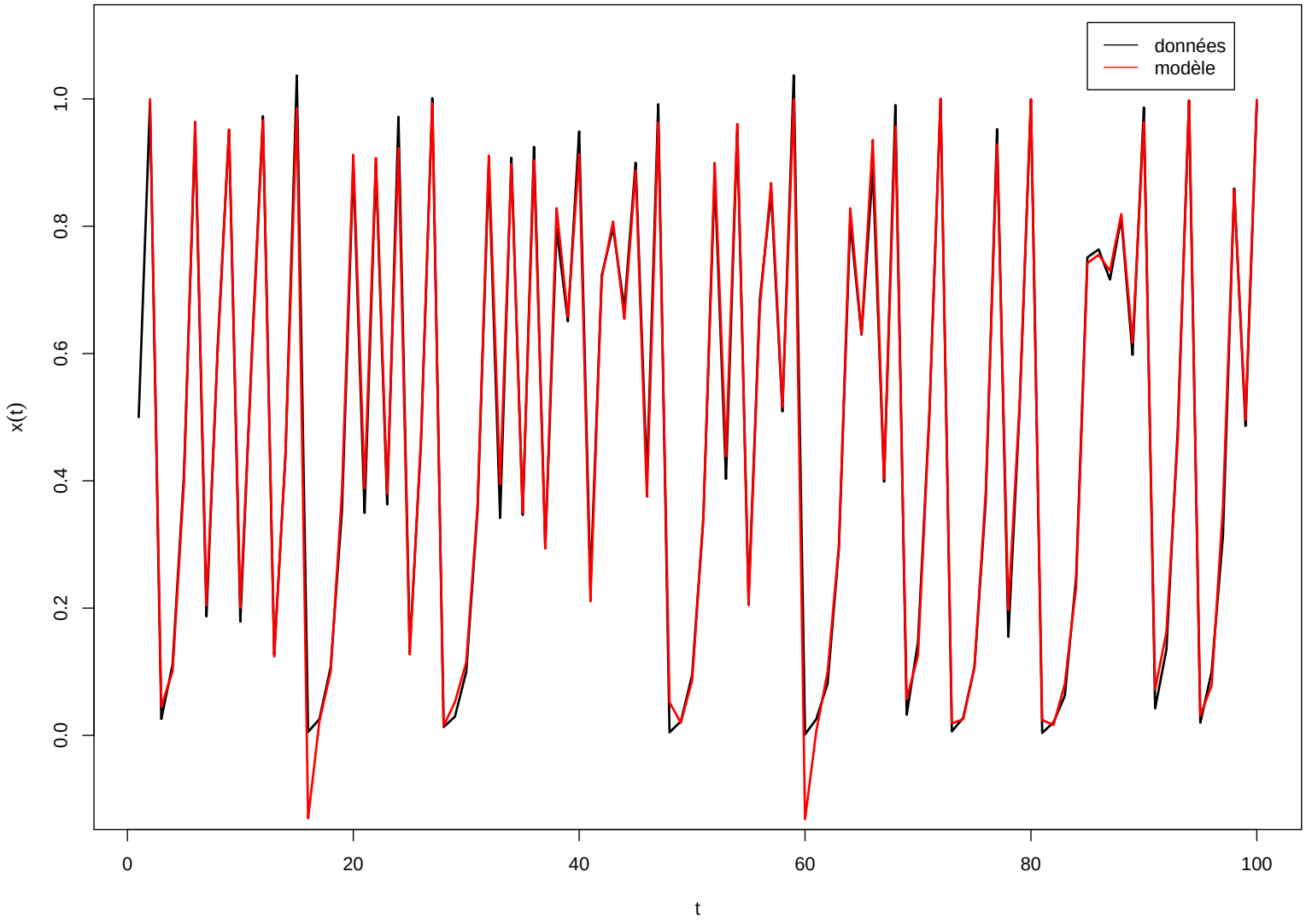


Représentation de $x(t)$ (2)

ou encore $x(t)$ comme fonction de $x(t-1)$ et de $x(t-2)$:



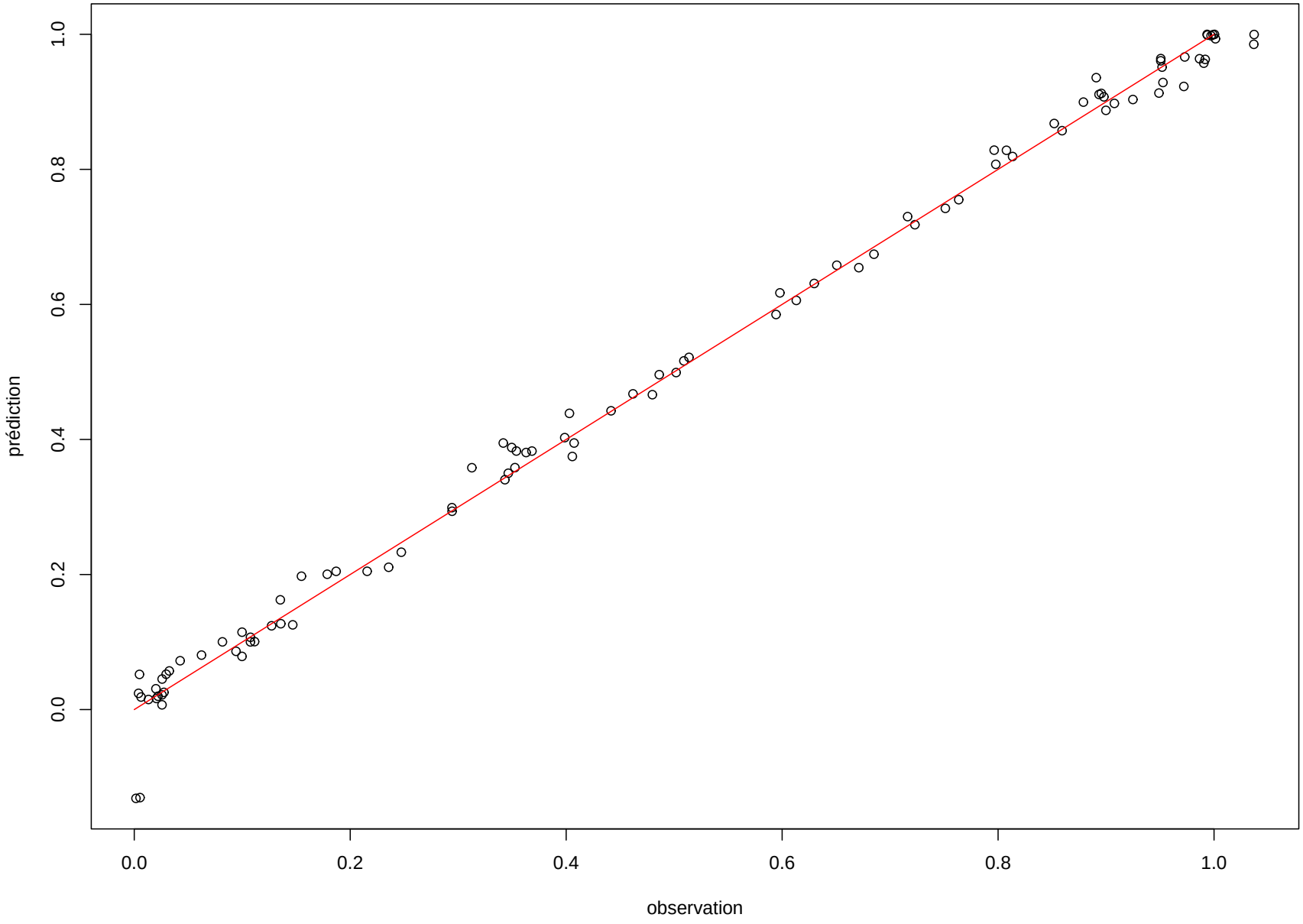
Solution



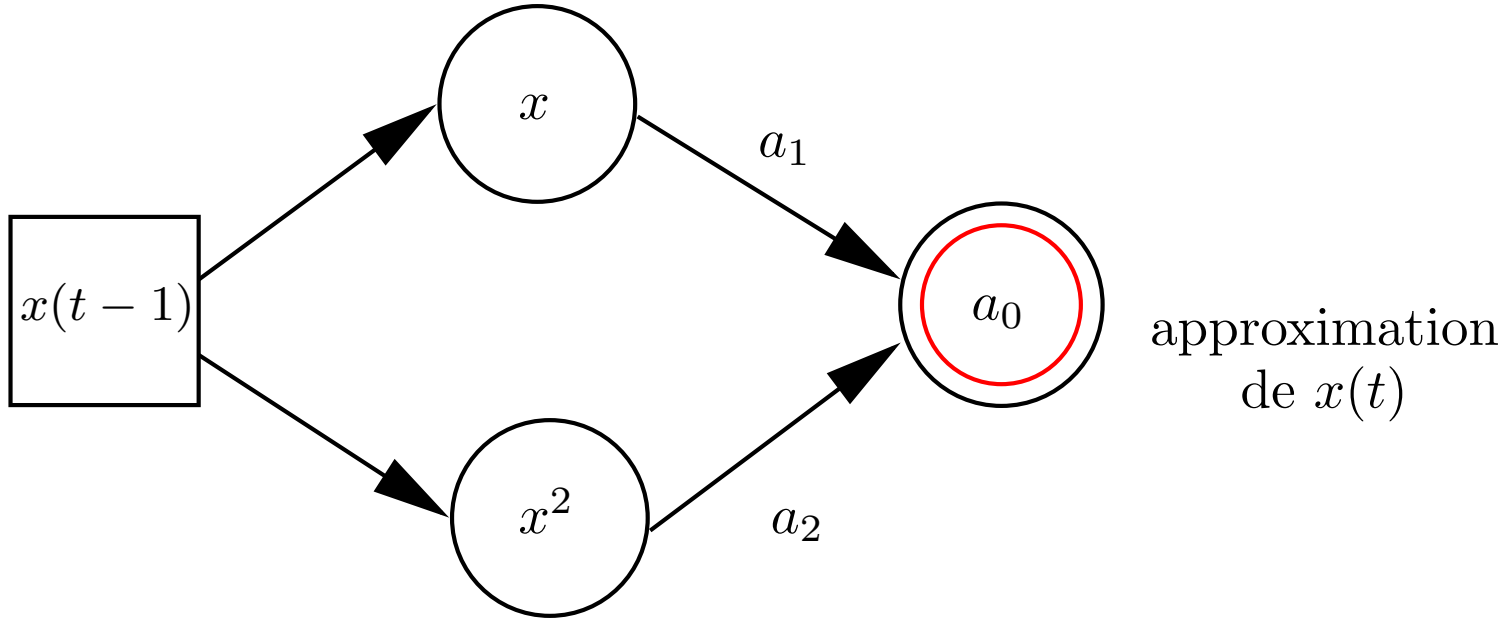
Modèle polynomial (degré 2) :

$$x(t) = a_2(x(t-1))^2 + a_1 x(t-1) + a_0$$

Qualité de la prédiction

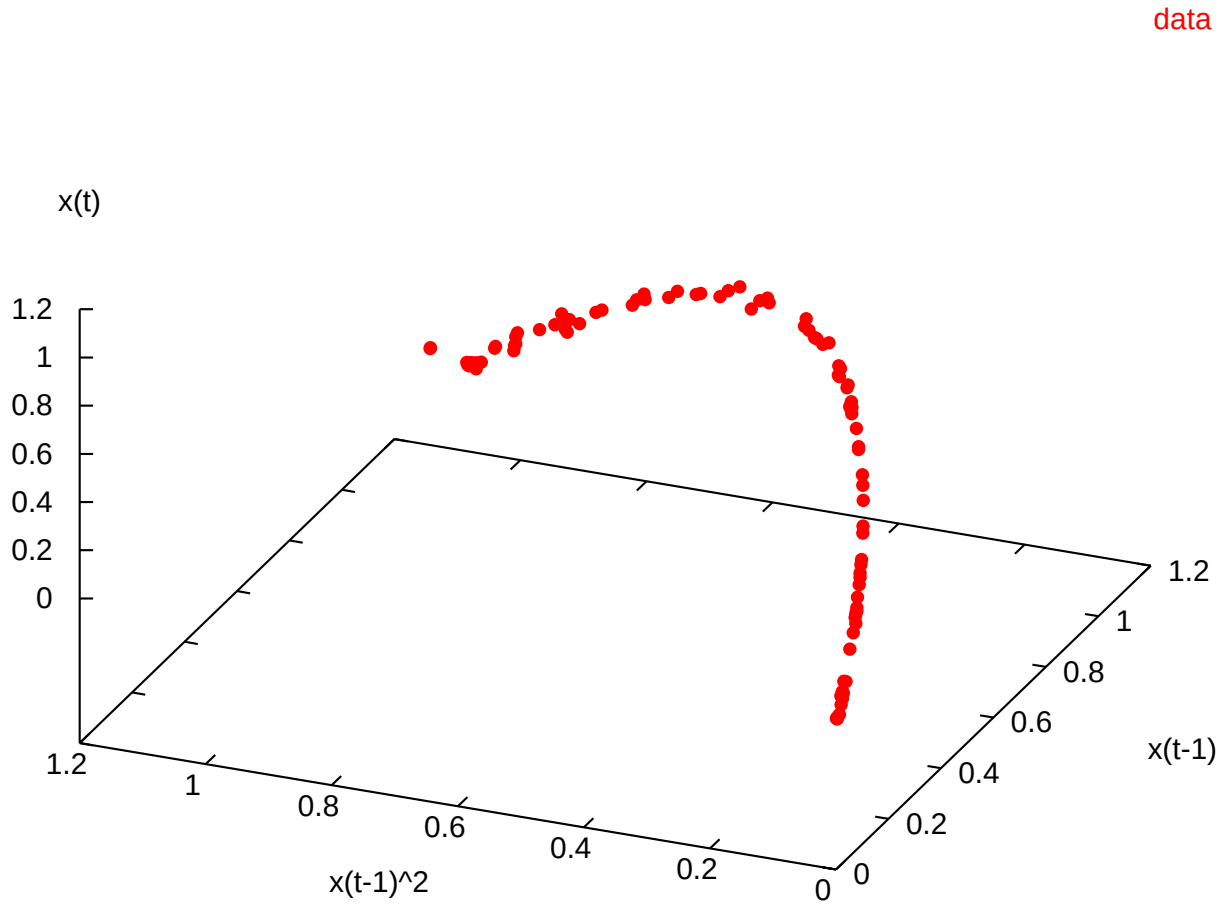


Réseau utilisé



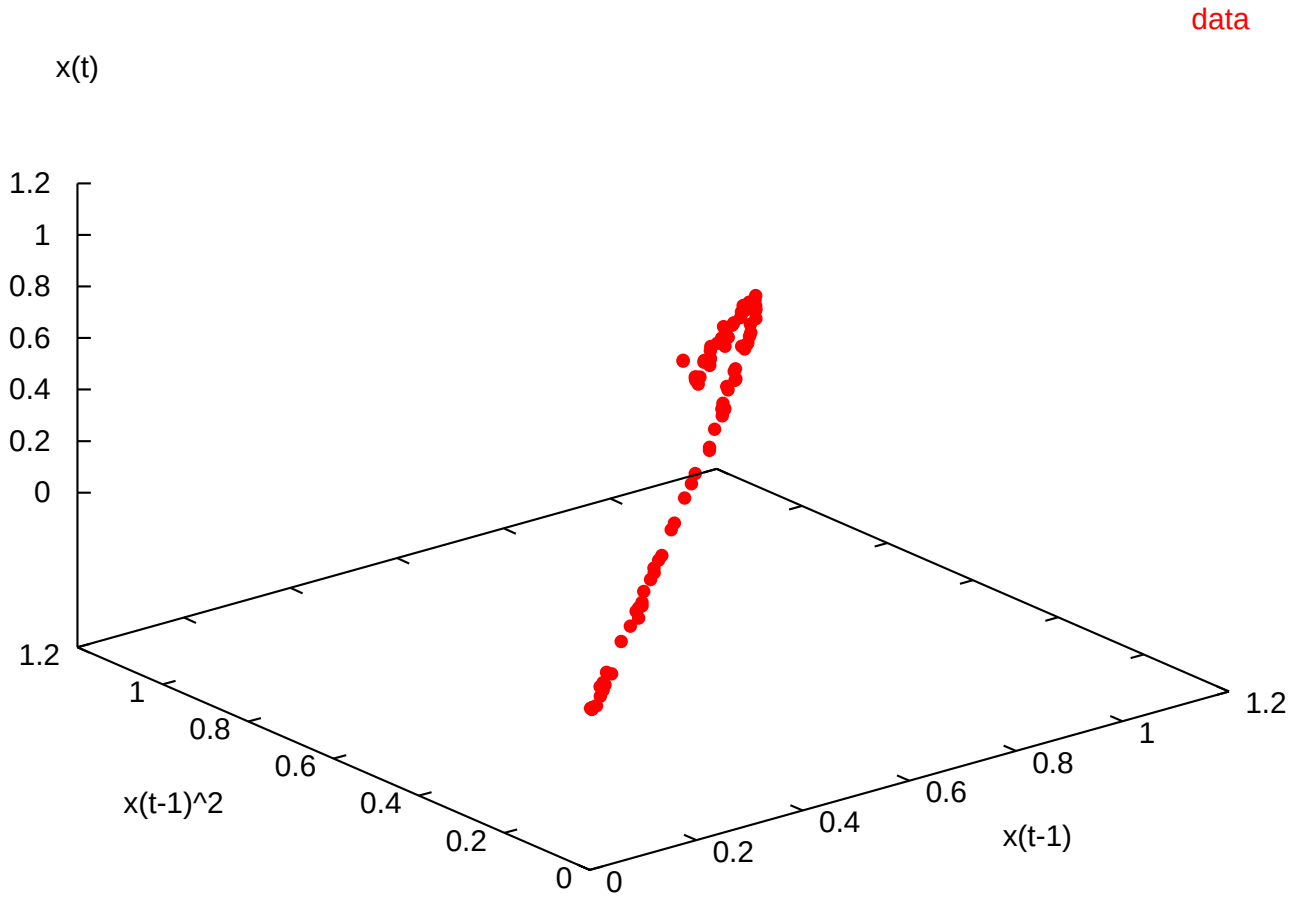
Optimisation (estimation ou apprentissage) : réglagle des paramètres a_1, a_2 (connexions synaptiques) et a_0 (seuil) pour approcher l'association $x(t-1)$ vers $x(t)$ contenue dans les données.

Transformations par la première couche



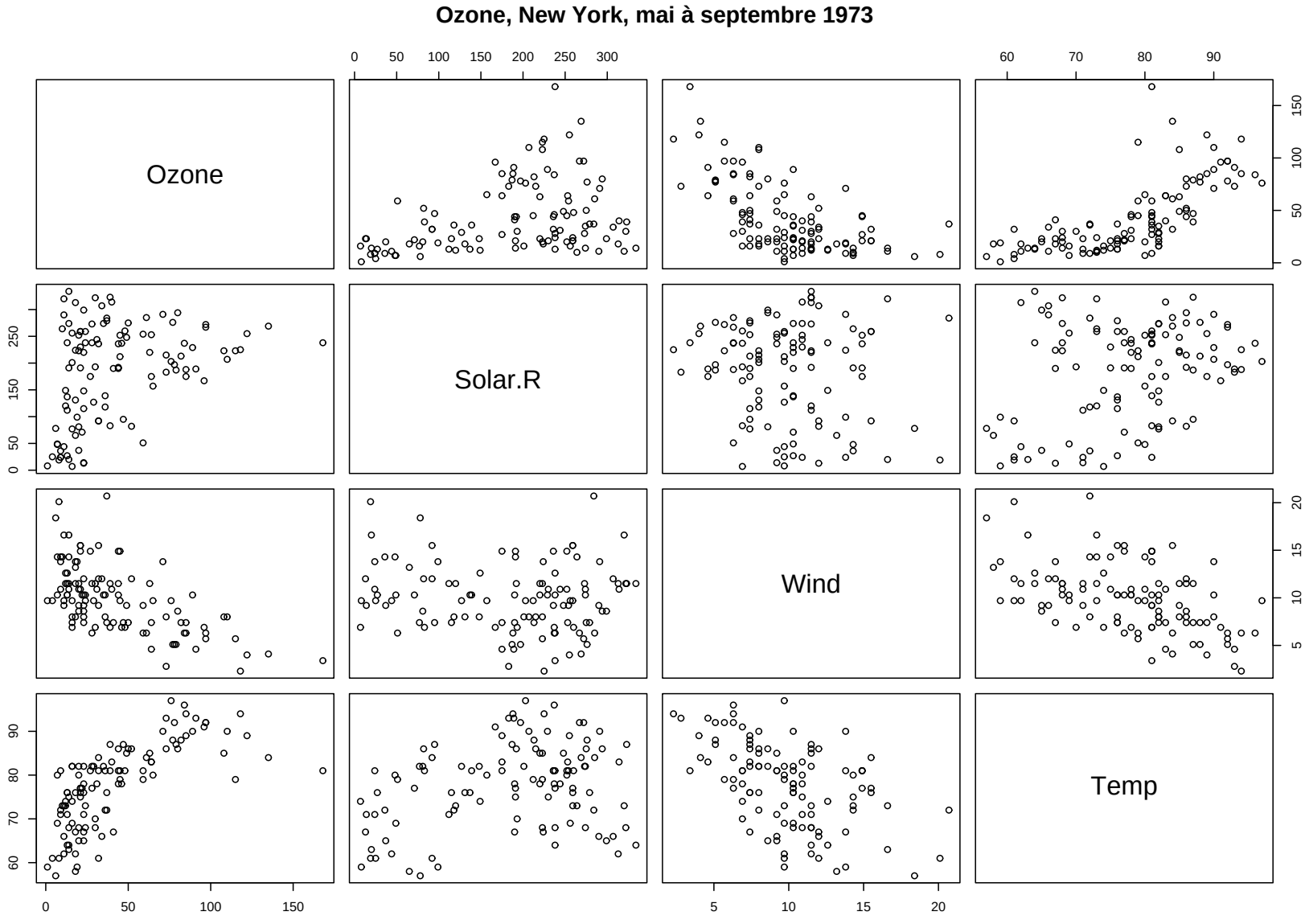
Le problème est maintenant linéaire !

Transformations par la première couche

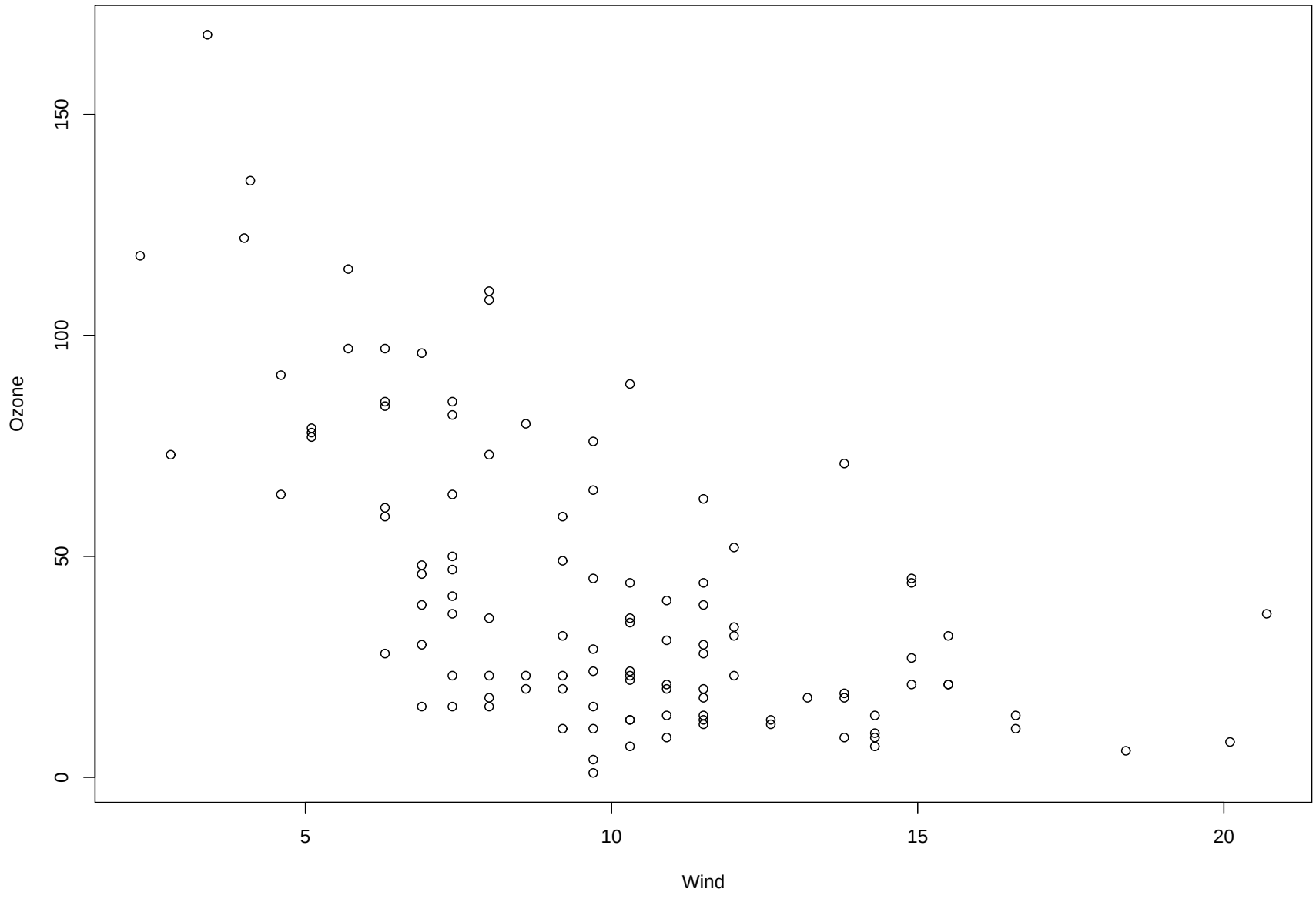


Le problème est maintenant linéaire !

