Analyse Numérique

Jacques Le Bourlot
Observatoire de Paris & Université Paris-Diderot

13 Janvier 2014

Interpolation linéaire 1D

- Cas le plus simple et le plus fréquent

Interpolation linéaire 1D

- Cas le plus simple et le plus fréquent
- On connaît $\{x_i\}$ et $\{y_i = f(x_i)\}$ pour $i \in [0, n]$

Interpolation linéaire 1D

- Cas le plus simple et le plus fréquent
- On connaît $\{x_i\}$ et $\{y_i = f(x_i)\}$ pour $i \in [0, n]$
- On cherche une valeur approchée $\bar{y}$ de $y = f(x)$ pour x quelconque.

Interpolation linéaire 1D

- Cas le plus simple et le plus fréquent
- On connaît $\{x_i\}$ et $\{y_i = f(x_i)\}$ pour $i \in [0, n]$
- On cherche une valeur approchée $\bar{y}$ de $y = f(x)$ pour x quelconque.

1. Trouver j tel que $x_j \leqslant x < x_{j+1} \Rightarrow$ Problème de tri !
2. Calculer $\bar{y}$ par :

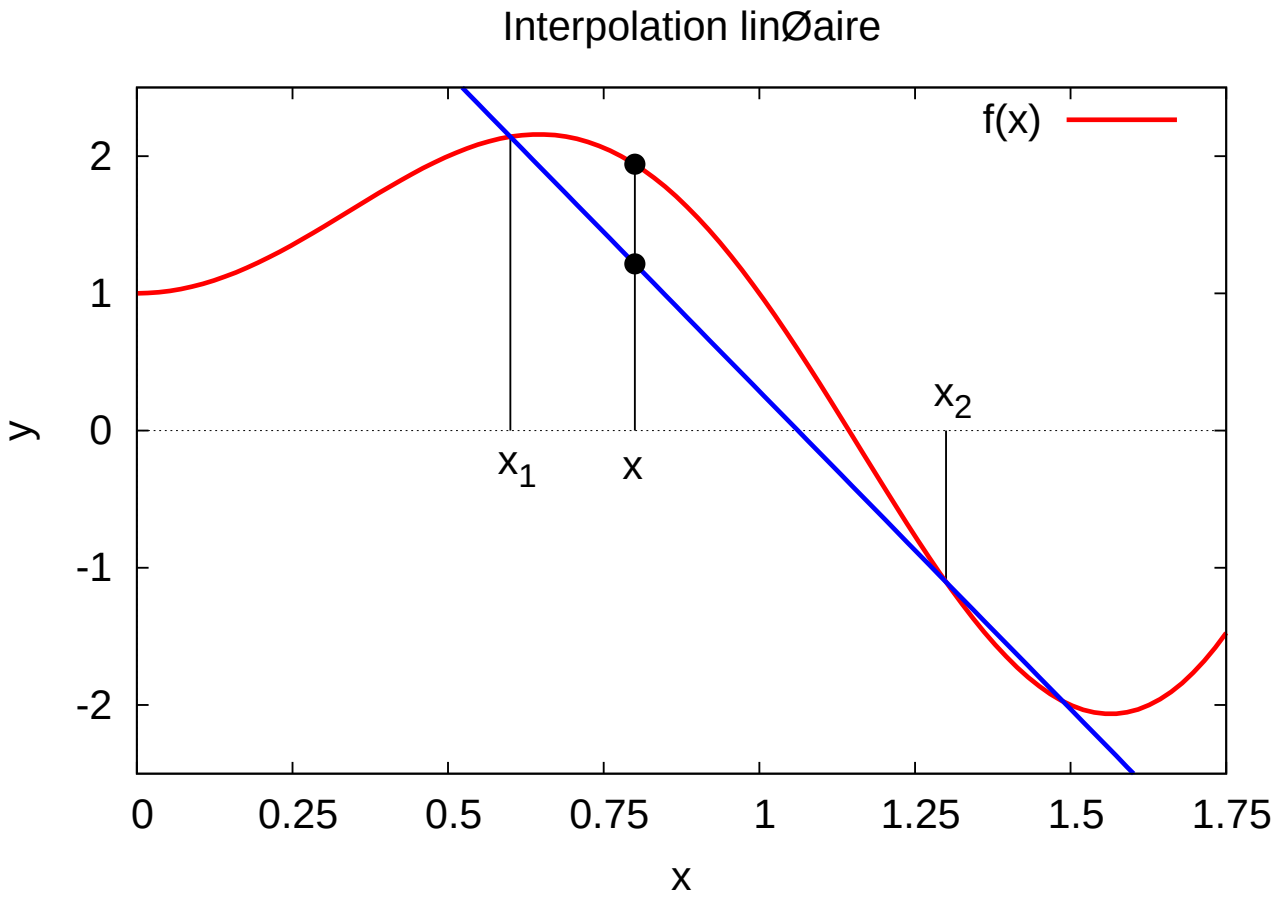
$$\bar{y} = y_j + (y_{j+1} - y_j) \times \frac{x - x_j}{x_{j+1} - x_j}$$