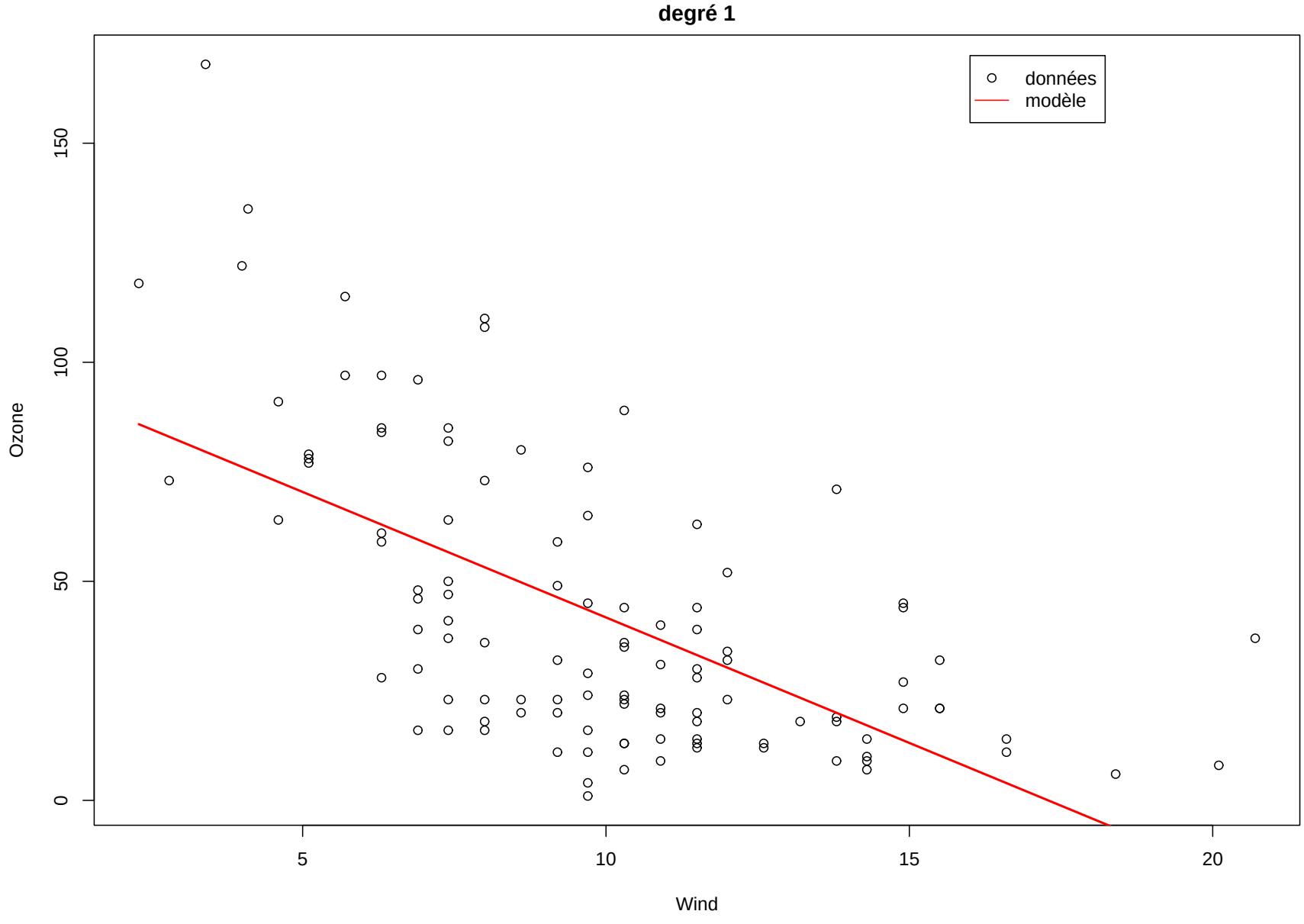
Exemple d'étude de données réelles



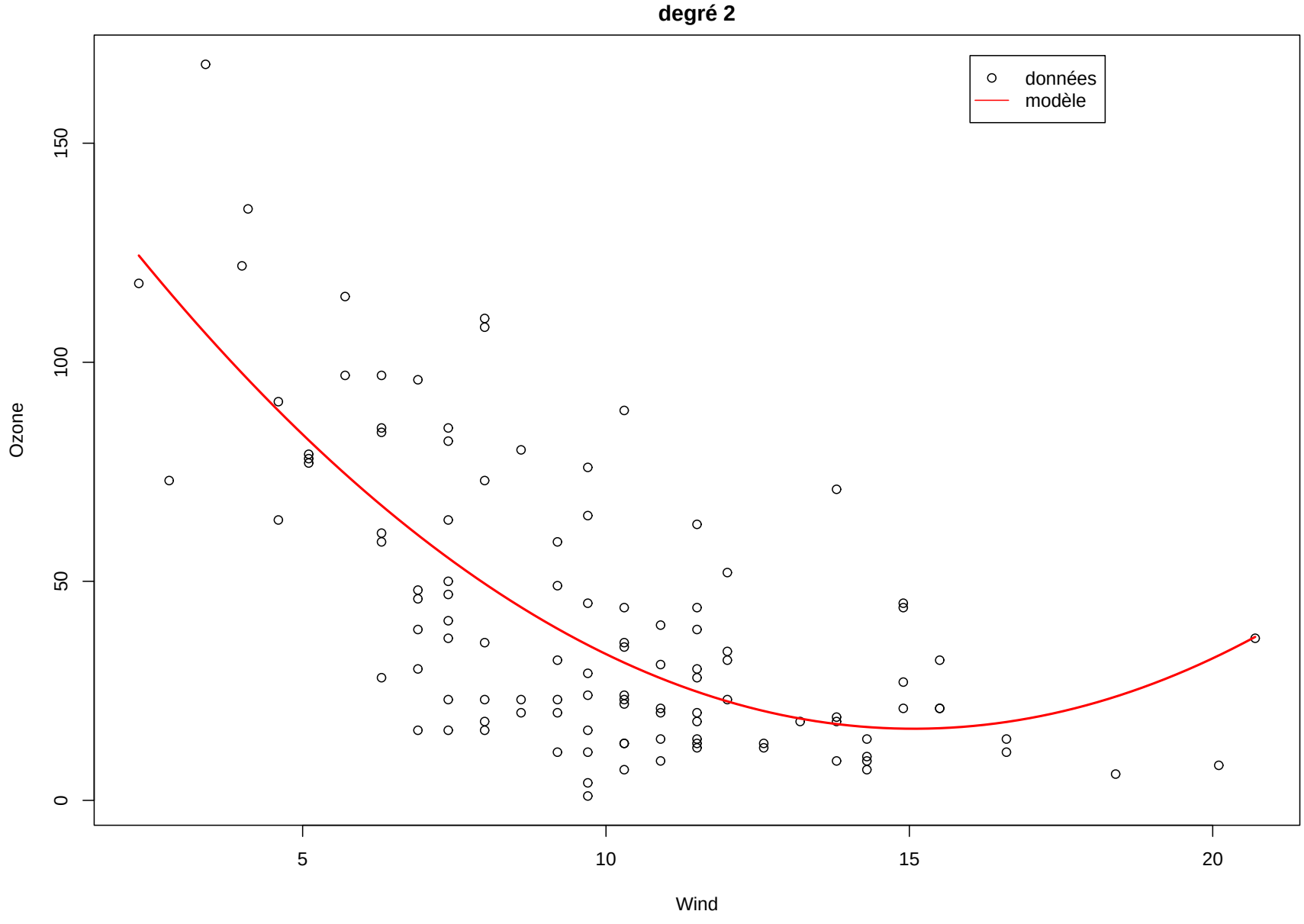
Prédire l'ozone à partir du vent



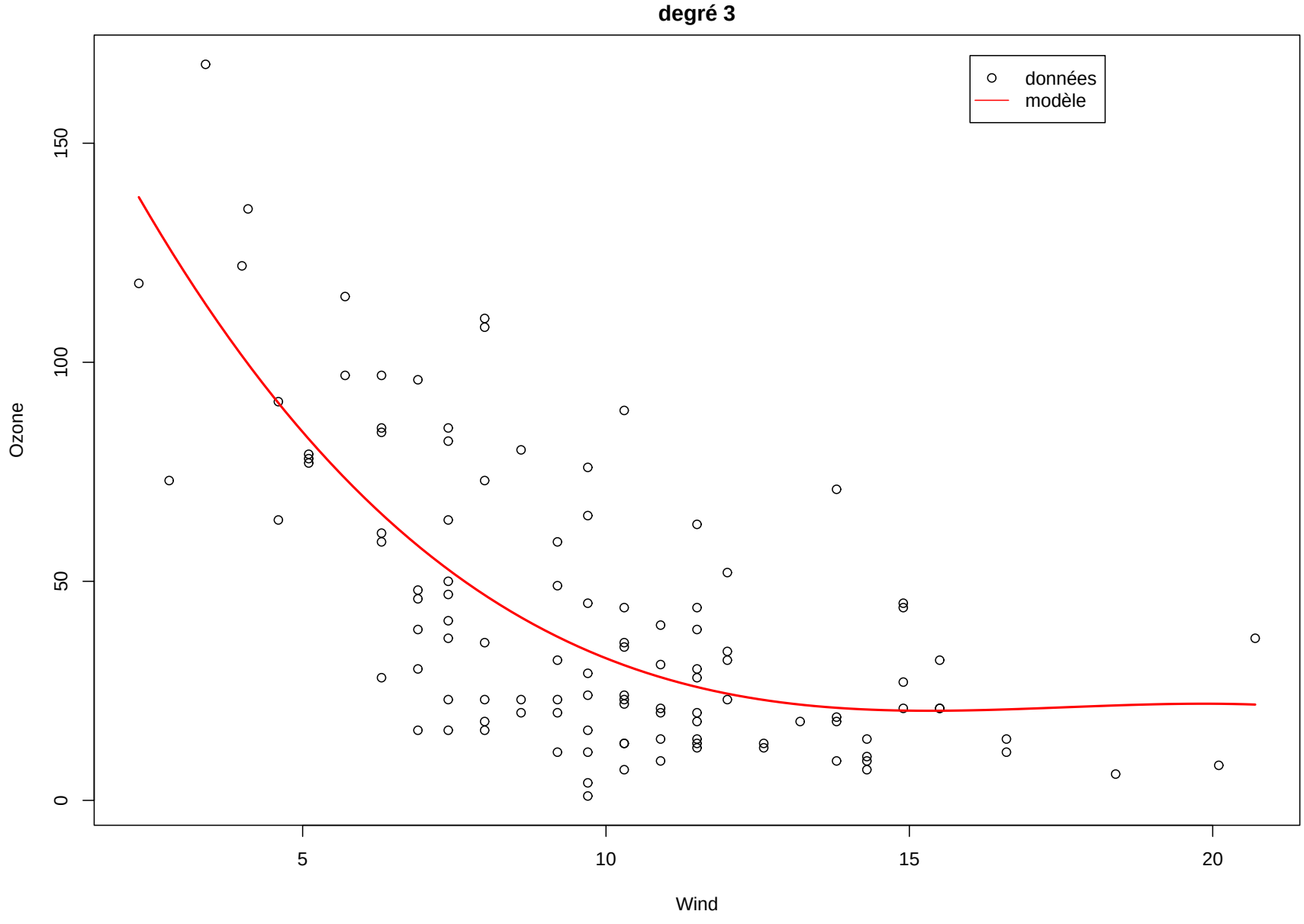
Prédire l'ozone à partir du vent



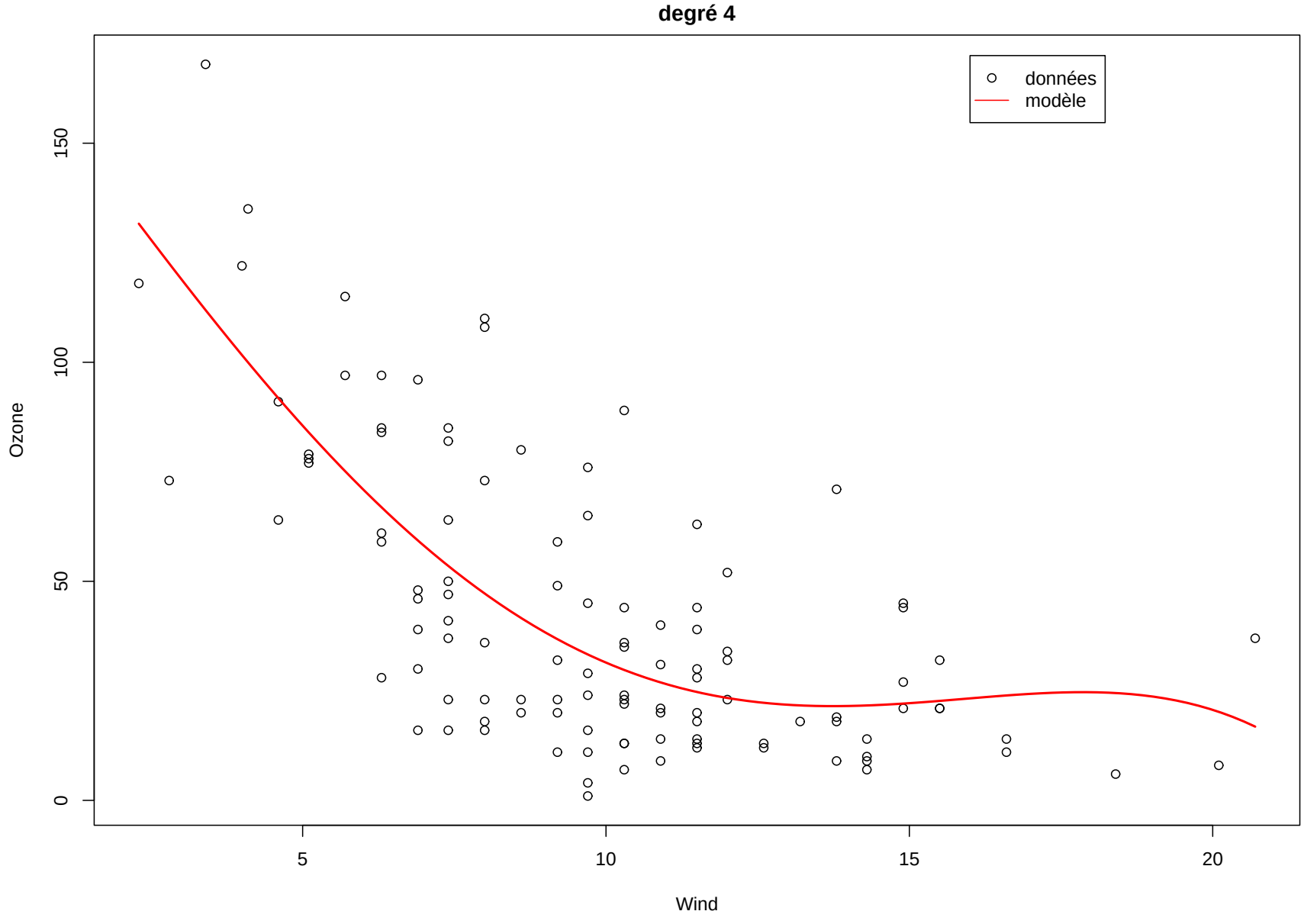
Prédire l'ozone à partir du vent



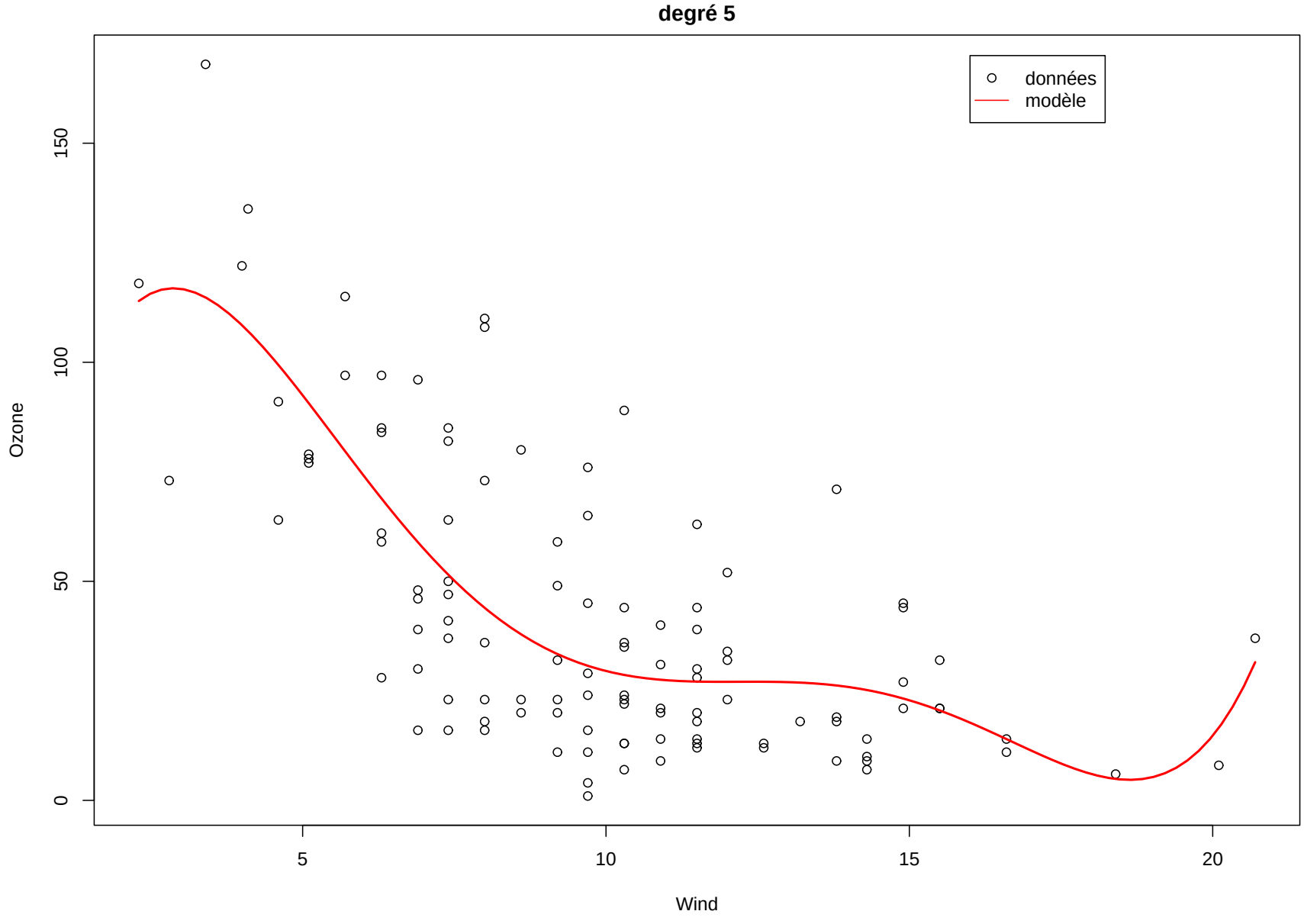
Prédire l'ozone à partir du vent



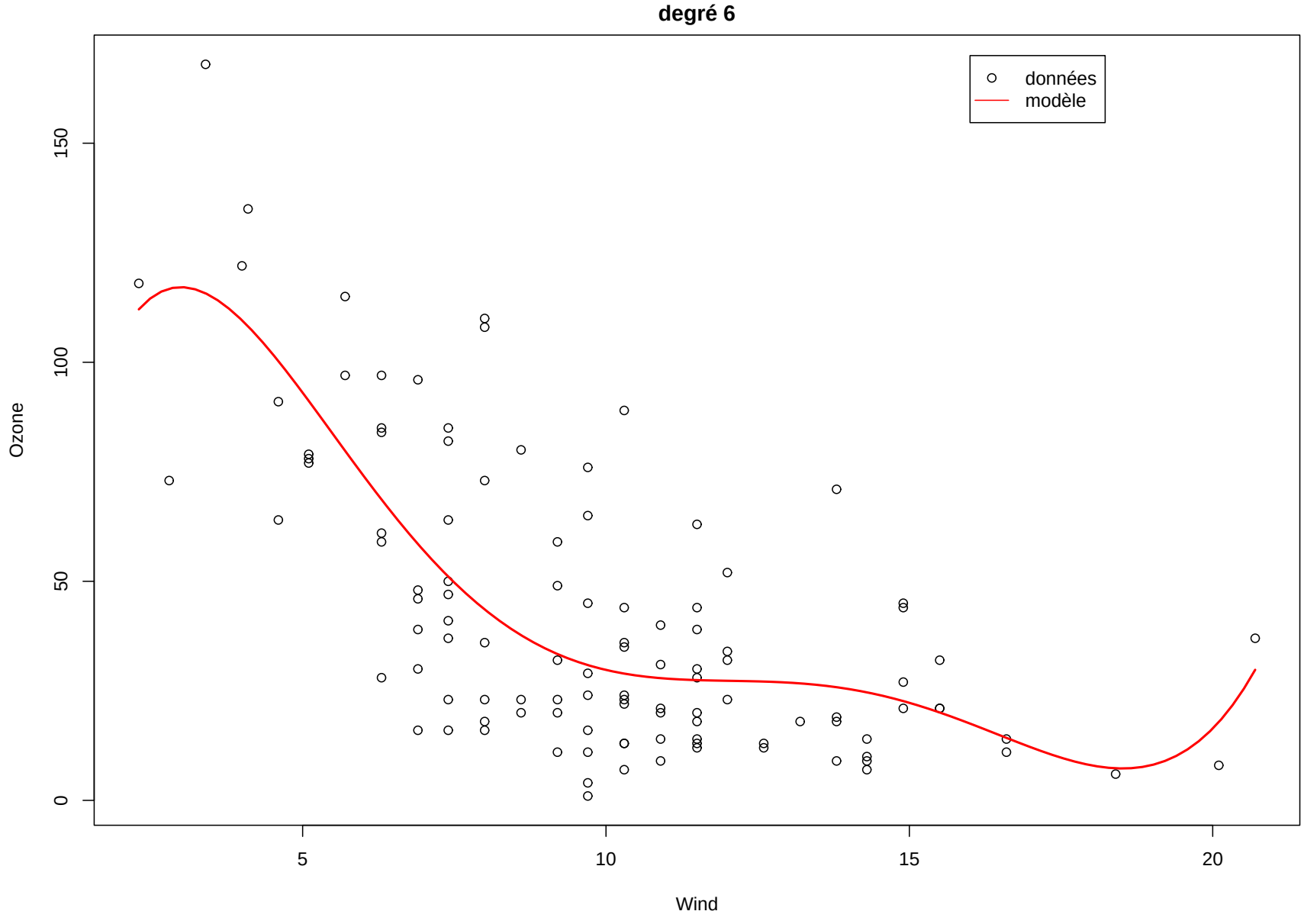
Prédire l'ozone à partir du vent



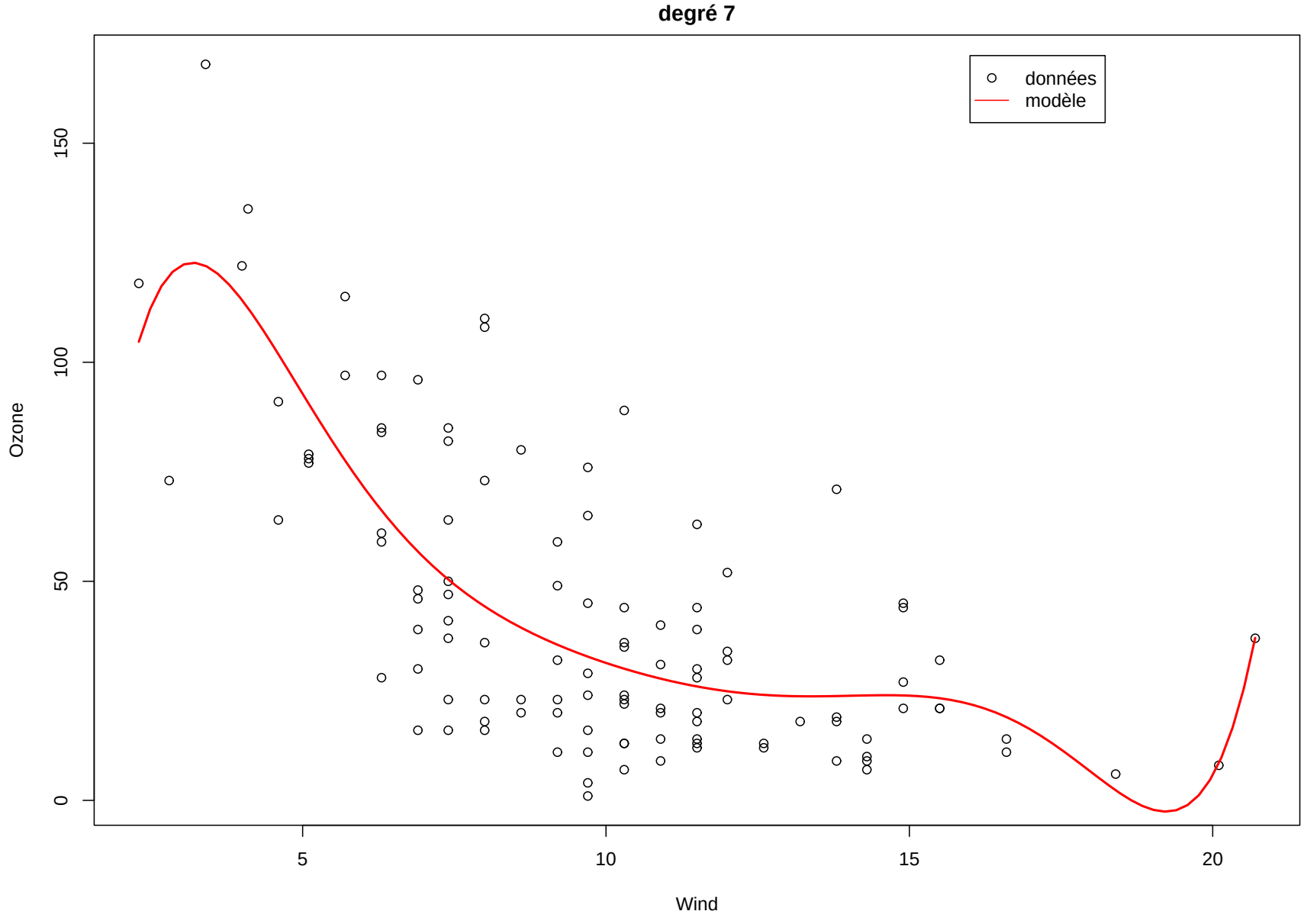
Prédire l'ozone à partir du vent



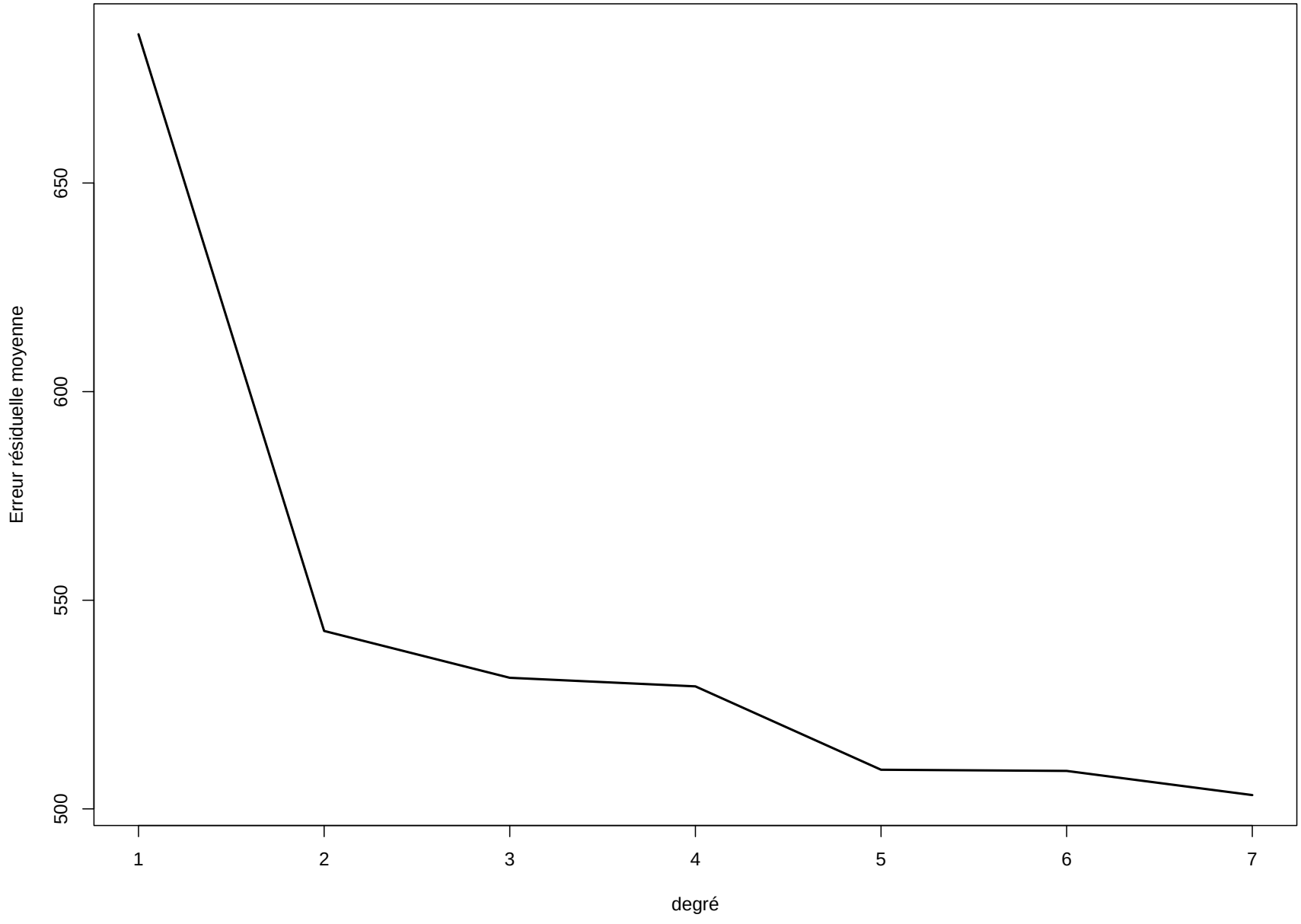
Prédire l'ozone à partir du vent



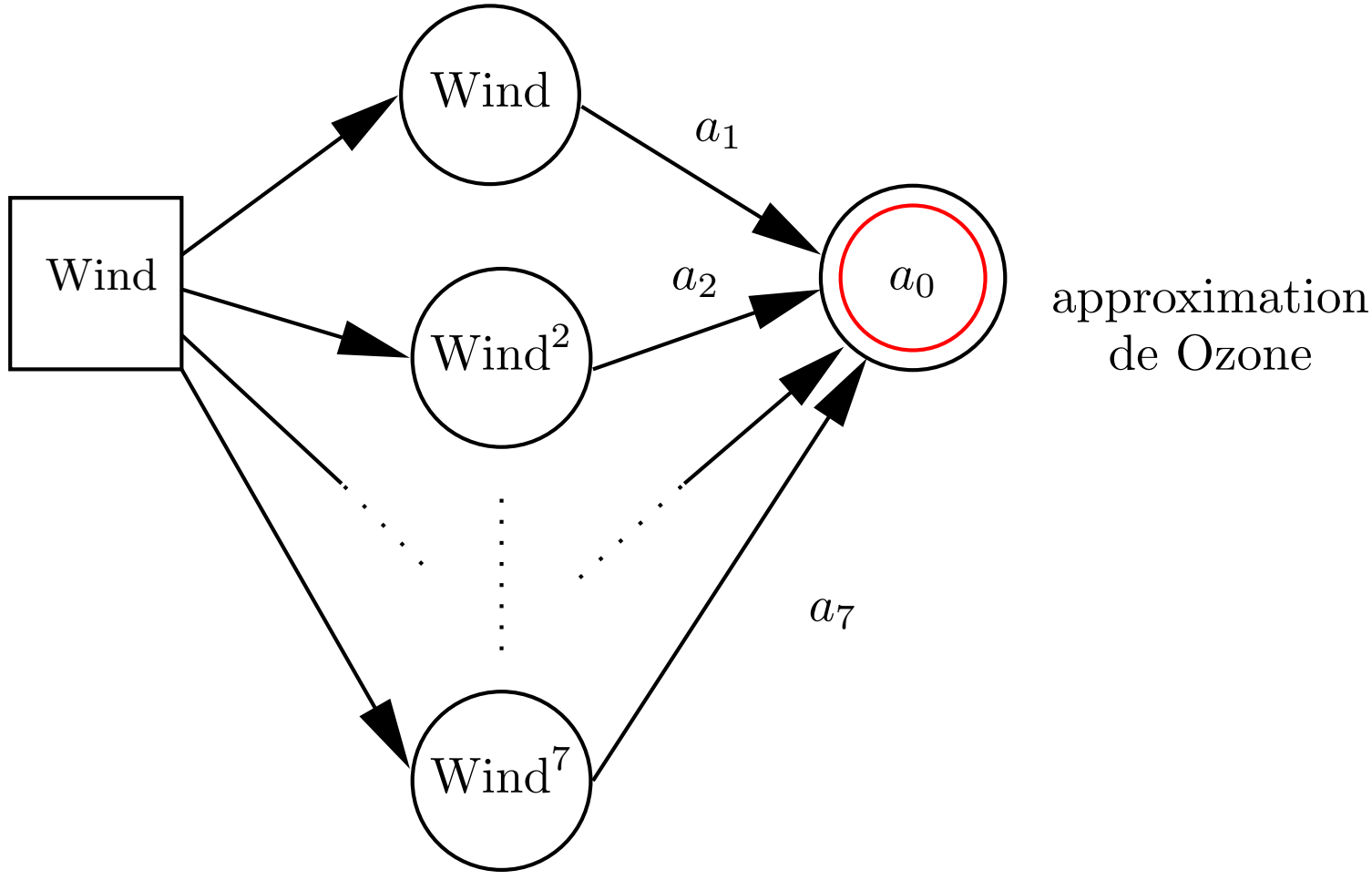
Prédire l'ozone à partir du vent



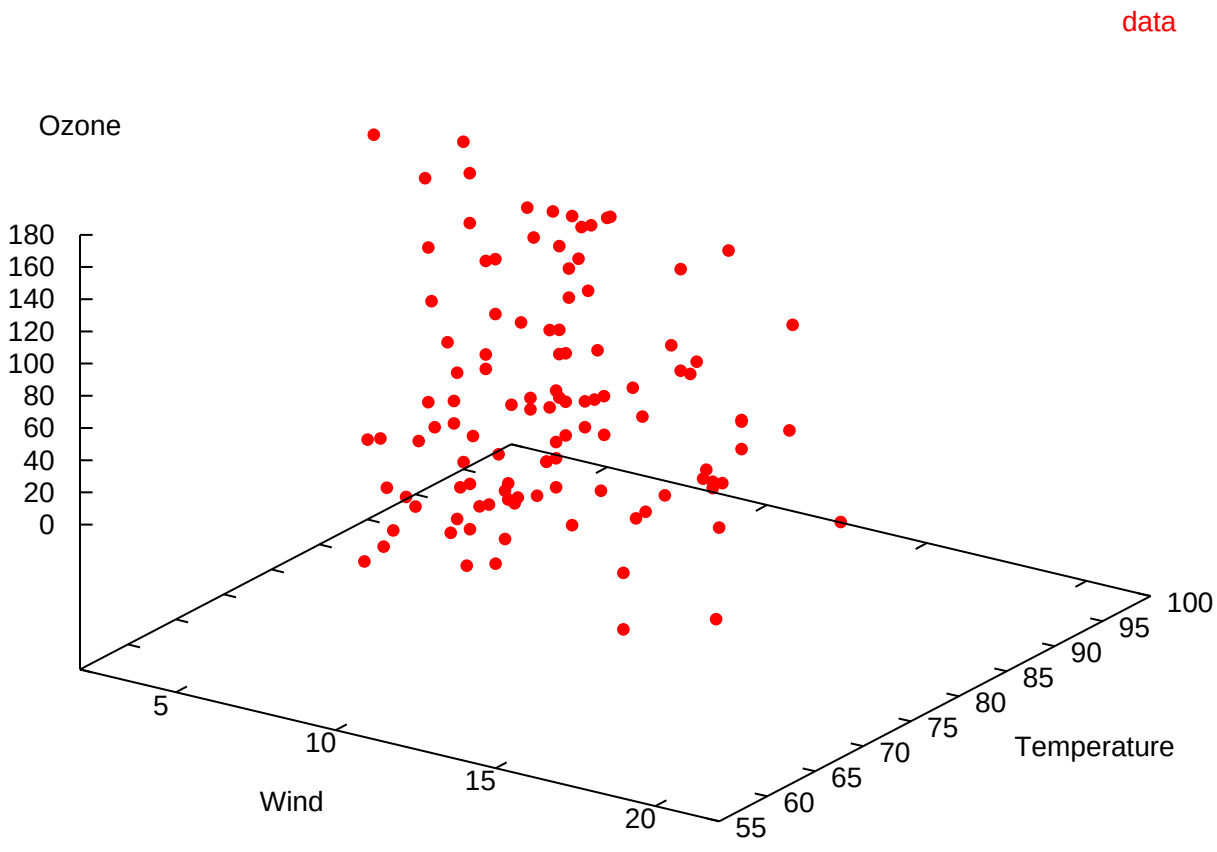
Evolution de l'erreur



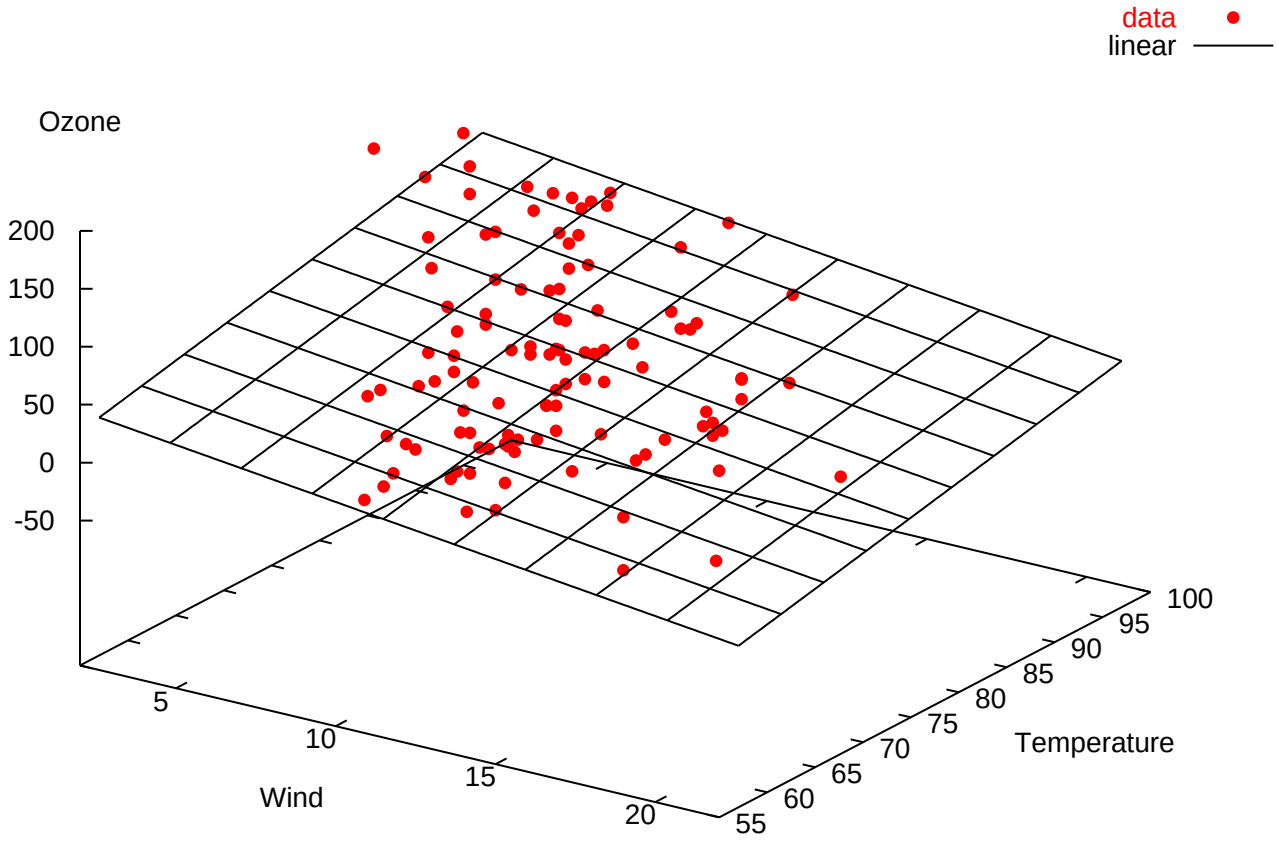
Réseau utilisé



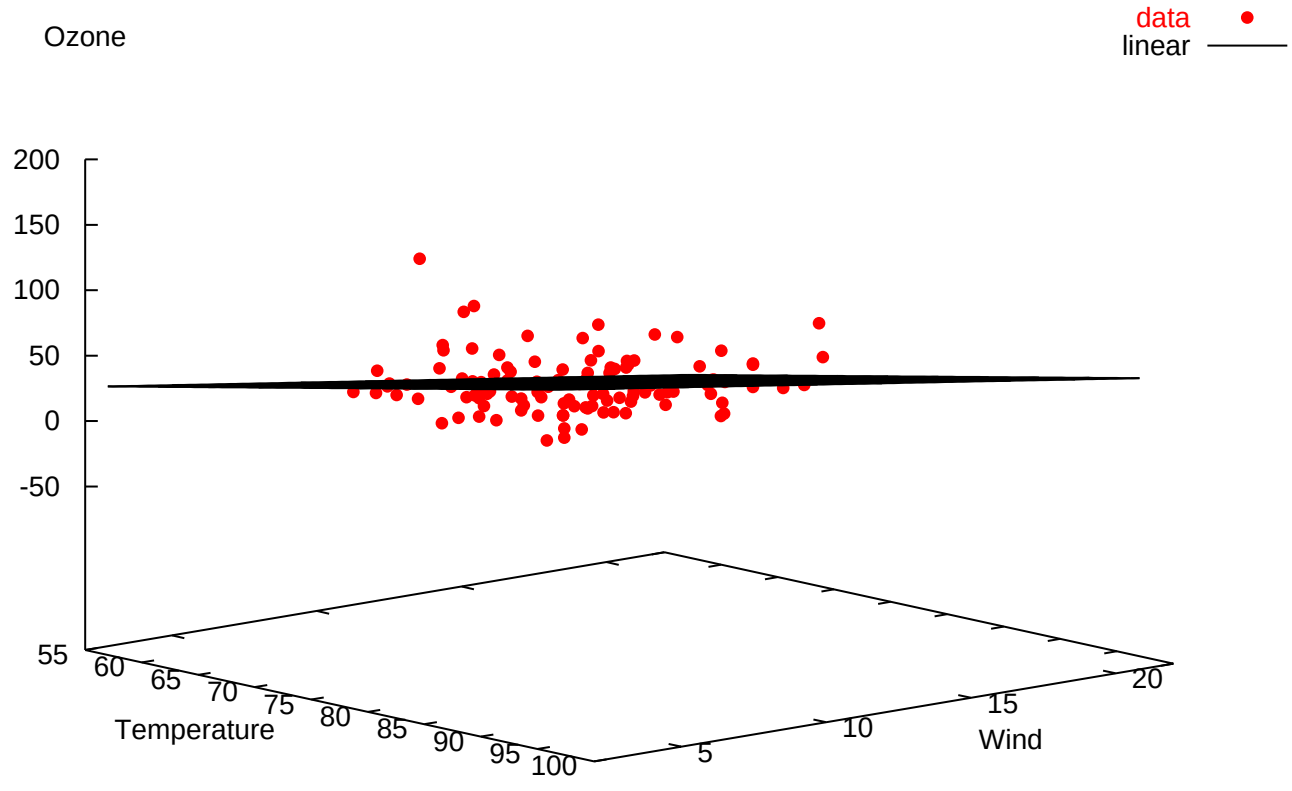
Avec deux variables



Modèle linéaire

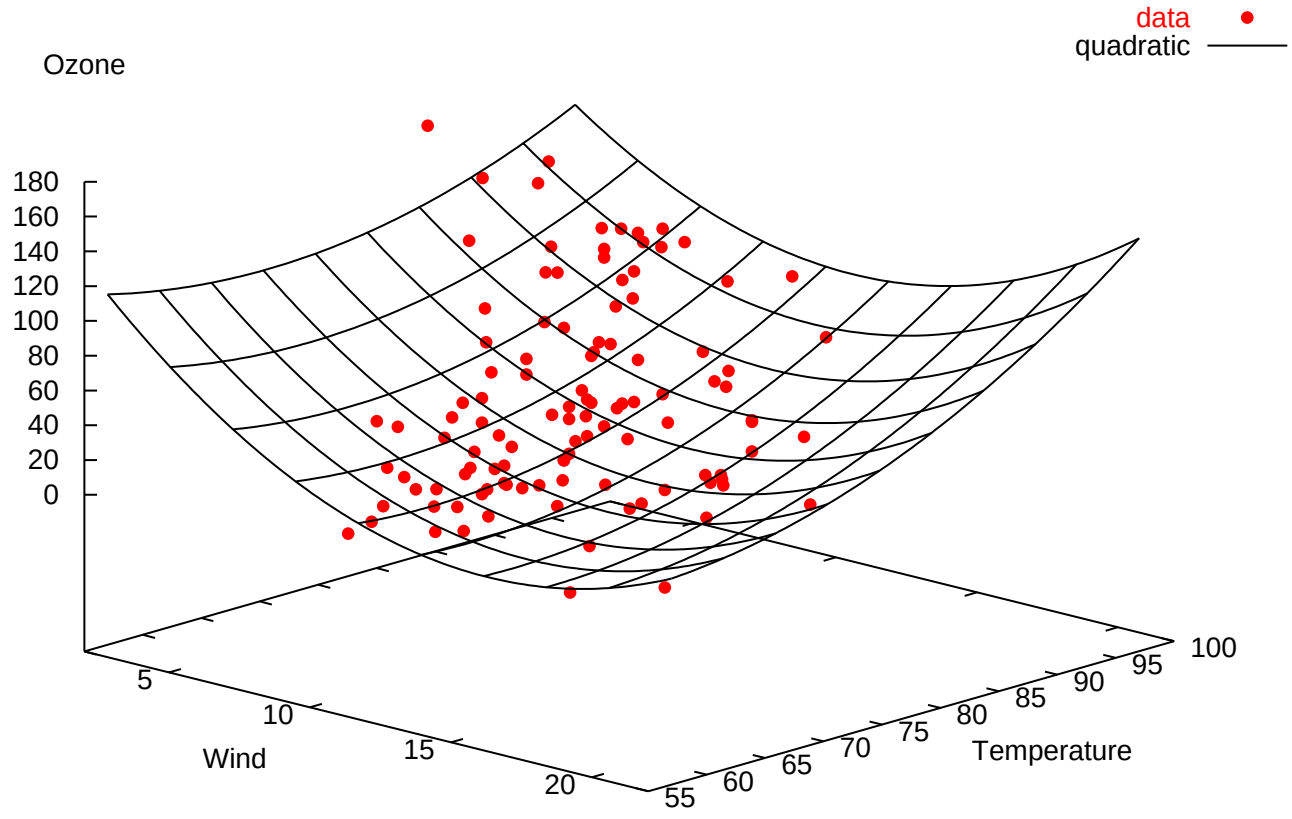


Modèle linéaire

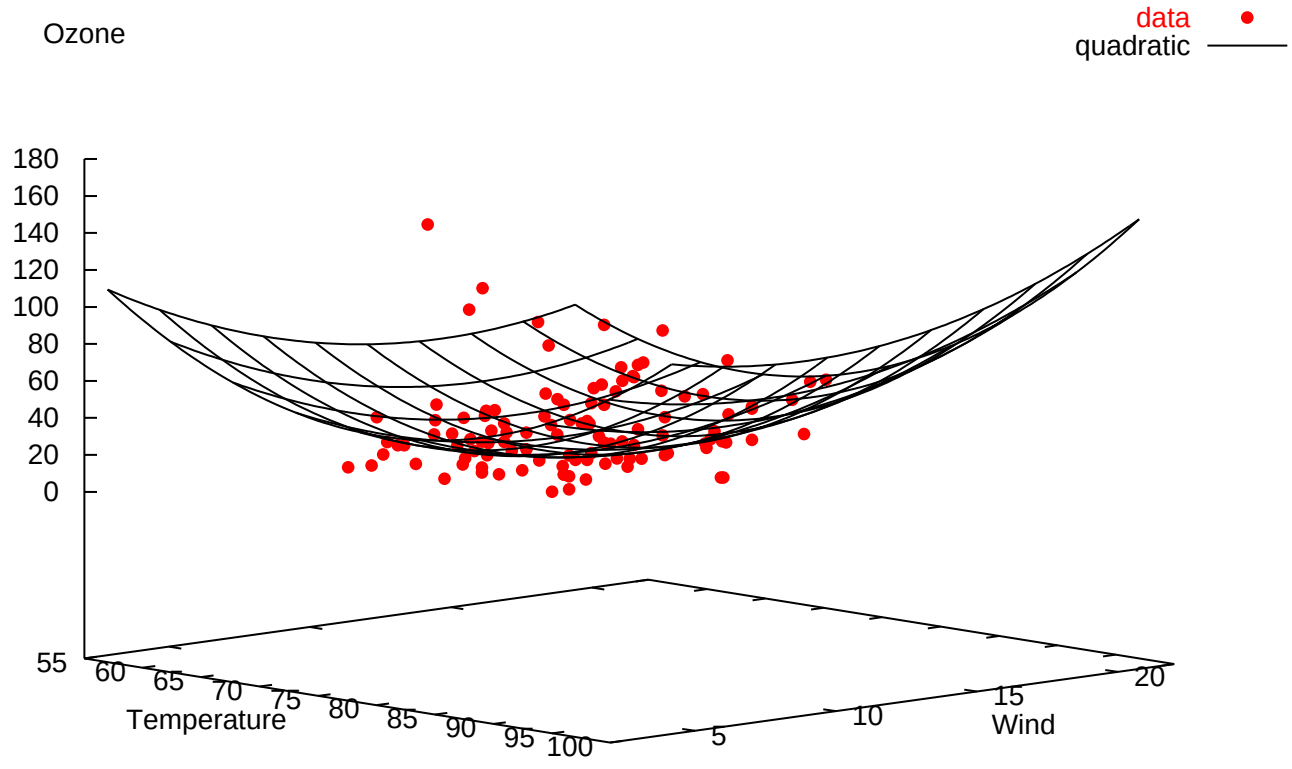


Erreur quadratique moyenne : 459 (contre 503)

Modèle quadratique

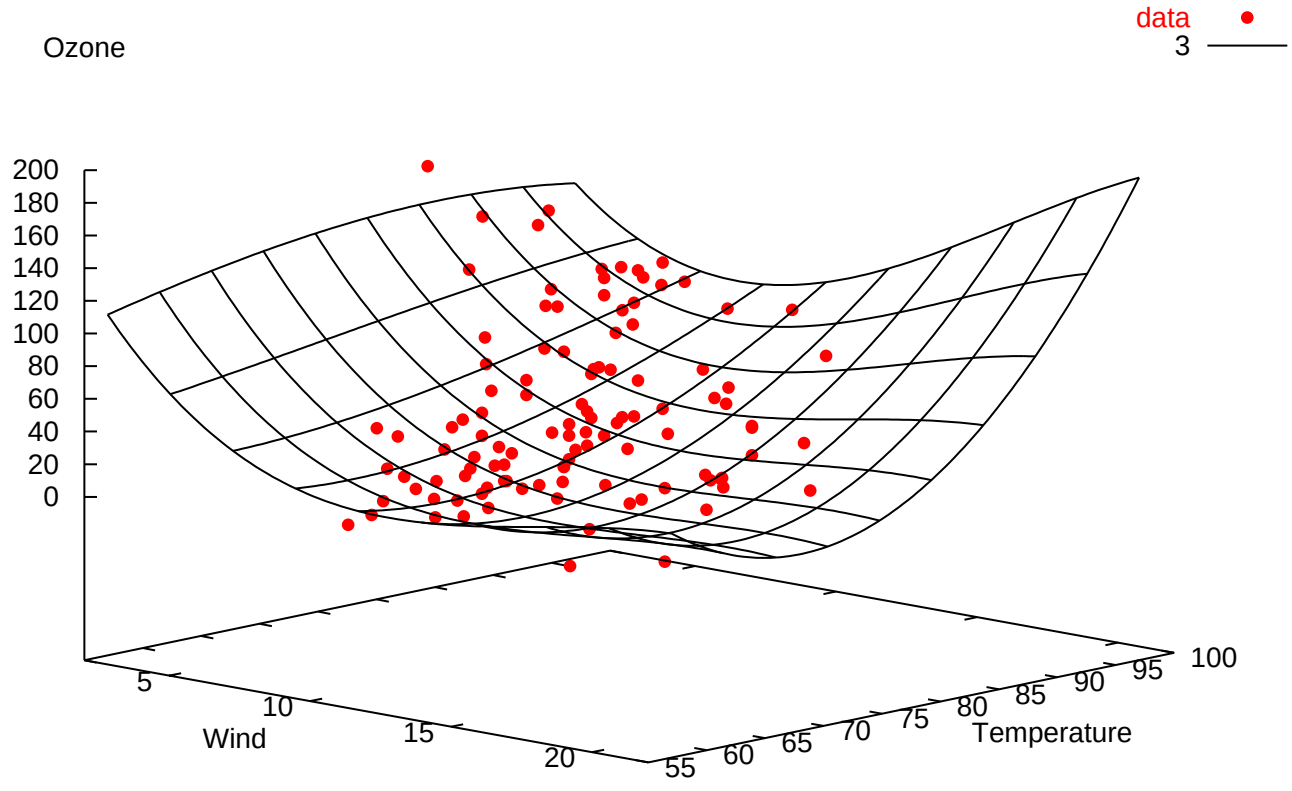


Modèle quadratique



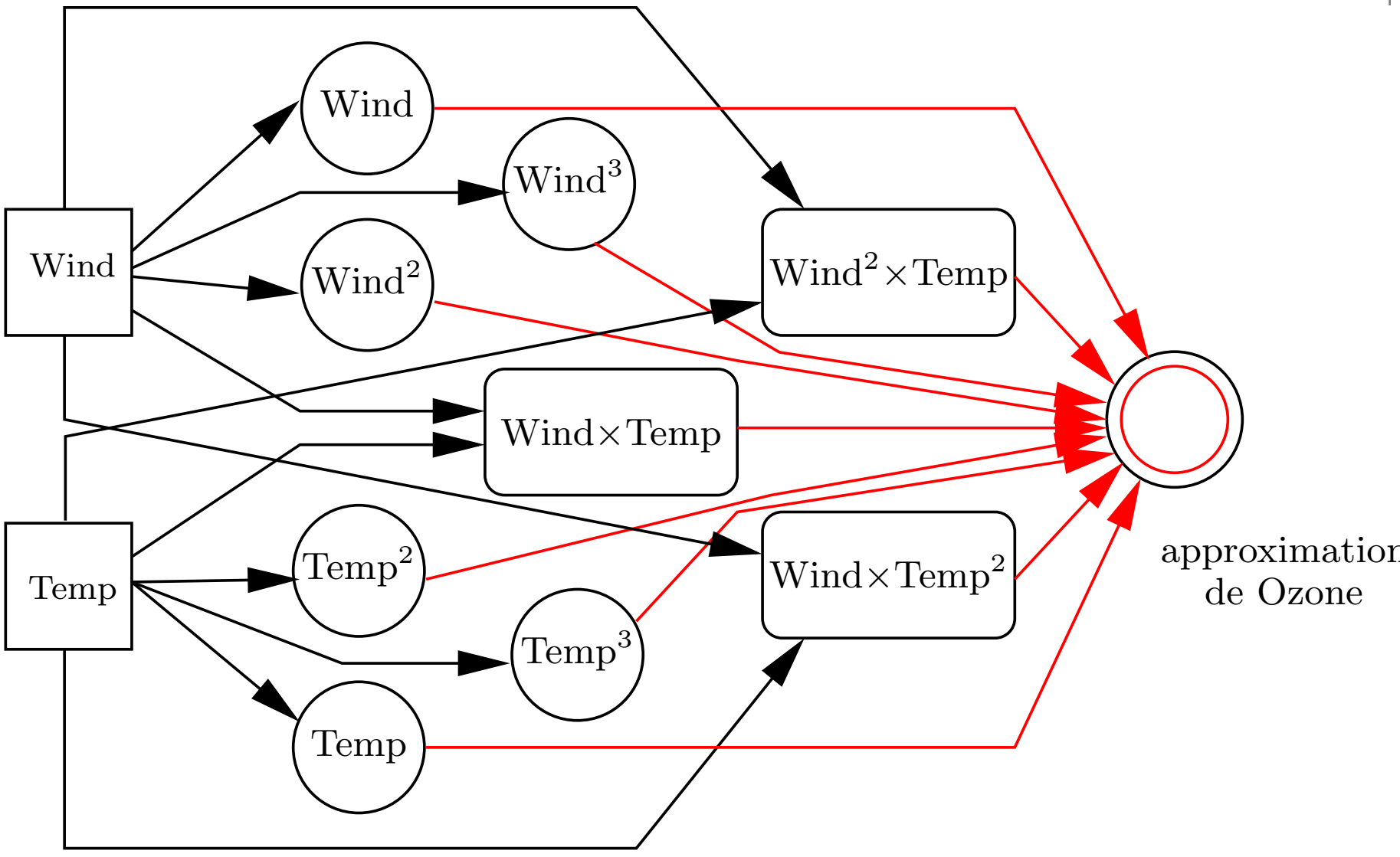
Erreur quadratique moyenne : 344

Modèle de degré 3



Erreur quadratique moyenne : 317

Réseau utilisé



Résumé

Variable	Degré	Neurones	Paramètres	Erreur
1	1	1	2	686
1	2	2	3	543
1	3	3	4	531
1	7	7	8	503
2	1	2	3	459
2	2	5	6	344
2	3	9	10	317

Le nombre de paramètres augmente rapidement avec le degré quand on augmente le nombre de variables : malédiction des grandes dimensions.

Propriétés importantes

Toutes les propriétés du modèle linéaire restent vraies :

- asymptotiquement correct (en augmentant le nombre de données, on s'approche du modèle optimal)
- sens des moindres carrés :
  - approximation de $E(y|x)$
  - maximum de vraisemblance

Les inconvénients aussi :

- estimation des performances
- choix du modèle

On gagne cependant :

- modèle strictement plus puissant
- techniques de régularisation

Choix des transformations non-linéaires

Toute l'"intelligence" du modèle réside dans les ϕ_i, i.e. les transformations non linéaires des entrées. Propriétés recherchées :

- simples à calculer : $\phi_i(x)$ se calcule rapidement
- "puissantes" : obtenir des fonctions complexes (pour $E(Y|X)$) en utilisant le moins de possible de ϕ_i
- faciles à choisir : idéalement indépendantes des données, c'est-à-dire fonctionnant correctement pour toute sorte de données

Formalisation mathématique :

- pouvoir d'approximation d'un ensemble de fonctions
- malédiction des grandes dimensions

Bases fonctionnelles

La puissance du modèle vient de la notion de base :

- dans un espace vectoriel, on peut toujours trouver une base :
  - des vecteurs linéairement indépendants $(e_i)_{i\in\mathbb{N}}$
  - tout vecteur s'écrit $x = \sum_{i\in\mathbb{N}} a_i e_i$ (avec un nombre fini de a_i non nuls)
- dans un espace hilbertien, on peut définir des bases "approximatives", i.e., tout vecteur est **limite** de la série $\sum_{i\in\mathbb{N}} (x|e_i)e_i$

$$\left\| x - \sum_{i=0}^{\infty} (x|e_i)e_i \right\|^2 \to 0$$

- plus généralement, on peut définir une base topologique : tout vecteur est limite d'une série fabriquée avec les vecteurs de la base.

Approximation universelle

En choisissant bien une suite de fonctions $(\phi_i)_{i\in\mathbb{N}}$, on peut représenter approximativement toute fonction (régulière).
Exemples :

- polynômes (Bernstein, Lagrange, etc.)
- B-splines (fonctions polynômiales par morceaux)
- fonctions trigonométriques (séries de Fourier)
- etc.

On a une propriété d'**approximation universelle** si la suite $(\phi_i)_{i\in\mathbb{N}}$ est bien choisie : pour toute fonction "régulière" f et toute précision $\epsilon > 0$, il existe un réseau de neurones à deux couches basés sur la suite $(\phi_i)_{i\in\mathbb{N}}$ (pour la première couche) et utilisant des neurones linéaires dans la seconde couche, qui calcule une fonction proche de f à ϵ près.

Conséquences pratiques

A priori, on n'a que des avantages :

- Le modèle linéaire généralisé est **strictement plus puissant** que le modèle linéaire (approximation universelle)
- l'apprentissage reste rapide (calcul exact des estimateurs par inversion ou SVD)

Il y a quand même des inconvénients :

- le temps de calcul augmente avec le nombre de neurones :
  - modèle linéaire : matrice ZZ^T de taille $(n+1)^2$, algorithmes en n^3 ou n^4
  - modèle pseudo-linéaire : matrice ZZ^T de taille $(k+1)^2$, algorithmes en k^3 ou k^4
- choix difficile de la puissance du modèle (nombre de neurones et fonctions)

Autres aspects pratiques

Critères pratiques pour le choix de la base :

- on évite les polynômes dans la base canonique (i.e., 1, X, X^2, etc.) : la matrice ZZ^T est souvent singulière ou mal conditionnée (difficile à inverser)
- fonctions trigonométriques : très utiles pour modéliser des fonctions pédiodiques
- B-splines : les plus utilisées pour des nombreuses raisons :
  - matrice ZZ^T avec une structure bande (inversion rapide)
  - fonctions très régulières
  - comportement numérique très satisfaisant

B-splines

On se donne $m+1$ nœuds, $t_0 \leq t_1 \leq \ldots \leq t_{m+1}$. Une spline de degré k est une fonction de $[t_0, t_{m+1}]$ dans $\mathbb{R}$

- polynomiale de degré k sur chaque intervalle $]t_i, t_{i+1}[$
- de classe C^{k-1} sur tout $[t_0, t_{m+1}]$

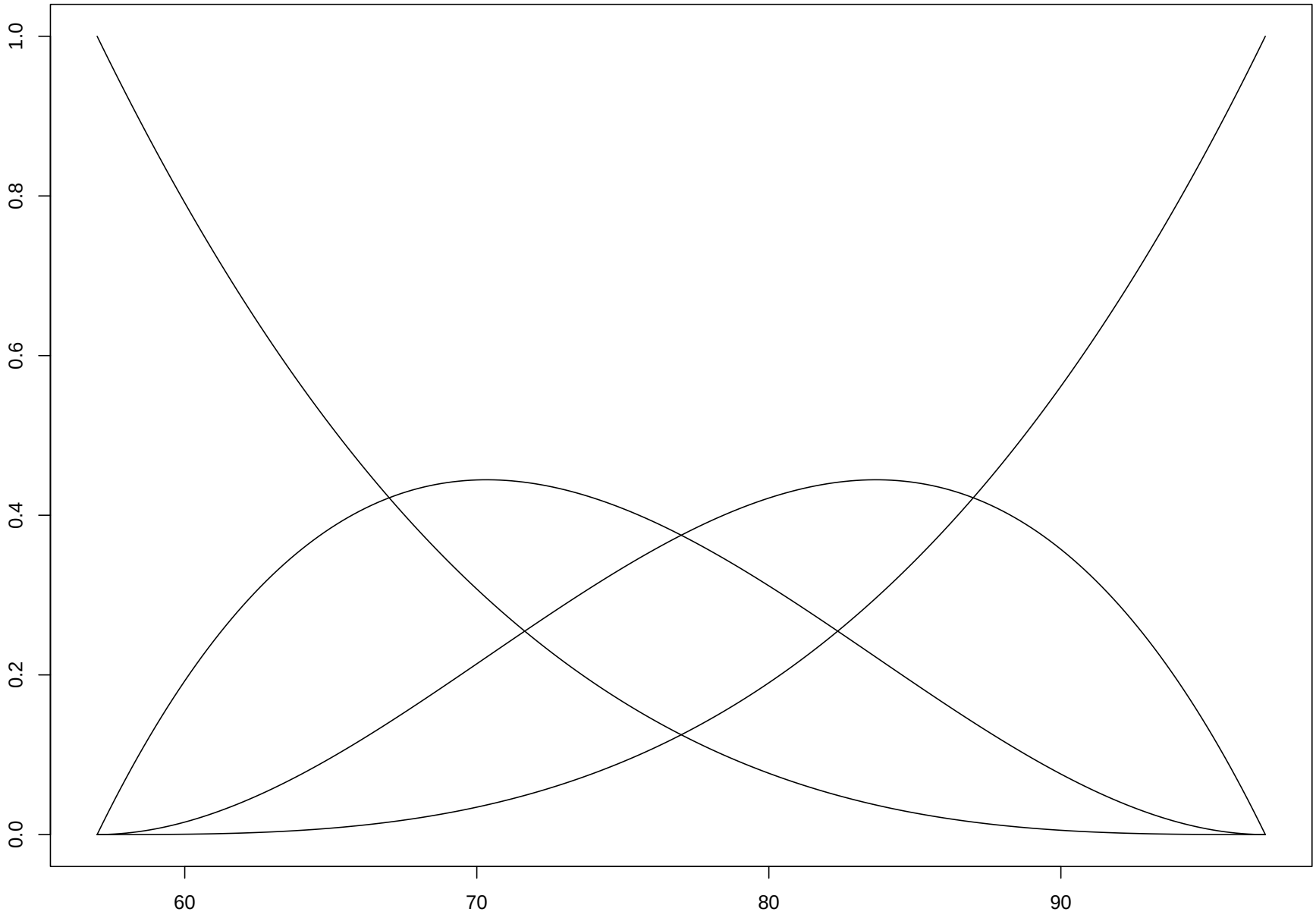
On définit par récurrence une base de l'ensemble des splines de degré k, les B-splines. Si on note $N_{i,k}$ la i fonction de la base de splines de degré k, on a :

$$N_{i,0}(t) = \begin{cases} 1 & \text{si } t_i \leq t < t_{i+1} \\ 0 & \text{sinon} \end{cases}$$

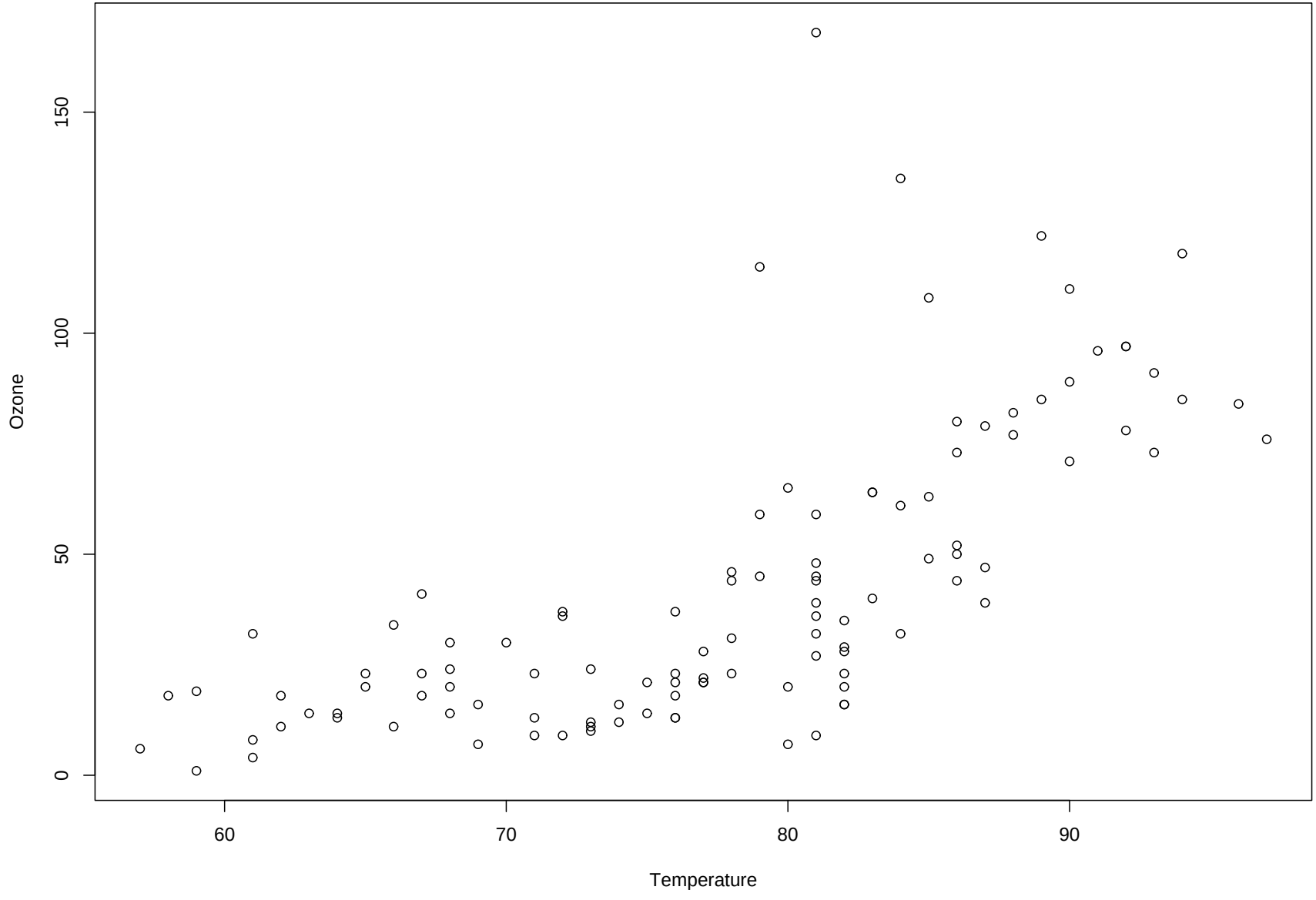
$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t)$$

Il faut que les nœuds extrêmes soient de multiplicité $k+1$.

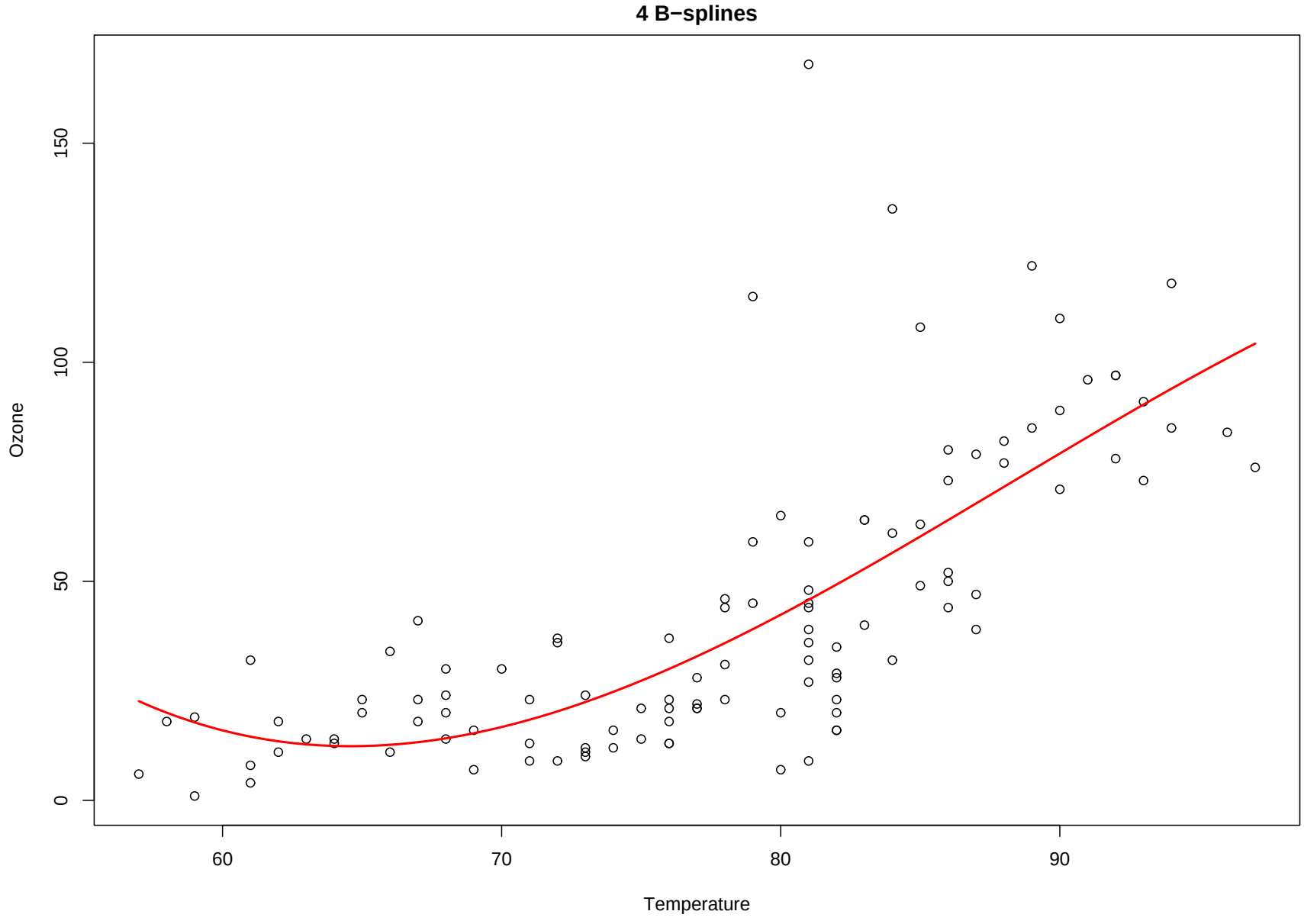
Exemple : B-splines de degré 3 (8 noeuds)