Note : Cette expression peut se mettre sous la forme (Lagrange) :

$$\bar{y} = y_j \frac{x - x_{j+1}}{x_j - x_{j+1}} + y_{j+1} \frac{x - x_j}{x_{j+1} - x_j}$$

Interpolation linéaire 1D

Graphiquement :



Tri

- Algorithme direct : $\propto n^2$
- Algorithme optimisé (Quicksort, etc...) : $\propto n \log n$
- Pour un tableau de 10^4 points, le rapport est :

$$\frac{n}{\log n} = \frac{10^4}{9.2} > 10^3$$

...La question ne se pose pas !

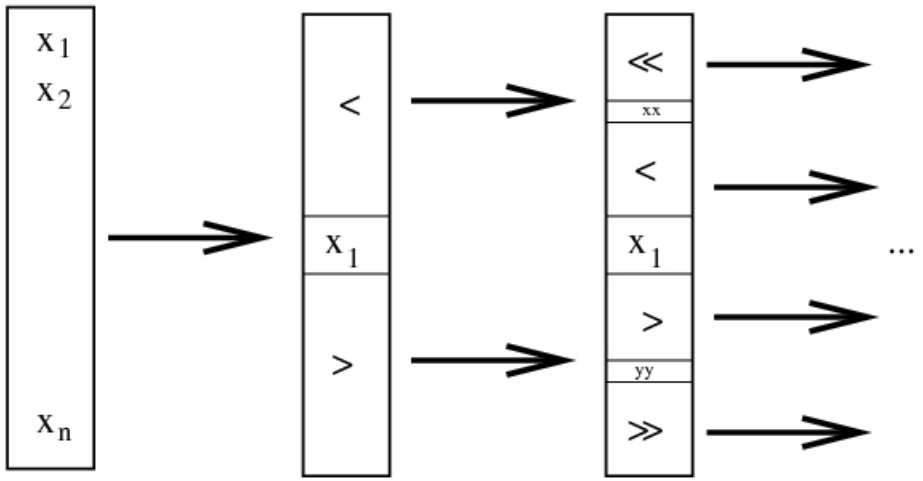
Tri

- Algorithme direct : $\propto n^2$
- Algorithme optimisé (Quicksort, etc...) : $\propto n \log n$
- Pour un tableau de 10^4 points, le rapport est :

$$\frac{n}{\log n} = \frac{10^4}{9.2} > 10^3$$

...La question ne se pose pas !

Quicksort :



Interpolation Polynomiale 1D

- Utilise plus de points de chaque coté.

Interpolation Polynomiale 1D

- Utilise plus de points de chaque coté.
- Polynomiale (interpolation de Lagrange) : Polynôme de degré n, passant par (x_i, y_i), avec $i \in [0, n]$:

$$P_n(x) = \sum_{i=0}^{n} y_k \frac{\prod_{j=0, j\neq k}^{n} (x - x_j)}{\prod_{j=0, j\neq k}^{n} (x_i - x_j)}$$