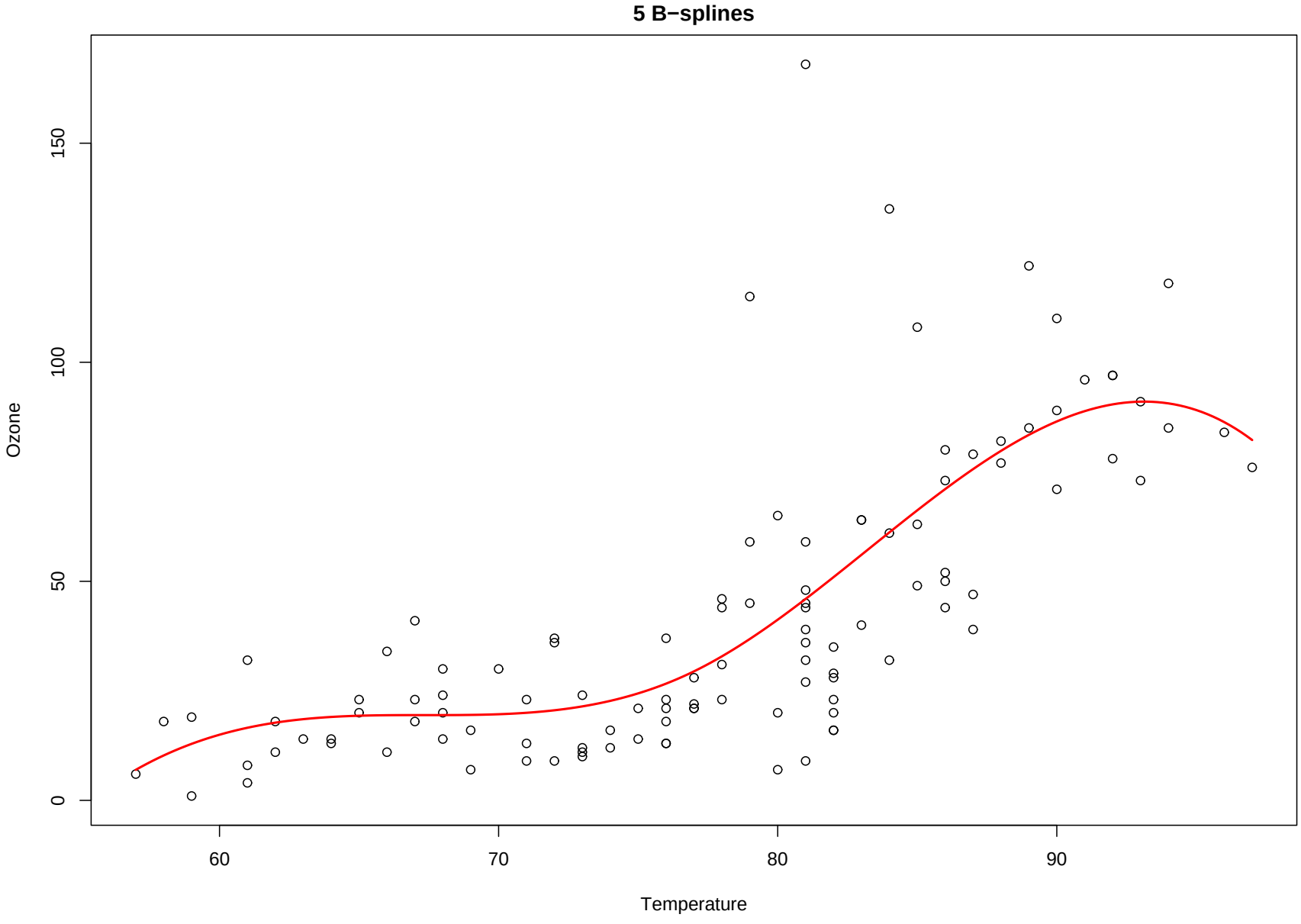
Exemple de modèle linéaire B-splines



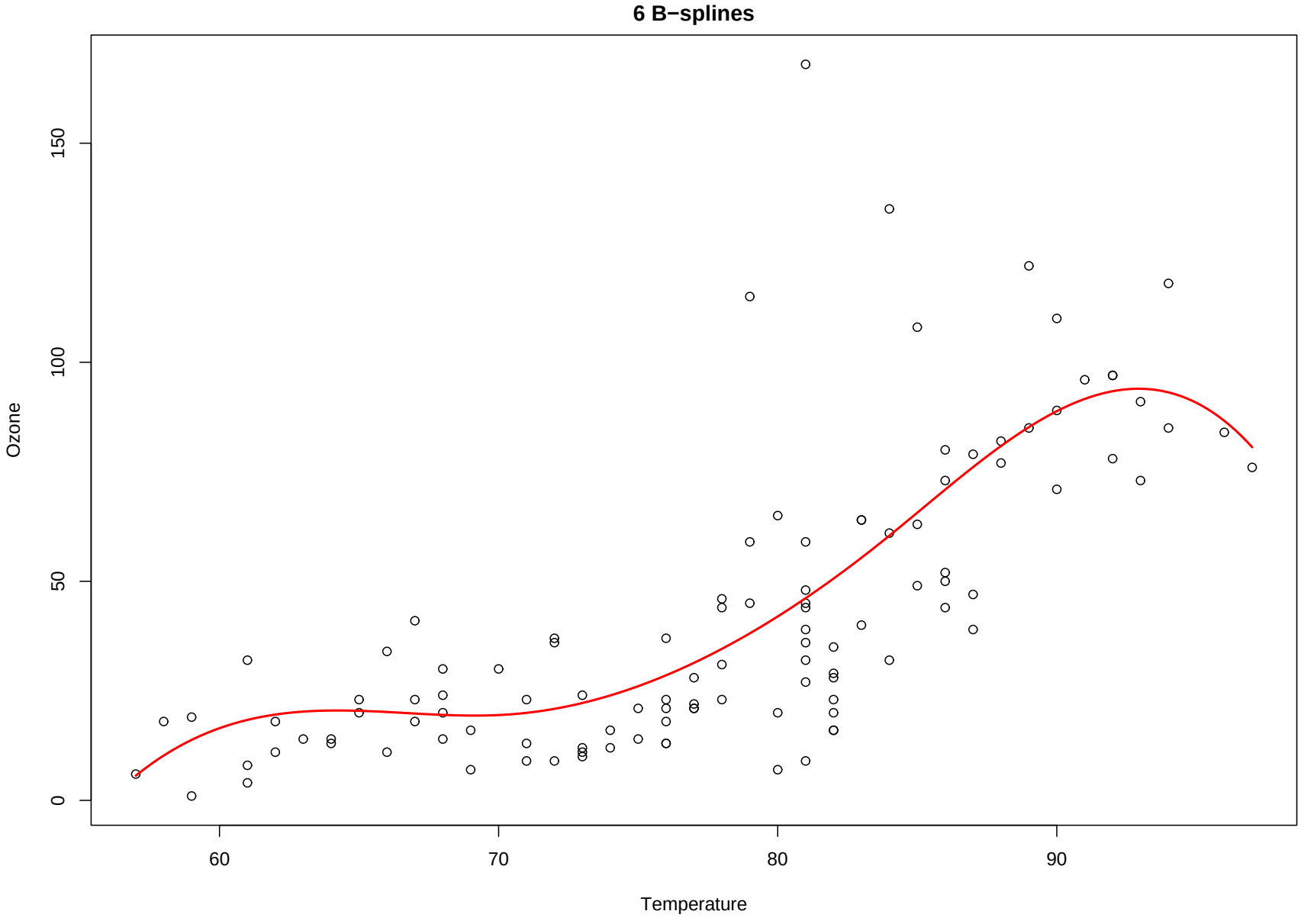
Exemple de modèle linéaire B-splines



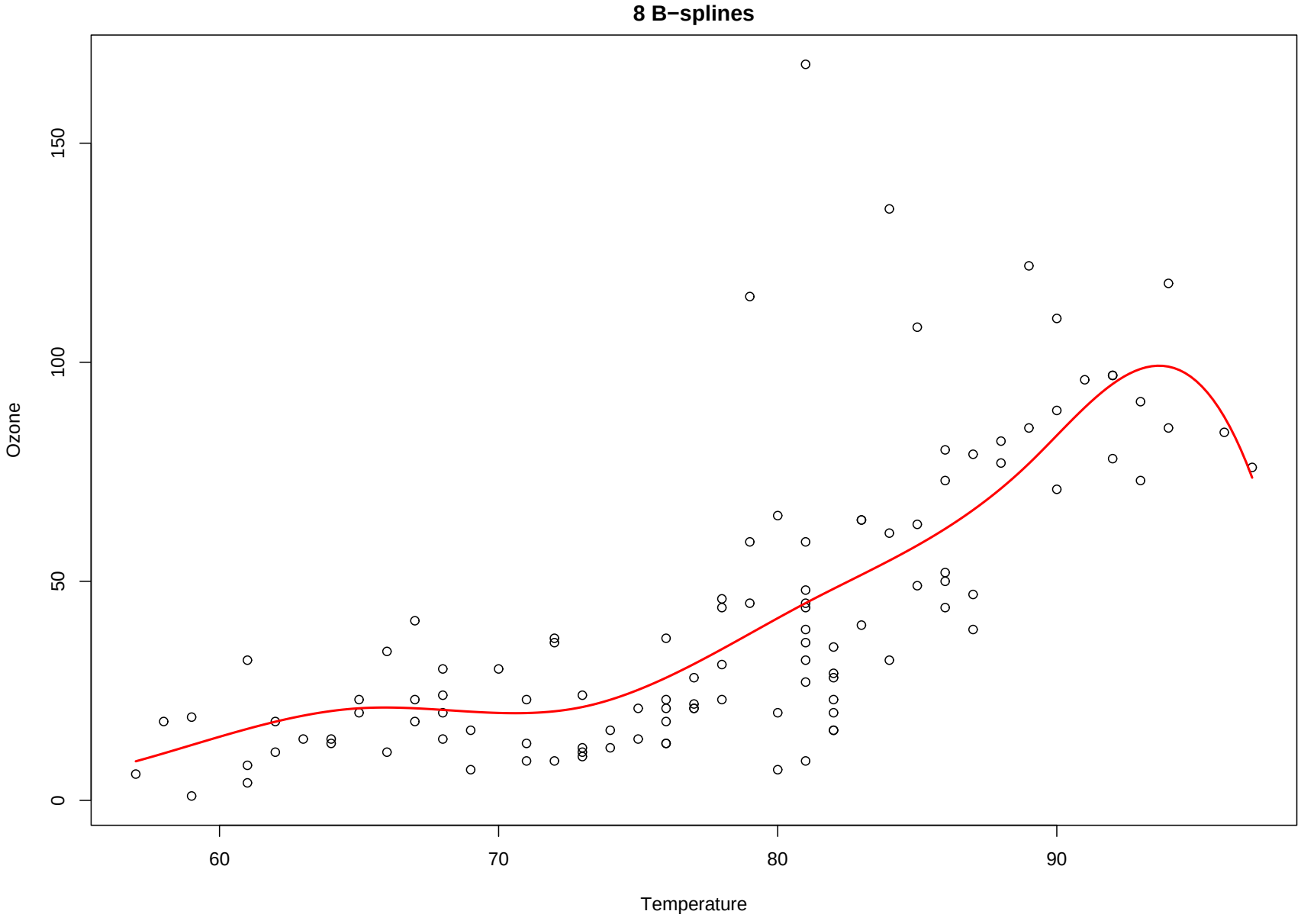
Exemple de modèle linéaire B-splines



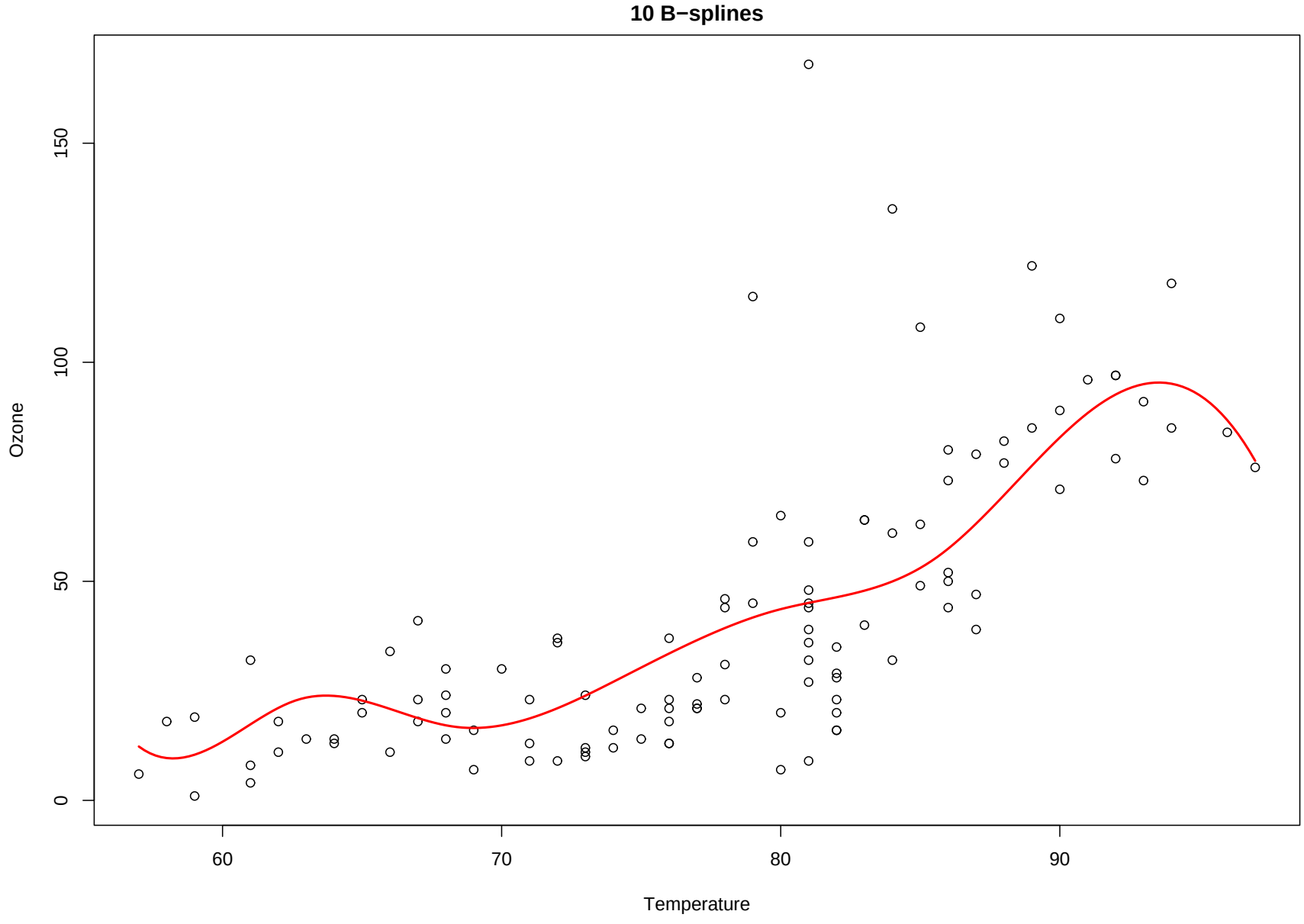
Exemple de modèle linéaire B-splines



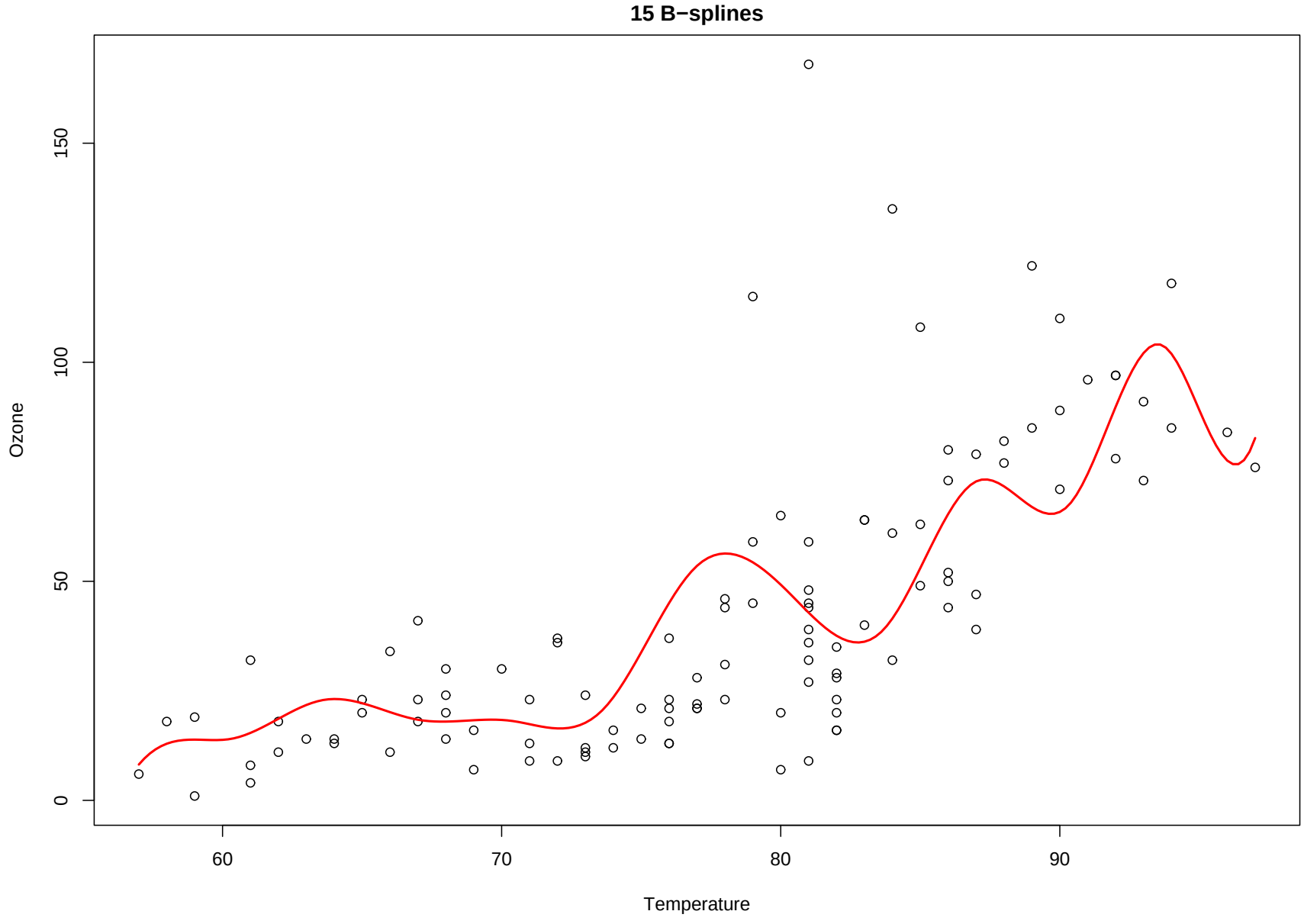
Exemple de modèle linéaire B-splines



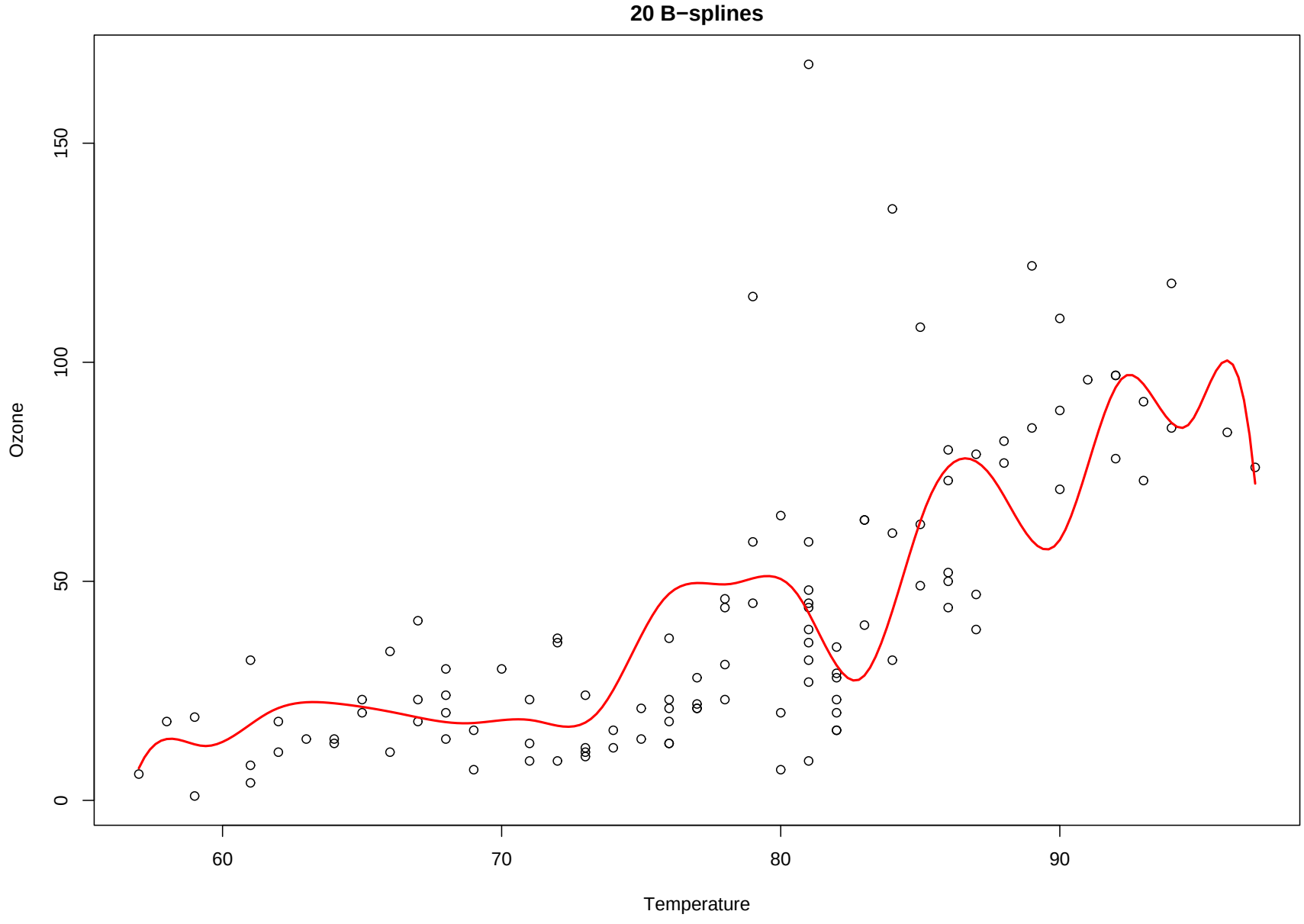
Exemple de modèle linéaire B-splines



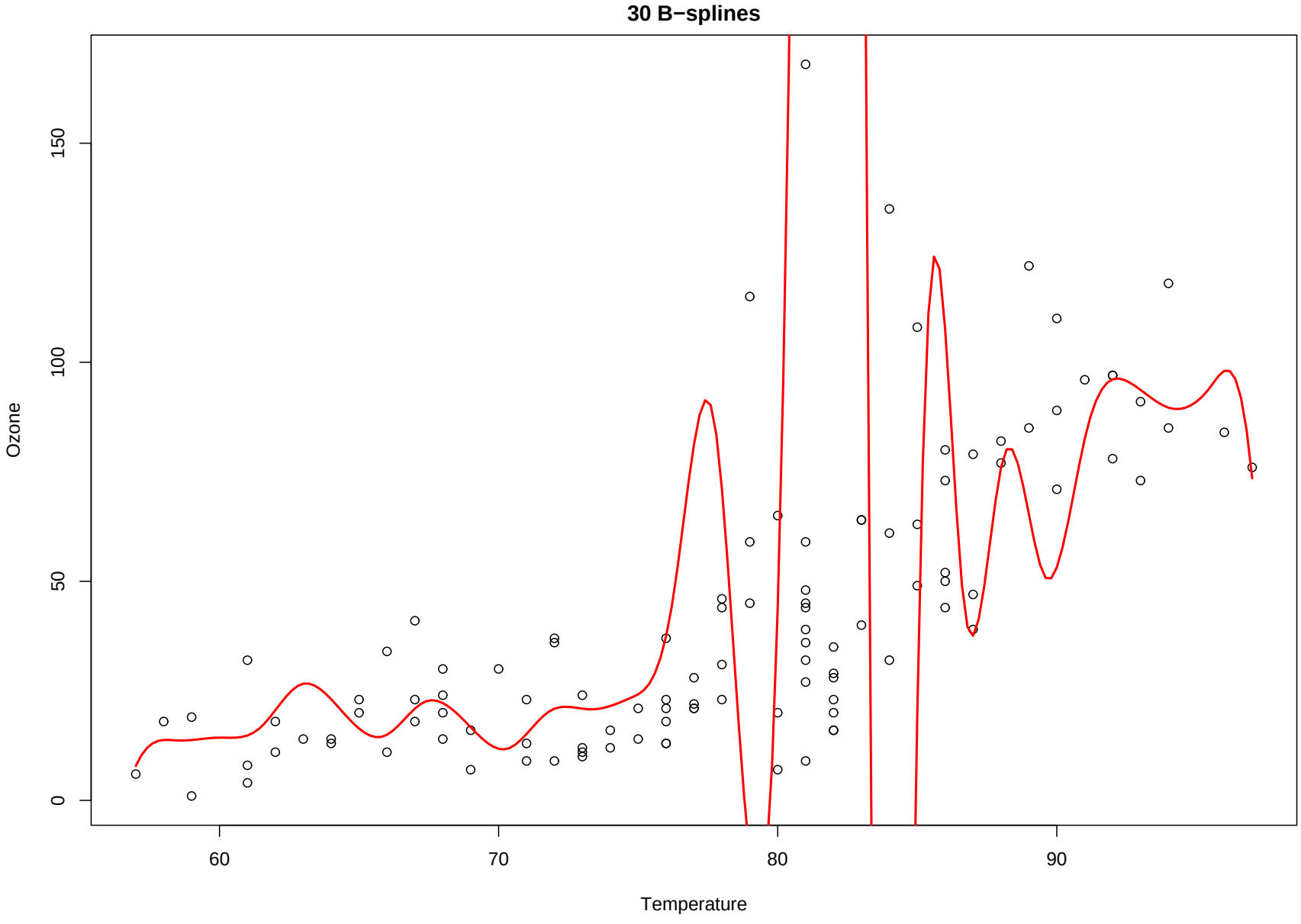
Exemple de modèle linéaire B-splines



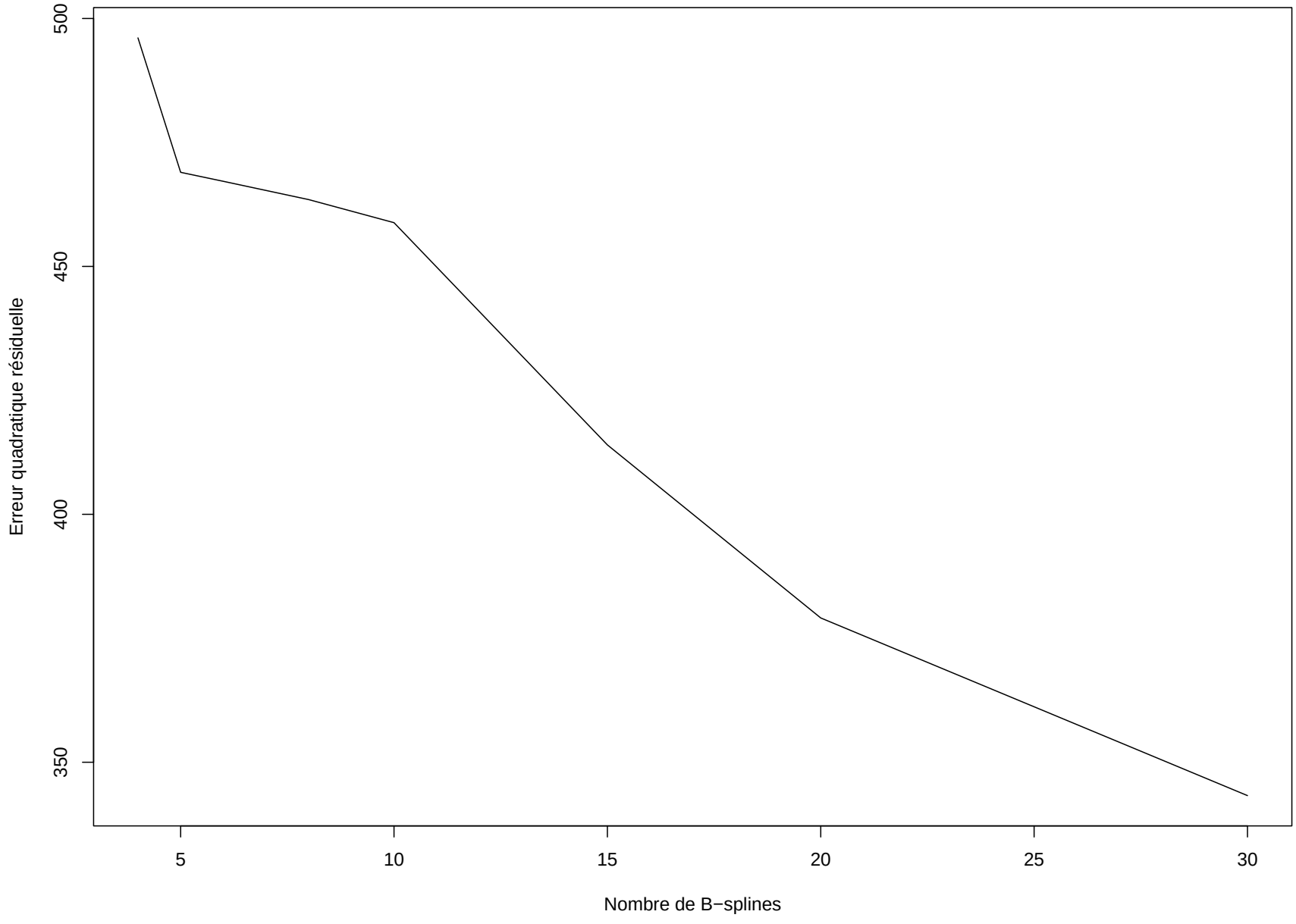
Exemple de modèle linéaire B-splines



Exemple de modèle linéaire B-splines



Evolution de l'erreur



Problème de dimension

Première apparition de la **malédiction des grandes dimensions** (*curse of dimensionality*). Polynômes :

- dans $\mathbb{R}$, $a_0 + a_1 x + a_2 x^2 + \ldots$: le degré k correspond à $k+1$ coefficients
- dans $\mathbb{R}^2$, $a_{0,0} + a_{1,0} x + a_{1,0} y + a_{1,1} xy + a_{2,0} x^2 + a_{0,2} y^2 \ldots$: le degré k correspond à $\frac{(k+1)(k+2)}{2}$ coefficients
- dans $\mathbb{R}^n$, le degré k correspond à $\frac{(k+n)!}{k!n!}$, soit $O(n^k)$ coefficients pour n grand

Pour toutes les bases classiques (B-splines, séries de fourier, etc.), on a des résultats similaires.

Problème de dimension (2)

Traduction pratique :

- plus la dimension d'entrée (le nombre de variables) est élevée, plus l'augmentation de puissance du modèle est coûteuse
- la croissance est **exponentielle**

Conséquences :

- les bases classiques sont délicates à utiliser pour $n > 2$
- le réglage de la puissance pose problème : pour passer du degré k au degré $k+1$, on doit **multiplier** le nombre de neurones (et donc de coefficients) par n !

On peut cependant revenir à une croissance **linéaire**, i.e., en dimension n, on peut augmenter la puissance du modèle en ajoutant les neurones 1 par 1, avec $\simeq n$ paramètres par neurone.

Fonction radiale de base (RBF)

Une solution simple pour éviter l'explosion du nombre de paramètres :

$$\phi_j(x) = \psi_j\left(\|x - \mu_j\|^2\right)$$

où ψ_j est une fonction de $\mathbb{R}$ dans $\mathbb{R}$. On connaît des conditions très larges sur ψ_j qui permettent de conserver la propriété d'approximation universelle. En général, on prend la fonction gaussienne :

$$\psi_j(x) = e^{-\frac{x}{2\sigma_j^2}}$$

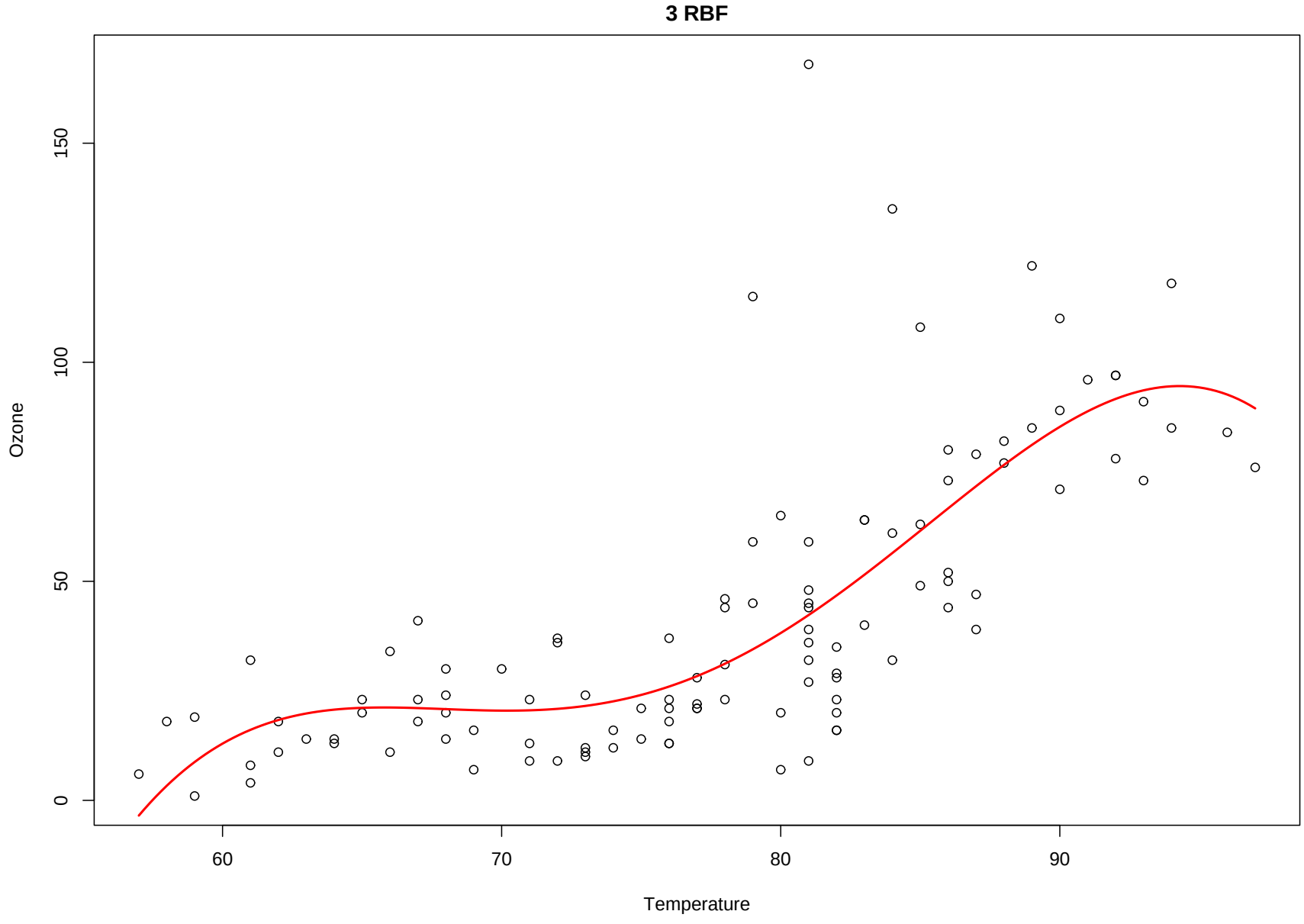
Augmentation de puissance : ajout d'un neurone, i.e. de $n+1$ paramètres numériques (μ et σ) $\Rightarrow$ linéaire.

Fonction radiale de base (2)

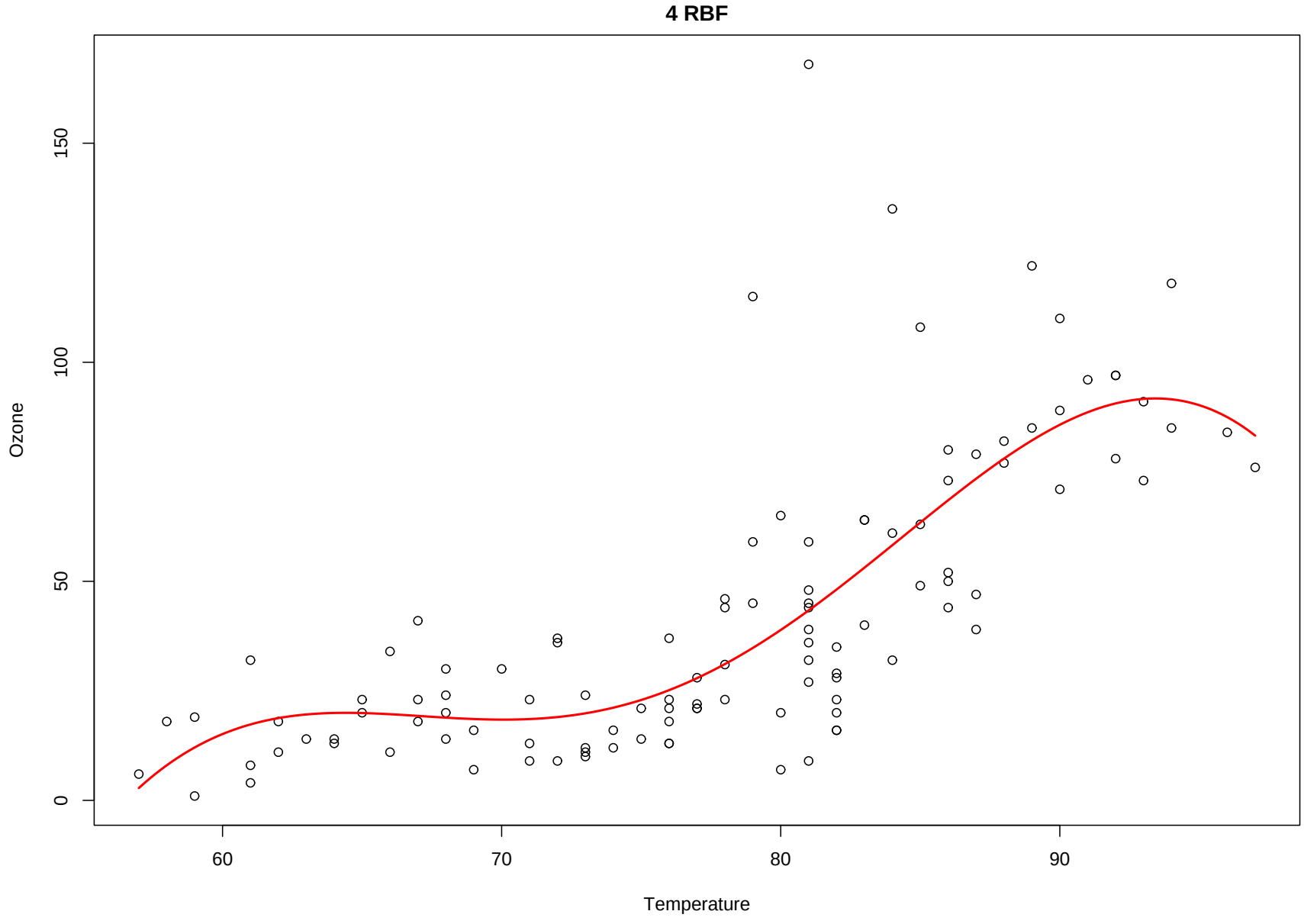
Choix des μ_j et σ_j :

- Solution de base :
  - répartition uniforme des μ_j
  - σ_j deux fois l'écart moyen entre les μ_k
- Solutions évoluées :
  - recherche active dans le domaine
  - adaptation aux X :
    - plus de neurones (i.e., de valeurs de μ_j) dans les zones de forte densité en observations
    - σ_j faible dans les zones de forte densité
  - adaptation aux Y :
    - plus de neurones dans les zones où Y varie rapidement
    - σ_j faible dans les zones d'évolution rapide

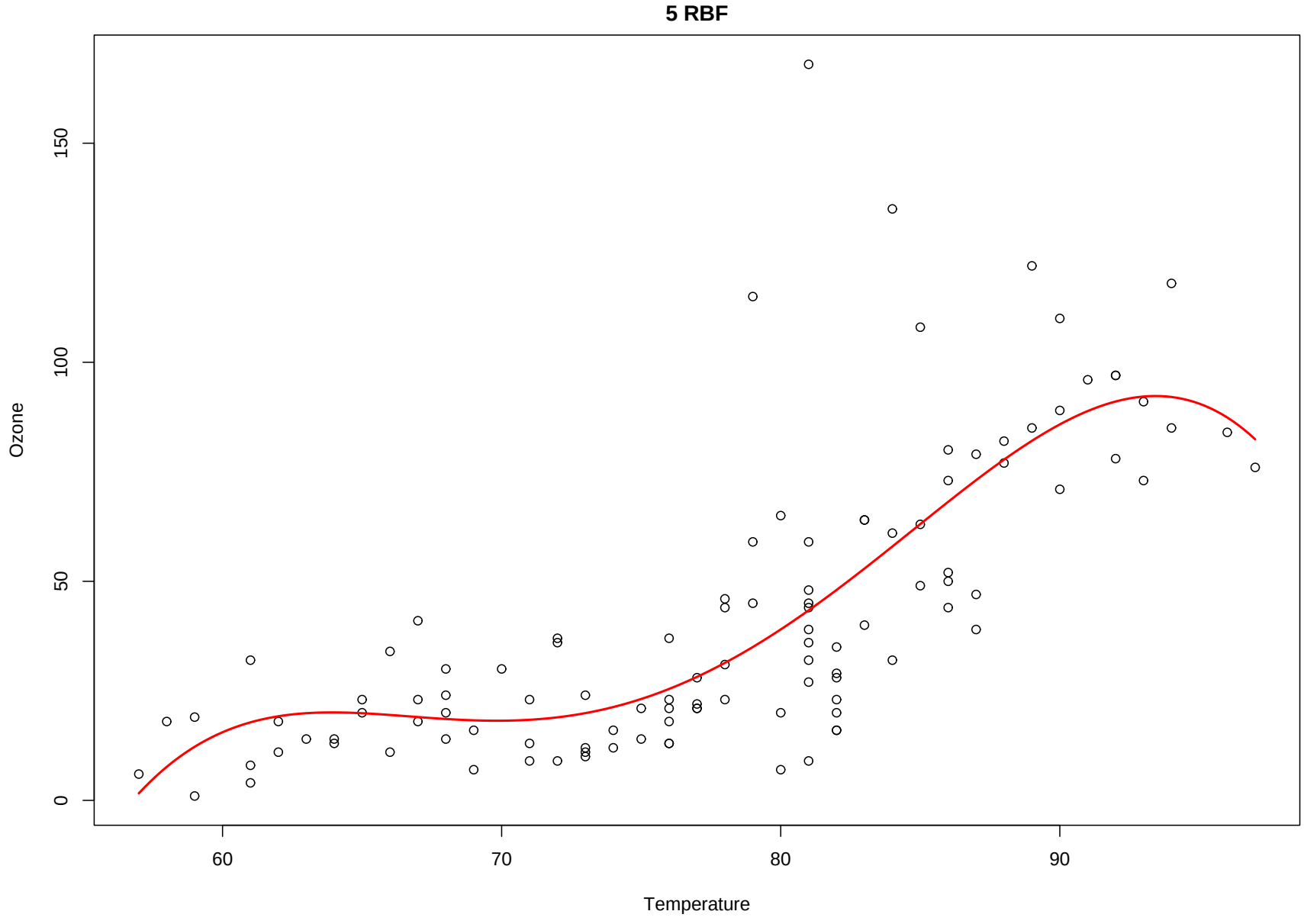
Exemple d'utilisation des RBF



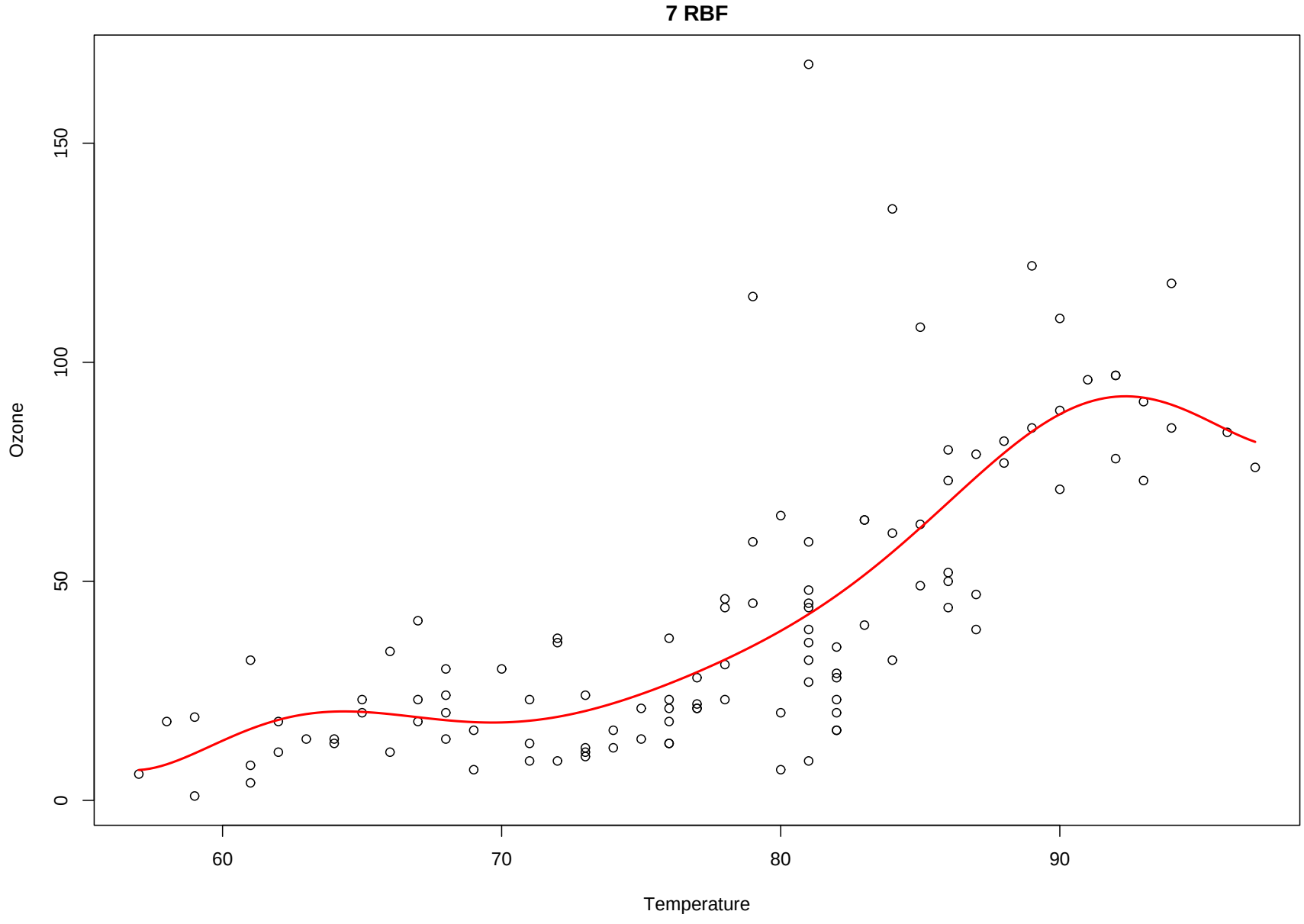
Exemple d'utilisation des RBF



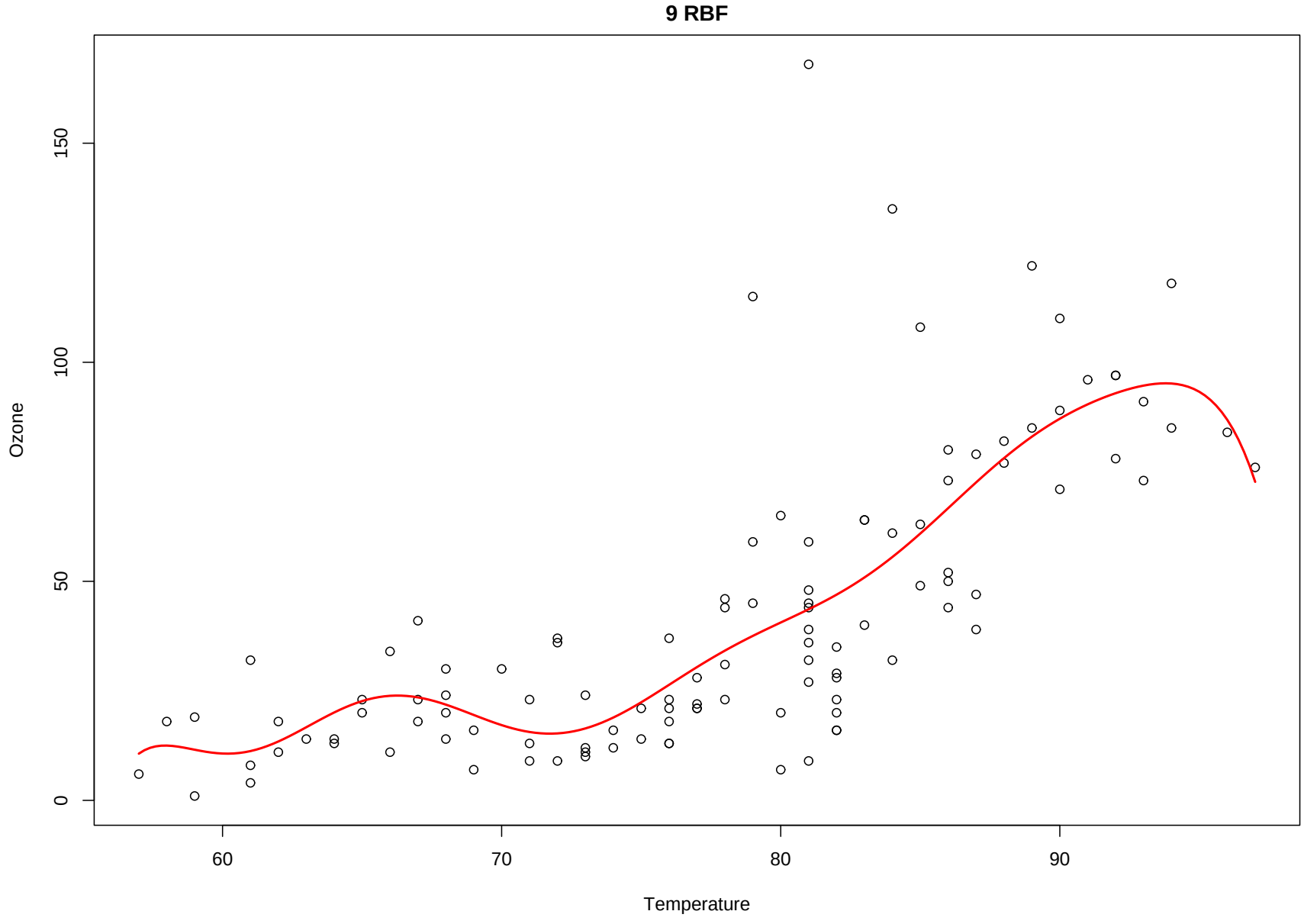
Exemple d'utilisation des RBF



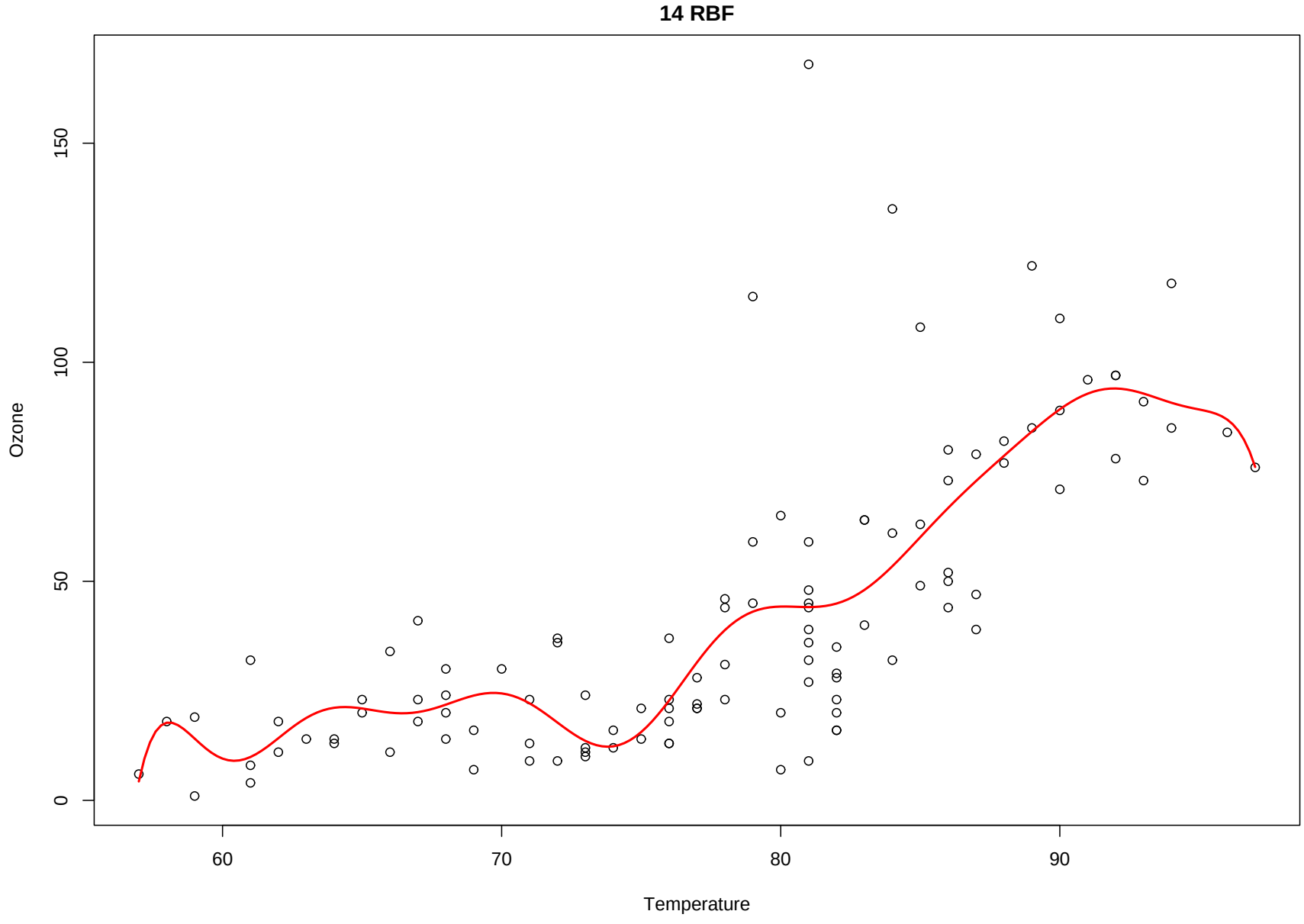
Exemple d'utilisation des RBF



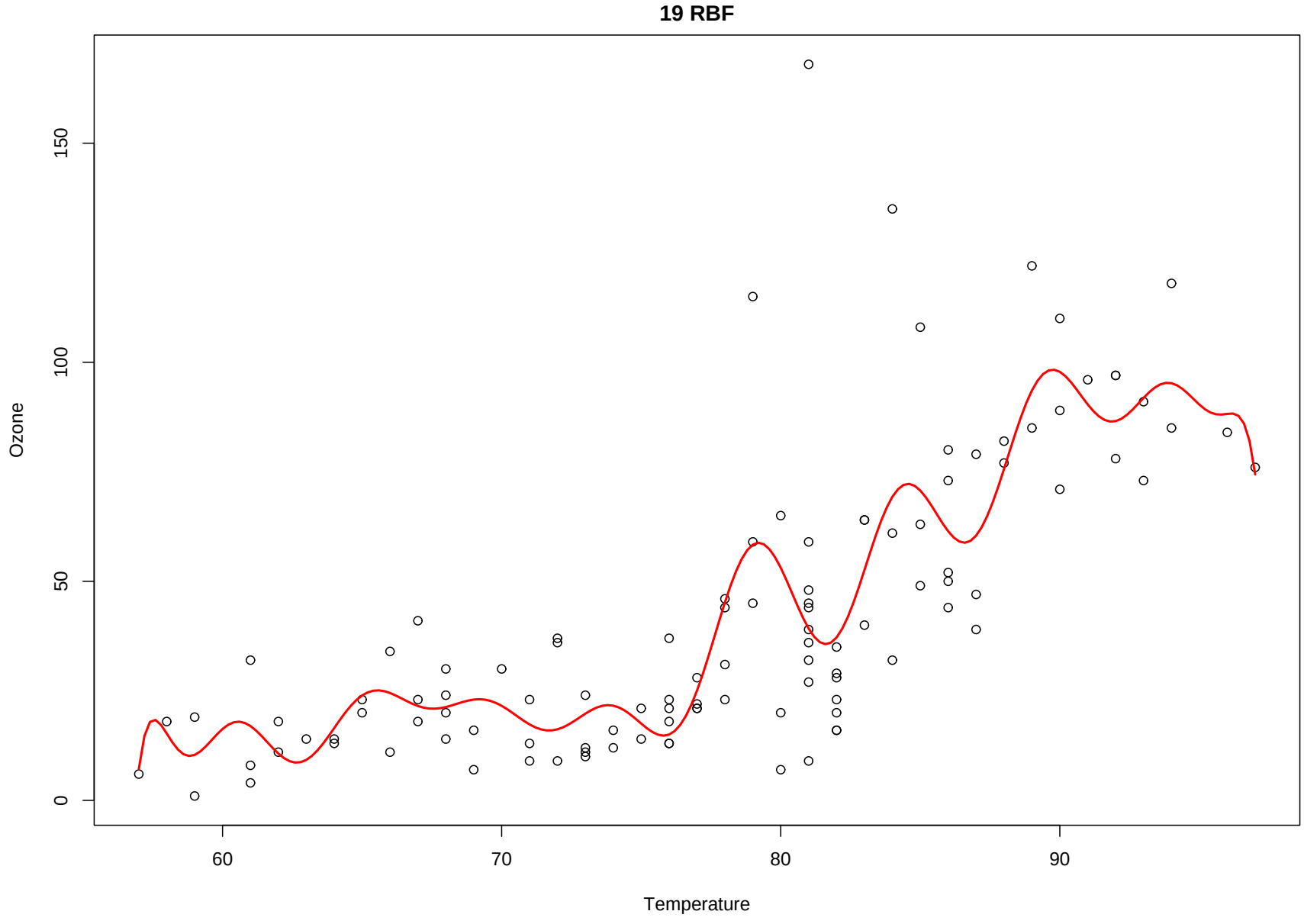
Exemple d'utilisation des RBF



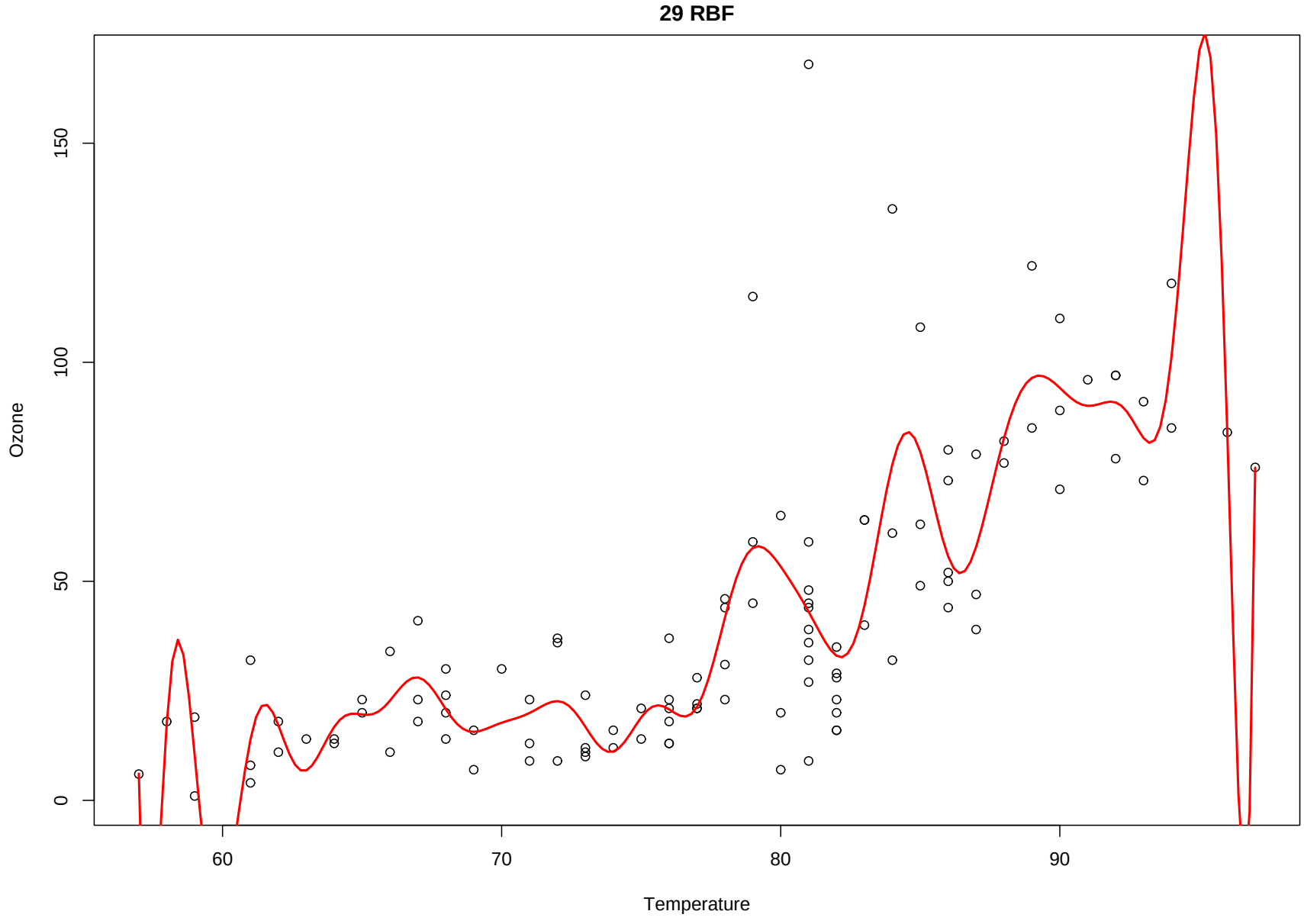
Exemple d'utilisation des RBF



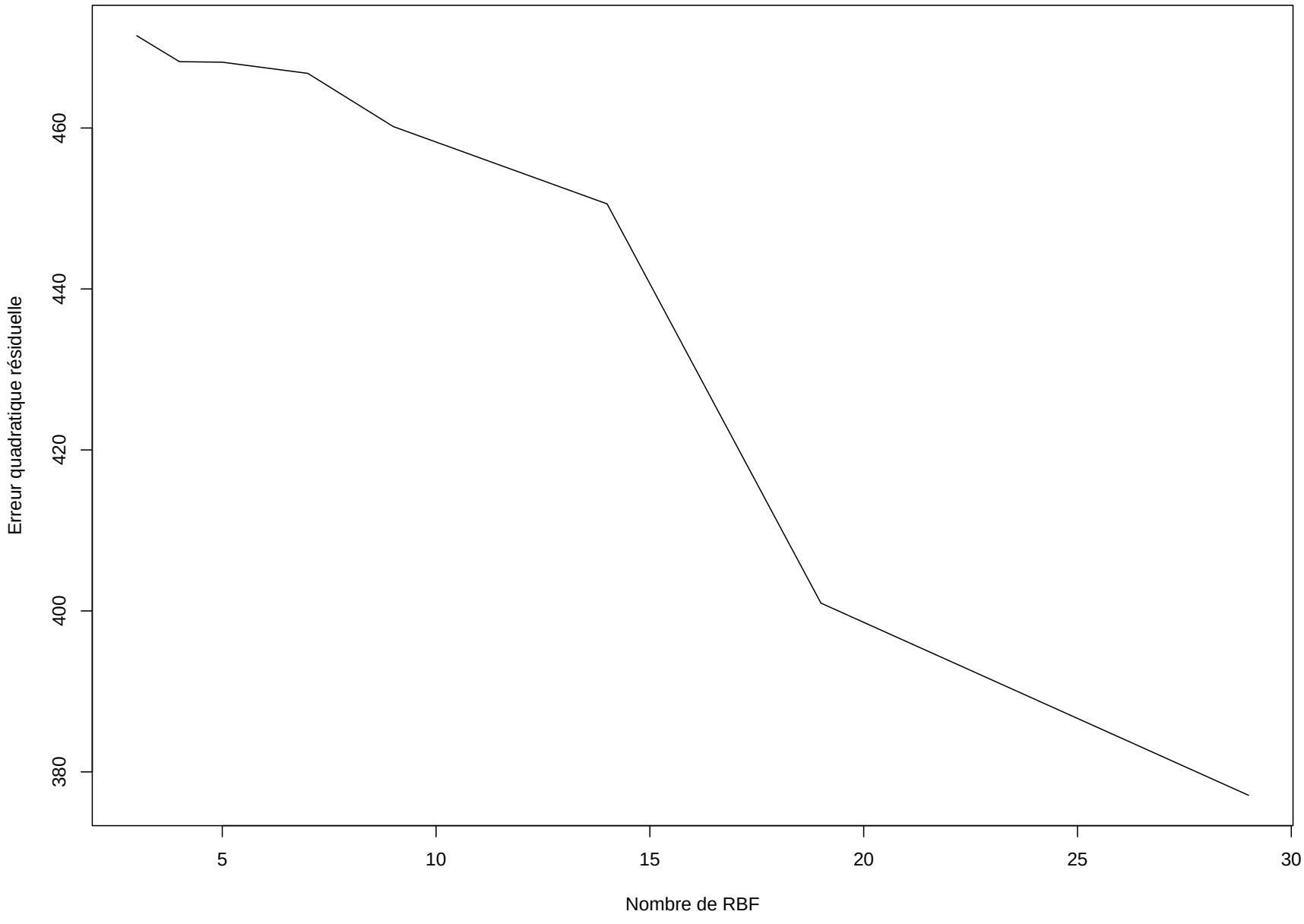
Exemple d'utilisation des RBF



Exemple d'utilisation des RBF



Evolution de l'erreur



On reste maudit...

Le problème des grandes dimensions est seulement en partie résolu. En effet, Barron a montré (en 93) les résultats suivants :

- étant donnée une base de fonctions $(\phi_i)_{i\in\mathbb{N}}$ fixée, la qualité de l'approximation d'une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ par cette base se comporte en $O\left(\frac{1}{k^{\frac{2}{n}}}\right)$ où k désigne le nombre de fonctions utilisées
- pour un réseau de neurones à 2 couches totalement adaptatives, la qualité de l'approximation se comporte en $O\left(\frac{1}{k}\right)$, où k désigne le nombre de neurones de la première couche

On gagne grâce à l'adaptativité de la première couche, mais l'optimisation n'est plus directe.

Solutions pratiques

Dans la pratique :

- pour les dimensions 1 et 2, on a intérêt à utiliser un modèle linéaire généralisé
- pour les dimensions > 2, plusieurs options :
  - on peut passer directement au modèle non linéaire (perceptron multi-couche)
  - on peut aussi procéder en deux temps :
    1. choix de la base par un algorithme rapide travaillant sur les données (e.g. RBF adapté aux données)
    2. puis modèle linéaire généralisé
  - on peut aussi rendre complètement adaptative la première couche : la solution linéaire n'est plus possible

Contrôle de complexité : exemple RBF

Une approche très (trop) simple d'adaptation aux données : on prend autant de neurones que de données i.e., $k = N$ et on fixe $\mu_i = x^i$. On a donc la fonction (cas linéaire)

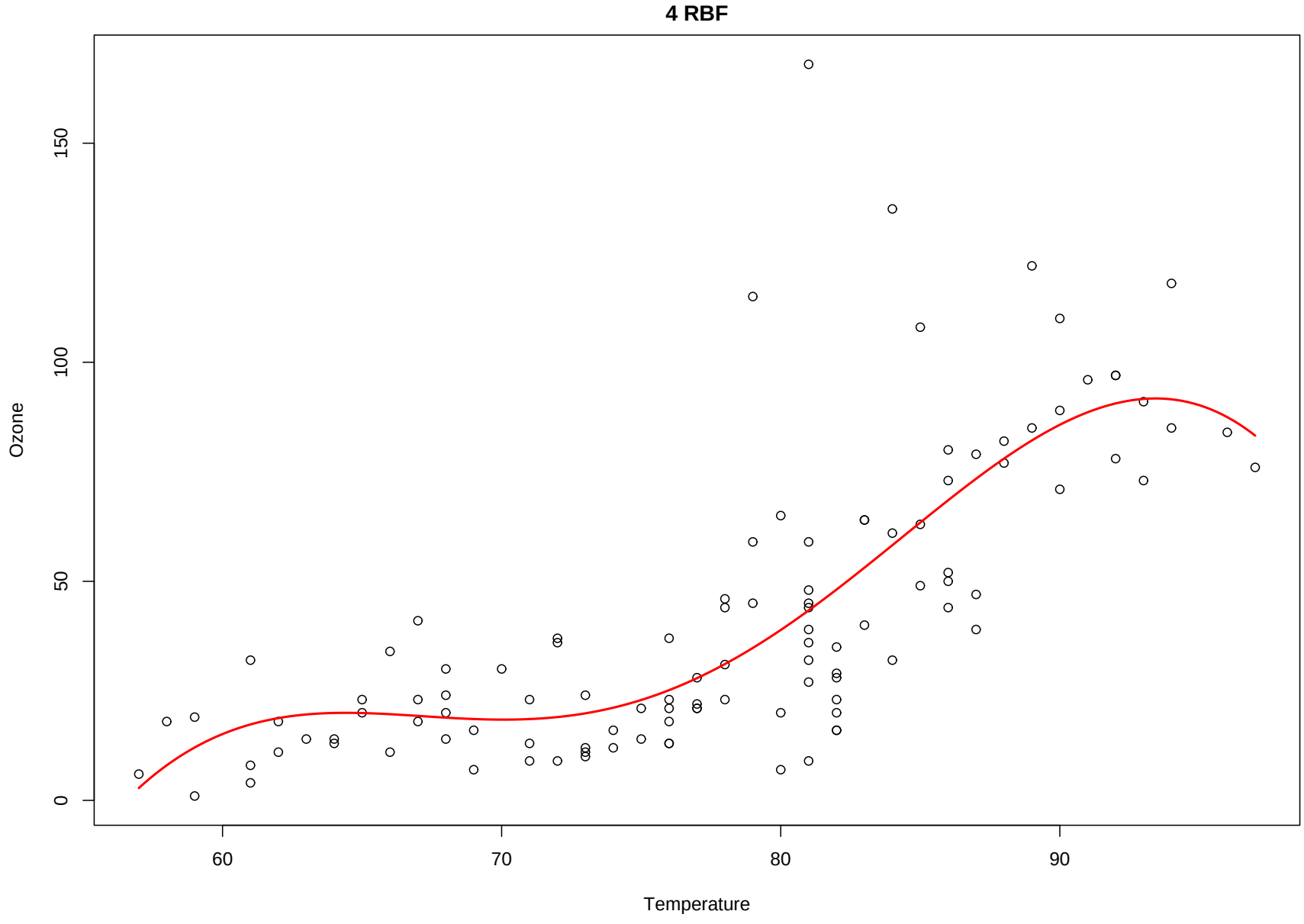
$$f(x)_j = \sum_{i=1}^{N} a_{ji} \phi_i \left(\left\| x - x^i \right\|^2 \right) + b_j$$

Pour ϕ_i gaussienne, il ne reste qu'à régler σ_i. En général, on le prend égal à deux fois la distance moyenne entre les exemples :

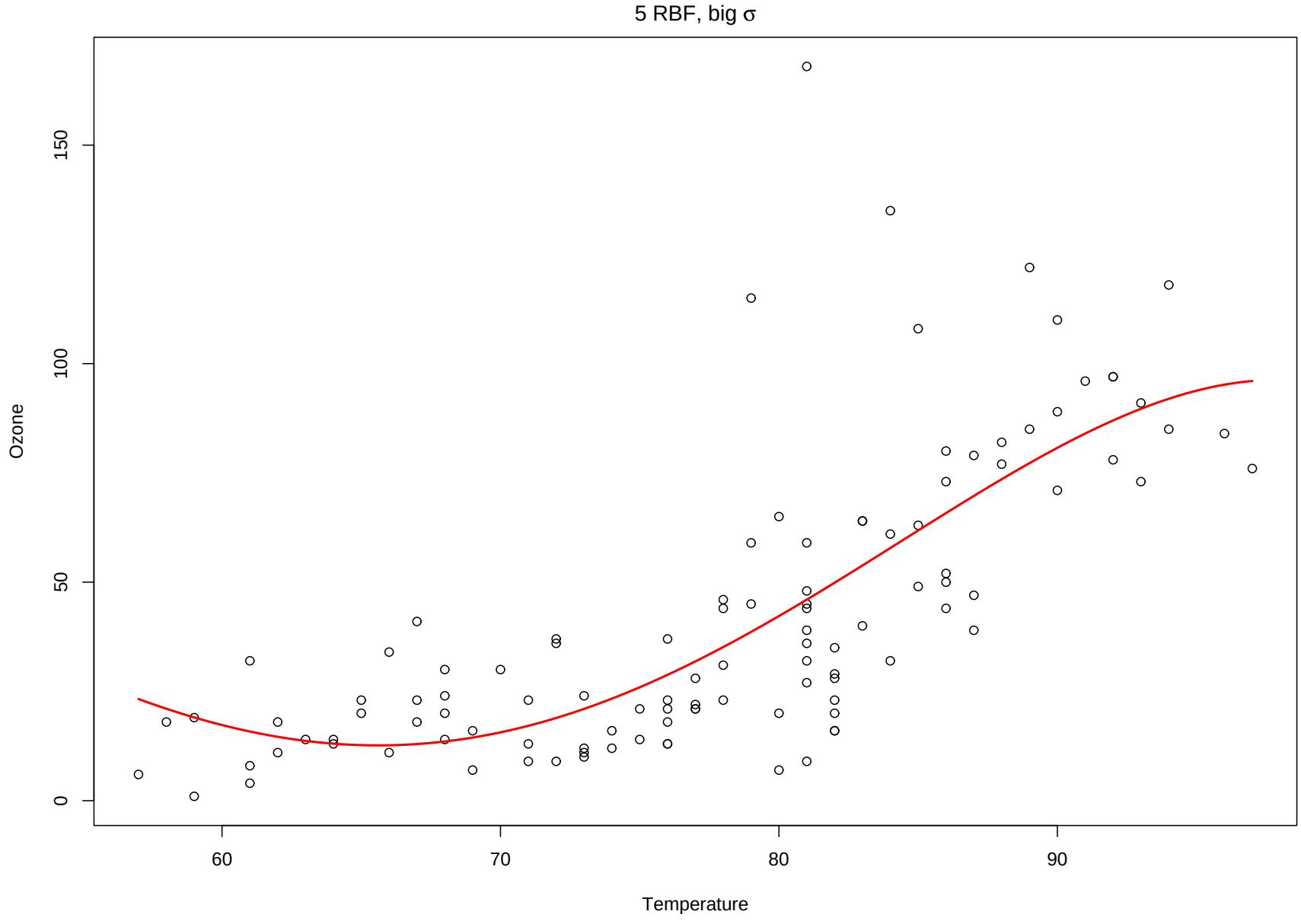
- trop petit : trop sensible aux données
- trop grand : trop insensible aux données

Sur l'exemple suivant, on a gardé k petit par rapport à N.

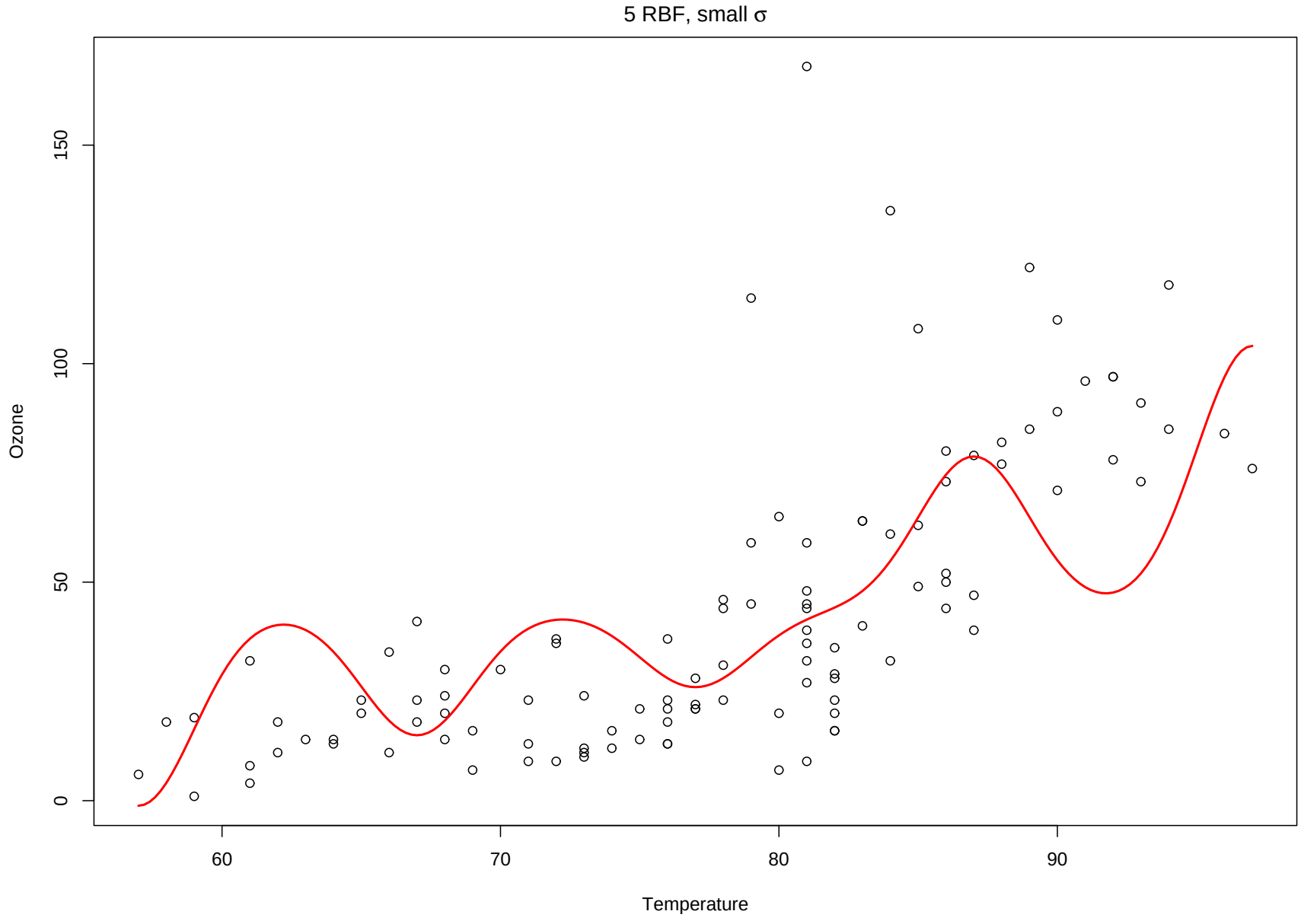
Exemple



Exemple



Exemple



Contrôle de complexité (2)

De façon plus générale, on doit toujours régler au moins un paramètre global qui contrôle la puissance du modèle :

- le degré dans le cas polynomial
- le nombre de fonctions dans le cas général (B-splines, séries de Fourier, etc.)

L'expérience prouve qu'on a toujours le compromis suivant :

- un modèle faible n'approche pas bien le lien entre Y et X (biais important)
- un modèle puissant est sensible au bruit (variance importante)

C'est le dilemme biais/variance. Un autre problème est la granularité du réglage : comment faire pour avoir des "demi"-neurones ?

Régularisation

Pour améliorer les résultats, on introduit une notion de régularisation :

- dans la nature, l'application $x \mapsto E(Y|X = x)$ doit être régulière
- régularité $\Rightarrow$ variations pas trop rapides $\Rightarrow$ bornes sur les dérivées (par exemple)
- principe : on cherche une solution régulière dans les solutions possibles

Mise en œuvre du principe : on ajoute à l'erreur quadratique une pénalité qui sera d'autant plus grande que la fonction est irrégulière.

Régularisation (2)

On a donc la nouvelle erreur :

$$\mathcal{C}(w) = \mathcal{E}(w) + \nu\mathcal{P}(w)$$

avec

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

$\mathcal{P}(w)$ est le terme de pénalité. L'importance de ce terme par rapport à l'ajustement aux données est réglé par ν.

Dans la pratique, on essaie de prendre pour $\mathcal{P}$ une fonction quadratique pour conserver une résolution simple.

Régularisation (3)

Intérêts de la régularisation :

- Contrôle souple : on peut faire varier ν continuement
- Effet connu à l'avance : minimiser $\mathcal{C}(w)$ revient à minimiser $\mathcal{E}(w)$ sous la contrainte $\mathcal{P}(w) < \lambda$
- S'applique pour n'importe quelles fonctions ϕ_i

On maîtrise beaucoup mieux l'effet de la régularisation que l'effet de l'ajout ou de la suppression d'un neurone.

Pénalisations classiques :

- valeurs des paramètres numériques (*weight decay*)
- dérivée(s) première(s) de $E(Y|X = x)$
- dérivée(s) seconde(s) de $E(Y|X = x)$ (contrôle des pics)

Pénalité sur les poids

Le modèle le plus simple de pénalité est donné par

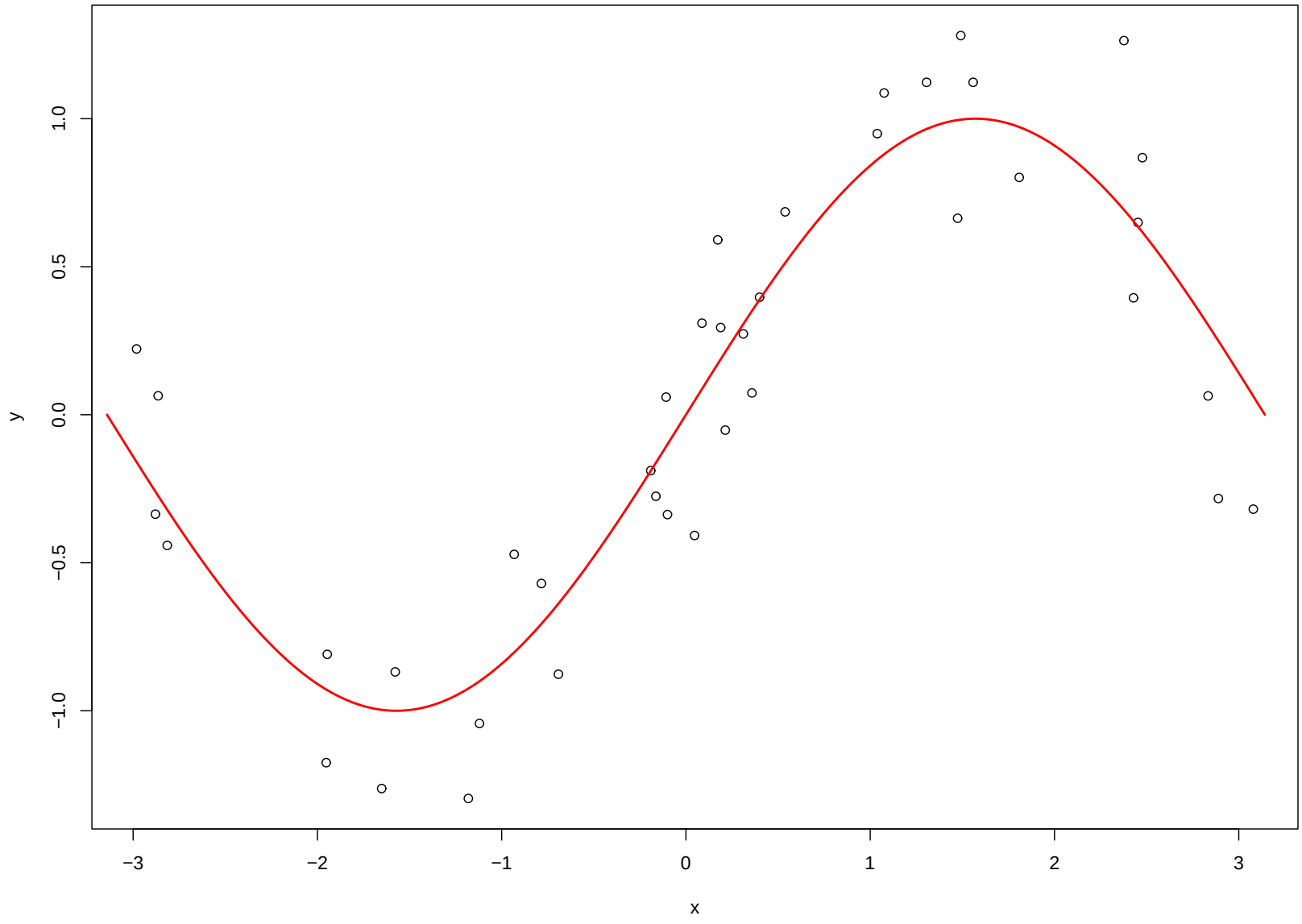
$$\mathcal{P}(w) = \|w\|^2 = \sum_{i=1}^{p} w_i^2$$

- Correspond aux dérivées d'ordre 1 pour le cas du modèle linéaire (si on ne compte pas le terme constant dans la pénalité)
- Utile en pratique pour les perceptrons multi-couches, pas vraiment pour le modèle pseudo-linéaire

Cette contrainte est souvent trop forte. Elle conduit à la nouvelle équation matricielle

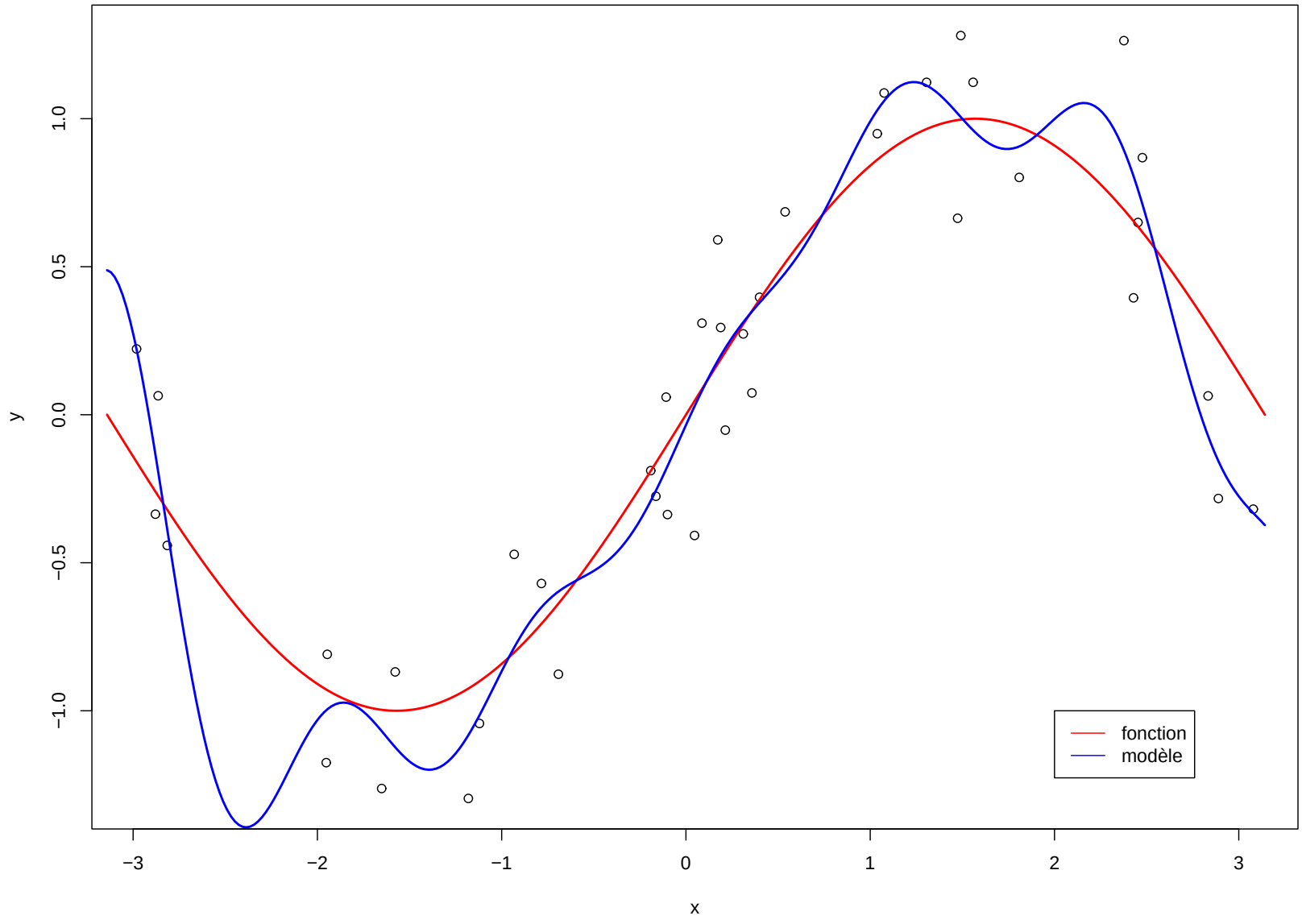
$$(ZZ^T + \nu Id)C^T = ZY^T$$

Exemple : RBF, pénalité sur les poids



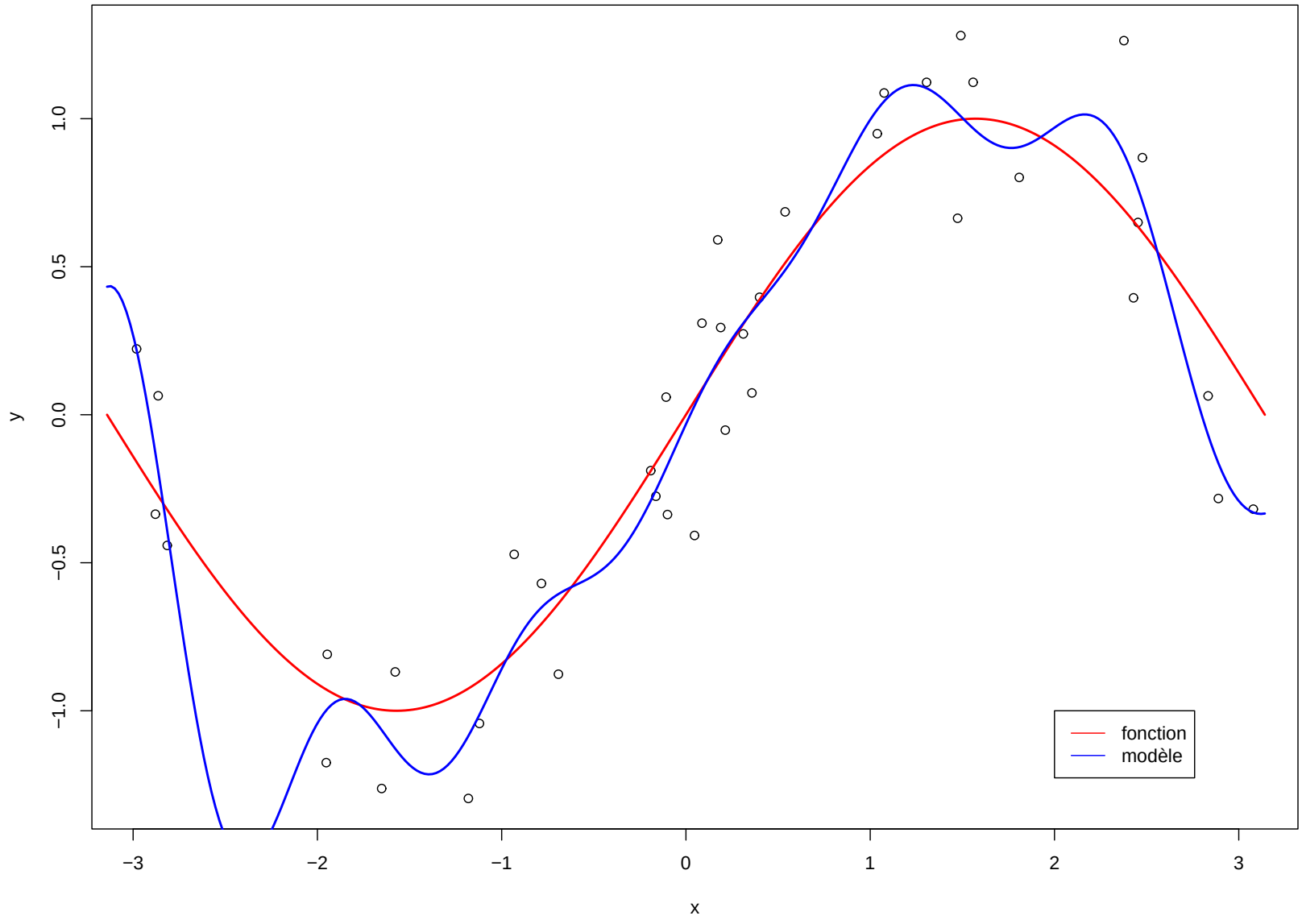
$$y(x) = \sin(x) + \epsilon(x)$$

Exemple : RBF, pénalité sur les poids



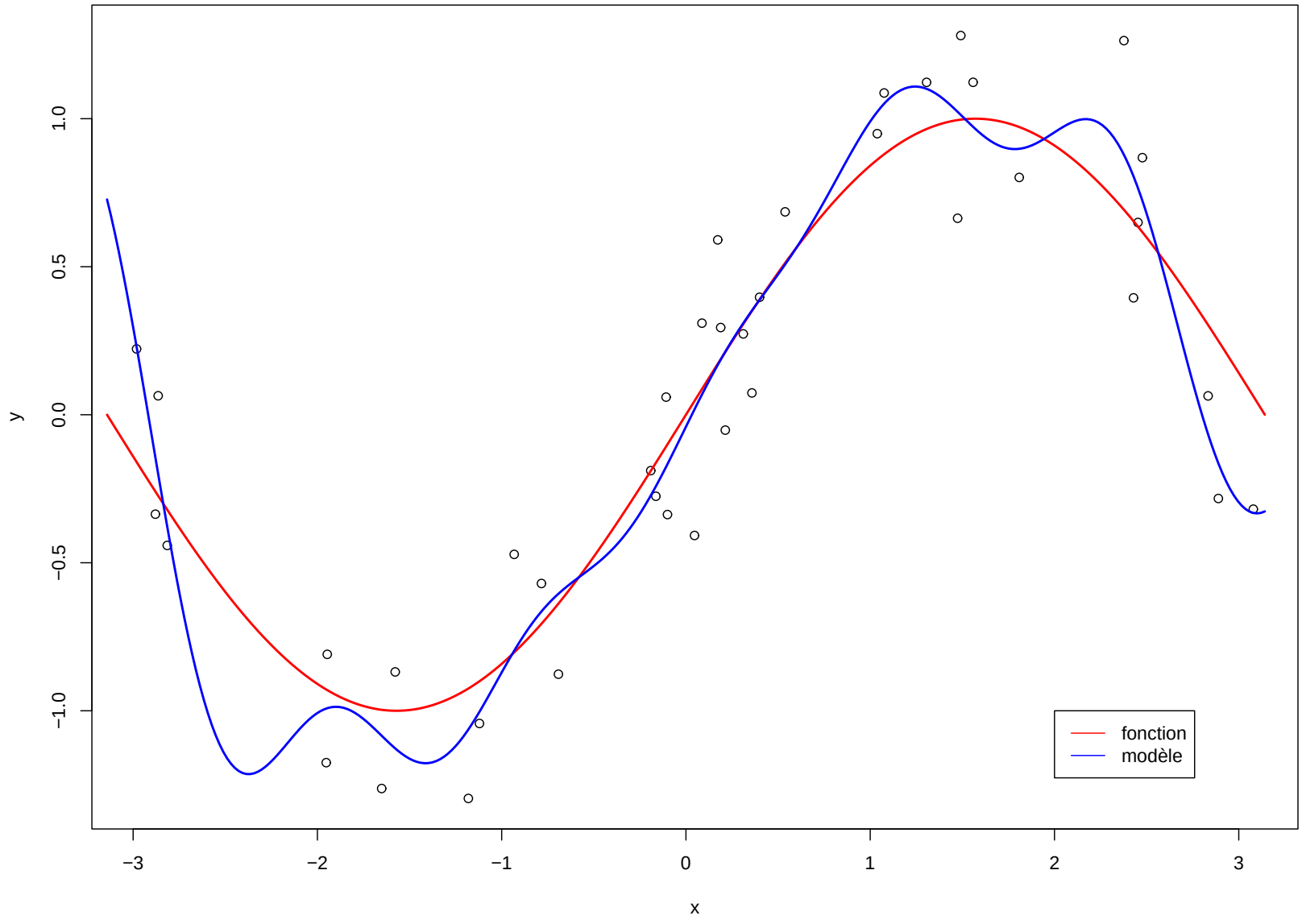
Réseau RBF à 20 neurones

Exemple : RBF, pénalité sur les poids



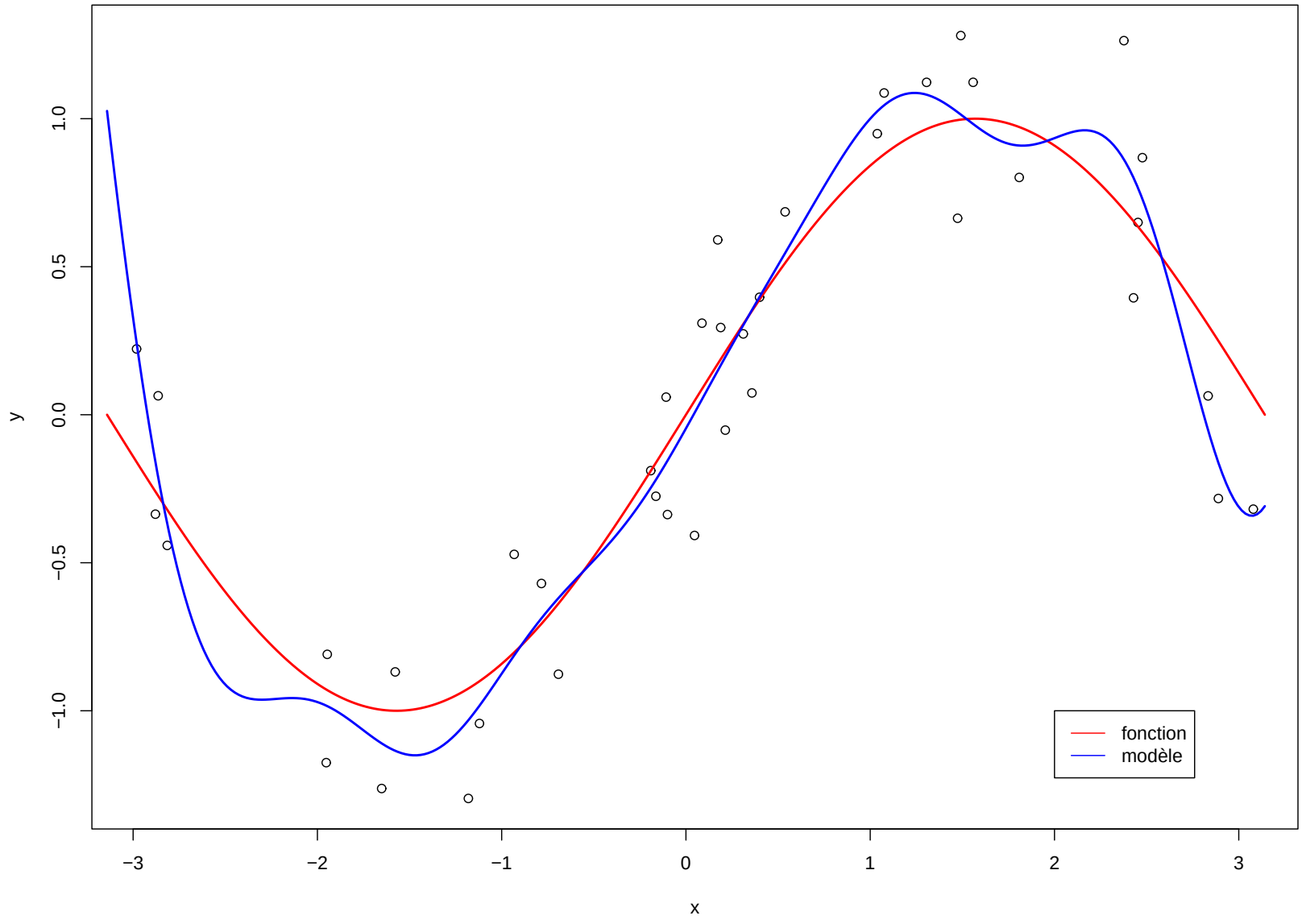
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-7}$)

Exemple : RBF, pénalité sur les poids



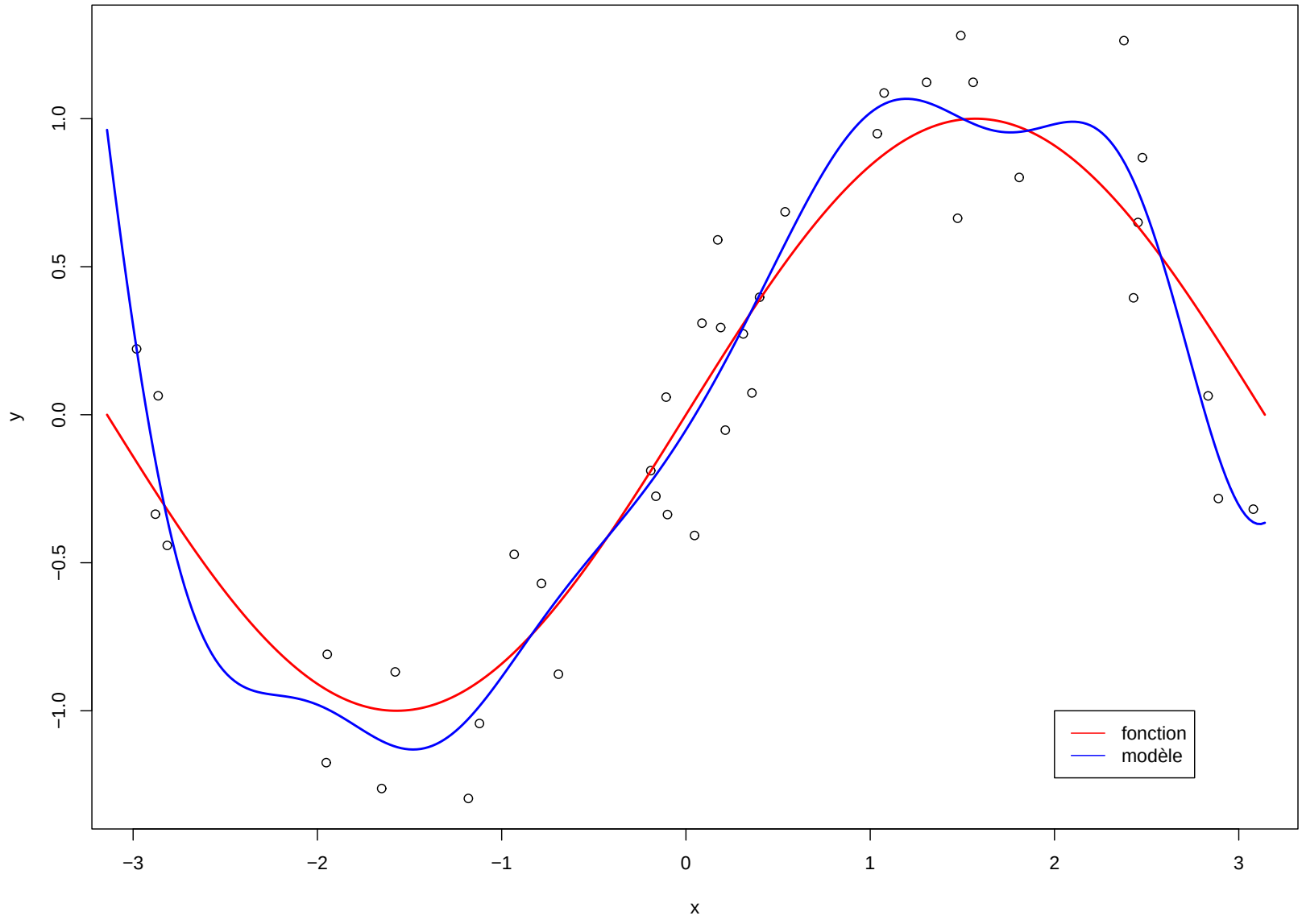
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-6}$)

Exemple : RBF, pénalité sur les poids



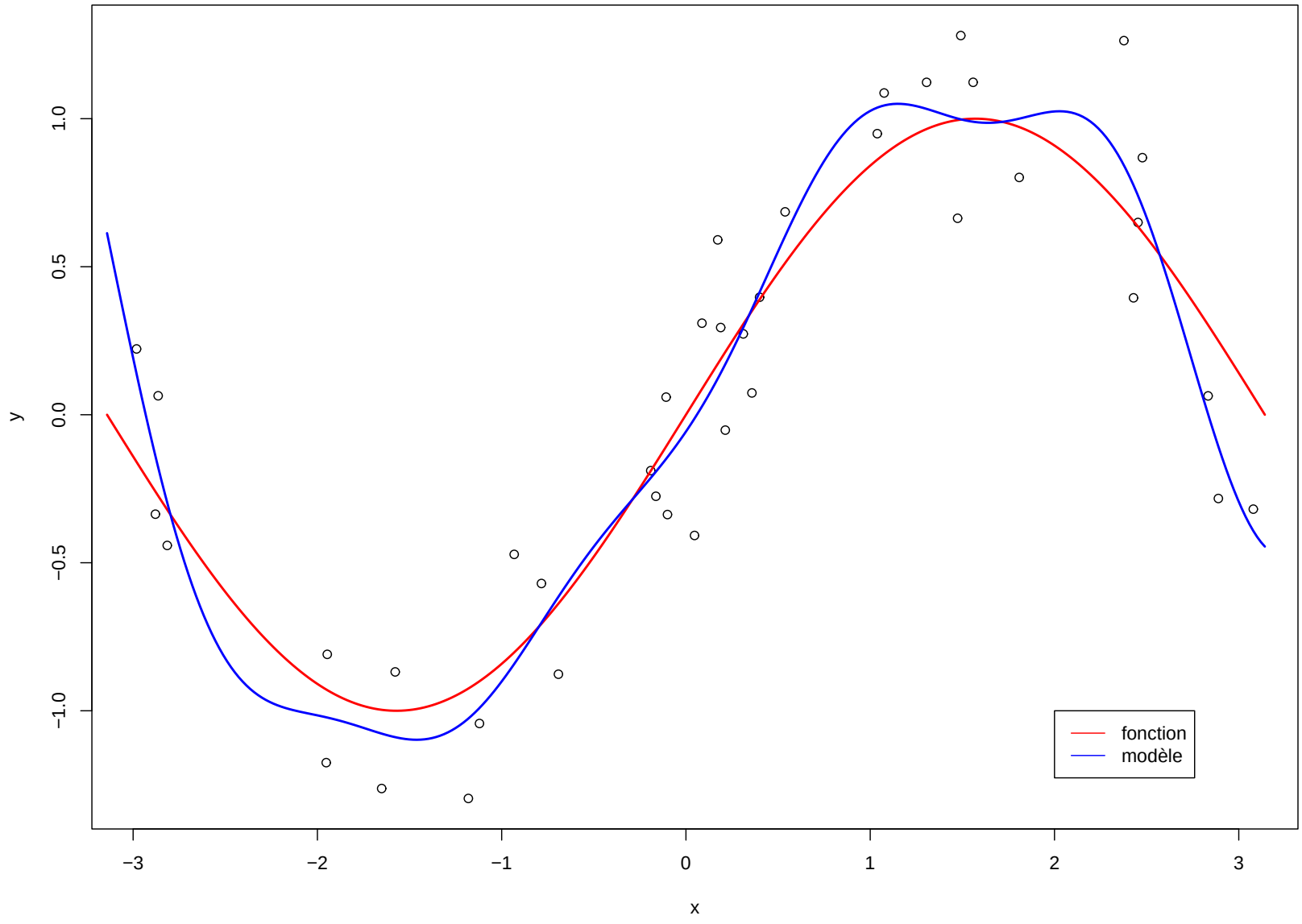
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-5}$)