Interpolation Polynomiale 1D

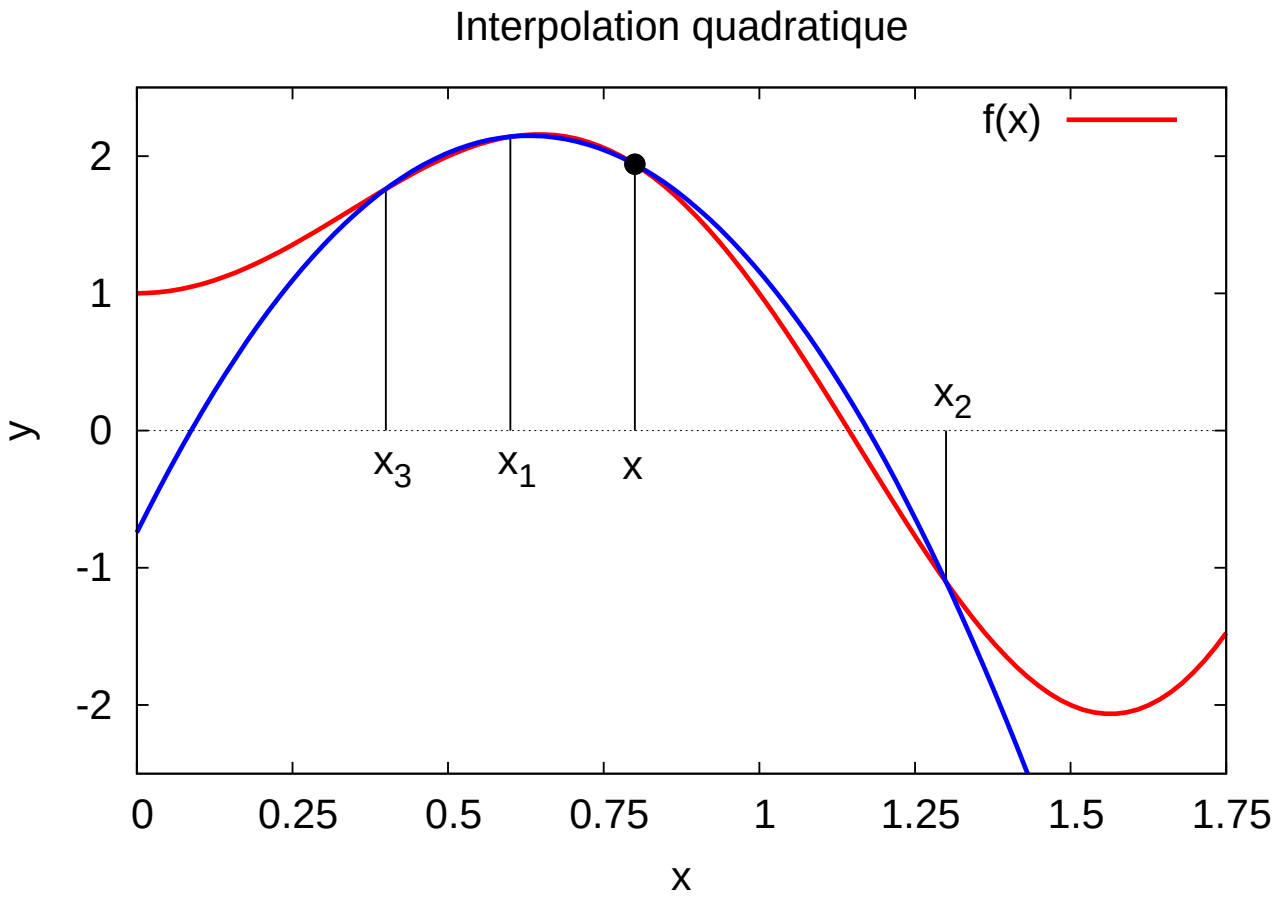
- Utilise plus de points de chaque coté.
- Polynomiale (interpolation de Lagrange) : Polynôme de degré n, passant par (x_i, y_i), avec $i \in [0, n]$:

$$P_n(x) = \sum_{i=0}^{n} y_k \frac{\prod_{j=0, j\neq k}^{n} (x - x_j)}{\prod_{j=0, j\neq k}^{n} (x_i - x_j)}$$

- Pour aller plus loin :
  - Newton : $P_{n+1}(x) = P_n(x) + a_{n+1} \prod_{i=0}^{n} (x - x_i)$
  - Différences divisées.
  - Polynômes de Chebyshev