Exemple : RBF, pénalité sur les poids



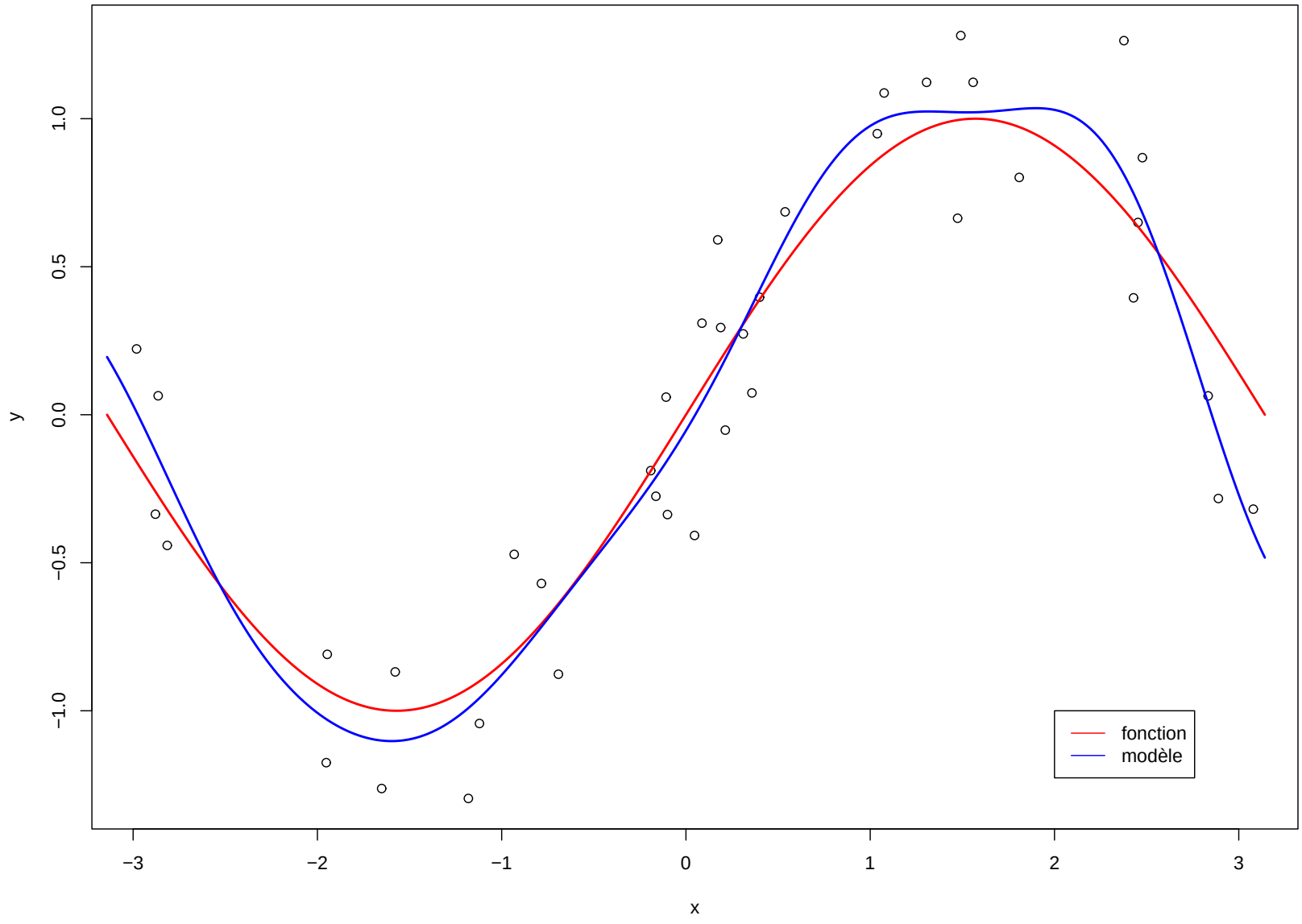
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-4}$)

Exemple : RBF, pénalité sur les poids



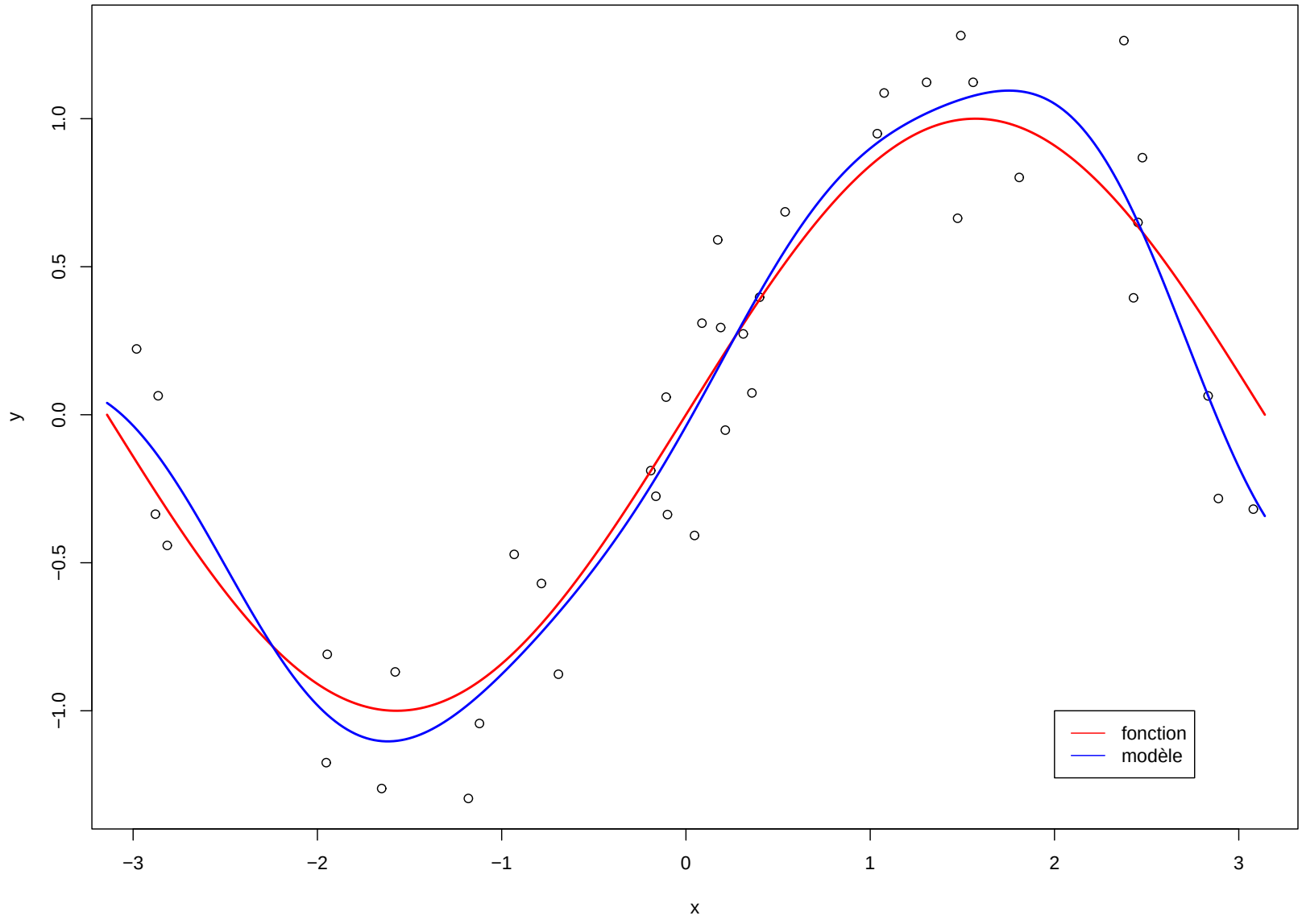
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-3}$)

Exemple : RBF, pénalité sur les poids



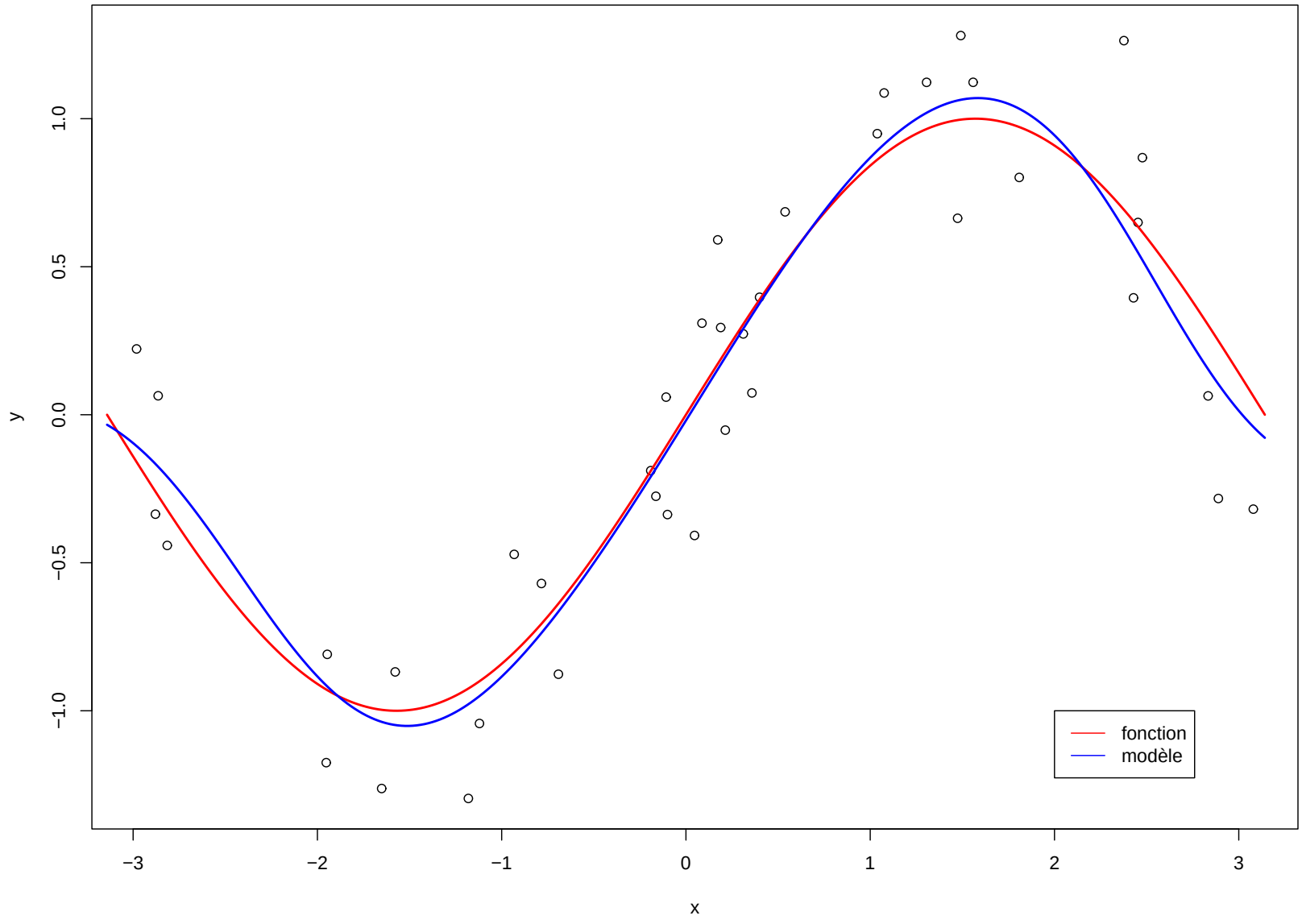
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-2}$)

Exemple : RBF, pénalité sur les poids



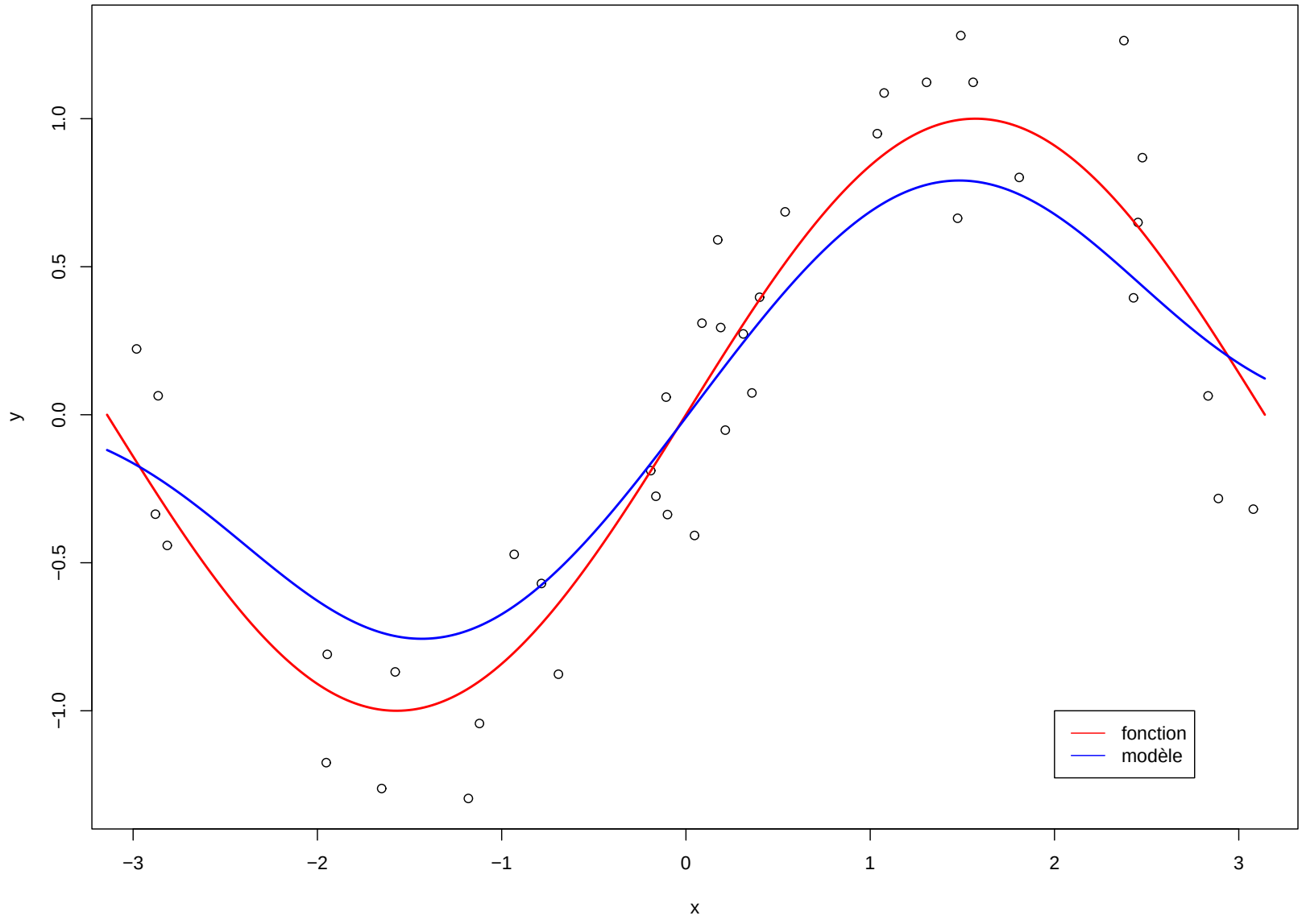
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-1}$)

Exemple : RBF, pénalité sur les poids



Réseau RBF à 20 neurones régularisé ($\nu = 1$)

Exemple : RBF, pénalité sur les poids



Réseau RBF à 20 neurones régularisé ($\nu = 10$)

Pénalité d'ordre un

On améliore la situation en s'intéressant aux dérivées. On choisit par exemple

$$\mathcal{P}(w) = \sum_{l=1}^{N} \left\| dF_x(w, x^l) \right\|^2 = \sum_{l=1}^{N} \sum_i \sum_j \left(\frac{\partial F_i}{\partial x_j}(w, x^l) \right)^2$$

Dans le cas linéaire généralisé $F = A\Phi(x) + b$, on a

$$\mathcal{P}(w) = \sum_{l=1}^{N} \left\| Ad\Phi(x^l) \right\|^2$$

Pénalité d'ordre un (2)

On obtient la nouvelle équation matricielle

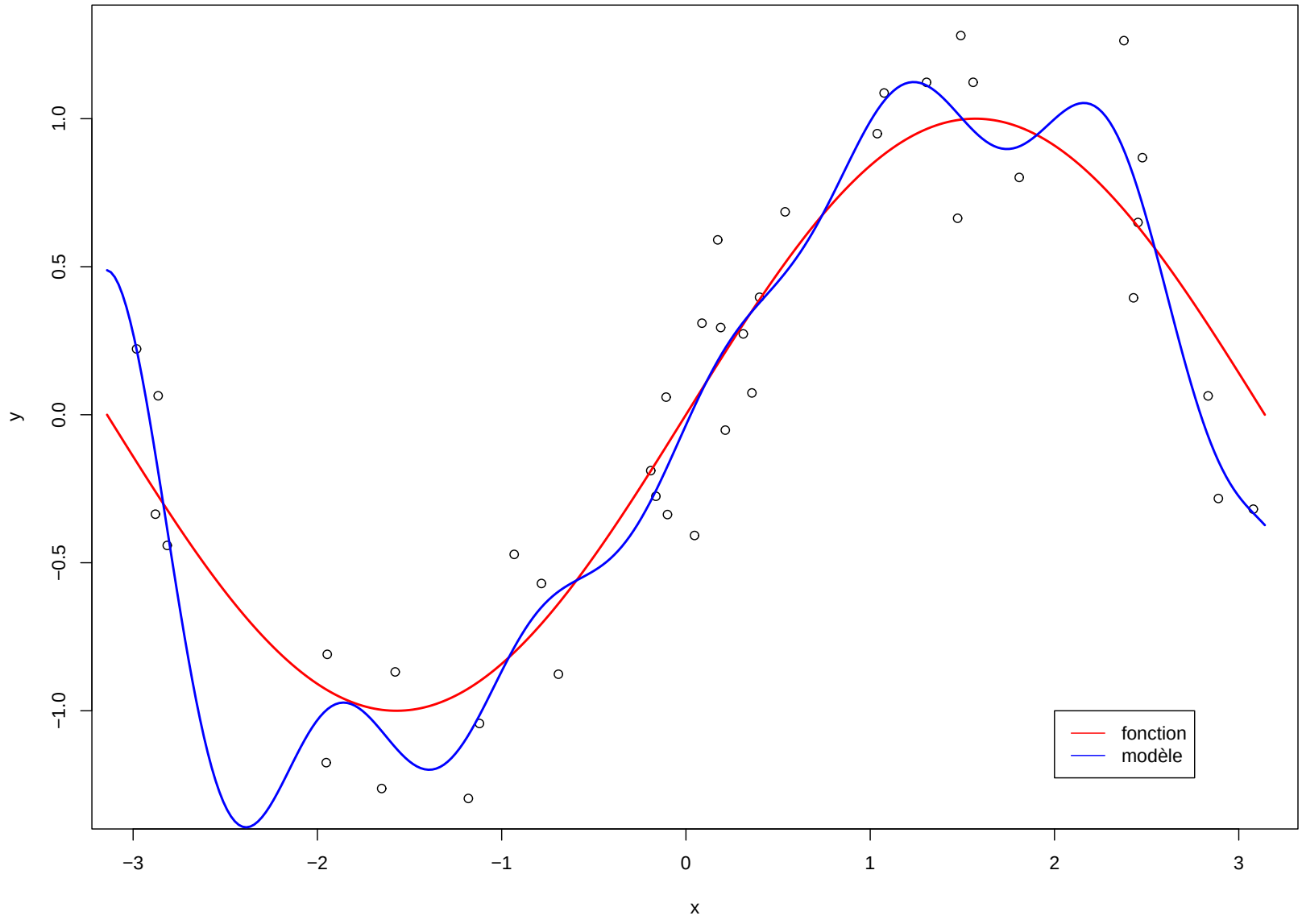
$$(ZZ^T + \nu P)C^T = ZY^T$$

où P est donnée par

$$P = \begin{pmatrix} \sum_{l=1}^{N} d\Phi(x^l) d\Phi(x^l)^T & 0 \\ 0 & 0 \end{pmatrix}$$

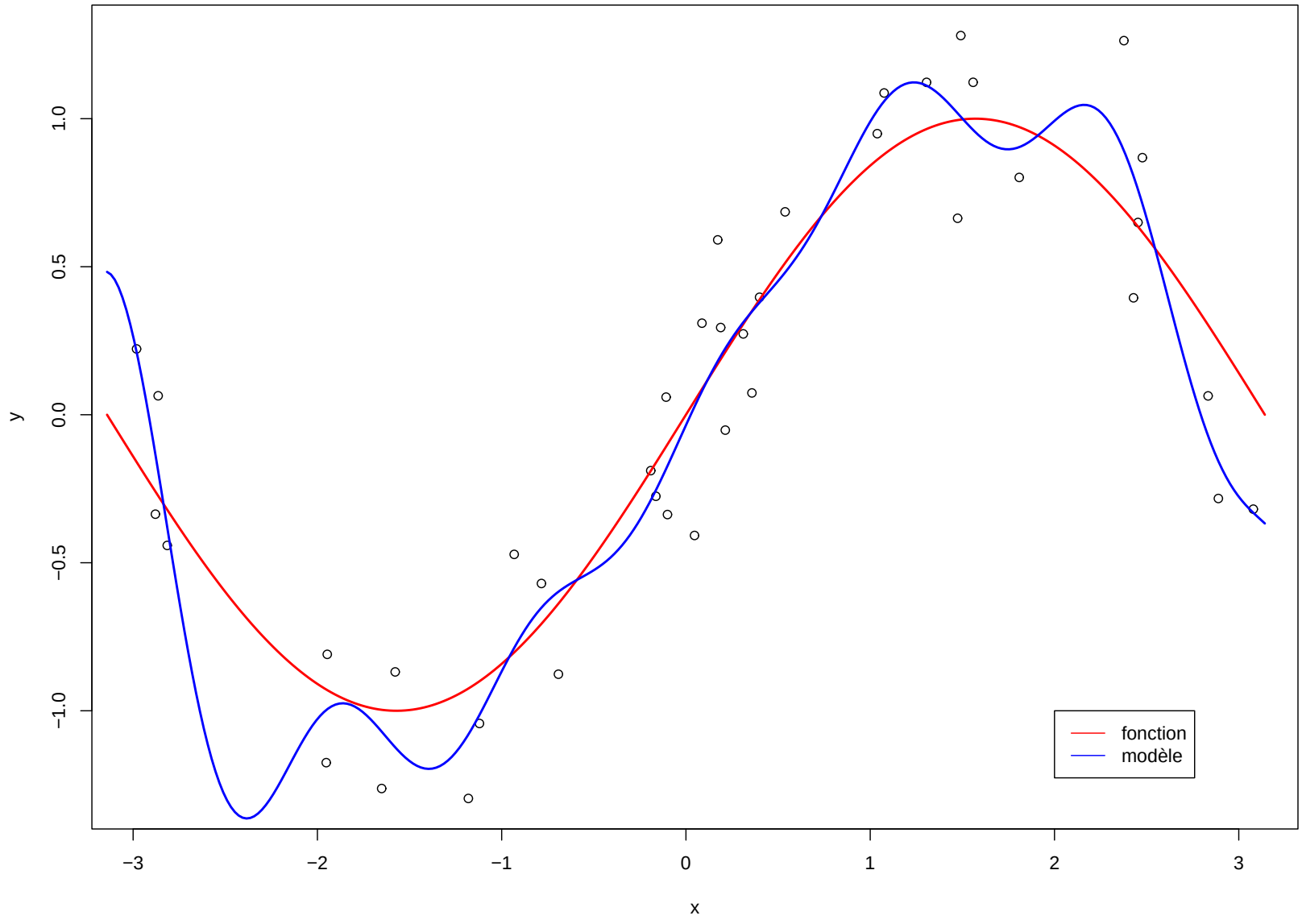
Les termes nuls viennent du fait que la pénalité ne tient pas compte des seuils.

Exemple : RBF, pénalité sur la dérivée première



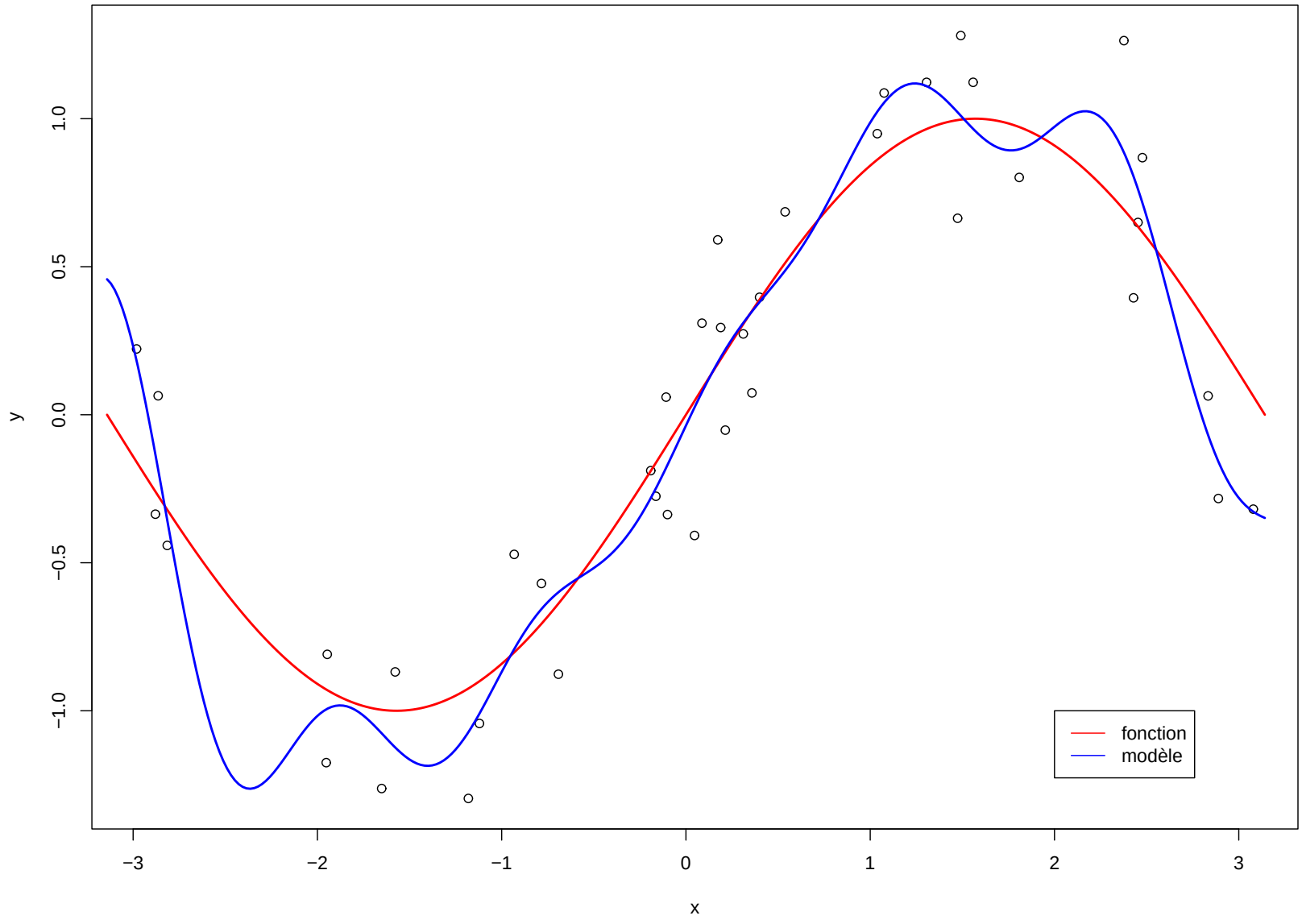
Réseau RBF à 20 neurones

Exemple : RBF, pénalité sur la dérivée première



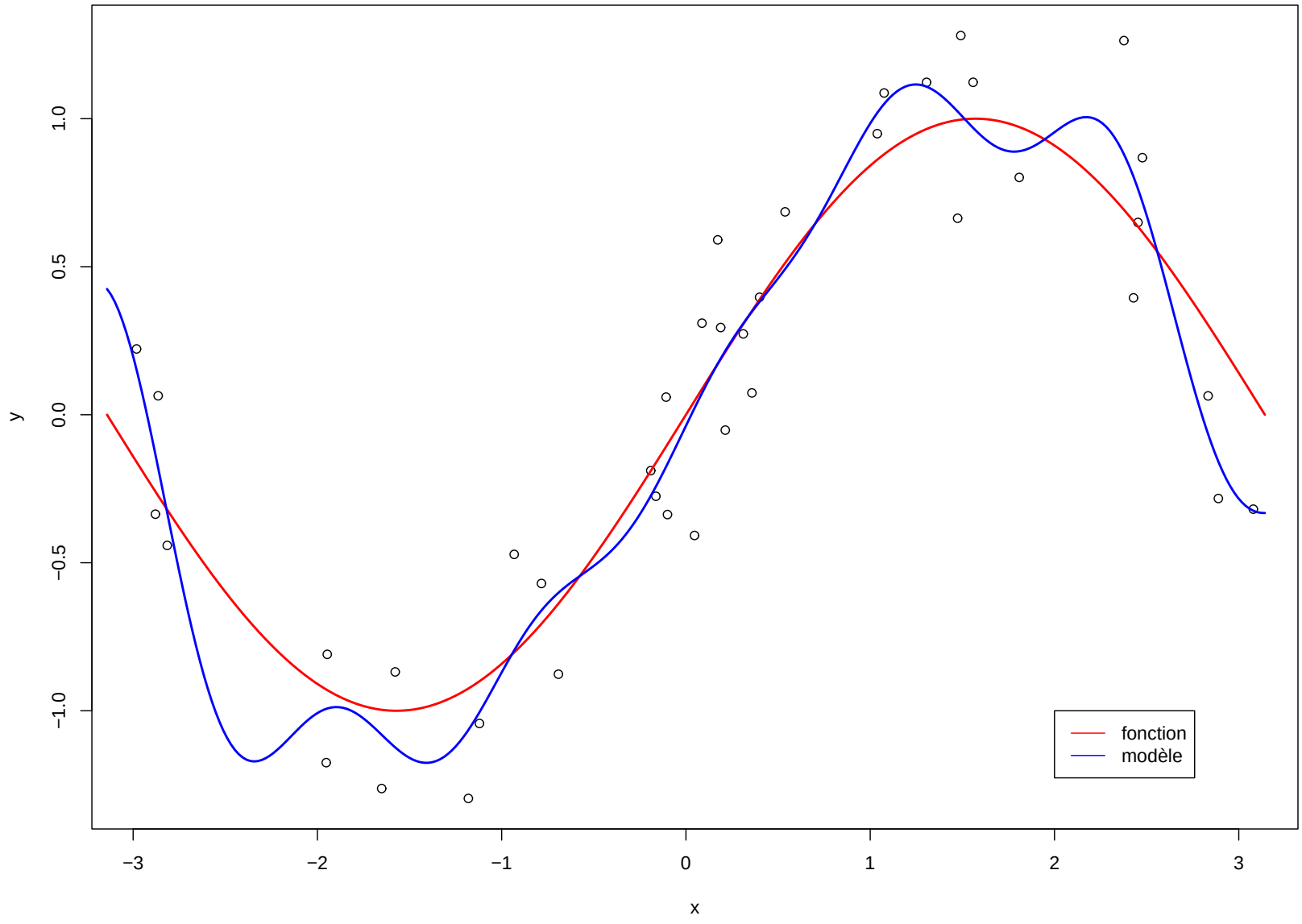
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-4}$)

Exemple : RBF, pénalité sur la dérivée première



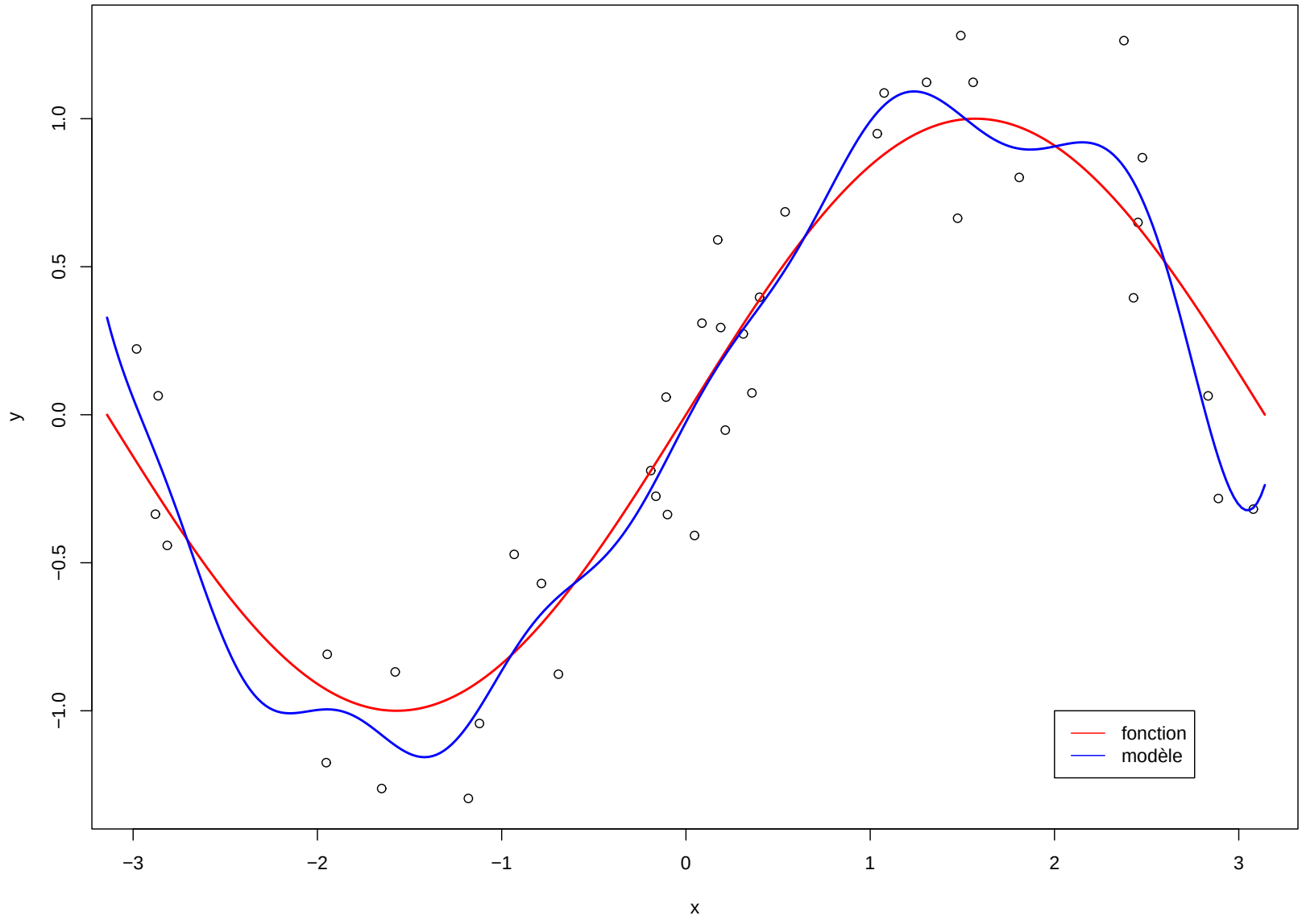
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-4}$)

Exemple : RBF, pénalité sur la dérivée première



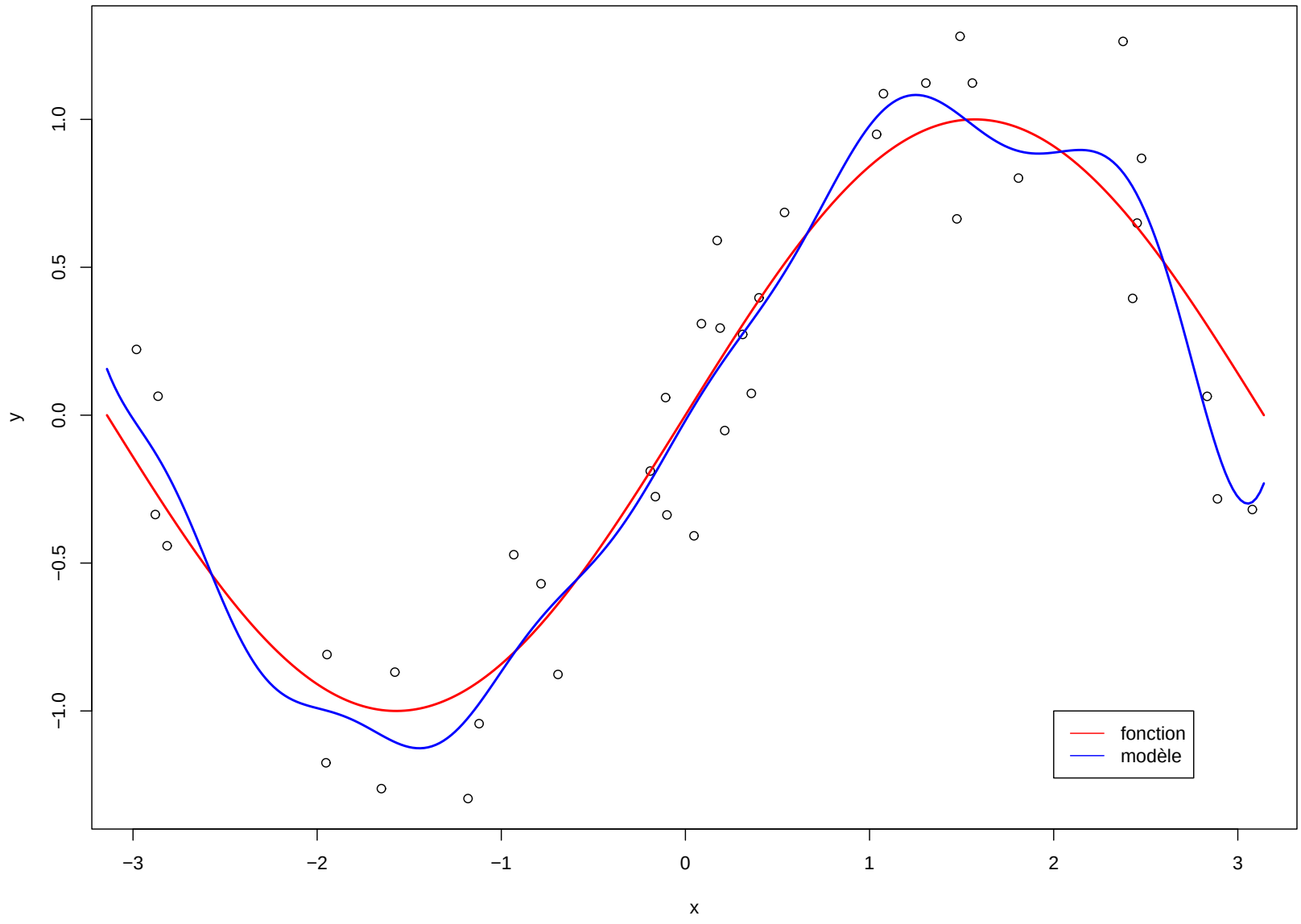
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-3}$)

Exemple : RBF, pénalité sur la dérivée première



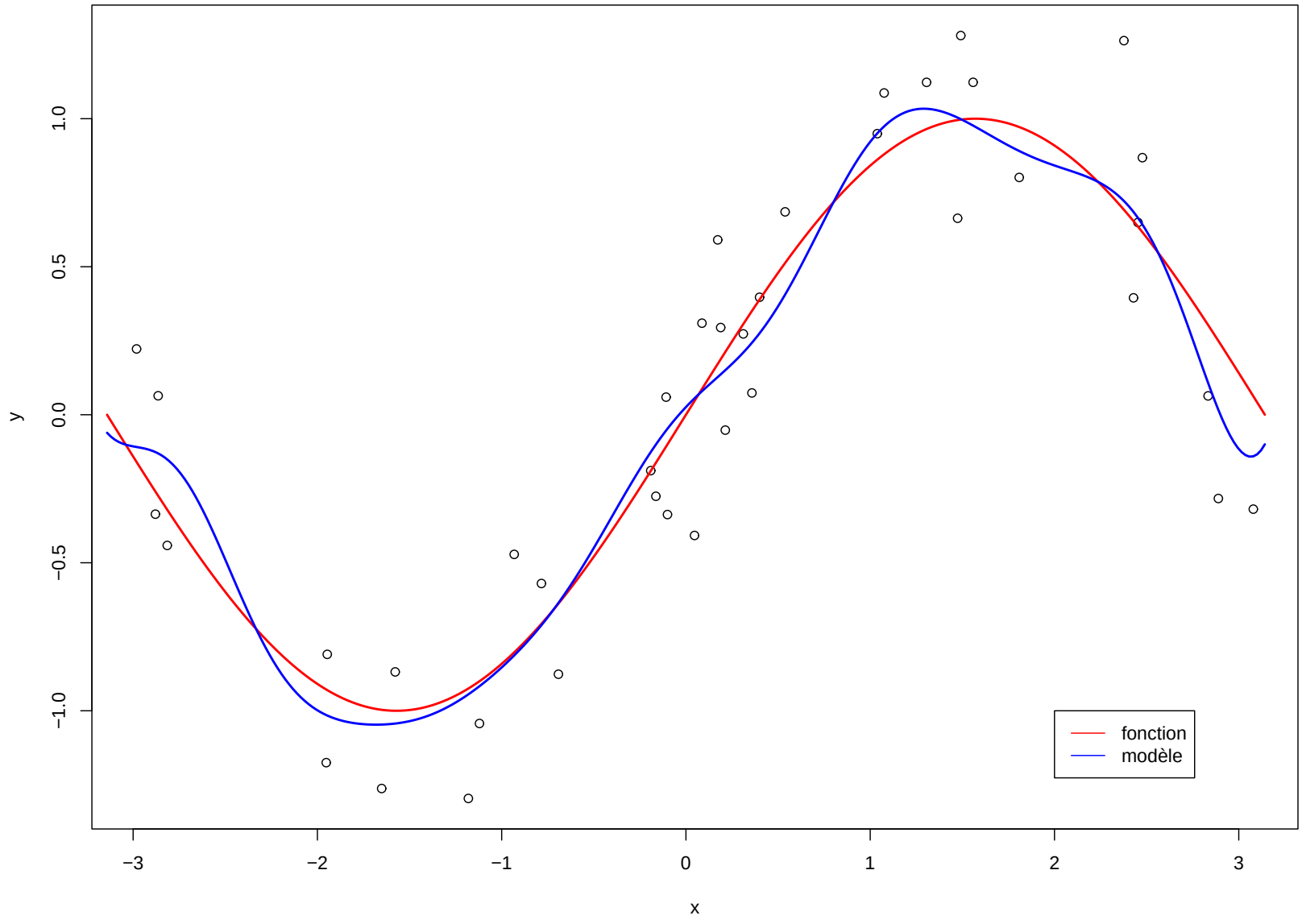
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-3}$)

Exemple : RBF, pénalité sur la dérivée première



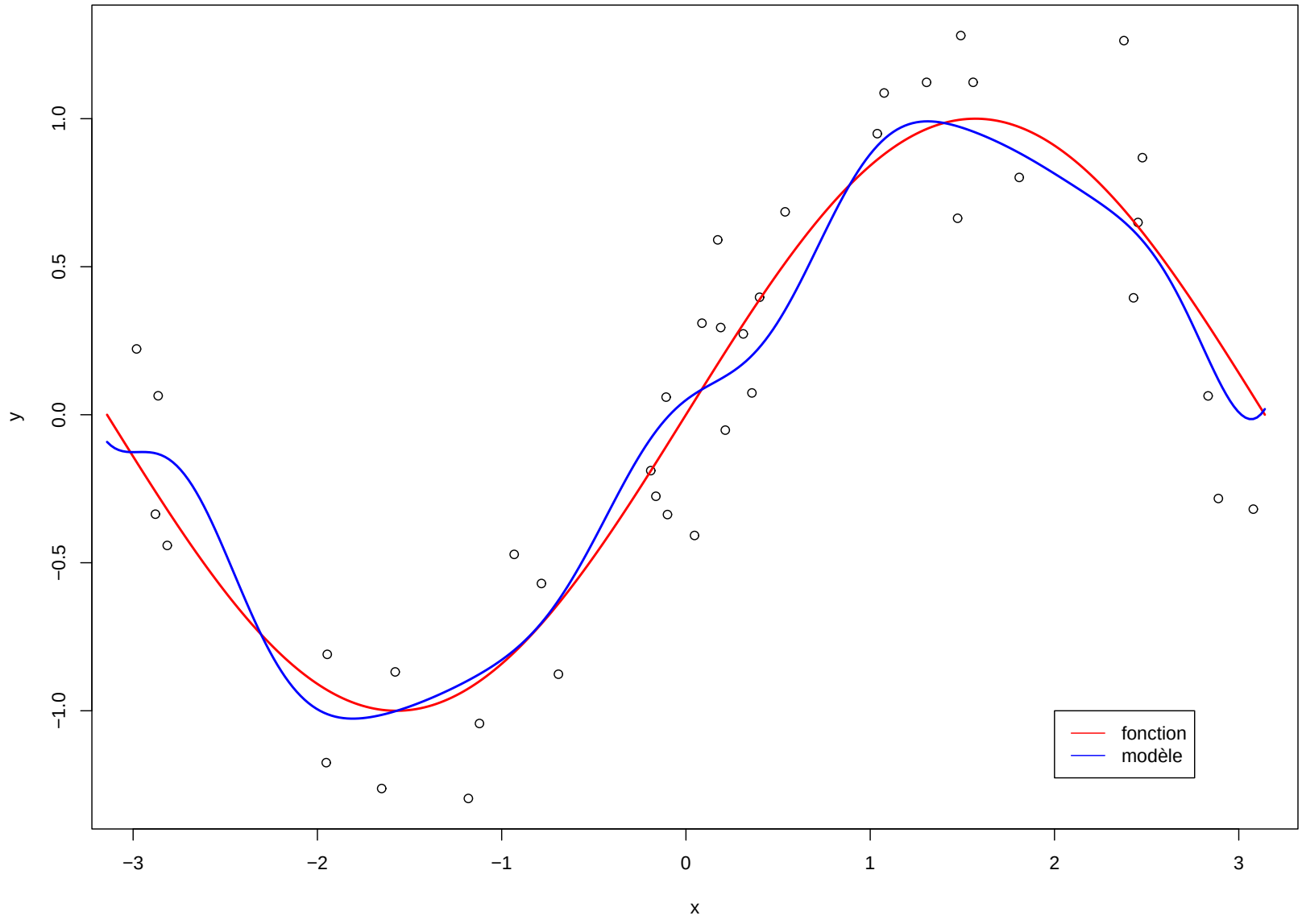
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-2}$)

Exemple : RBF, pénalité sur la dérivée première



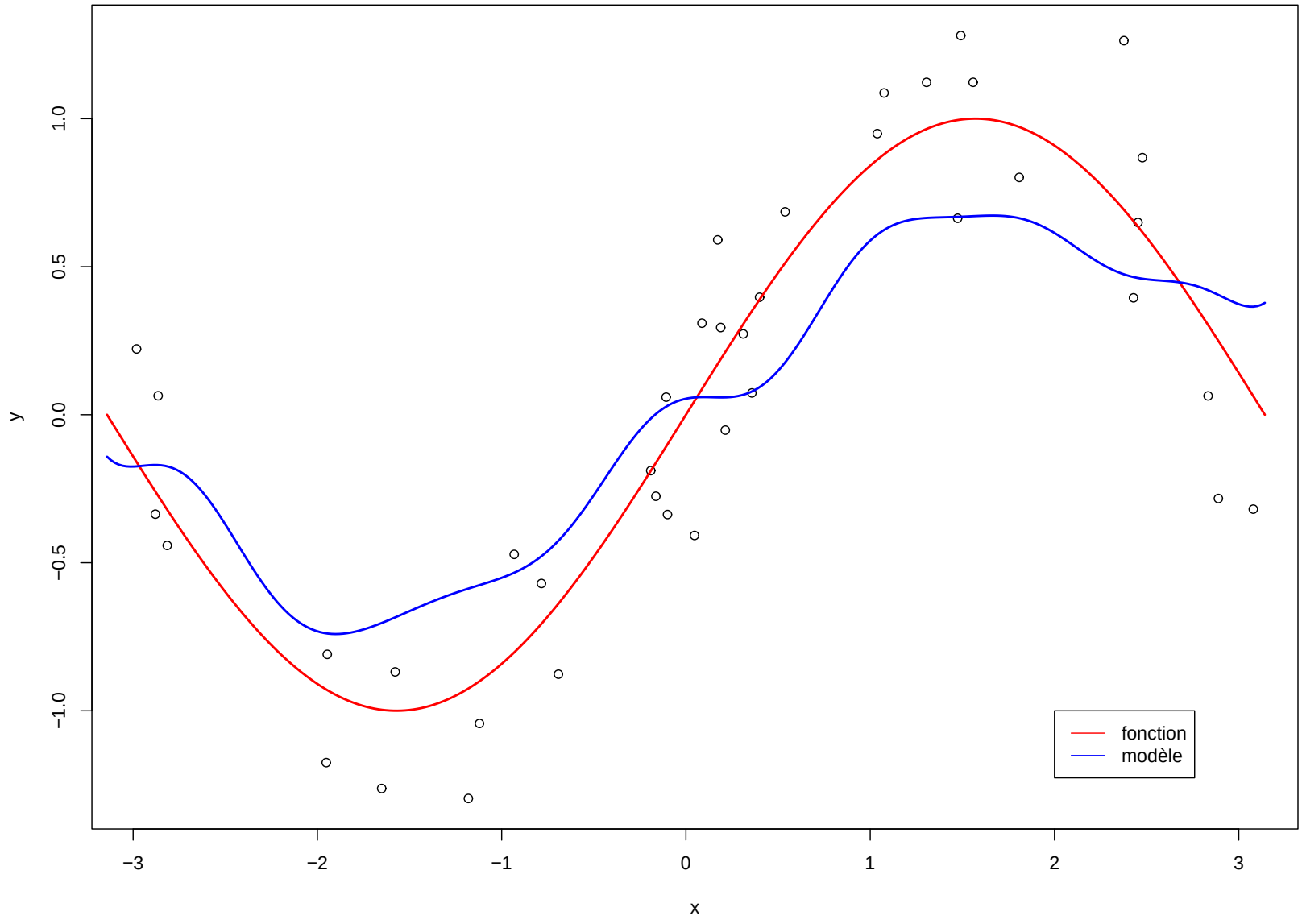
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-2}$)

Exemple : RBF, pénalité sur la dérivée première



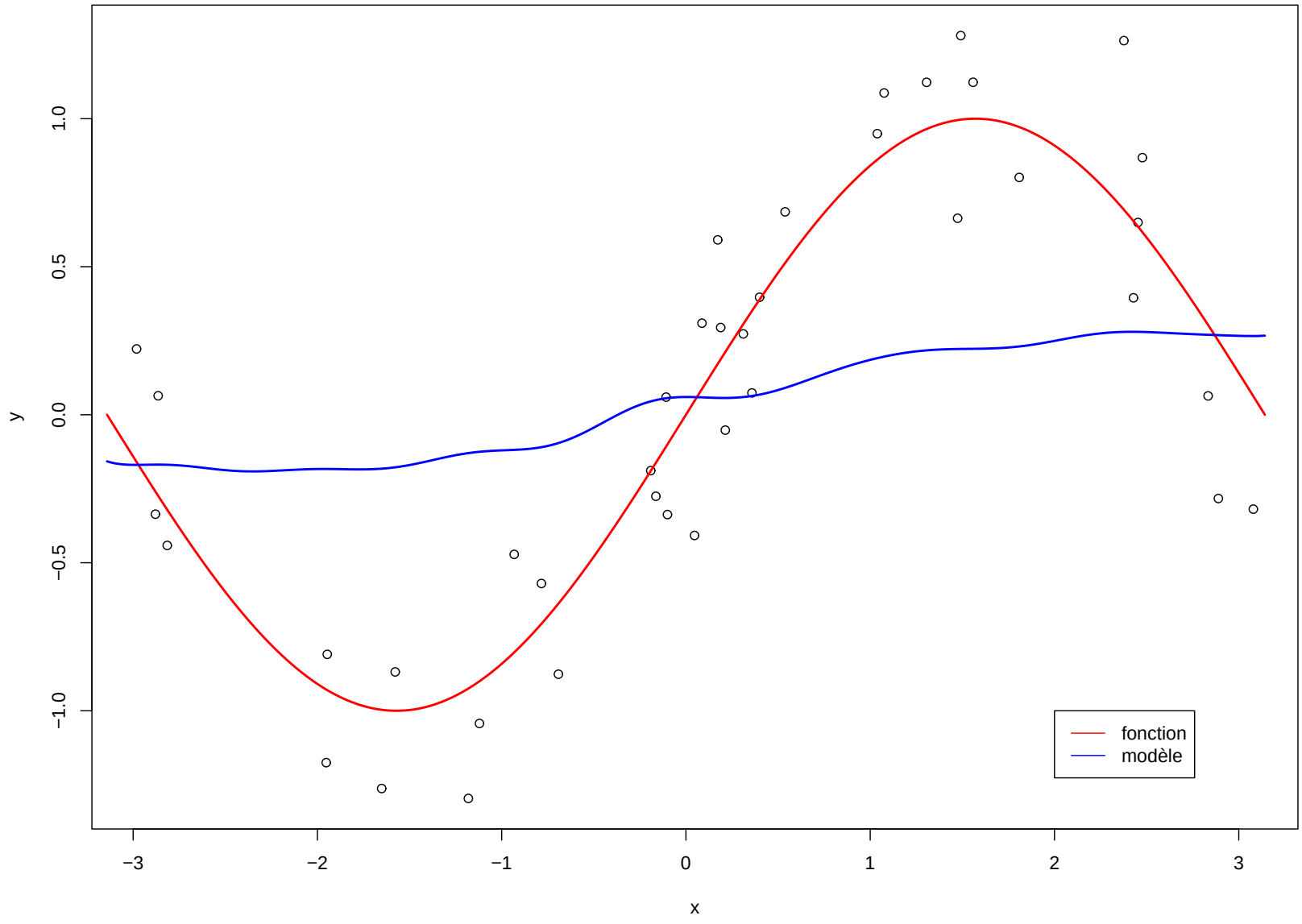
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-1}$)

Exemple : RBF, pénalité sur la dérivée première



Réseau RBF à 20 neurones régularisé ($\nu = 1$)

Exemple : RBF, pénalité sur la dérivée première



Réseau RBF à 20 neurones régularisé ($\nu = 10$)

Pénalité d'ordre deux

On peut aller plus loin en utilisant les dérivées secondes :

$$P(w) = \sum_{l=1}^{N} \left\| H_x(F)(w, x^l) \right\|^2$$

où H_x désigne la matrice hessienne de F. On se contente souvent de la diagonale de la hessienne, soit

$$P(w) = \sum_{l=1}^{N} \sum_{i} \sum_{j} \left(\frac{\partial^2 F_i}{\partial x_j^2}(w, x^l) \right)^2$$

Pénalité d'ordre deux (2)

Comme pour la pénalité d'ordre un, on montre, comme dans le cas linéaire généralisé, que la nouvelle équation matricielle s'écrit :

$$(ZZ^T + \nu M)C^T = ZY^T$$

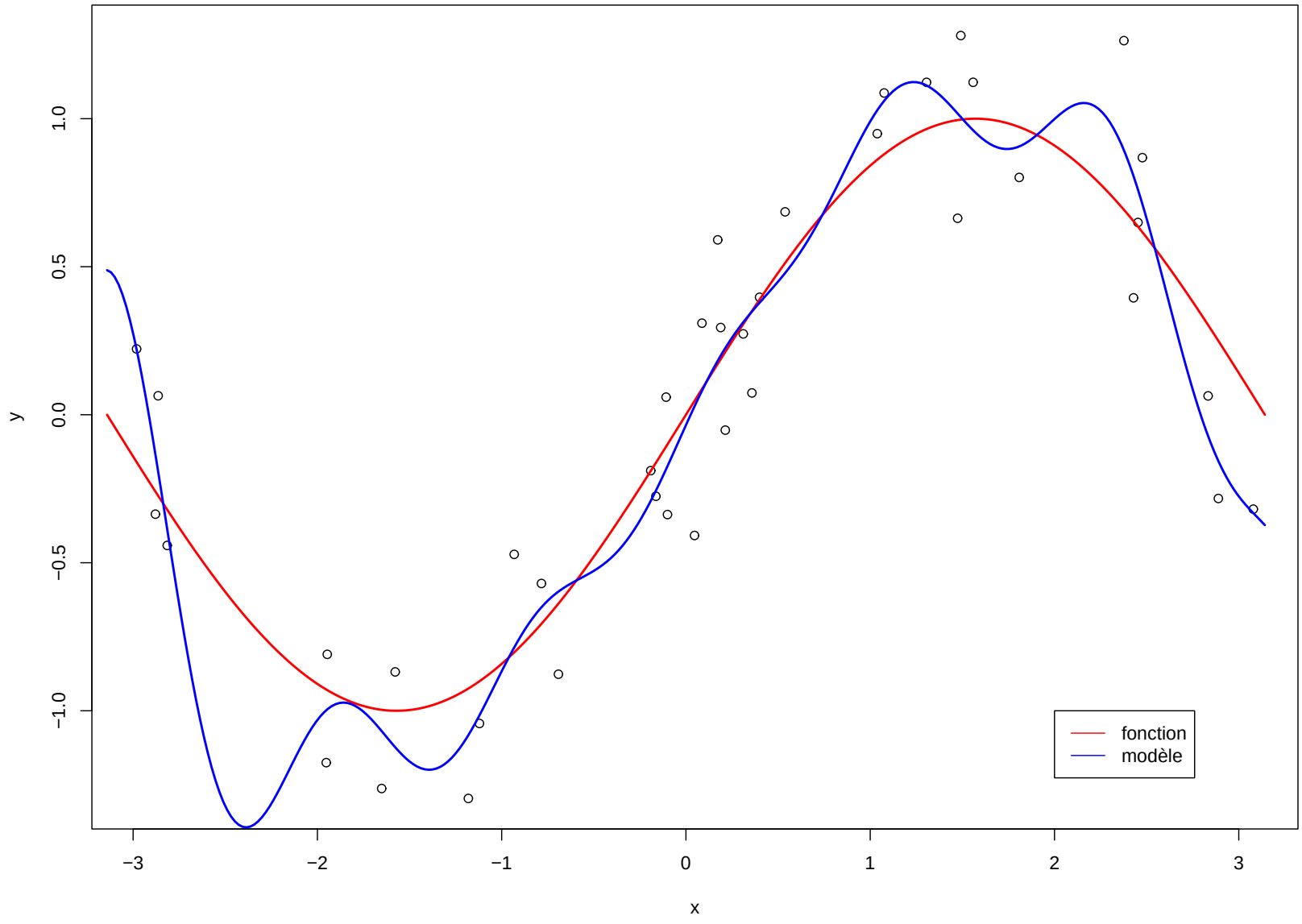
où M est donnée par

$$M = \begin{pmatrix} M' & 0 \\ 0 & 0 \end{pmatrix}$$

avec

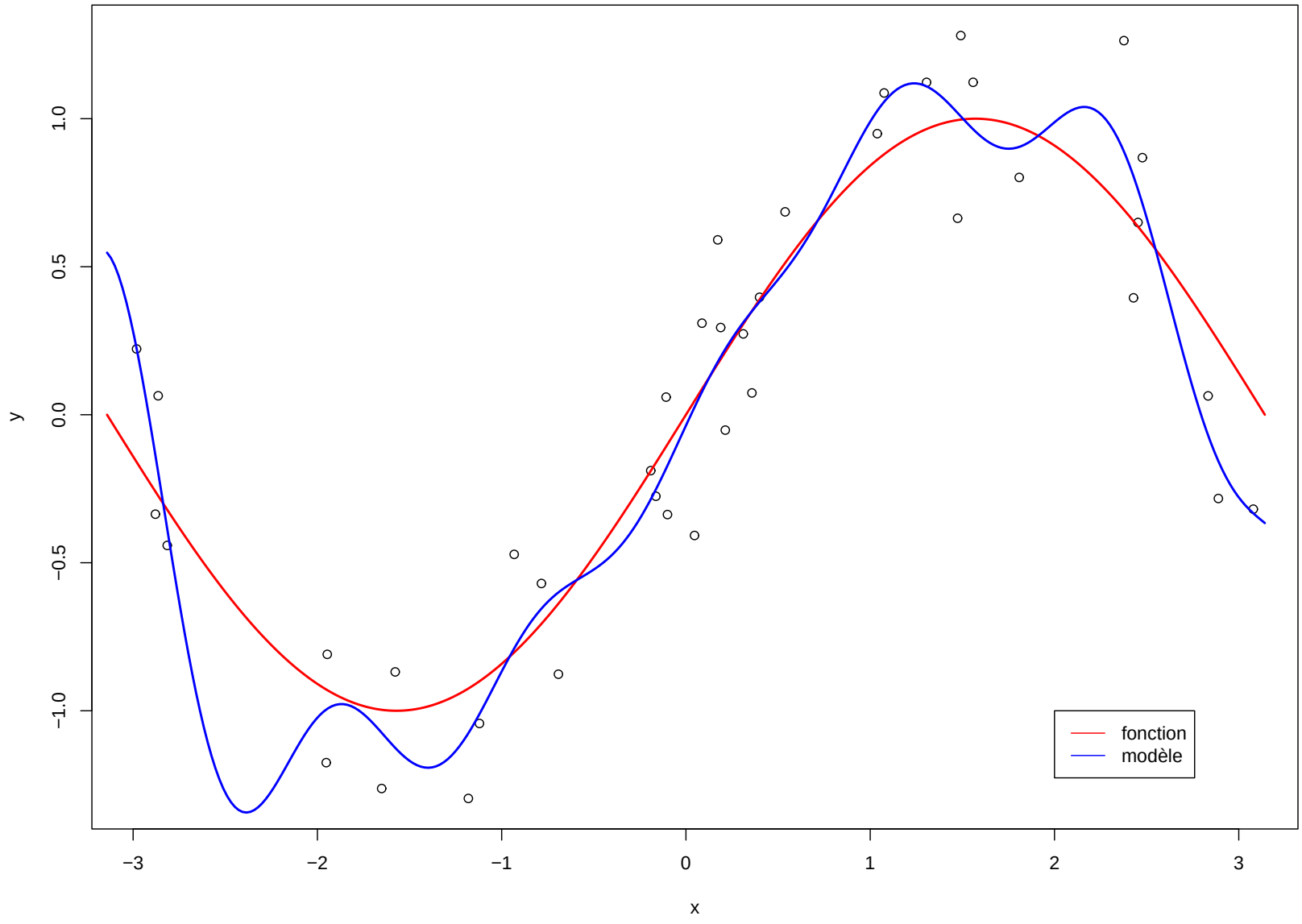
$$M'_{uv} = \sum_{l=1}^{N} \sum_{i} \sum_{j} \frac{\partial^2 \Phi_u}{\partial x_i \partial x_j}(x^l) \frac{\partial^2 \Phi_v}{\partial x_i \partial x_j}(x^l)$$

Exemple : RBF, pénalité sur la dérivée seconde



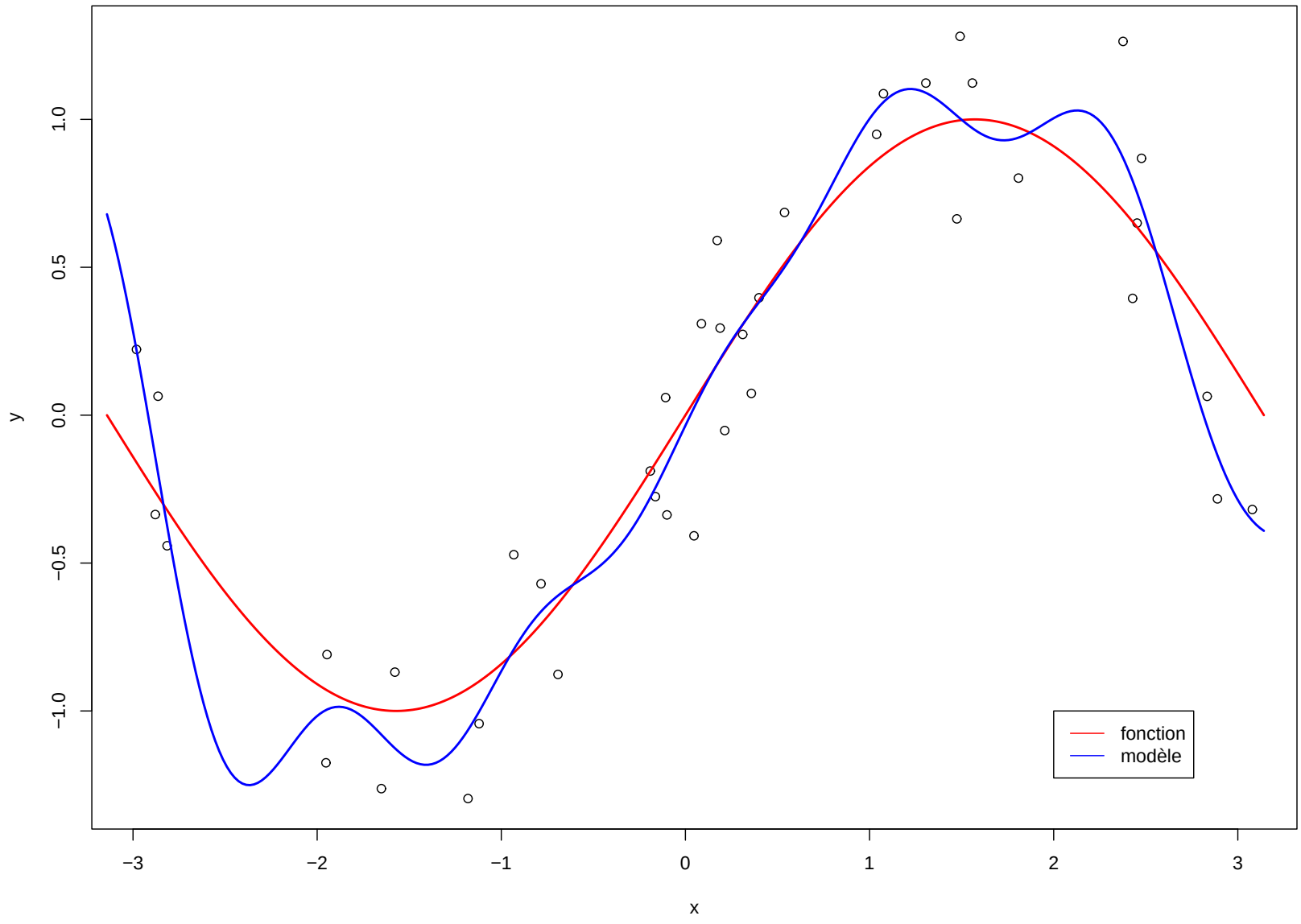
Réseau RBF à 20 neurones

Exemple : RBF, pénalité sur la dérivée seconde



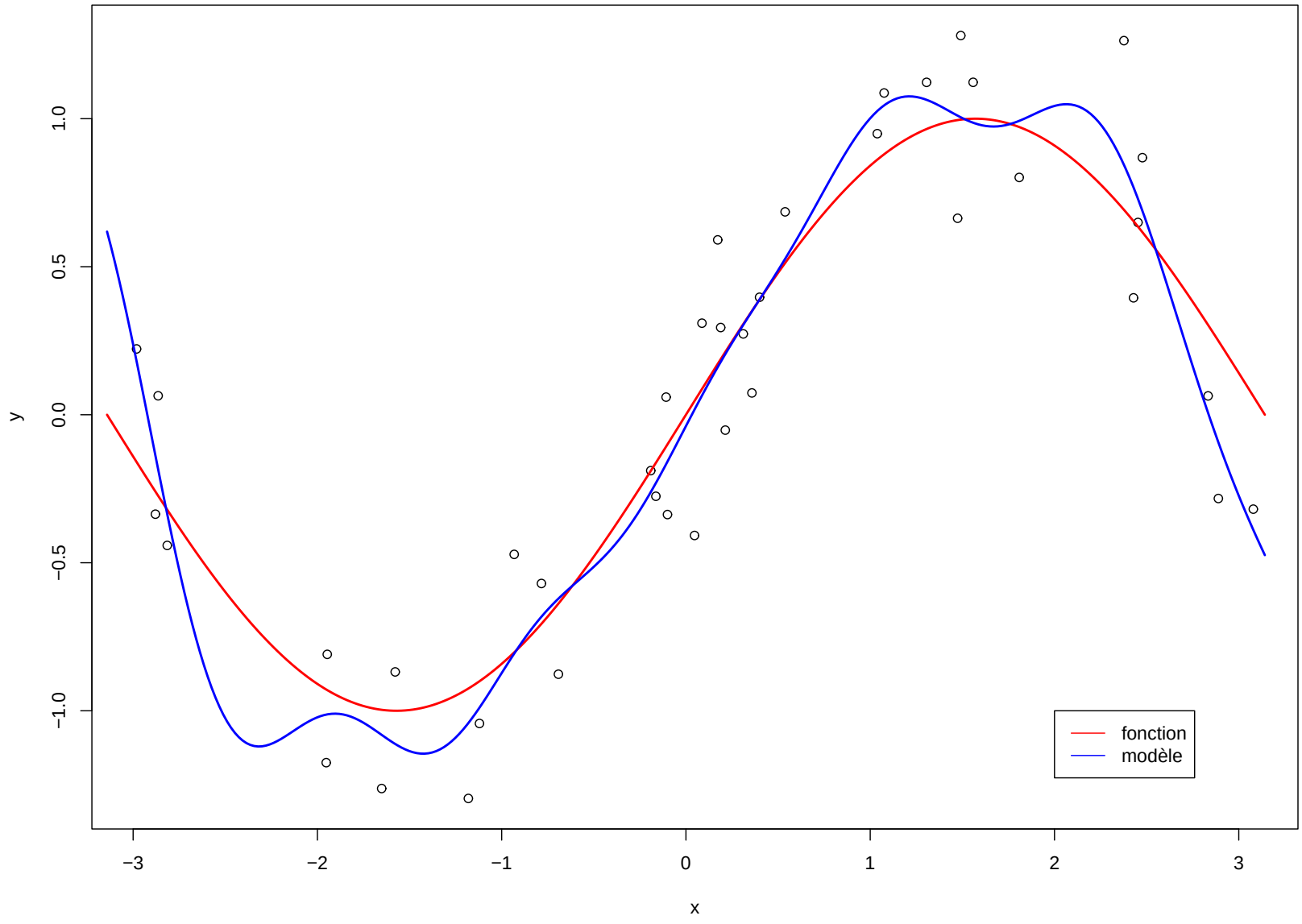
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-5}$)

Exemple : RBF, pénalité sur la dérivée seconde



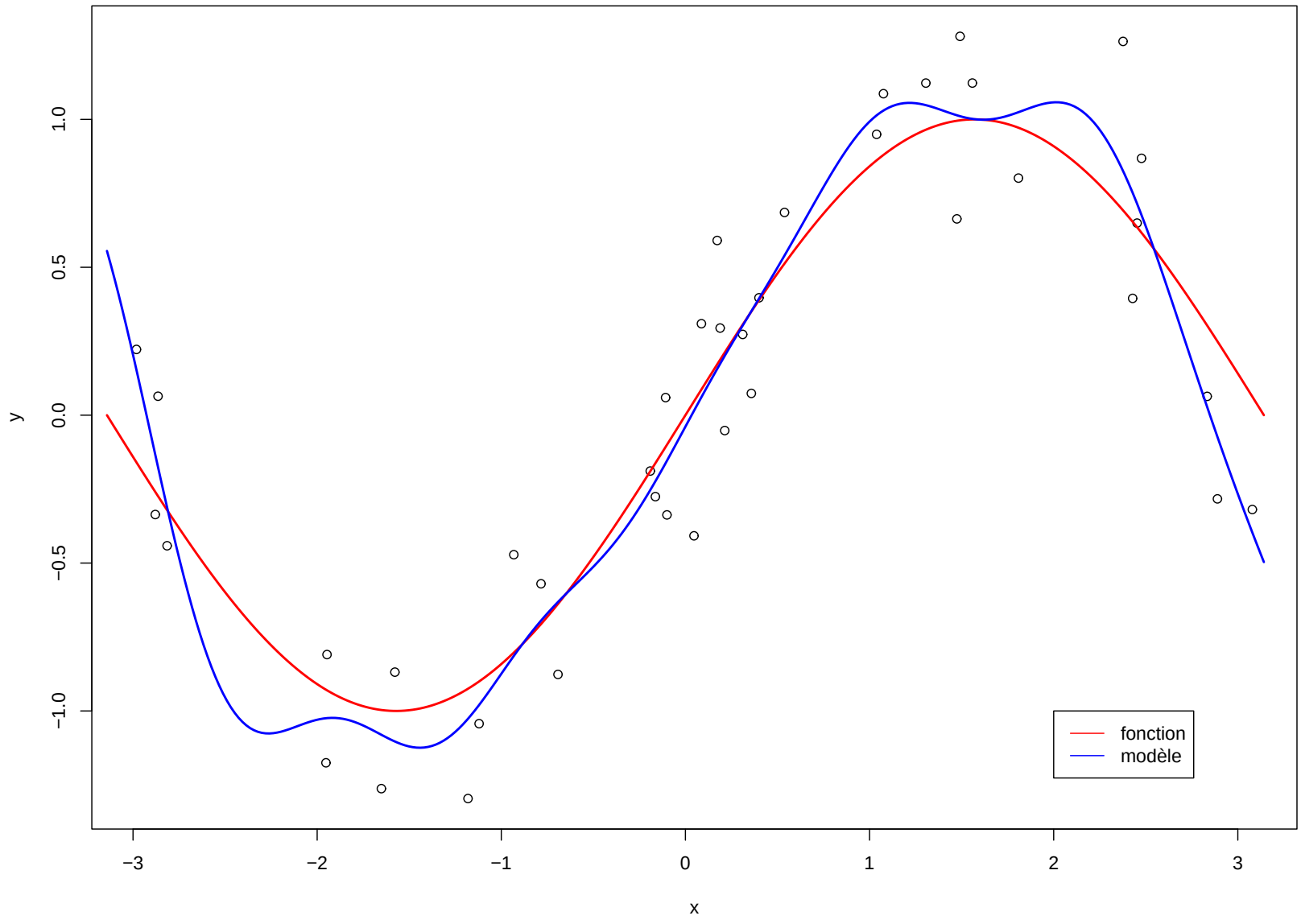
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-4}$)

Exemple : RBF, pénalité sur la dérivée seconde



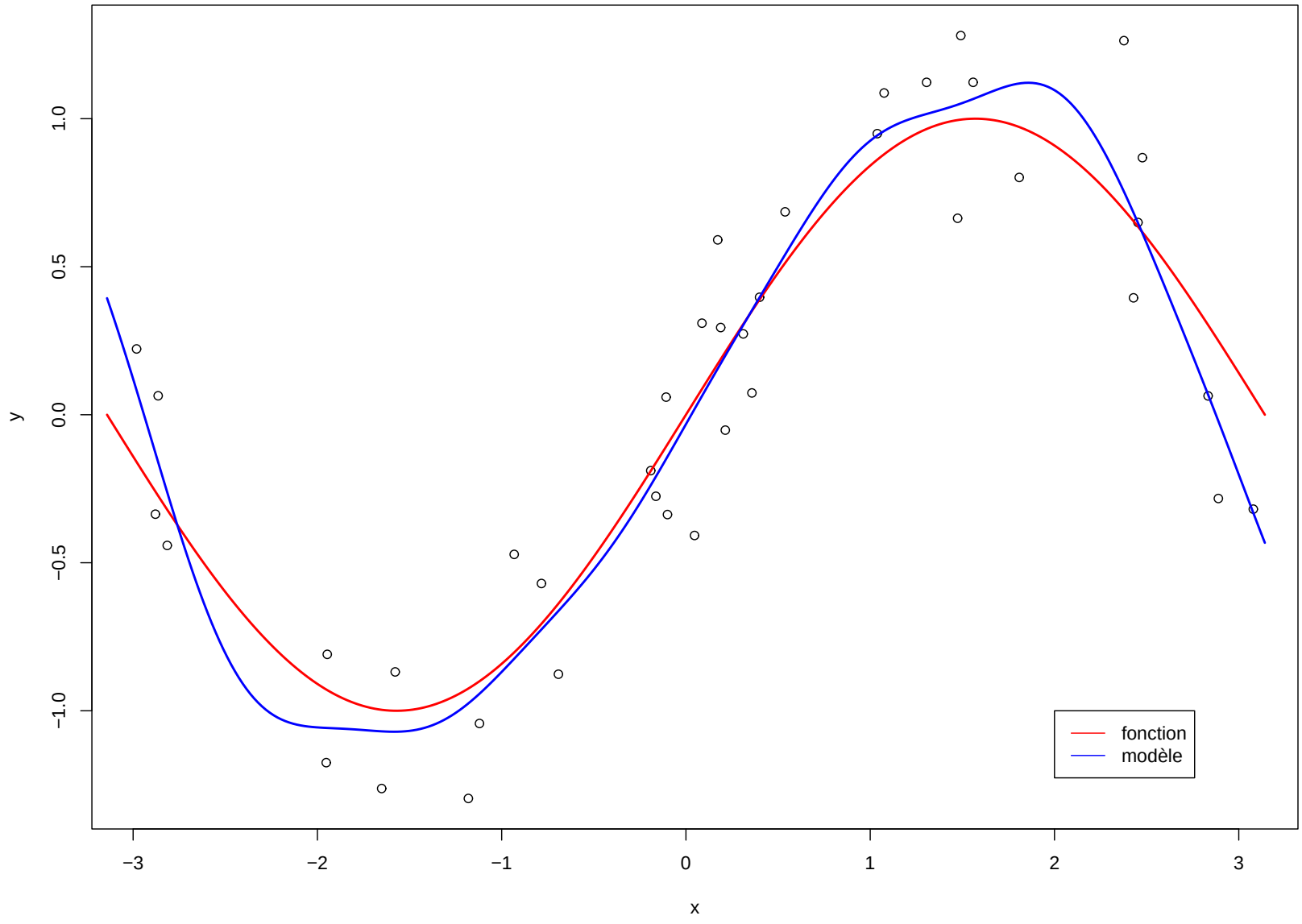
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-4}$)

Exemple : RBF, pénalité sur la dérivée seconde



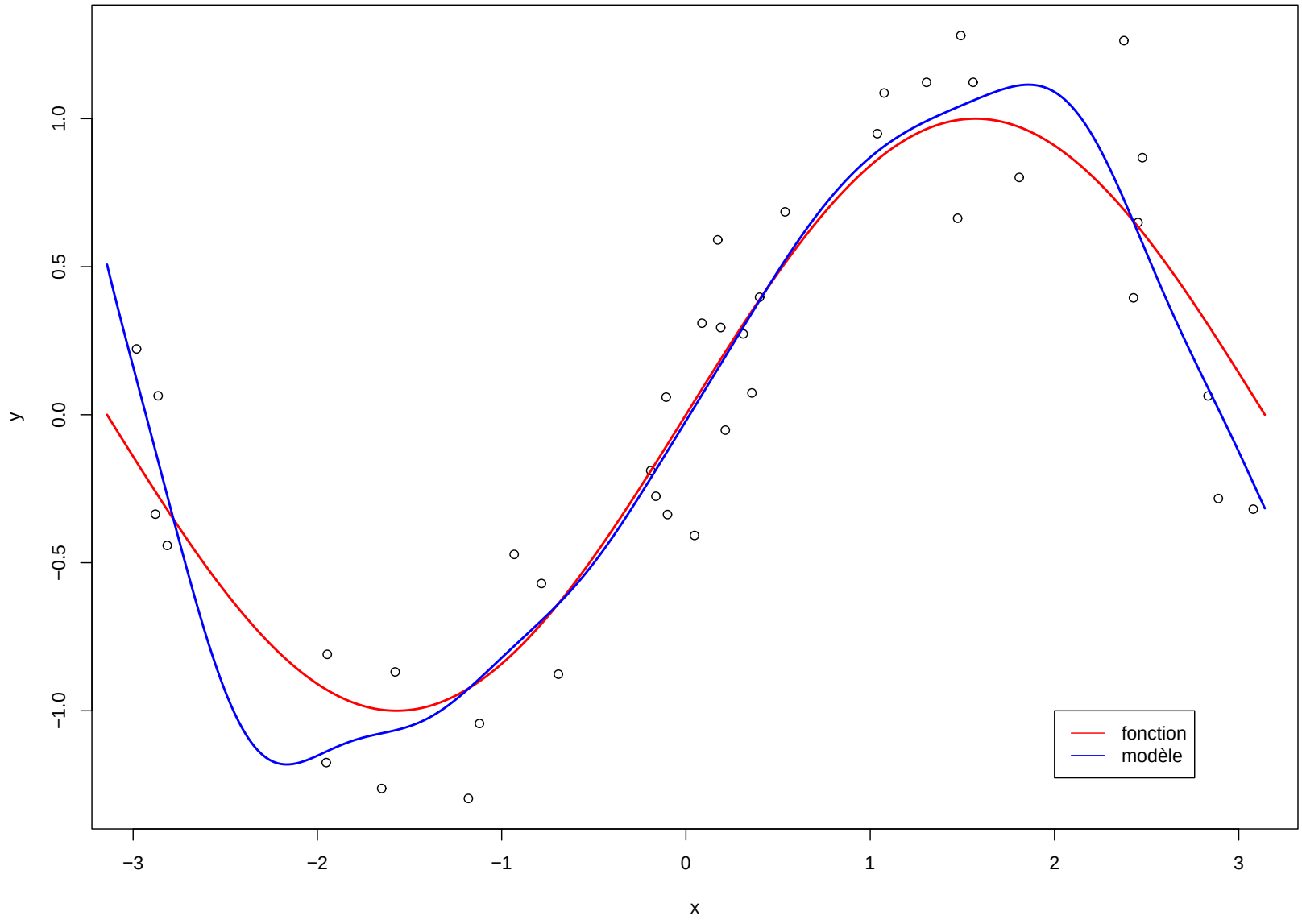
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-3}$)

Exemple : RBF, pénalité sur la dérivée seconde



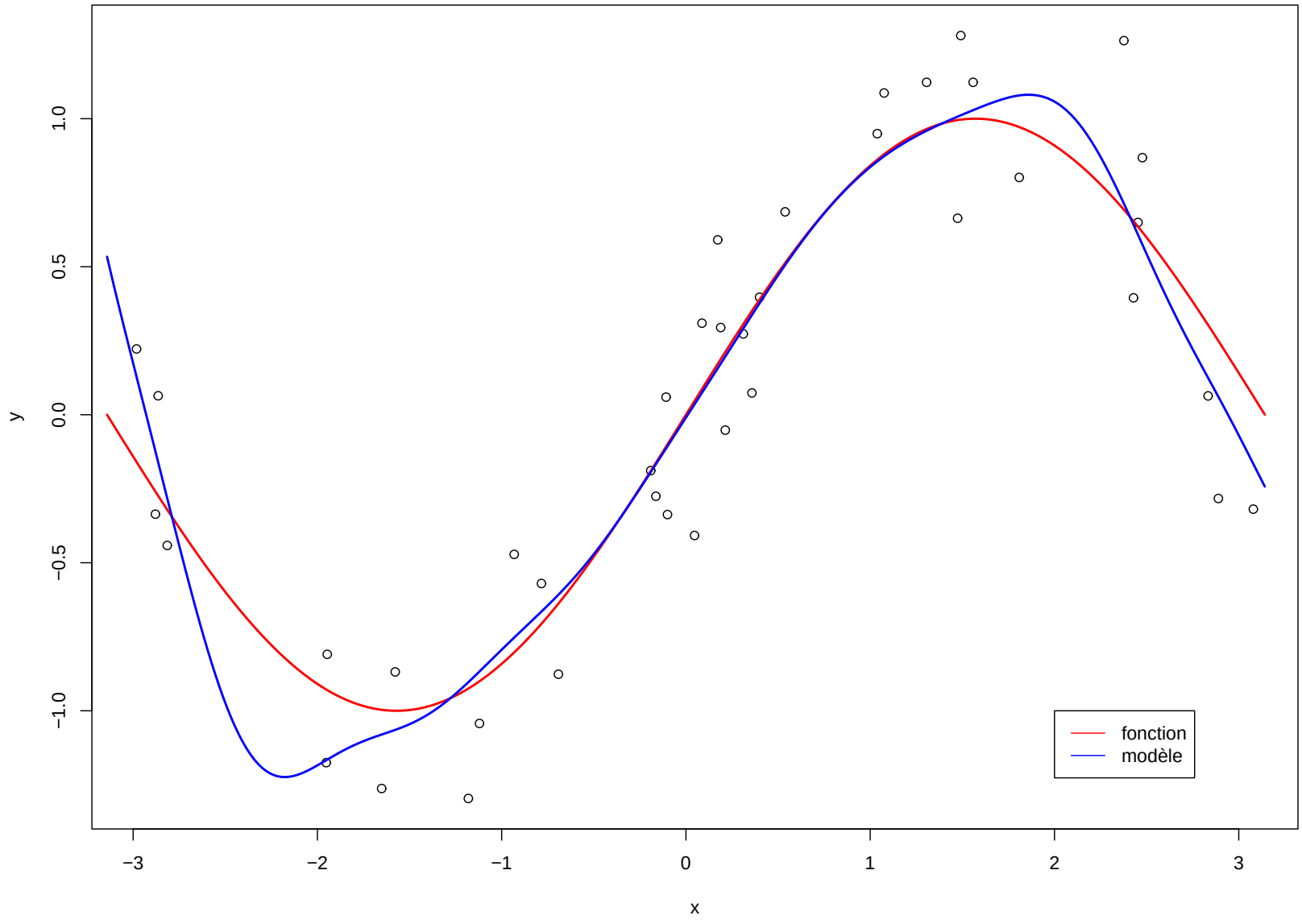
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-2}$)

Exemple : RBF, pénalité sur la dérivée seconde



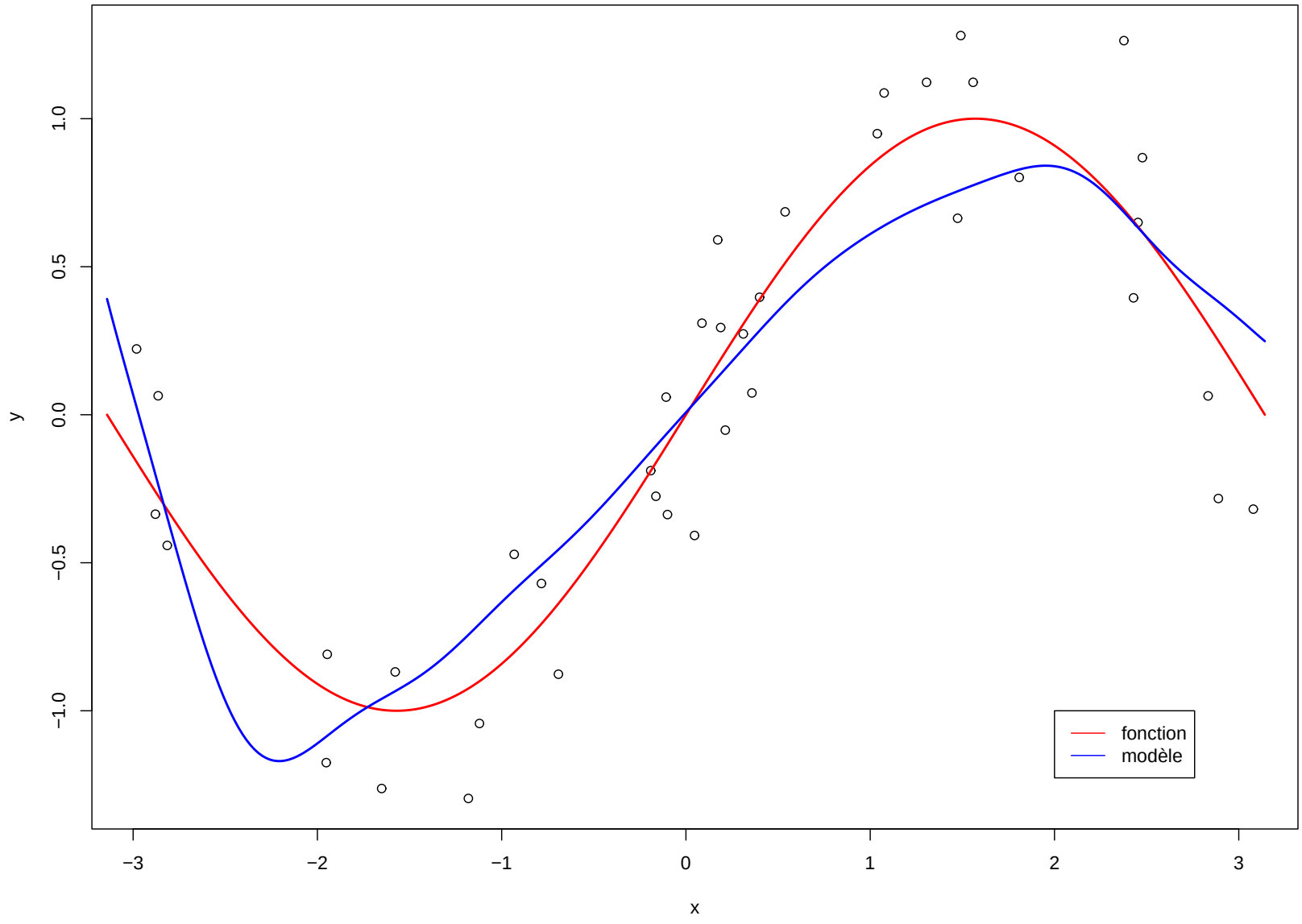
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-2}$)

Exemple : RBF, pénalité sur la dérivée seconde



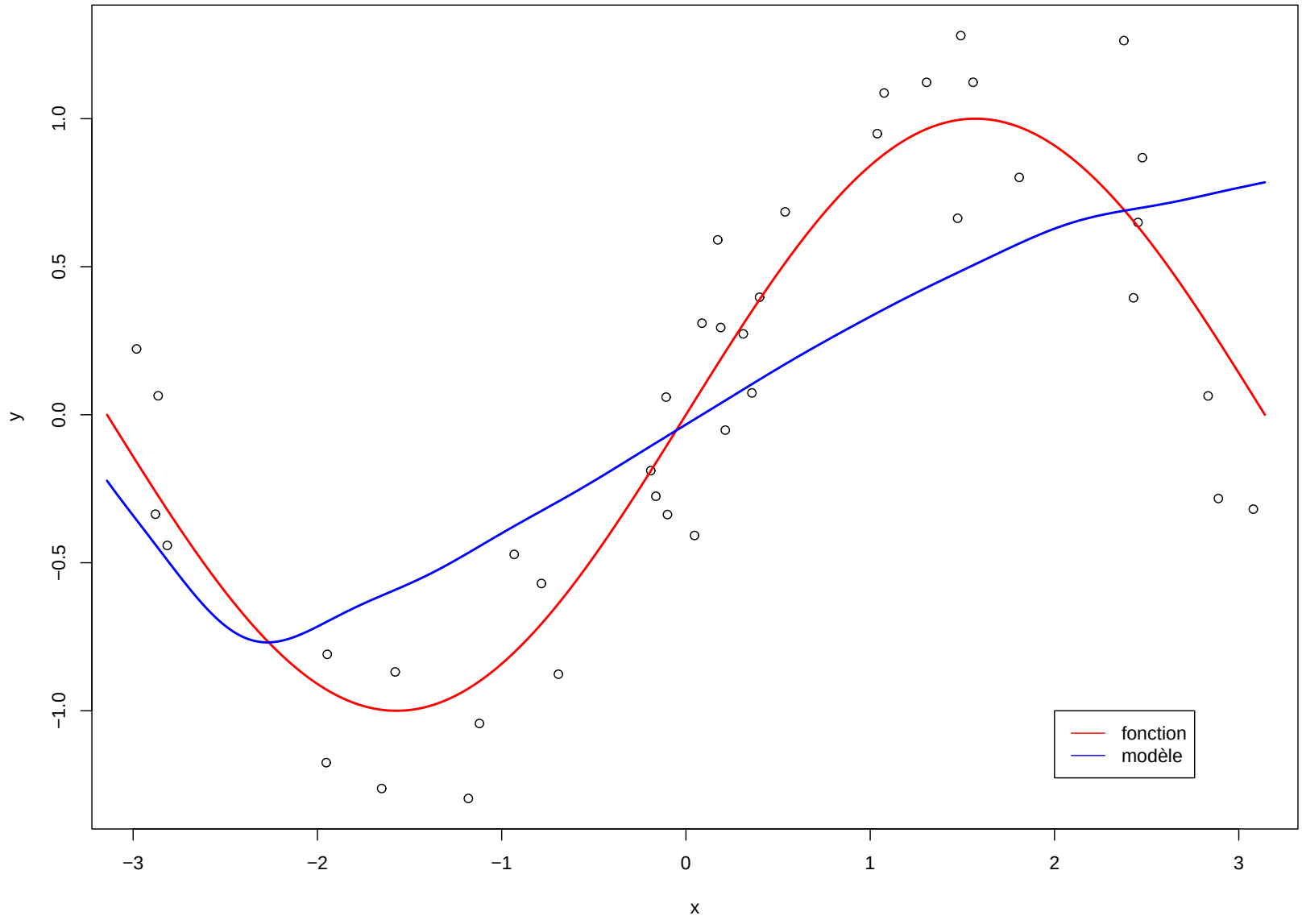
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-1}$)

Exemple : RBF, pénalité sur la dérivée seconde



Réseau RBF à 20 neurones régularisé ($\nu = 1$)

Exemple : RBF, pénalité sur la dérivée seconde



Réseau RBF à 20 neurones régularisé ($\nu = 10$)

Réduction directe ?

Intuitivement :

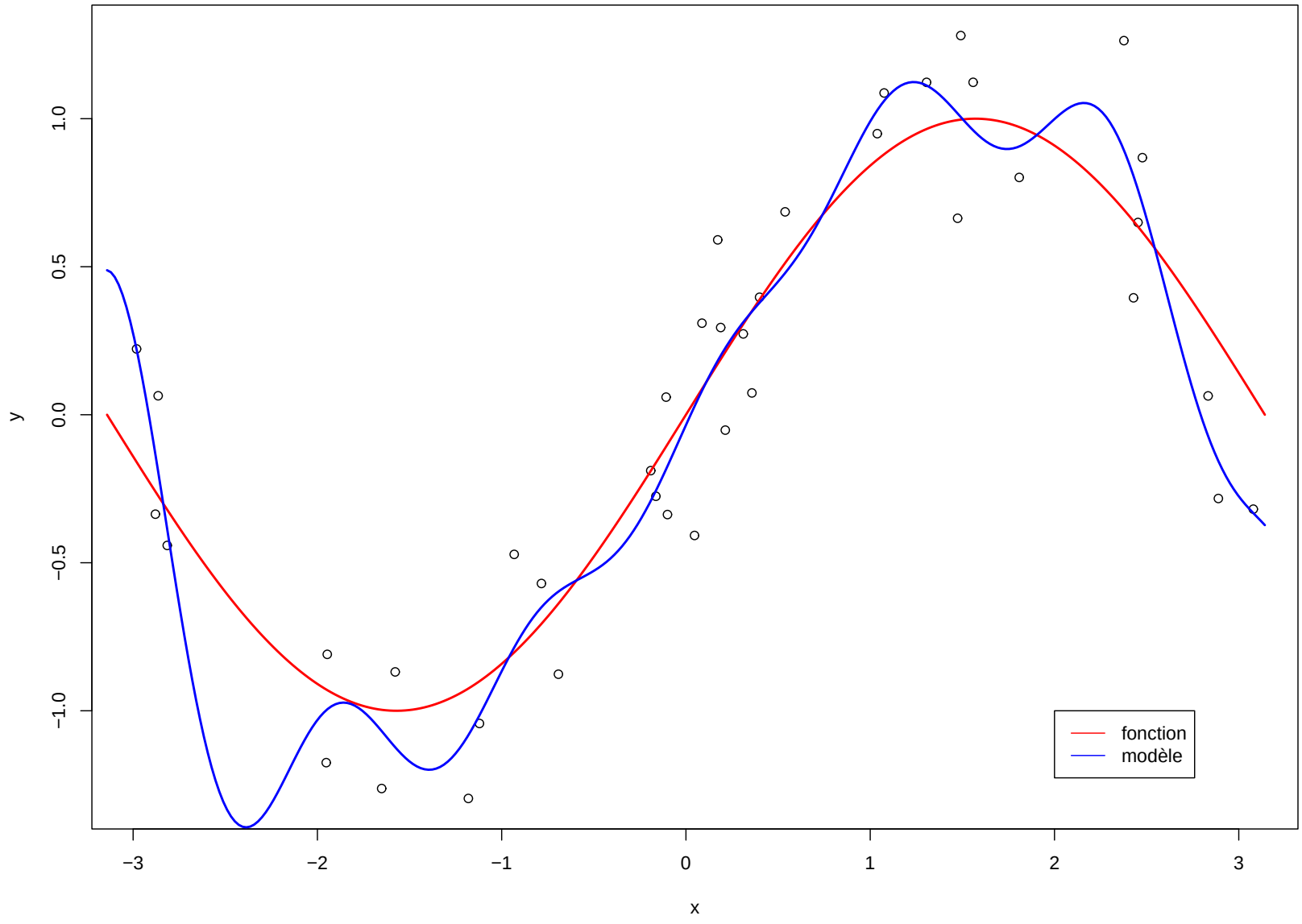
1. la régularisation réduit la puissance
2. la suppression de neurones réduit la puissance

La deuxième solution semble plus attractive :

- moins de neurones $\Rightarrow$ moins de calculs
- pas de terme de pénalité à calculer

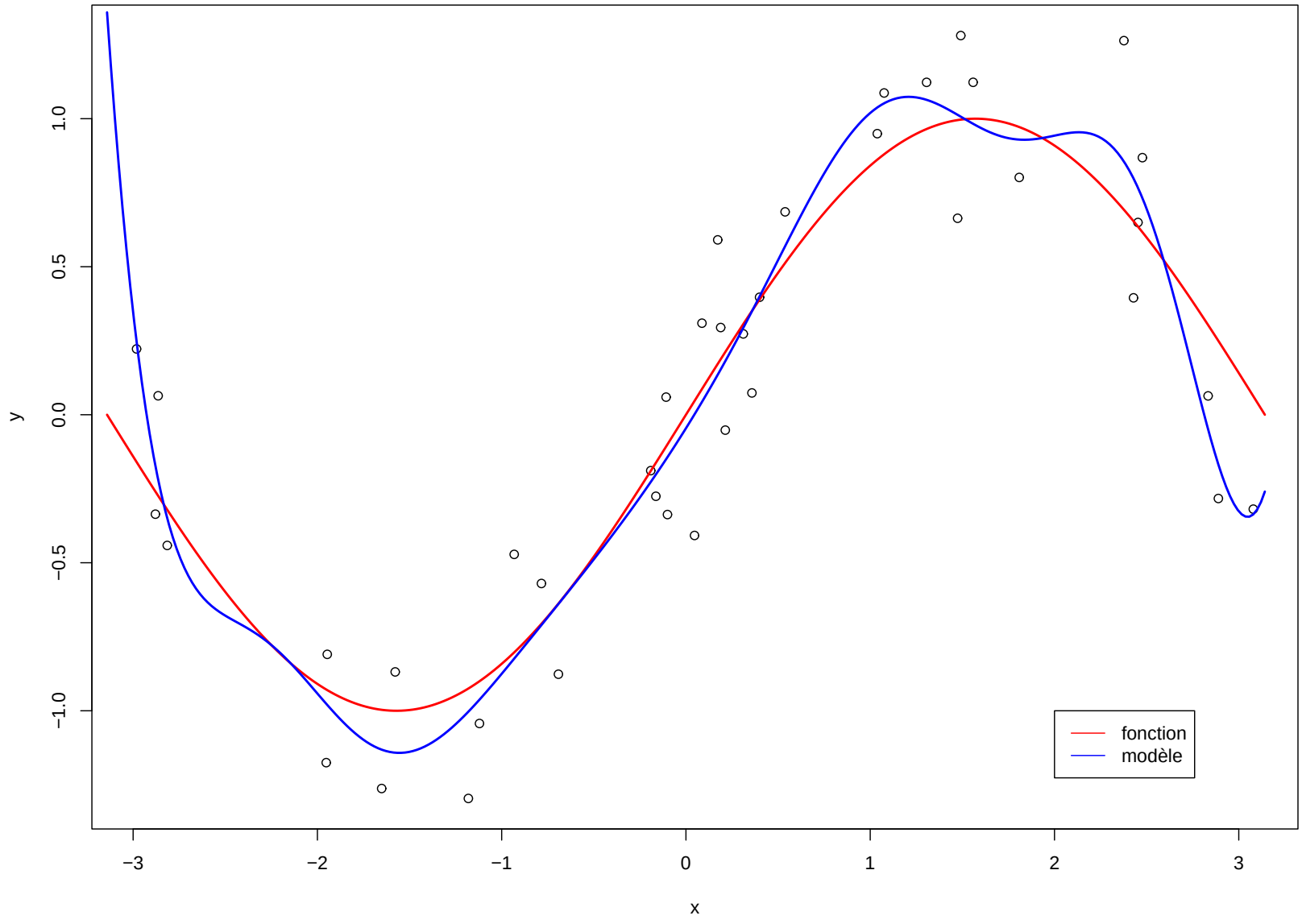
Mais en général la réduction de puissance ne rend pas la solution plus régulière (ou au contraire trop régulière). De plus, le contrôle n'est pas assez fin.