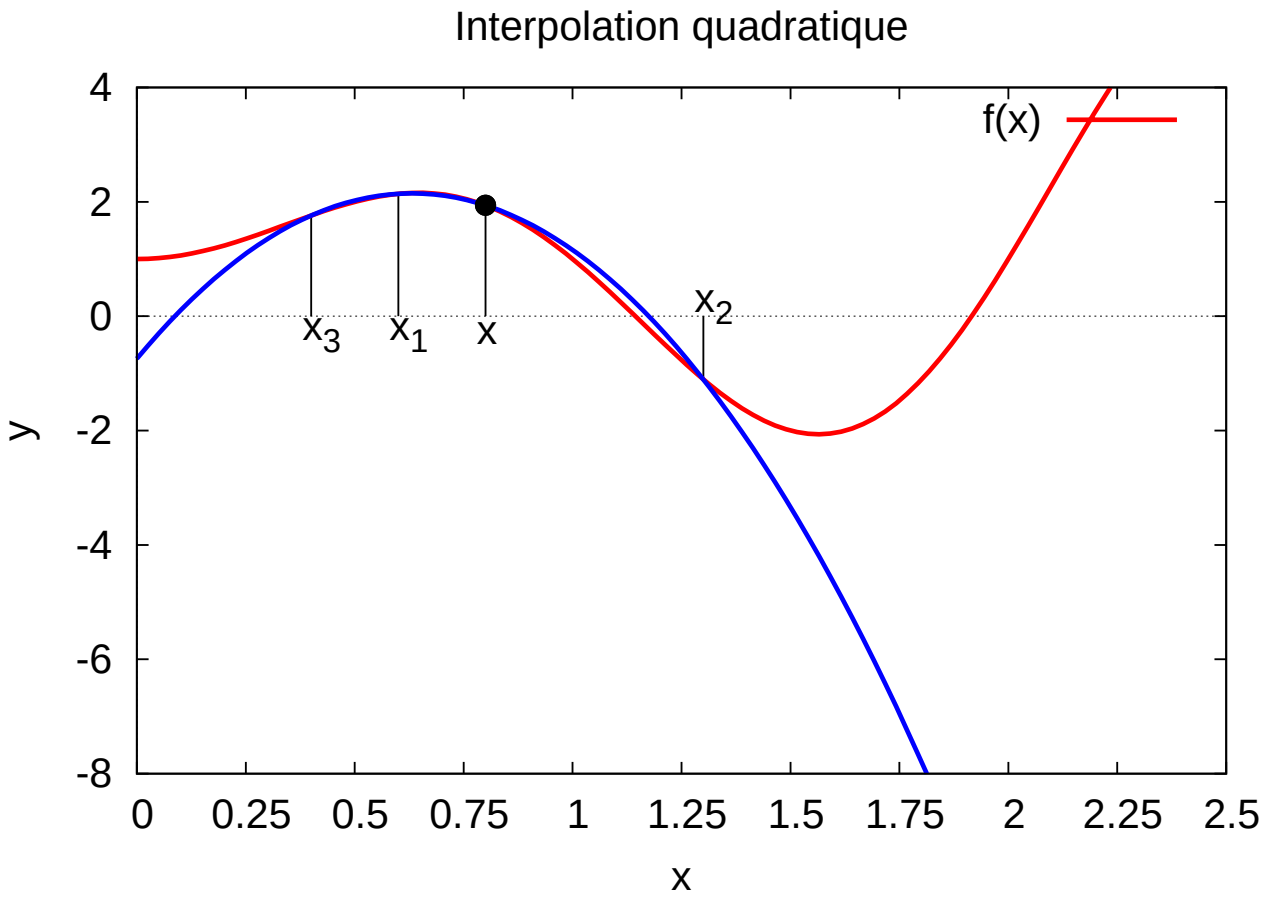
Interpolation Polynomiale 1D

Graphiquement :