Exemple : RBF



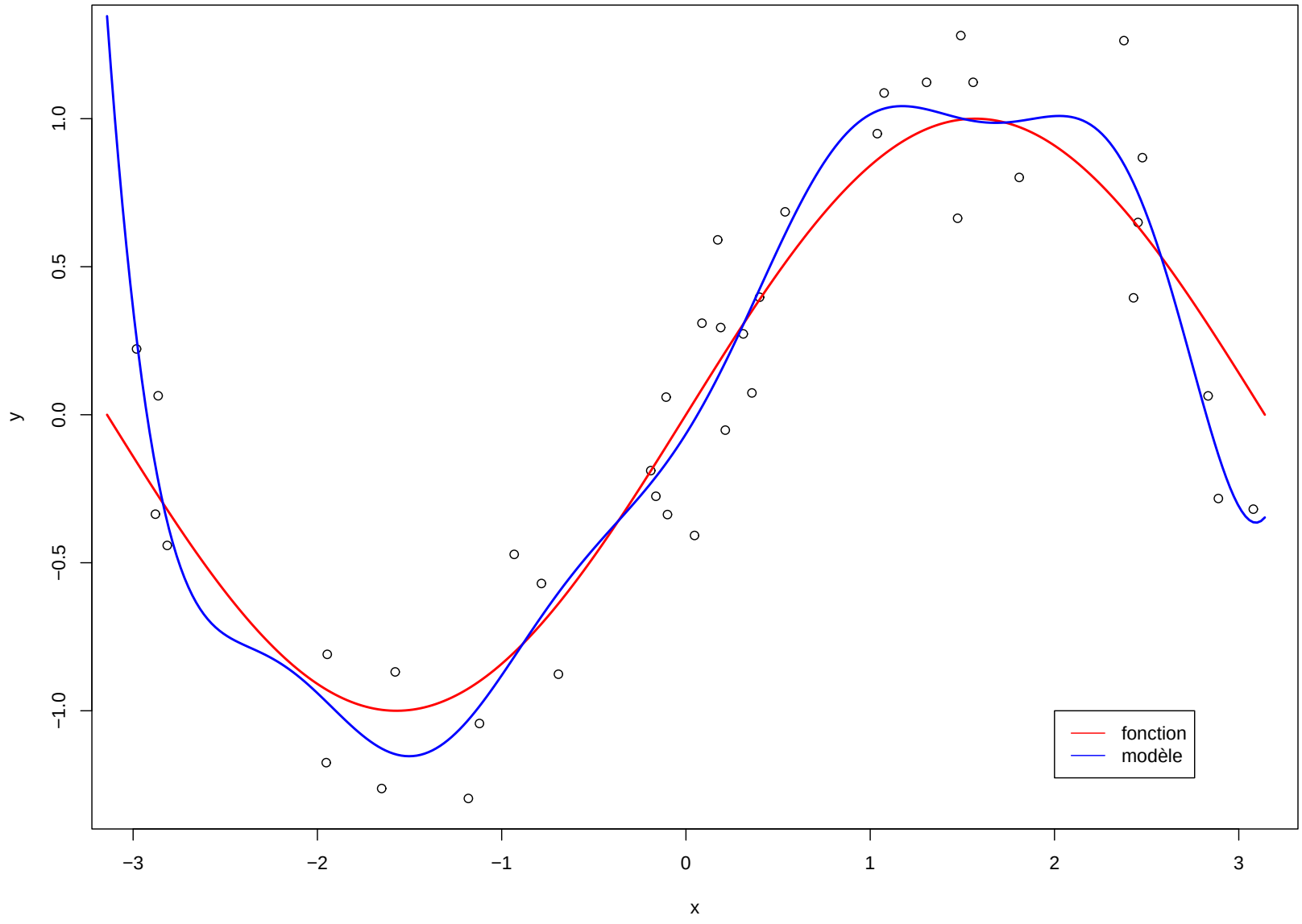
Réseau RBF à 20 neurones

Exemple : RBF



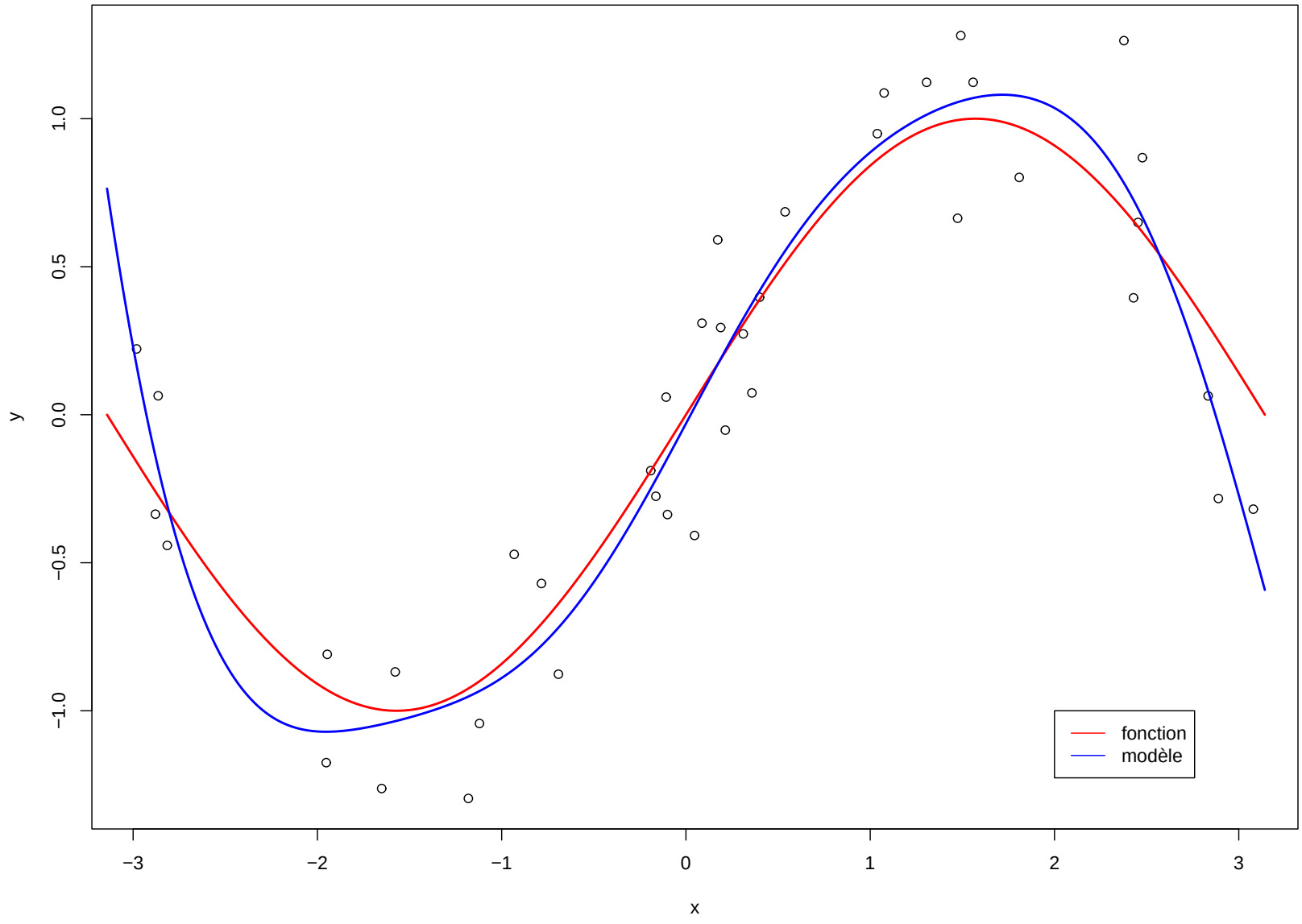
Réseau RBF à 15 neurones

Exemple : RBF



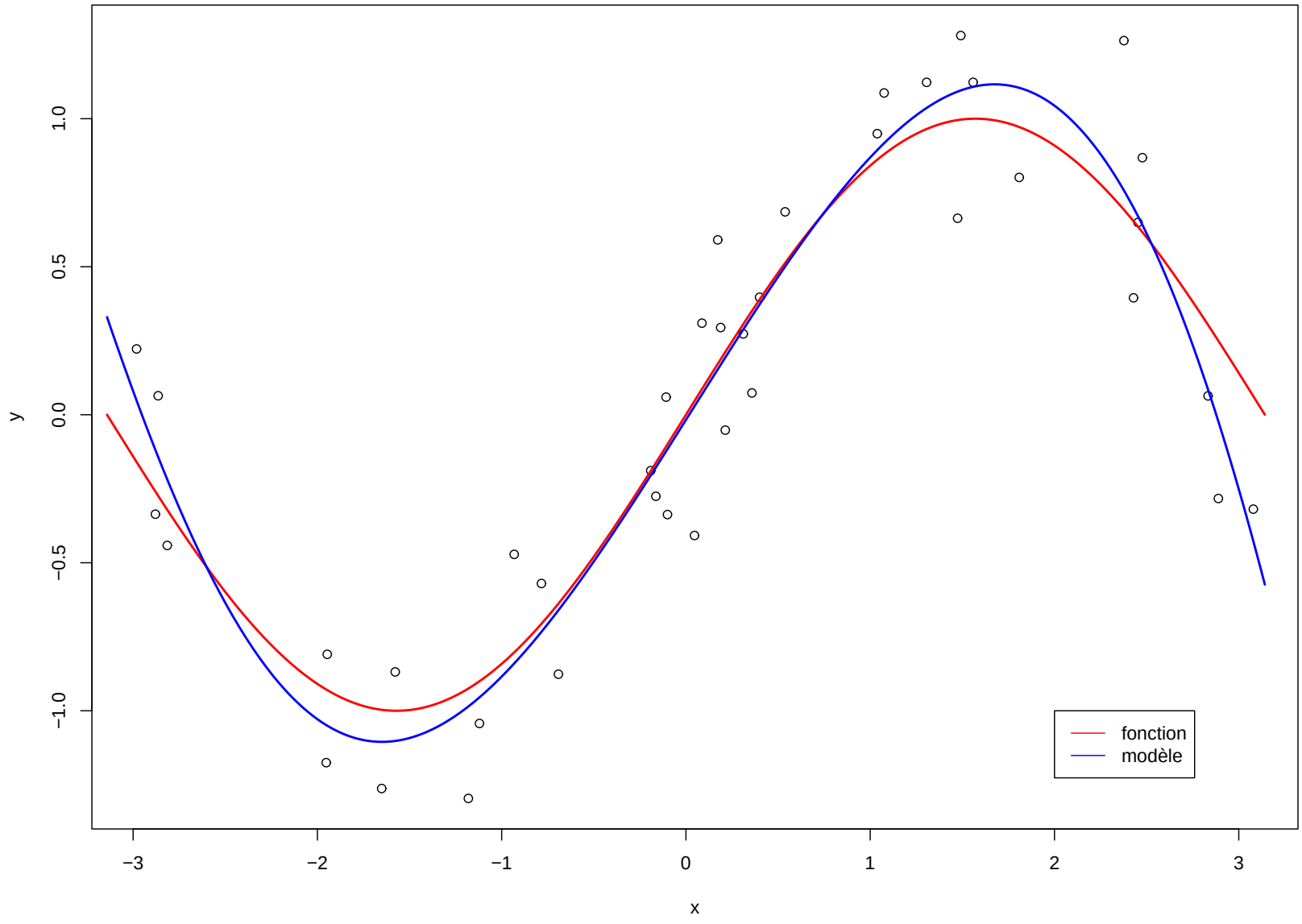
Réseau RBF à 12 neurones

Exemple : RBF



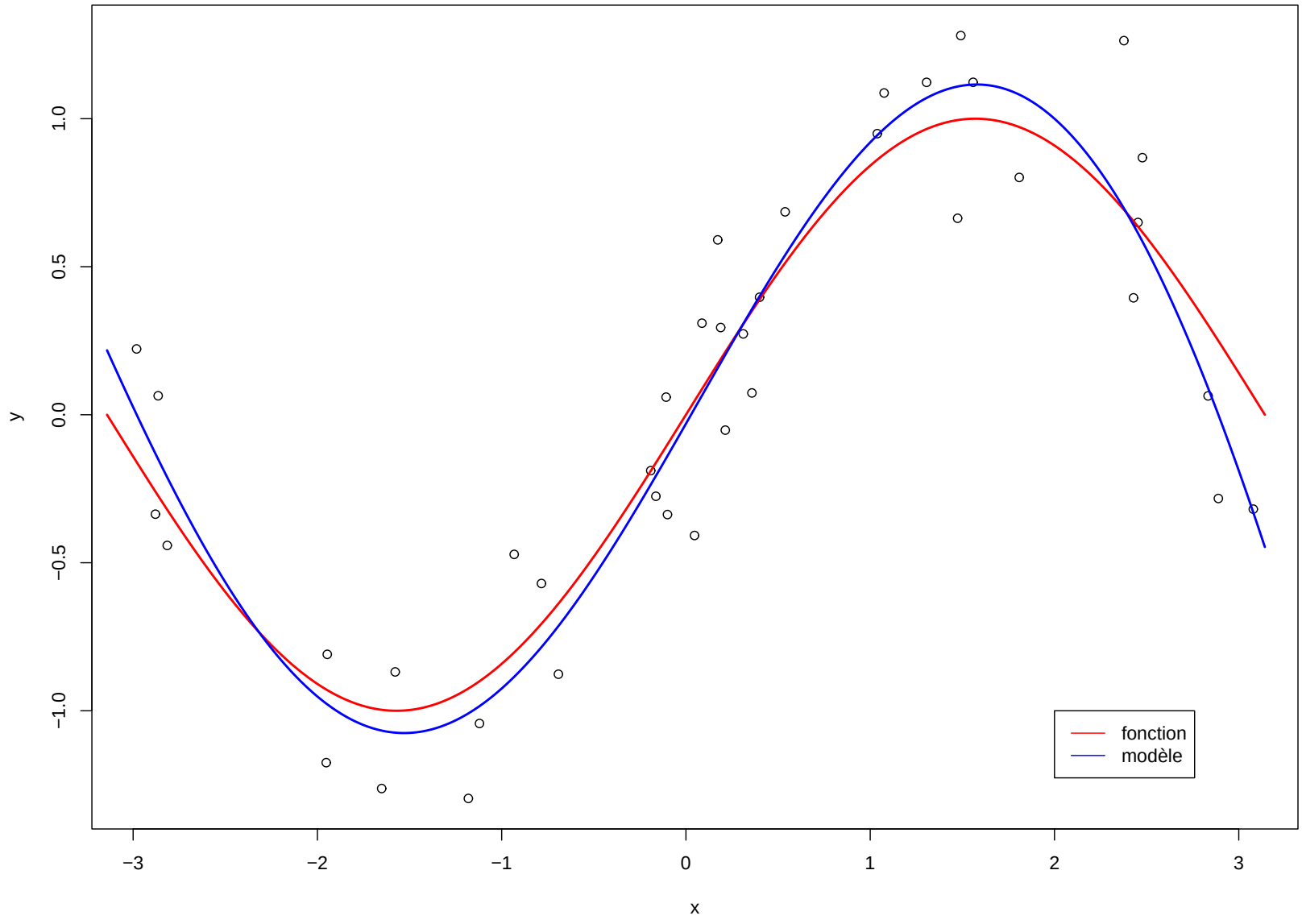
Réseau RBF à 9 neurones

Exemple : RBF



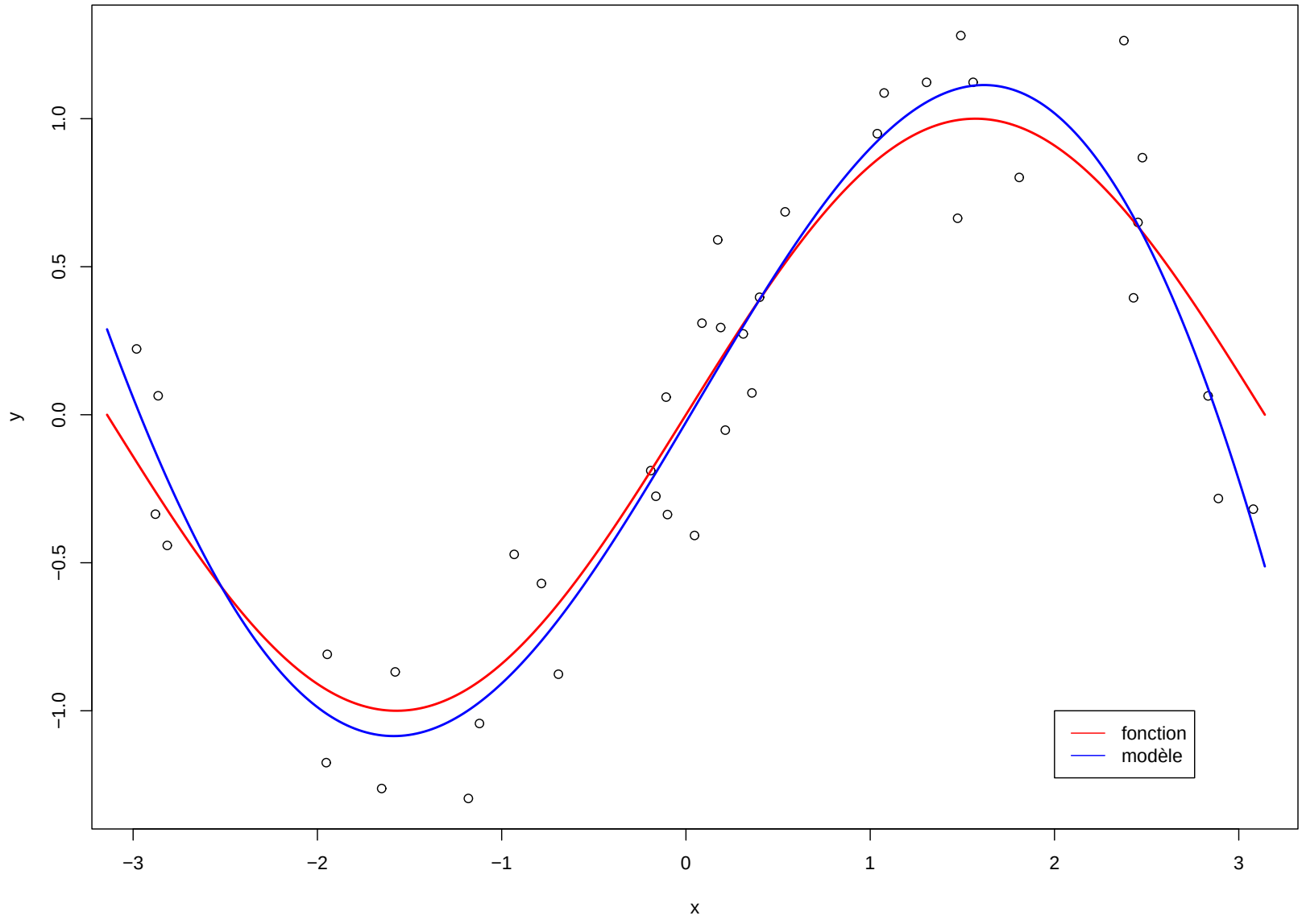
Réseau RBF à 6 neurones

Exemple : RBF



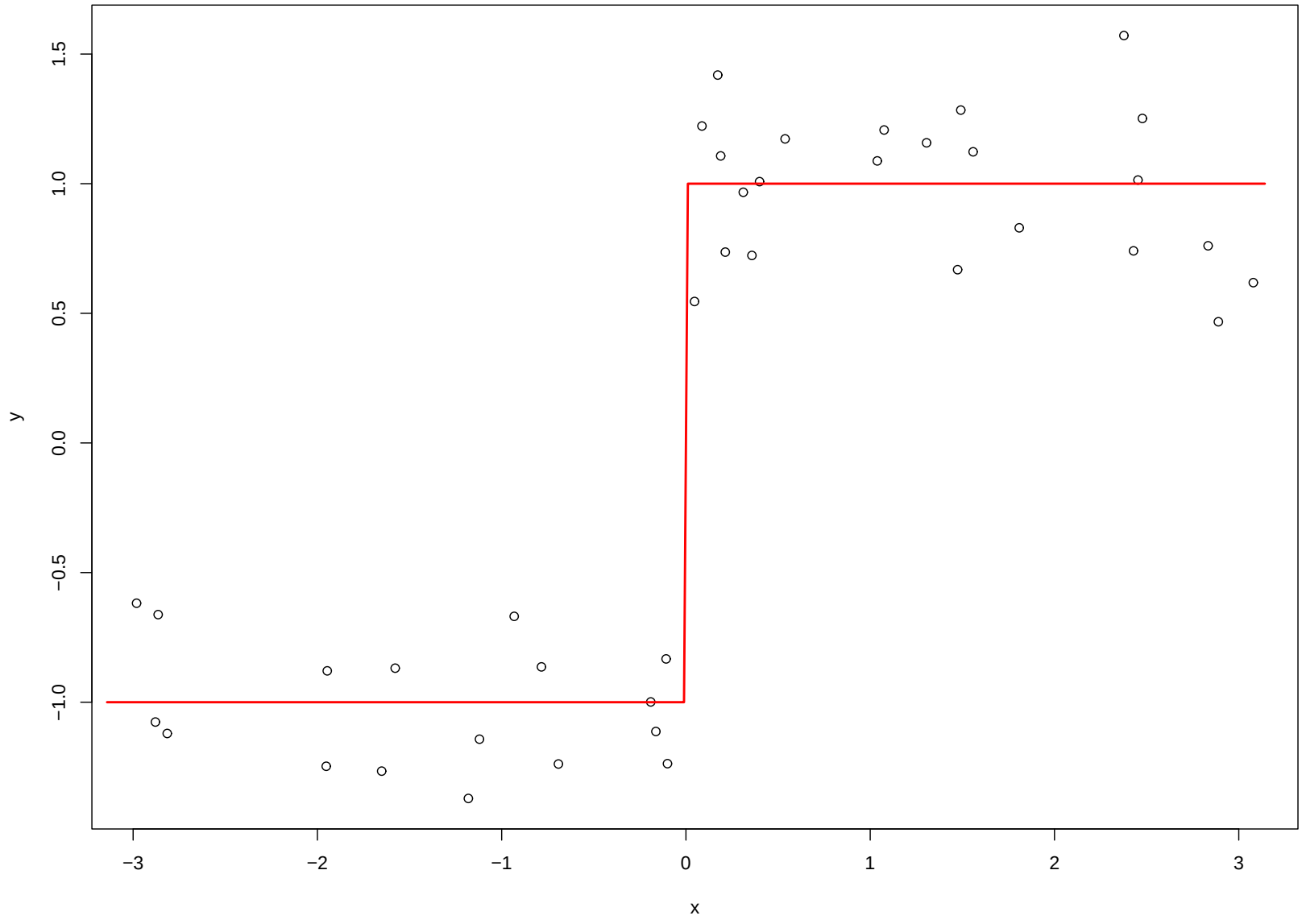
Réseau RBF à 5 neurones

Exemple : RBF

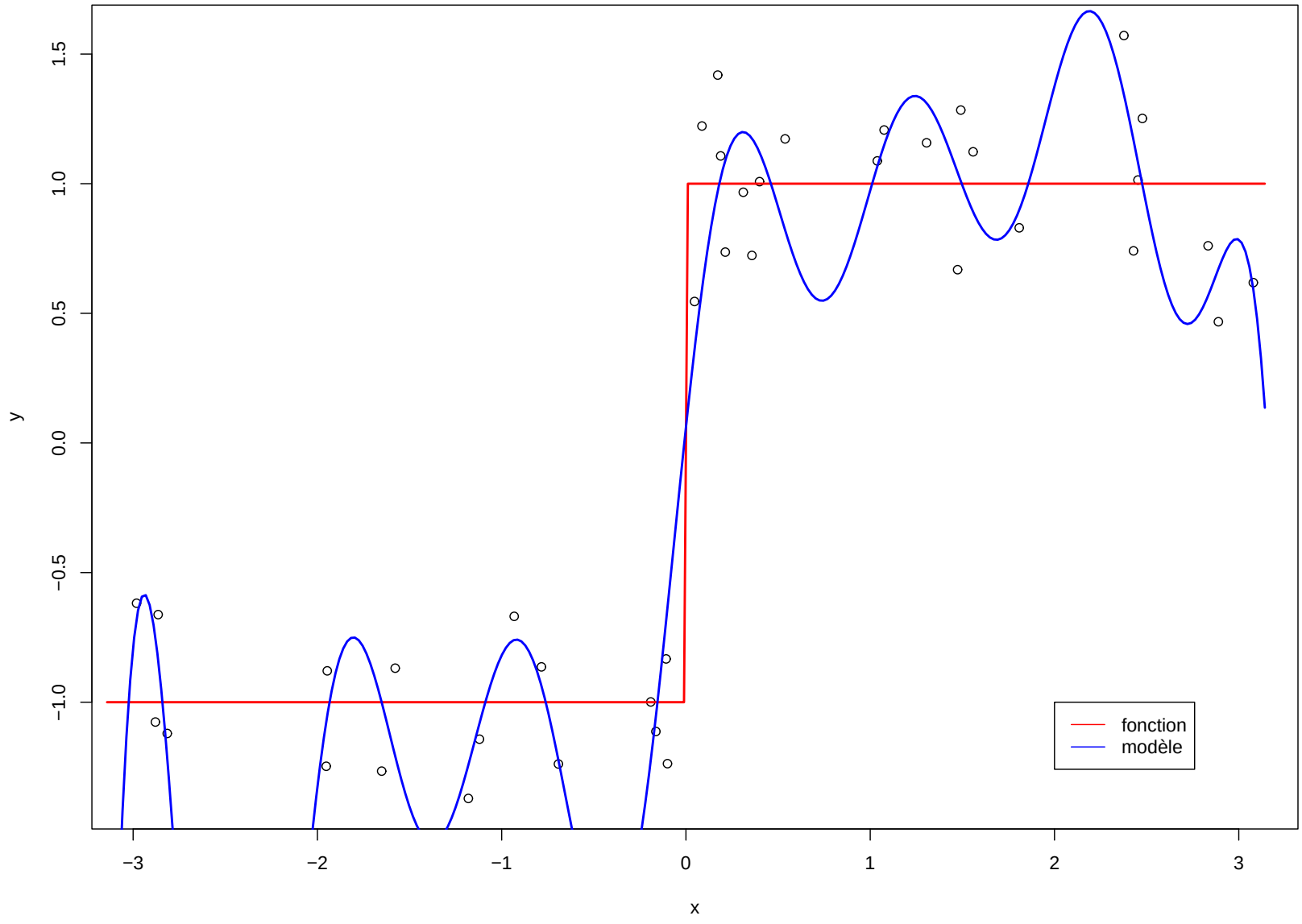


Réseau RBF à 4 neurones

Exemple : RBF, pénalité sur la dérivée seconde

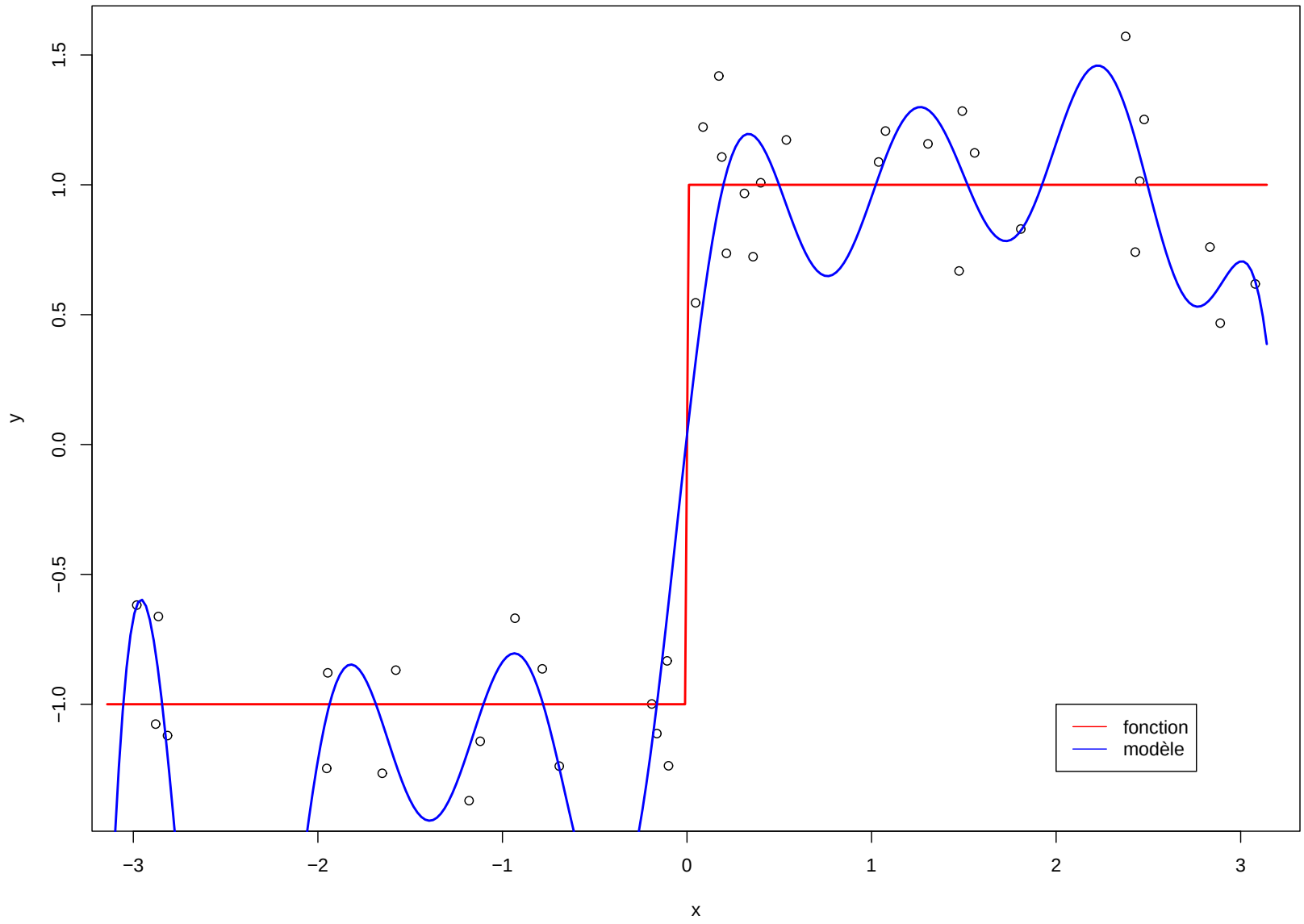


Exemple : RBF, pénalité sur la dérivée seconde



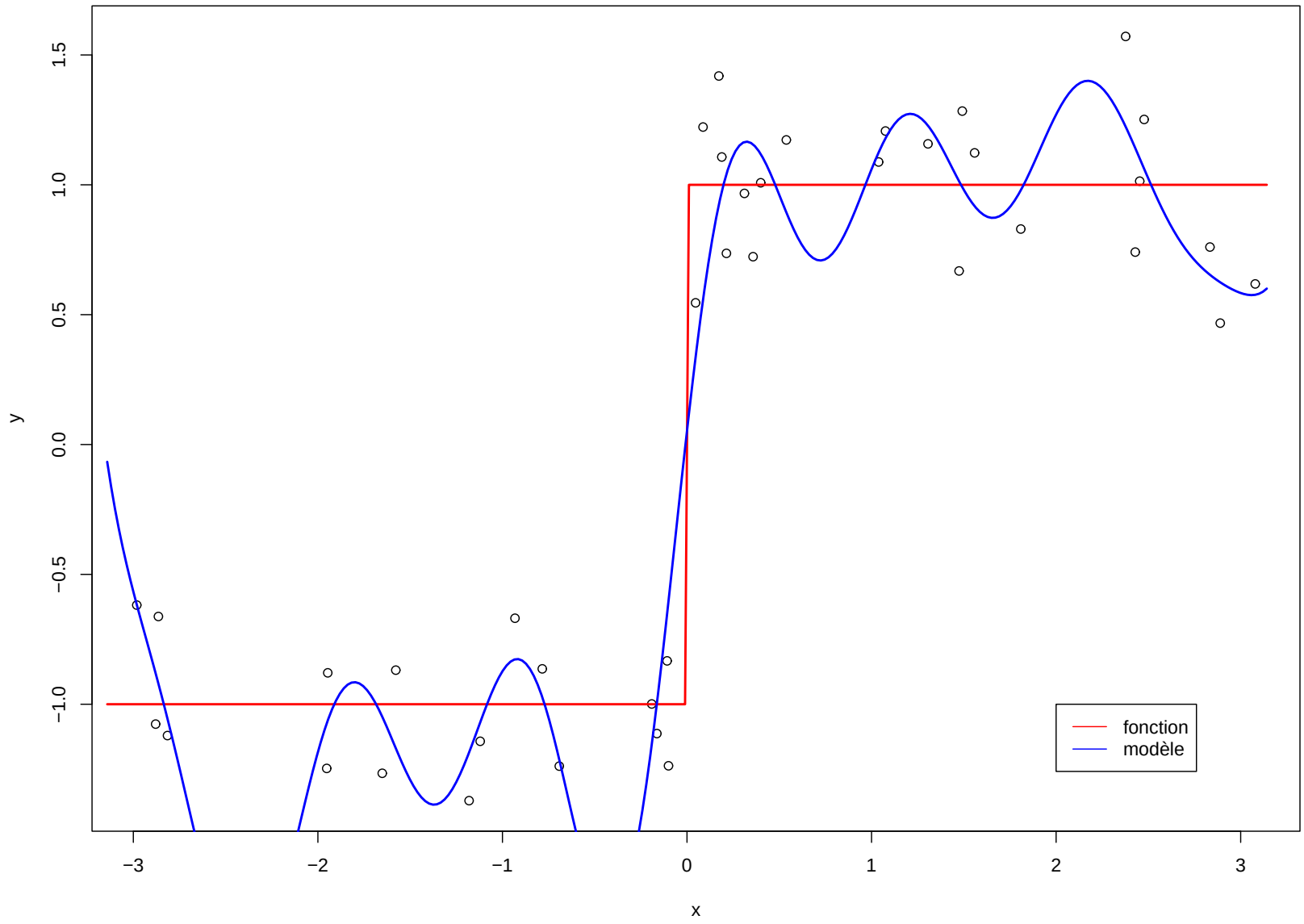
Réseau RBF à 20 neurones

Exemple : RBF, pénalité sur la dérivée seconde



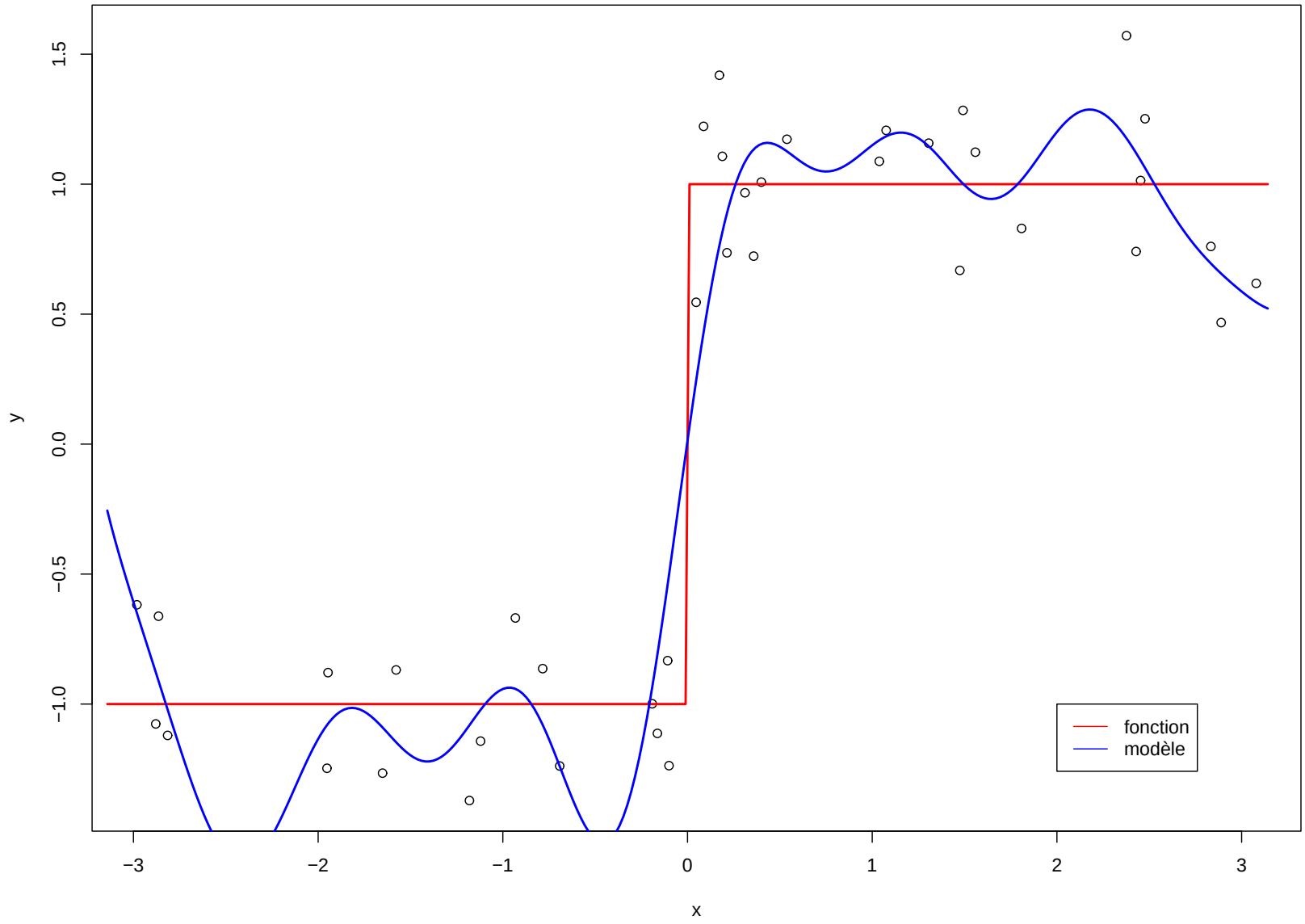
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-5}$)

Exemple : RBF, pénalité sur la dérivée seconde



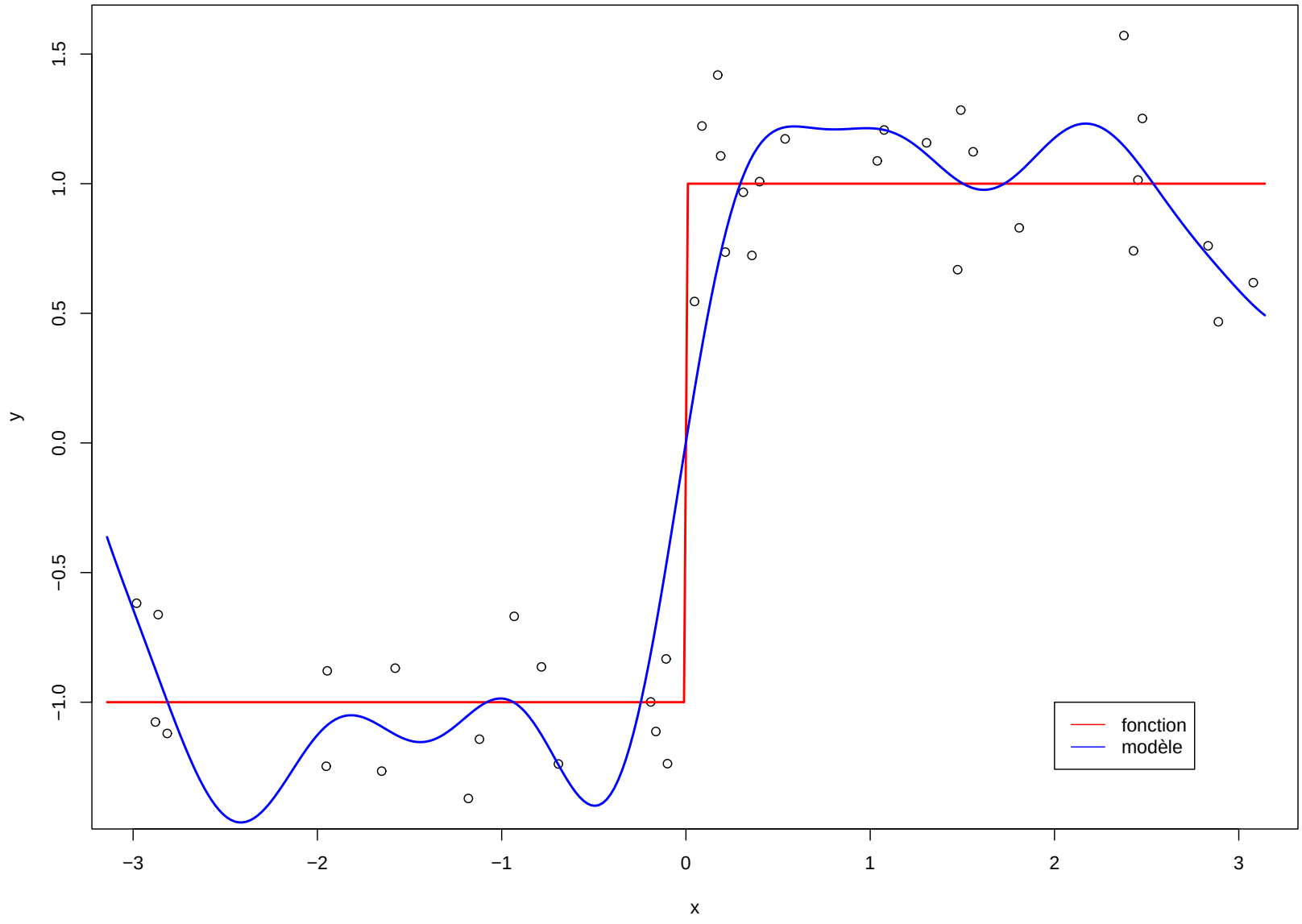
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-4}$)

Exemple : RBF, pénalité sur la dérivée seconde



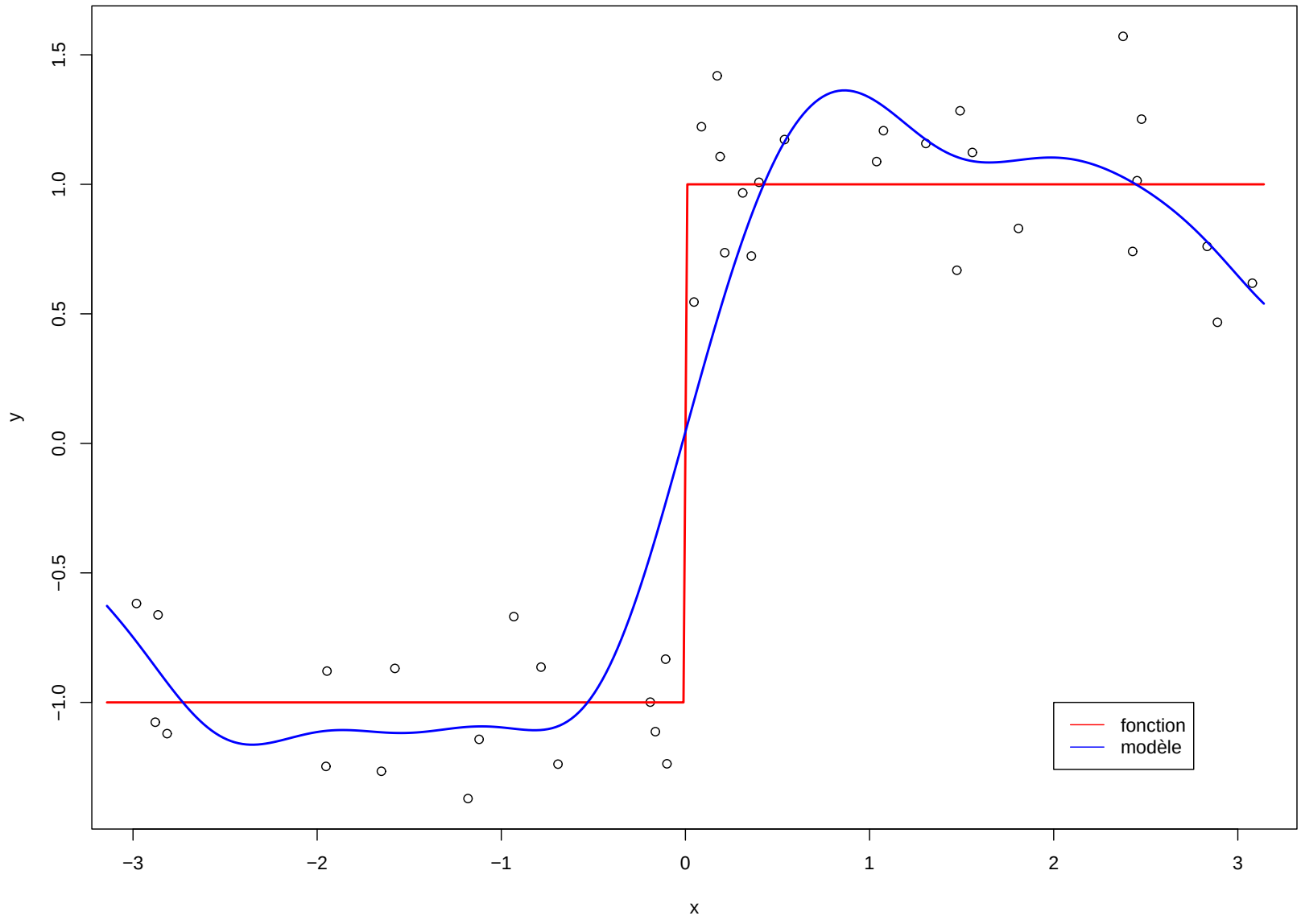
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-4}$)

Exemple : RBF, pénalité sur la dérivée seconde



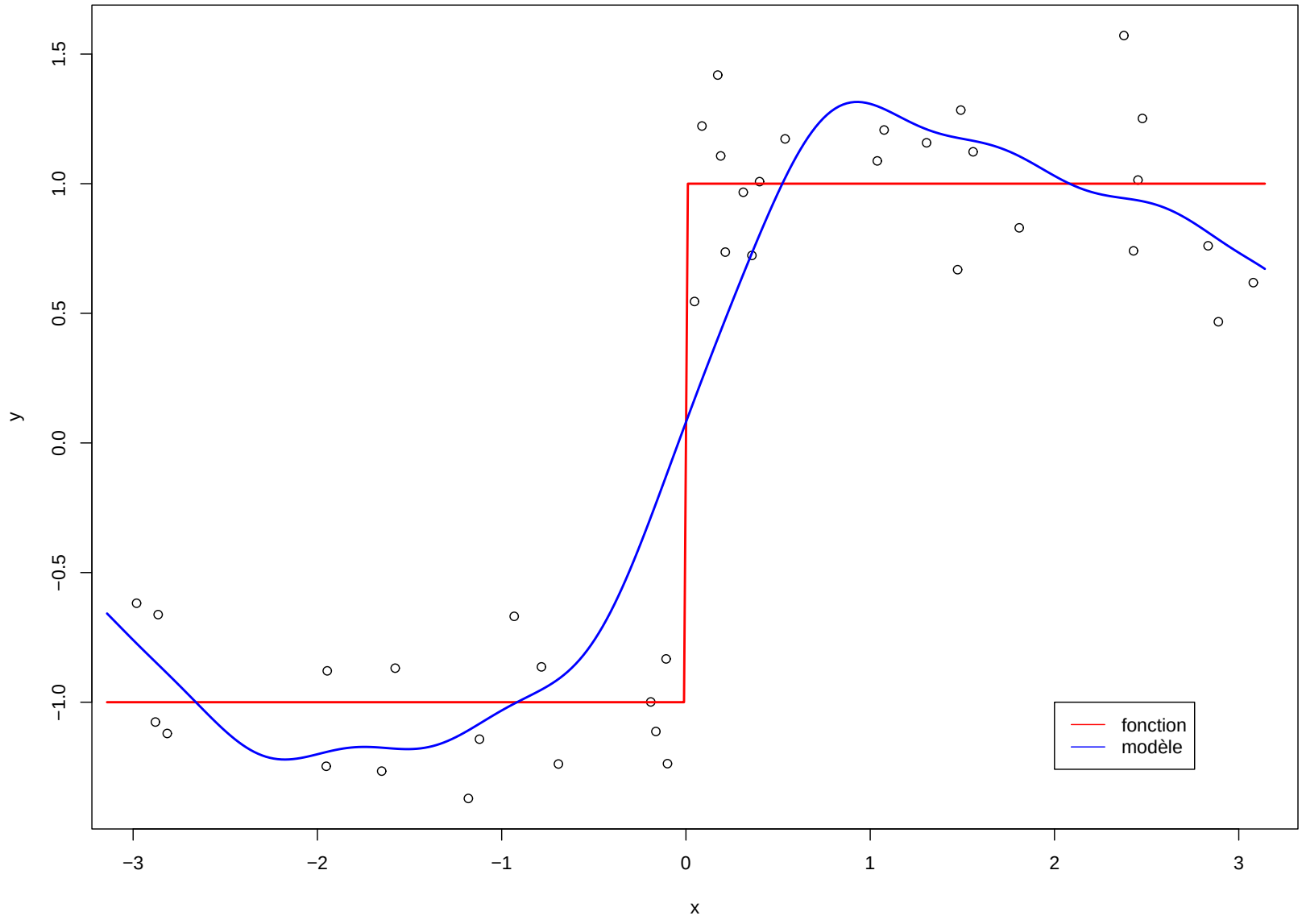
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-3}$)

Exemple : RBF, pénalité sur la dérivée seconde



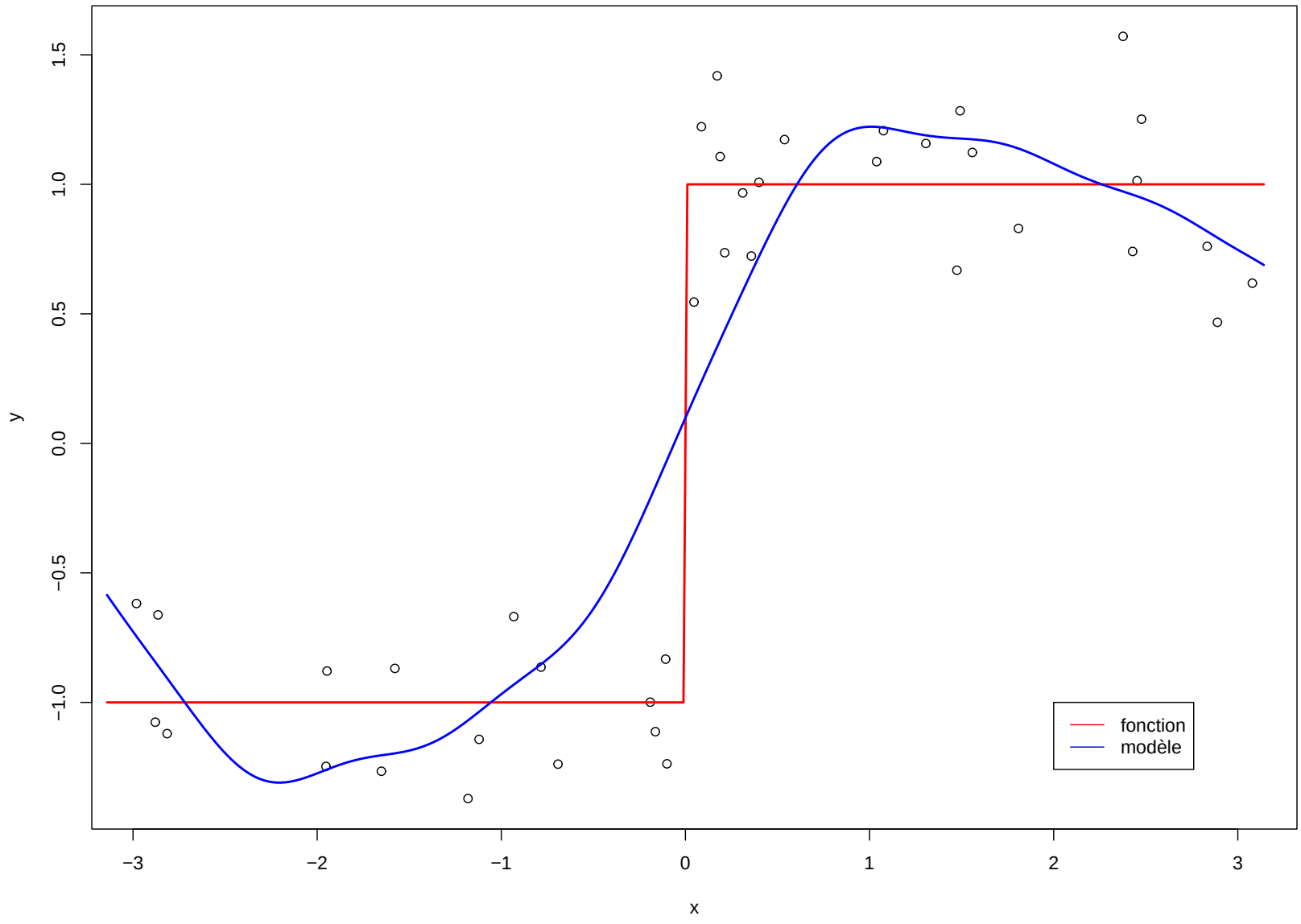
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-2}$)

Exemple : RBF, pénalité sur la dérivée seconde



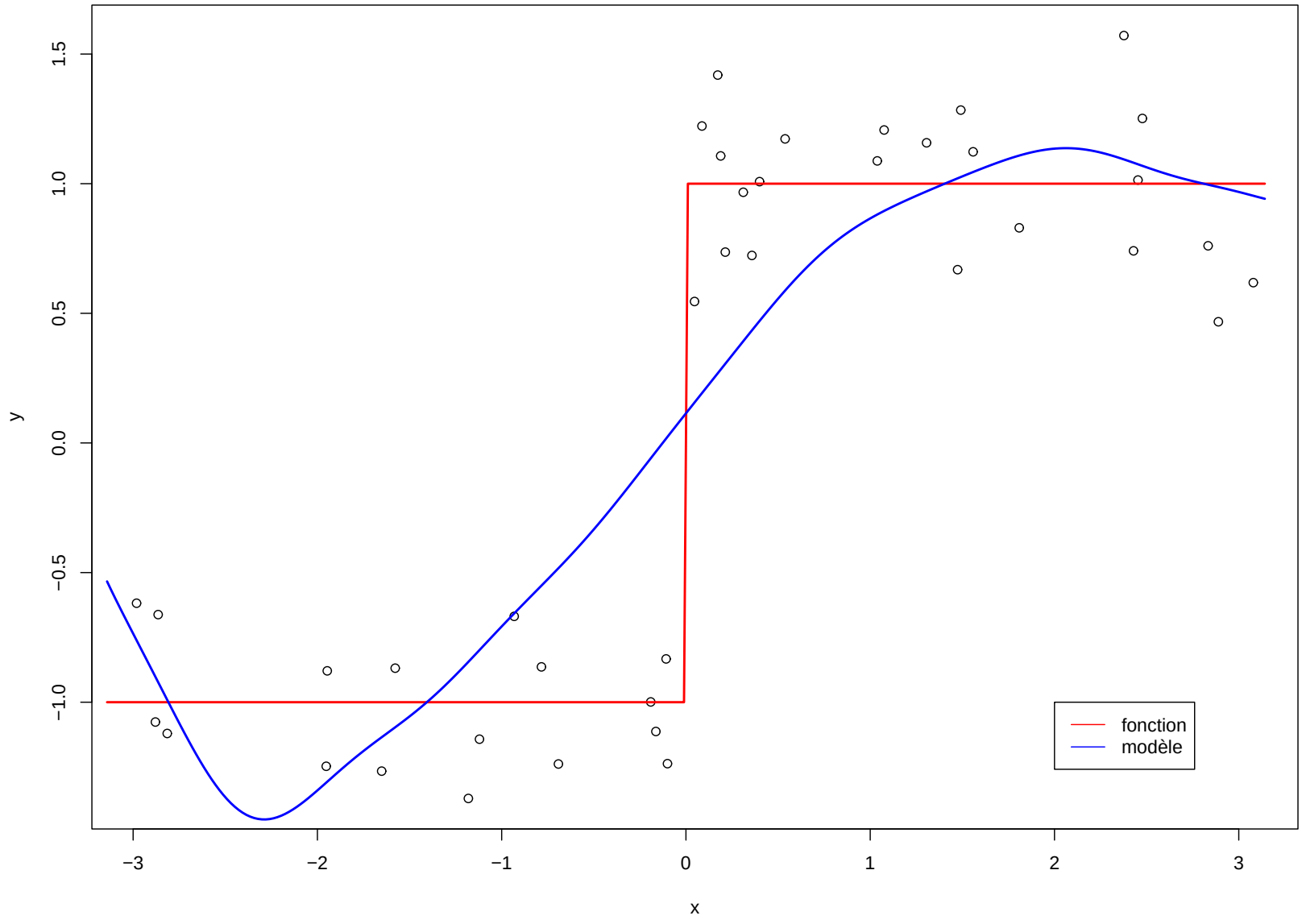
Réseau RBF à 20 neurones régularisé ($\nu = 5\,10^{-2}$)