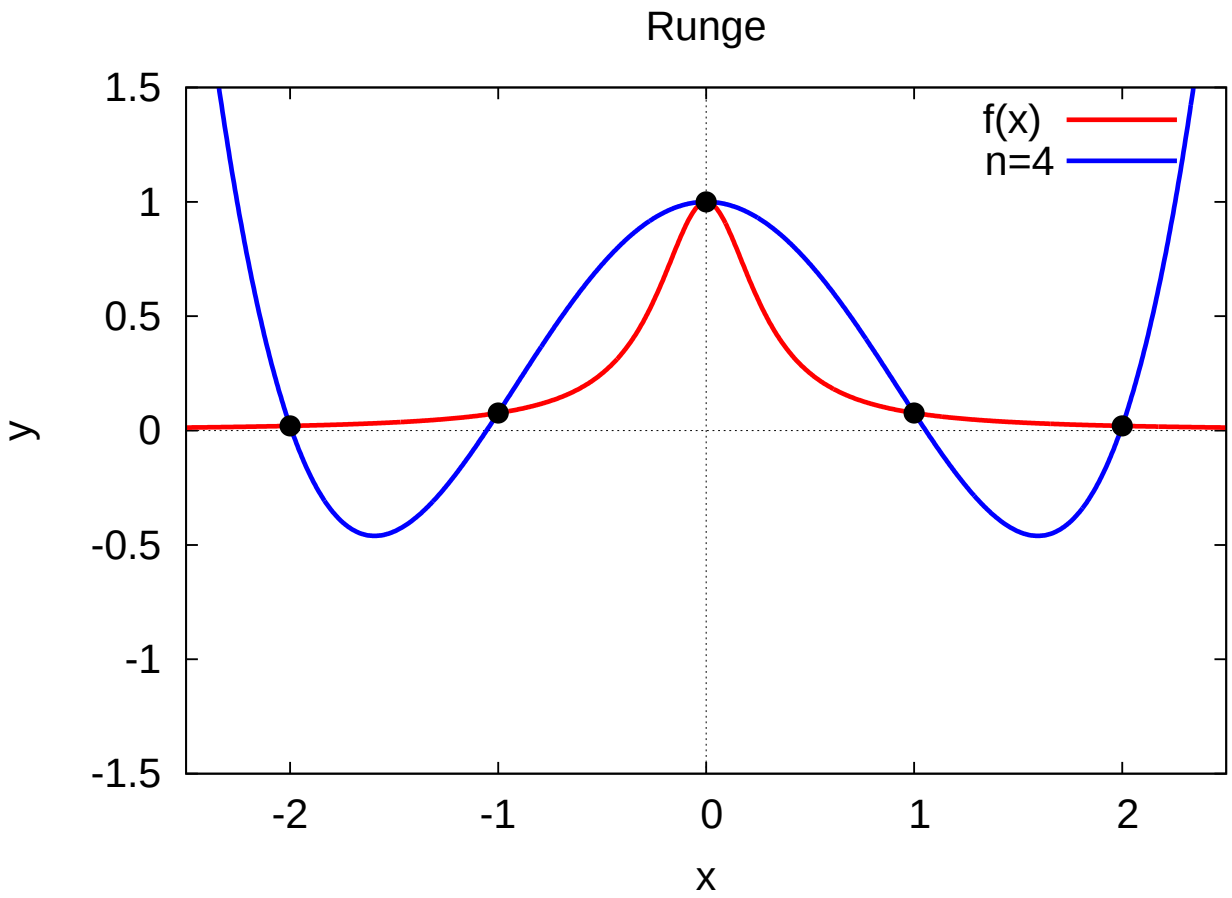
Extrapolation Polynomiale 1D

Attention : ne **JAMAIS** extrapoler avec un polynôme !