Exemple : RBF, pénalité sur la dérivée seconde



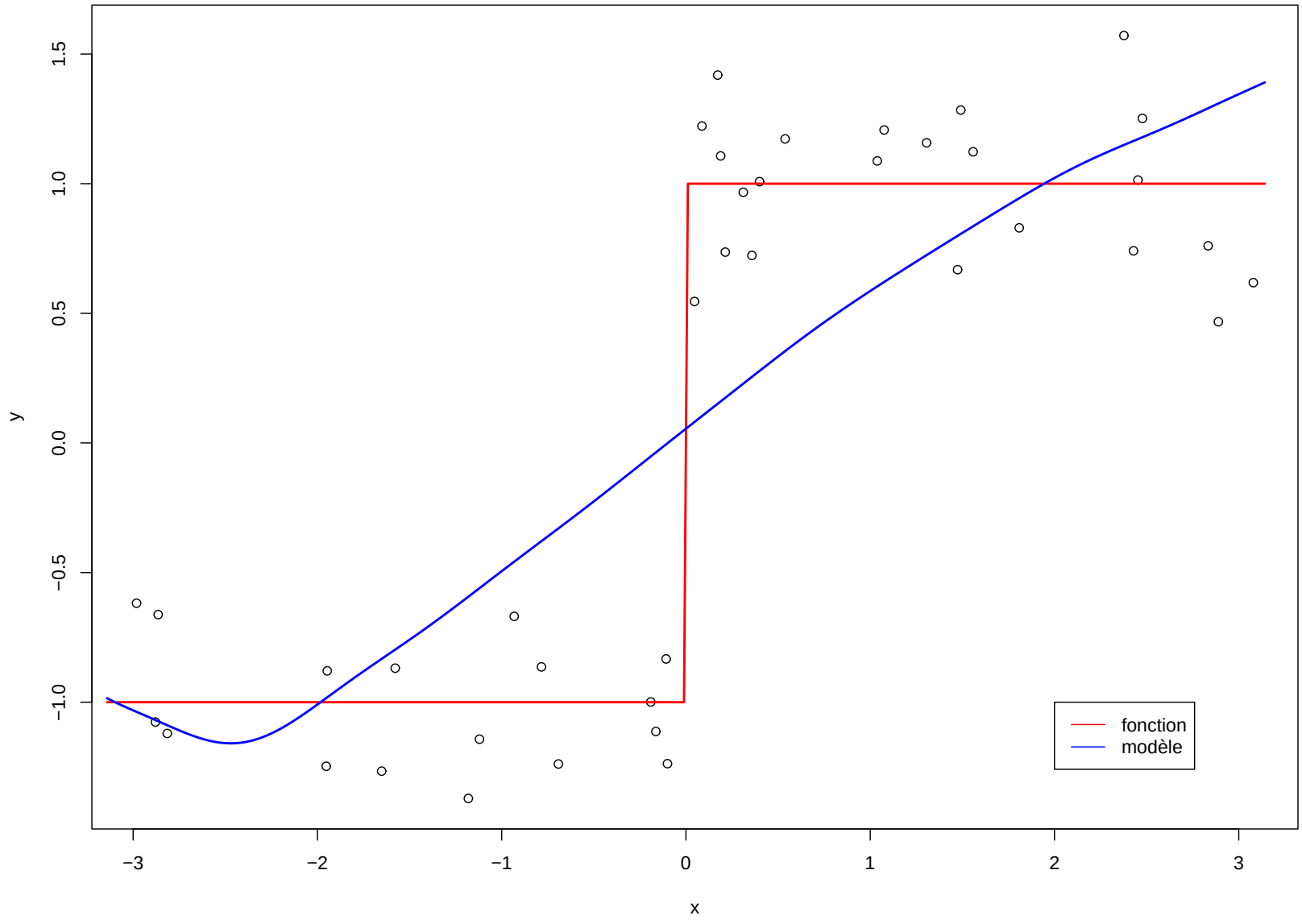
Réseau RBF à 20 neurones régularisé ($\nu = 10^{-1}$)

Exemple : RBF, pénalité sur la dérivée seconde



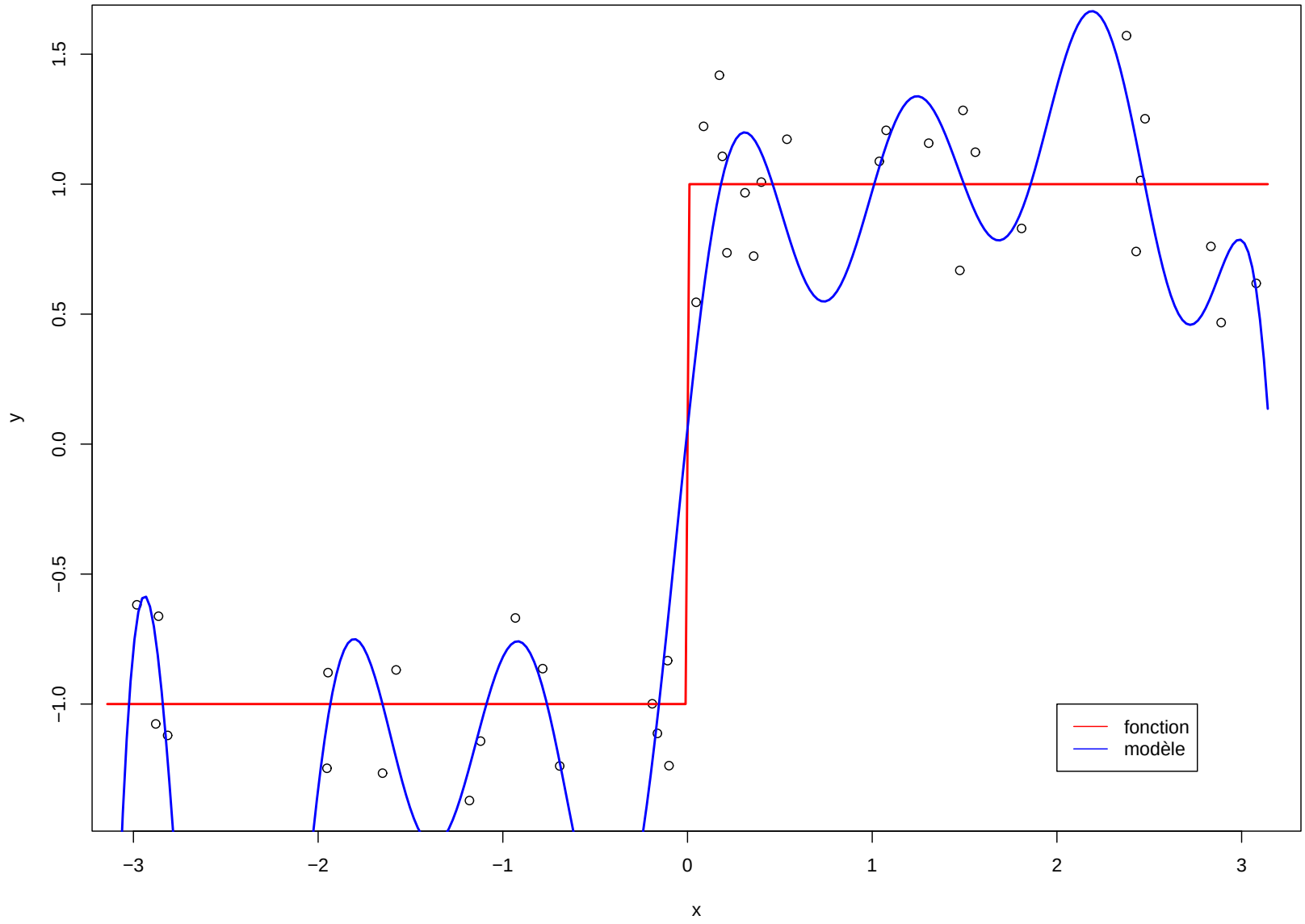
Réseau RBF à 20 neurones régularisé ($\nu = 1$)

Exemple : RBF, pénalité sur la dérivée seconde



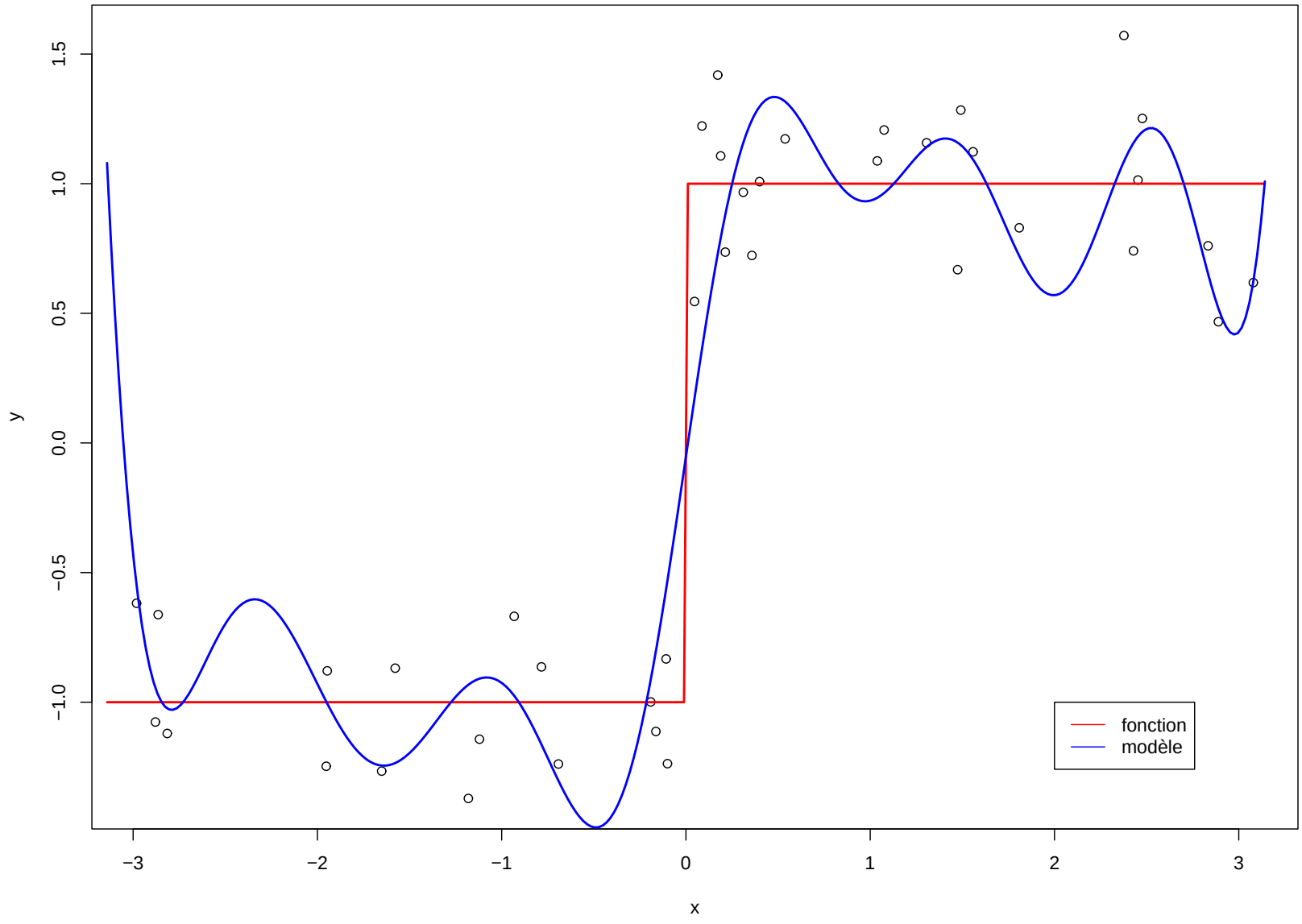
Réseau RBF à 20 neurones régularisé ($\nu = 10$)

Exemple : RBF



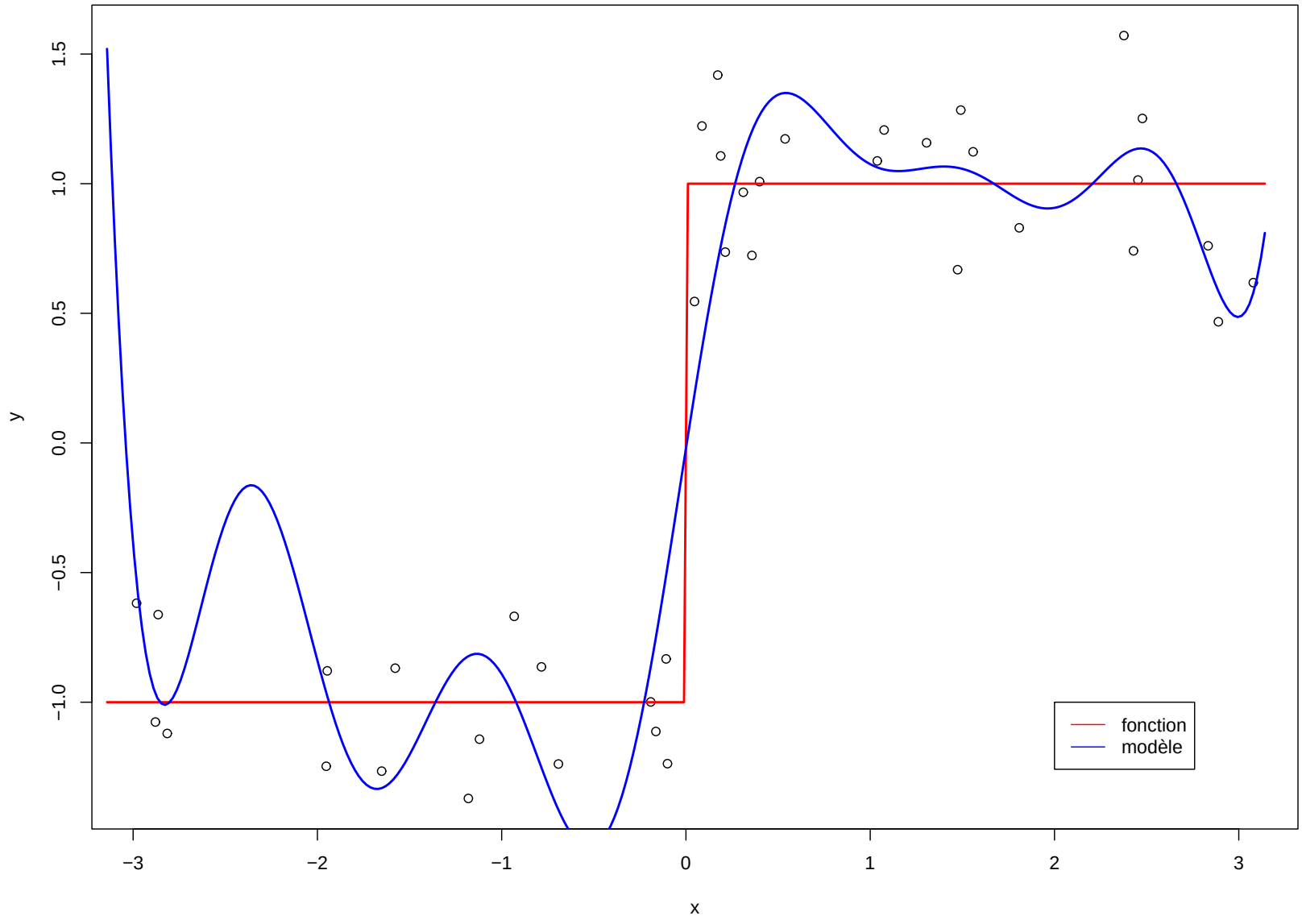
Réseau RBF à 20 neurones

Exemple : RBF



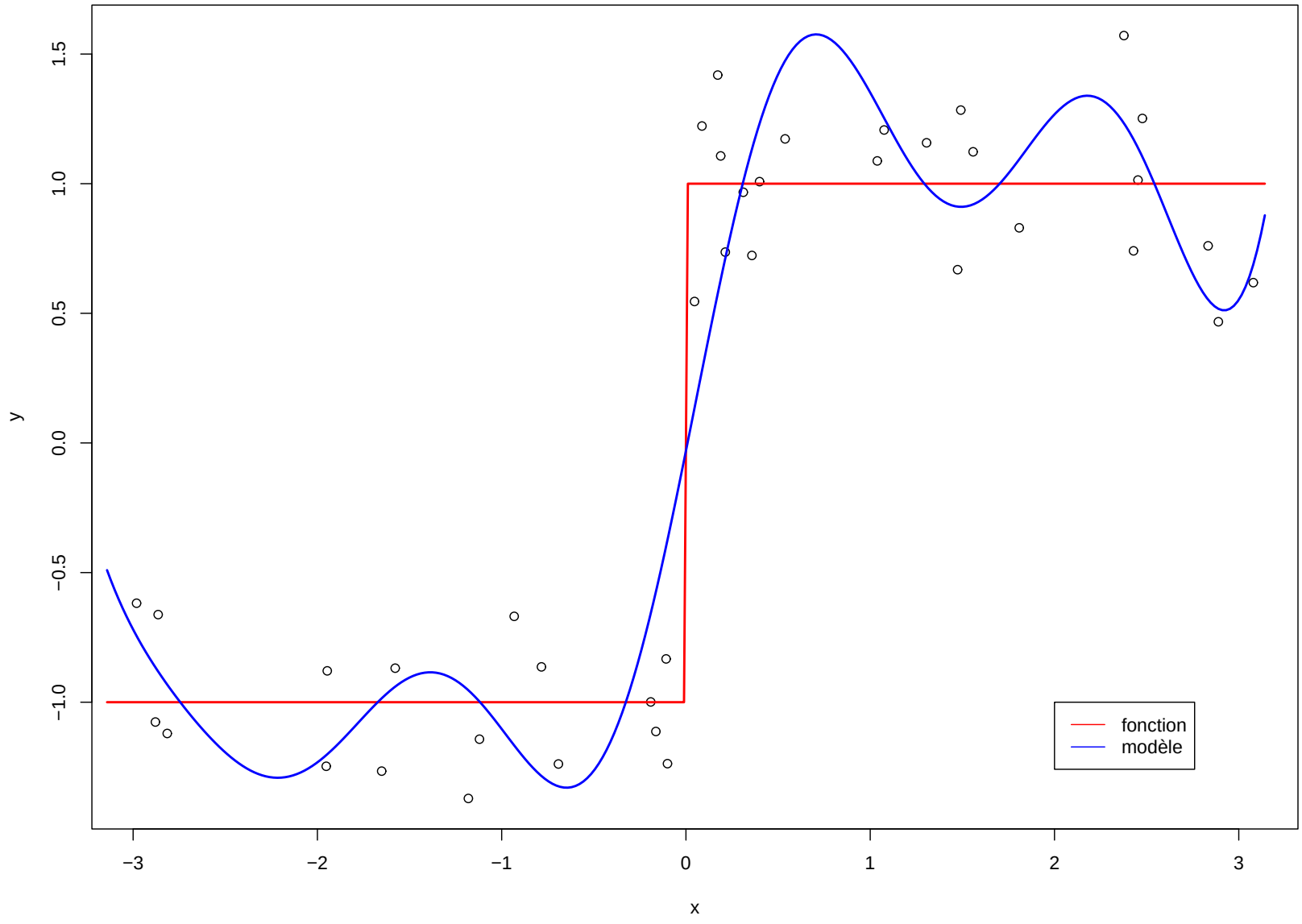
Réseau RBF à 18 neurones

Exemple : RBF



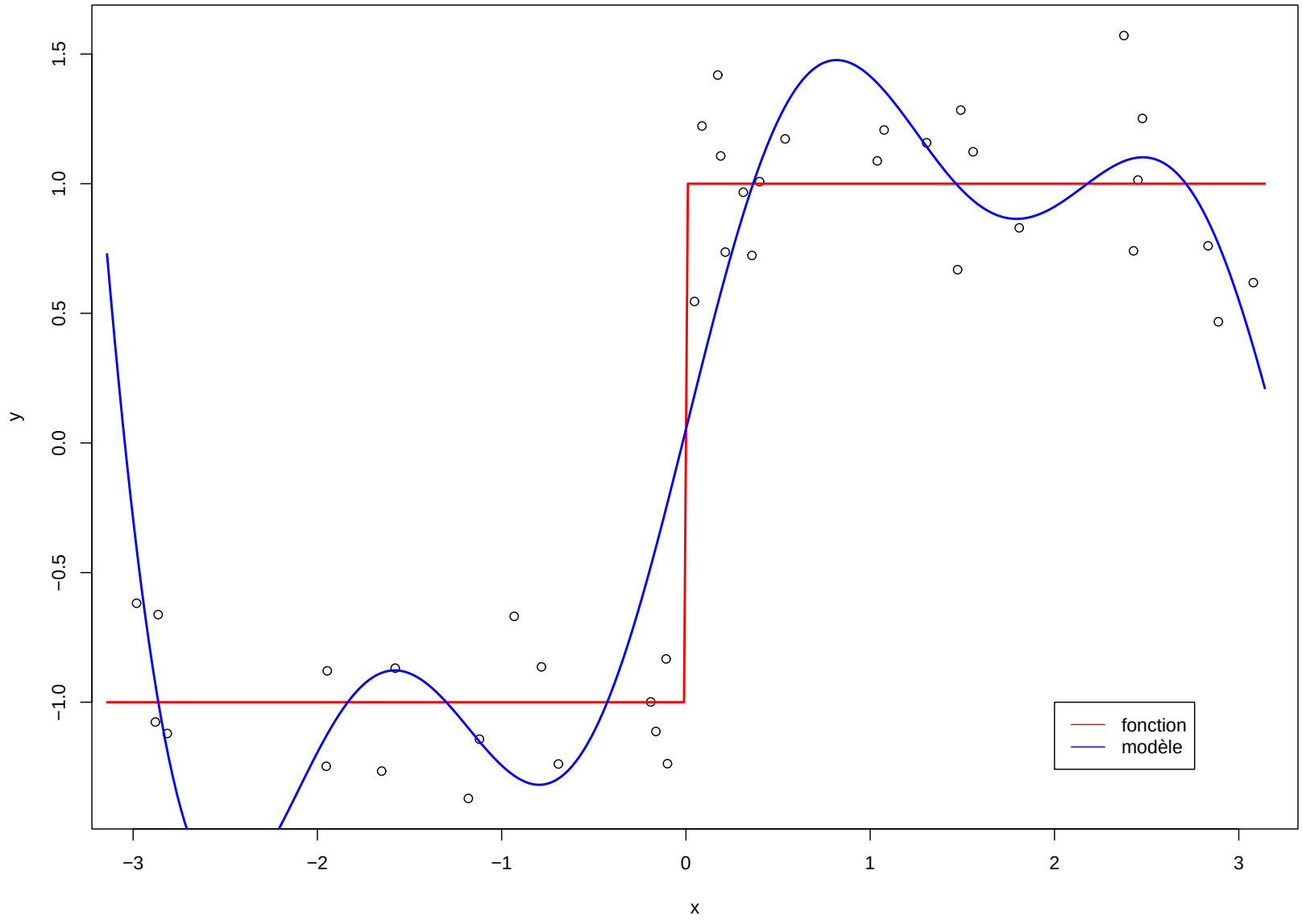
Réseau RBF à 15 neurones

Exemple : RBF



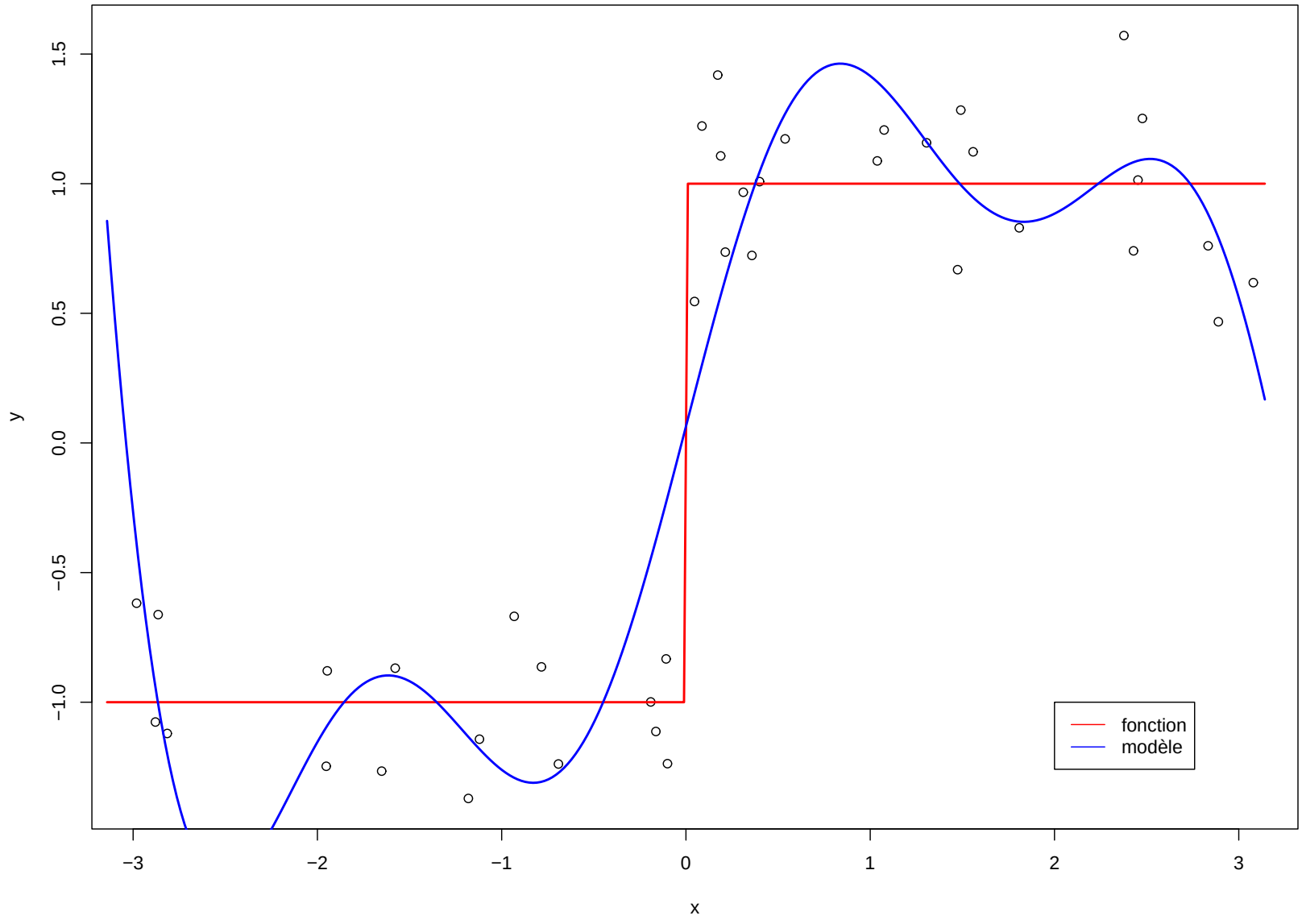
Réseau RBF à 12 neurones

Exemple : RBF



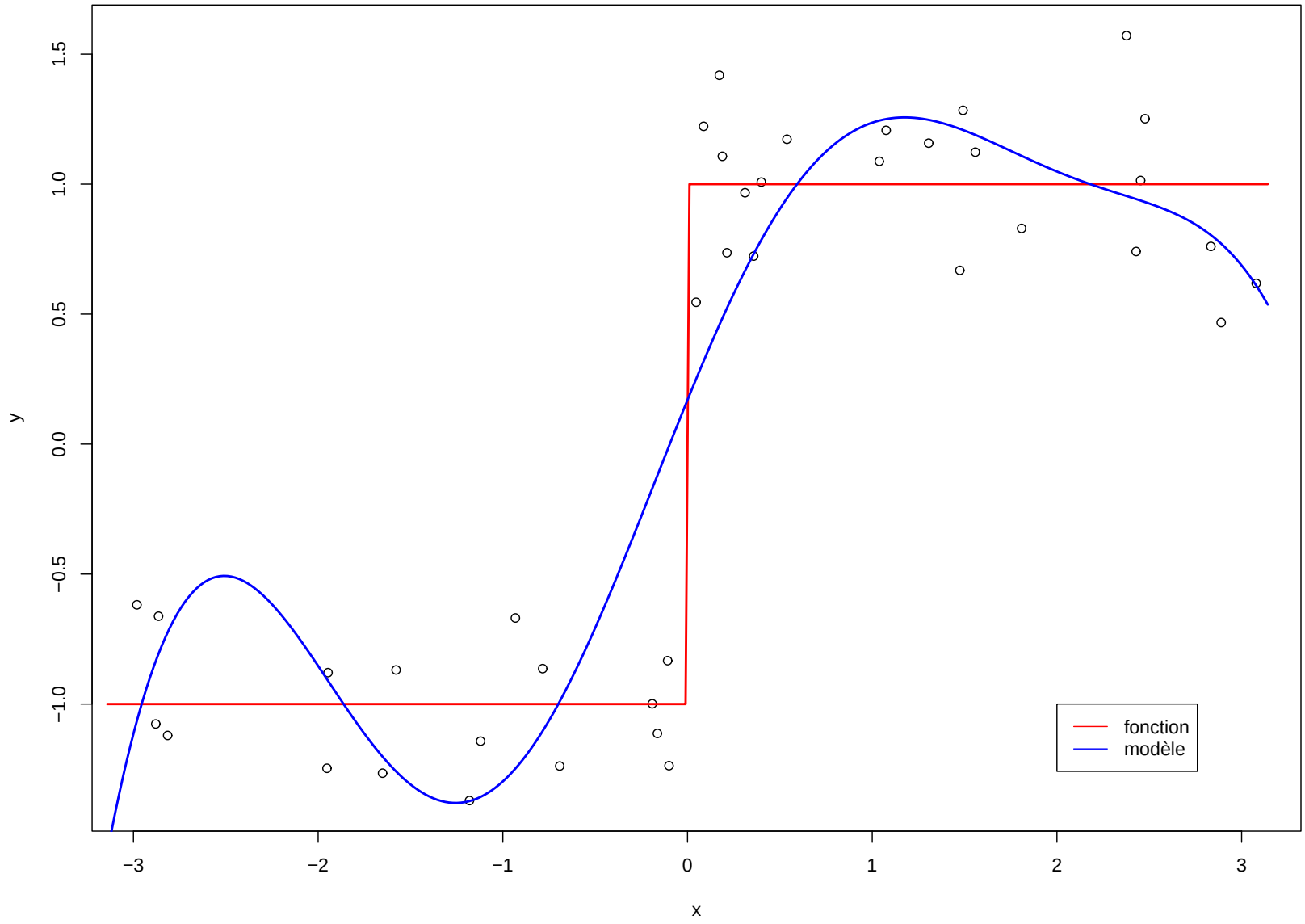
Réseau RBF à 10 neurones

Exemple : RBF



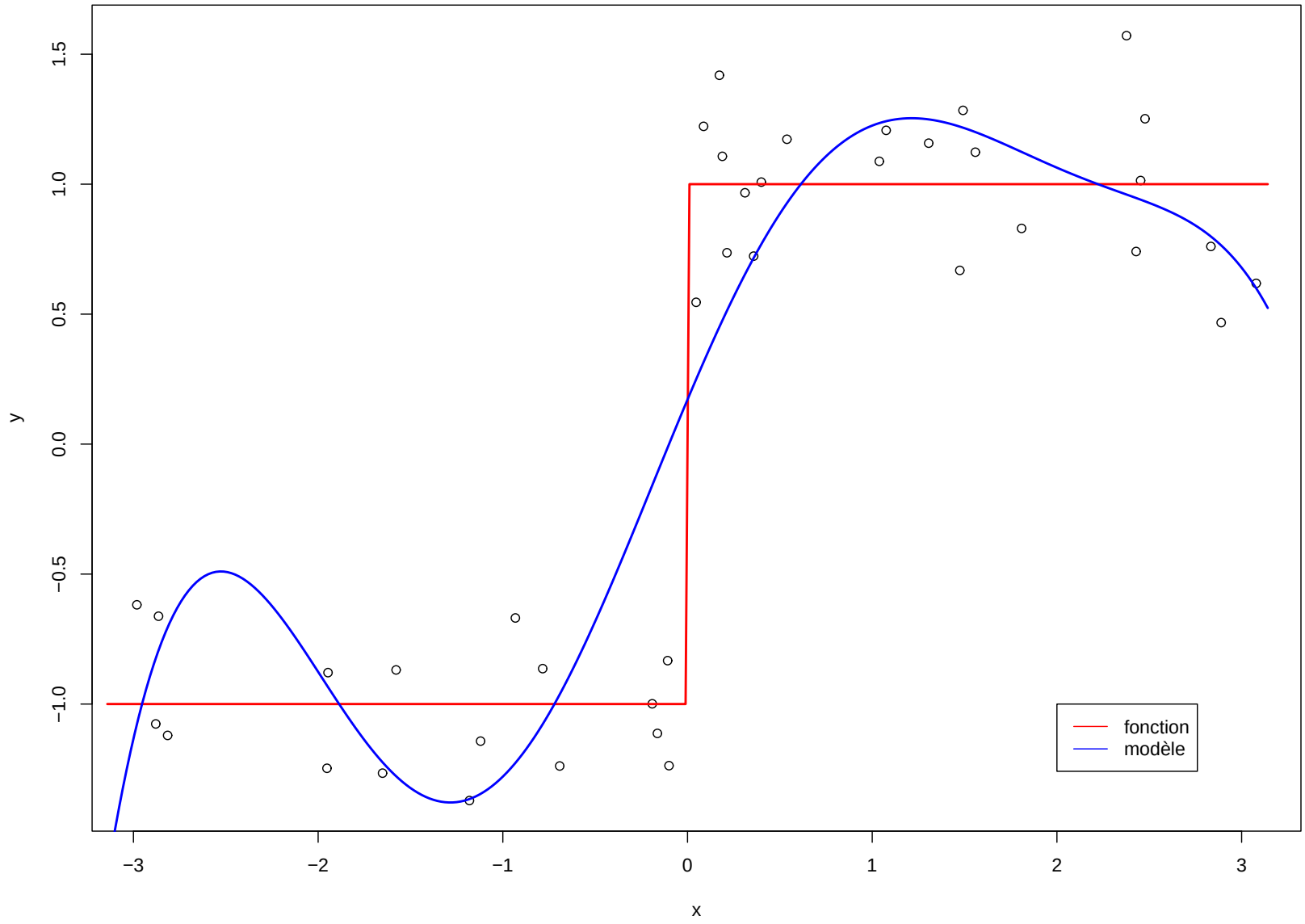
Réseau RBF à 9 neurones

Exemple : RBF



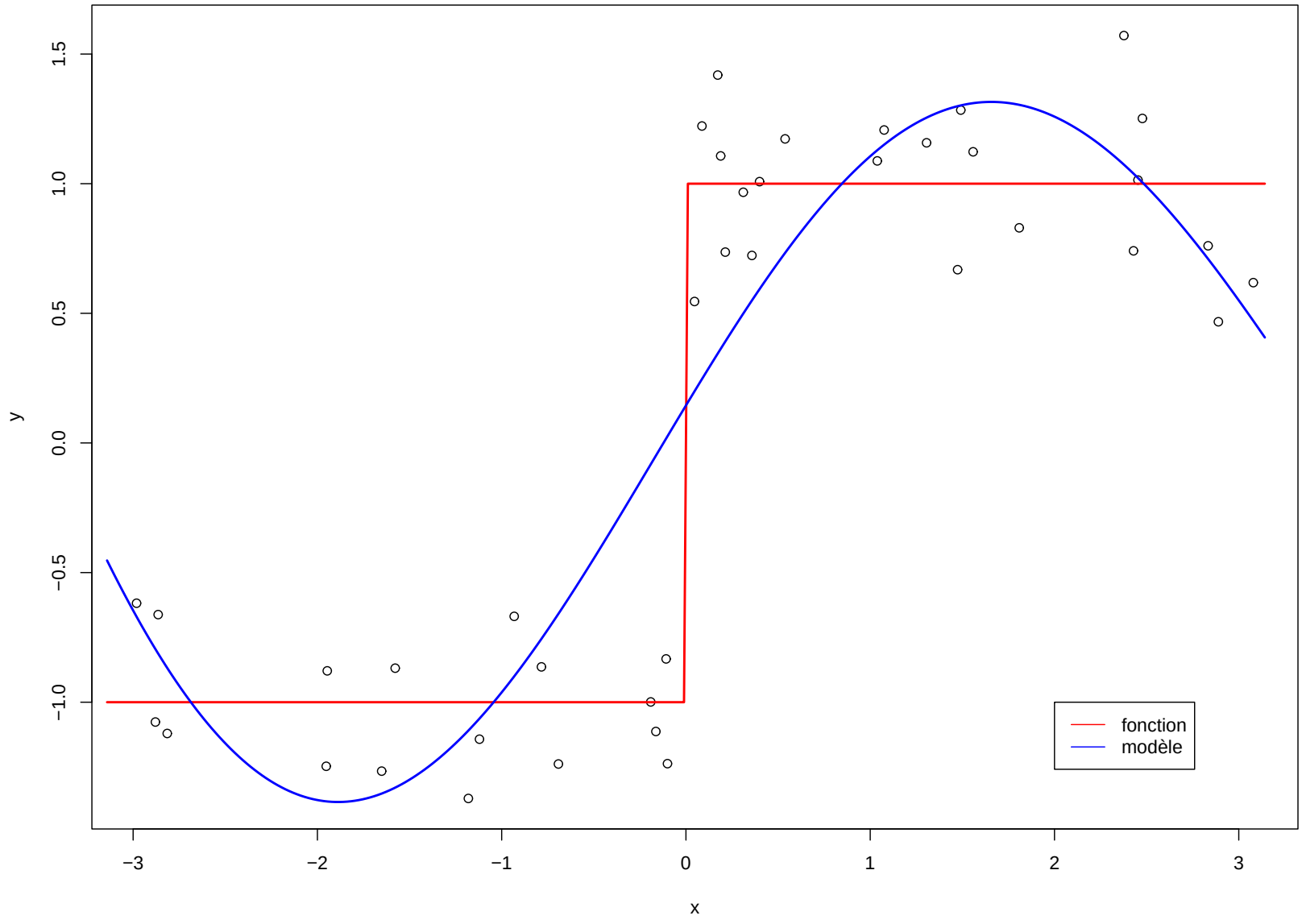
Réseau RBF à 7 neurones

Exemple : RBF



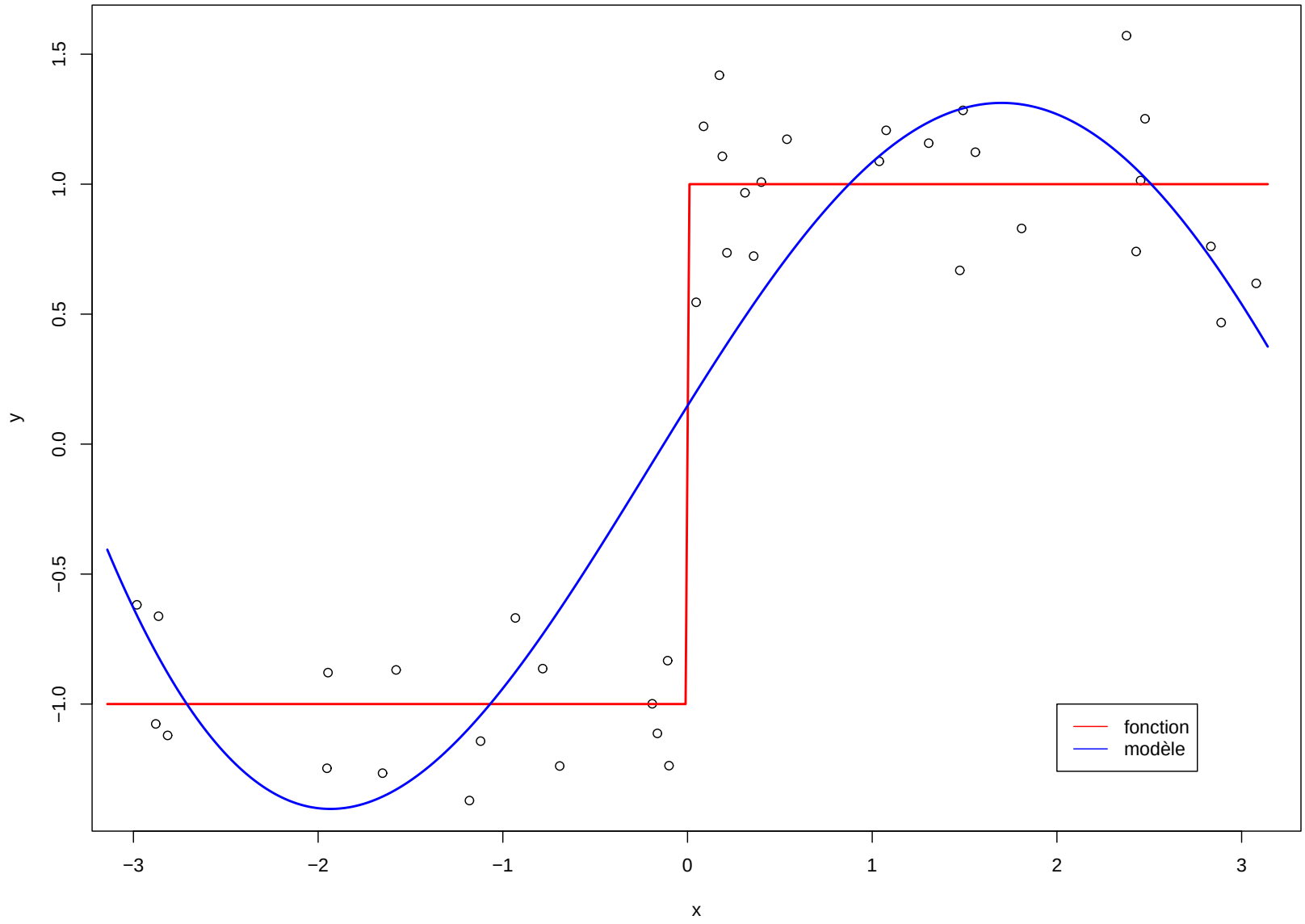
Réseau RBF à 6 neurones

Exemple : RBF



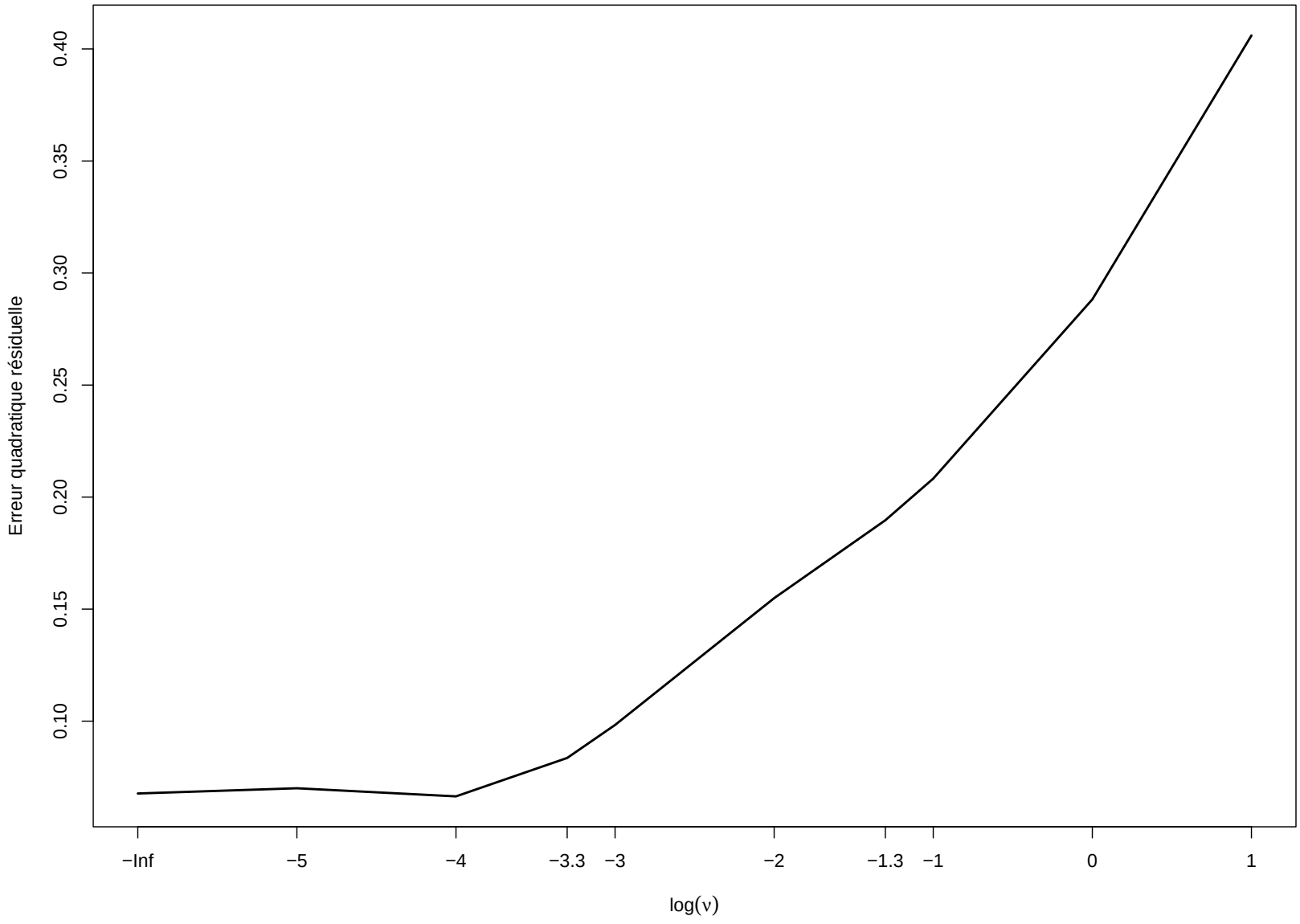
Réseau RBF à 5 neurones

Exemple : RBF



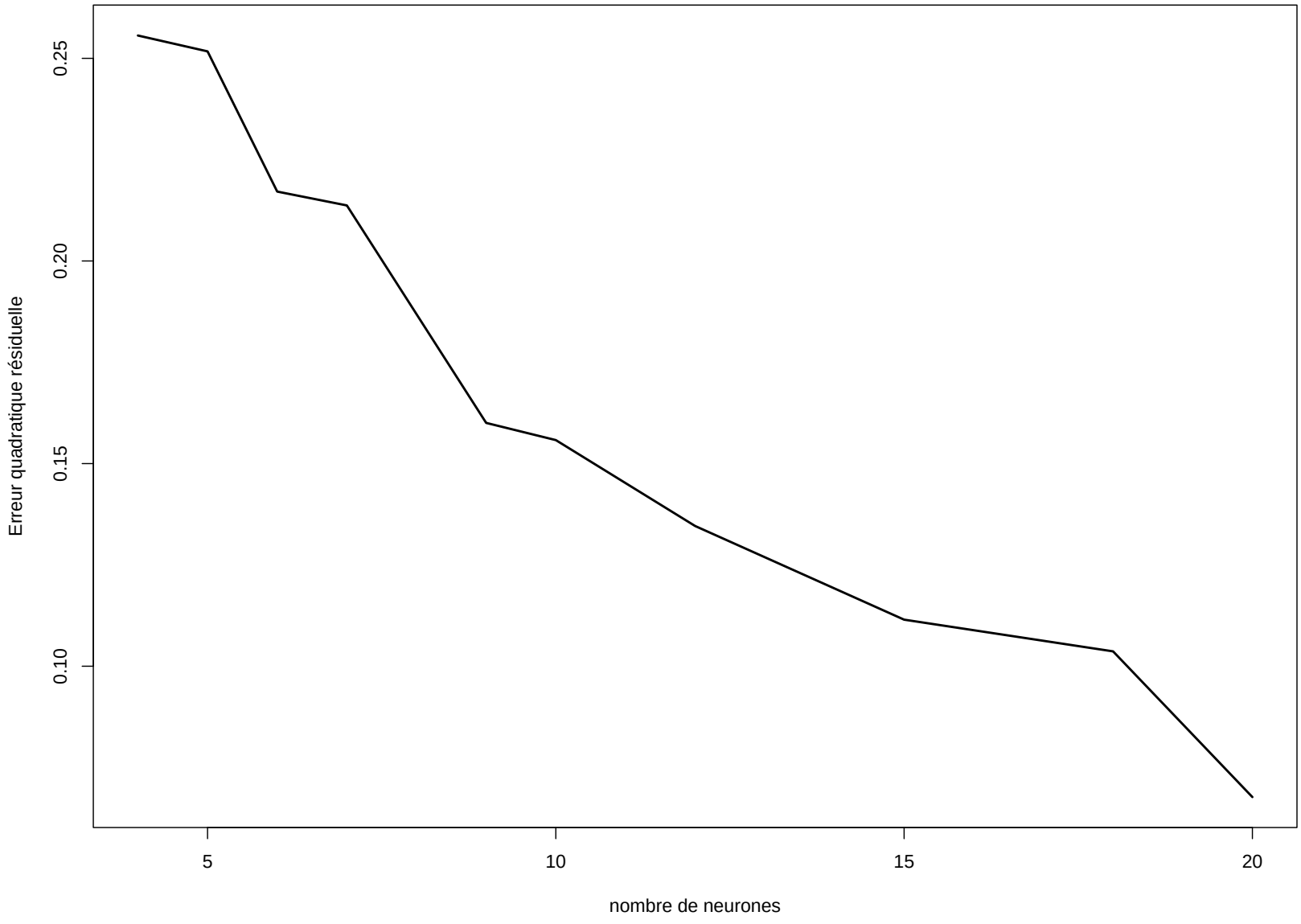
Réseau RBF à 4 neurones

Choix du modèle ?



Réseau RBF à 20 neurones régularisé

Choix du modèle ?



Réseau RBF à sans régularisation

Discrimination

Aucune différence de principe avec le modèle linéaire :

- Codage disjonctif complet
- Estimateurs des moindres carrés consistants
- Maximum de vraisemblance $\neq$ moindres carrés

Comme pour la régression, on gagne par rapport au modèle linéaire :

- modèle strictement plus puissant
- contrôle par régularisation

et on perd :

- temps de calcul plus élevé
- modèle plus puissant, donc évaluation plus difficile

Discrimination à plus de 2 classes

A partir de $k \geq 3$ classes, trois stratégies :

- méthode classique (codage disjonctif complet, $Y \in \mathbb{R}^k$)
- méthode 1 contre $k-1$:
  - on fusionne les classes $\mathcal{C}_i$ pour $i \neq k$
  - nécessite la construction de k classifieurs
- méthode 1 contre 1 :
  - on se contente de séparer $\mathcal{C}_i$ et $\mathcal{C}_j$ pour $i \neq j$
  - nécessite la construction de $\frac{k(k-1)}{2}$ classifieurs

Dans le cas pseudo-linéaire strict, les trois méthodes sont équivalentes. Par contre, si on adapte la partie non linéaire aux données, les résultats peuvent être différents.

Résumé

Modèle linéaire généralisé. Points positifs :

- Modèle strictement plus puissant que le modèle linéaire
- Estimateur des moindres carrés pour les paramètres :
  - consistant
  - équivalent au maximum de vraisemblance pour la régression avec bruit gaussien
  - calcul rapide
- Régularisation

Points négatifs :

- Temps de calcul plus élevés que pour le modèle linéaire
- Choix de la base
- Evaluation des performances