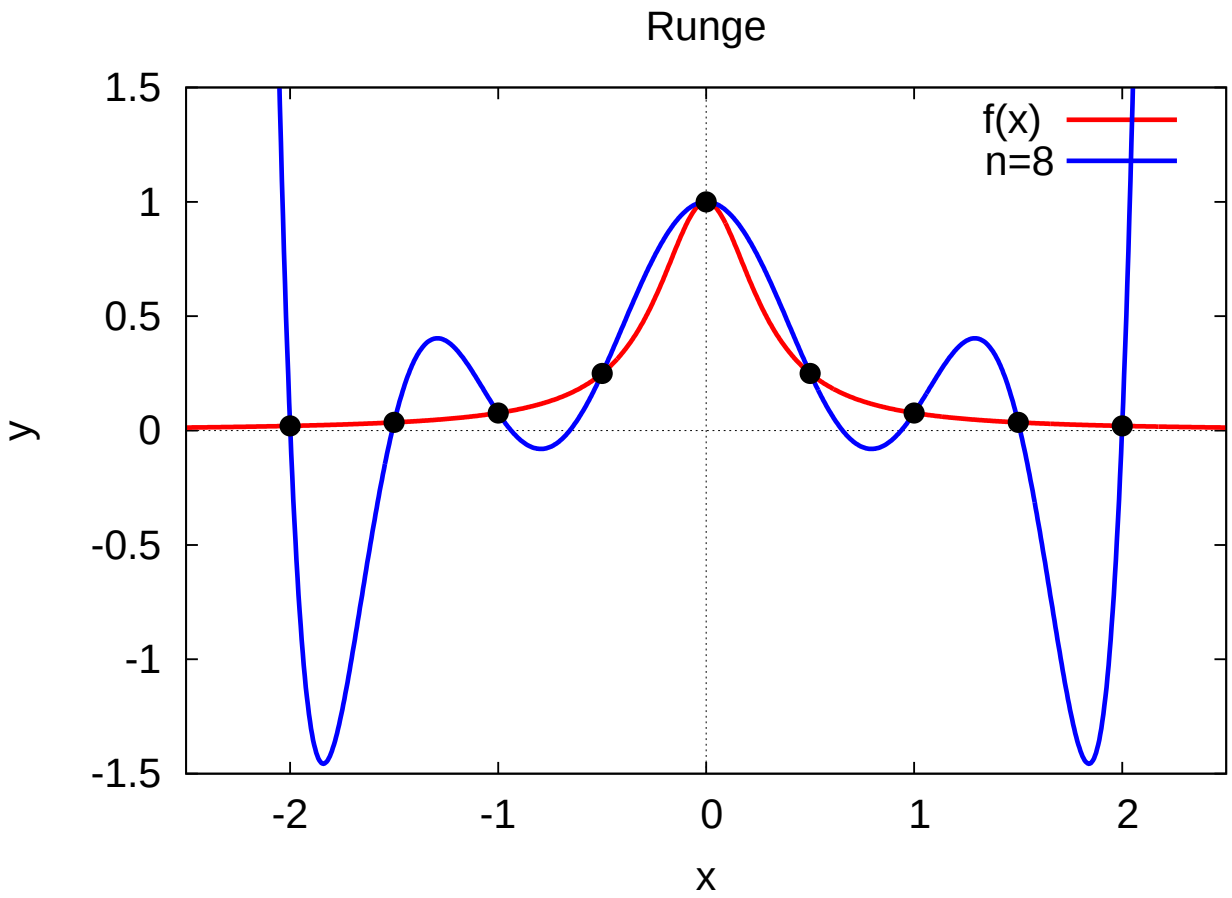
Phénomène de Runge

Les polynômes sont souvent de très mauvais choix :



Phénomène de Runge

Les polynômes sont souvent de très mauvais choix :



Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) =$
  $s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) =$
  $s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$
- $S(x_i) = y_i$ pour tout $i \in [0, n]$

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) =$
  $s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$
- $S(x_i) = y_i$ pour tout $i \in [0, n]$
- $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$
- $S(x_i) = y_i$ pour tout $i \in [0, n]$
- $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$
- $S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$
- $S(x_i) = y_i$ pour tout $i \in [0, n]$
- $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$
- $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ pour tout $i \in [0, n-2]$
- $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$ pour tout $i \in [0, n-2]$

Splines cubiques

Pour une série de points (x_i, y_i) avec $i \in [0, n]$, l'interpolation Spline $S(x)$ est formée de n polynômes $S_i(x)$ définis entre x_i et x_{i+1} tels que :

- $S(x) = S_i(x) = s_{i,0} + s_{i,1}(x - x_i) + s_{i,2}(x - x_i)^2 + s_{i,3}(x - x_i)^3$
  $x \in [x_i, x_{i+1}]$ pour tout $i \in [0, n-1]$
- $S(x_i) = y_i$ pour tout $i \in [0, n]$
- $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$
- $S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$
- $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})$ pour tout $i \in [0, n-2]$

La courbe passe par tous les points, elle est continue et ses dérivées premières et secondes aussi.
Il y a $(n+1) + 3 \times (n-1) = 4n - 2$ contraintes pour $4n$ paramètres : deux degrés de liberté (aux extrémités).

Fits

Ce problème est largement indépendant du précédent :

- Interpolation : on passe exactement par tous les points
- Fits : On ne passe **pas** par les points observés.

On cherche à trouver la forme analytique la “plus proche” en ajustant des **paramètres** libres.
En général : moindre carrés. Soit x_i les absices, y_i les valeurs observées avec une incertitude σ_i, et f une fonction analytiques, on pose :

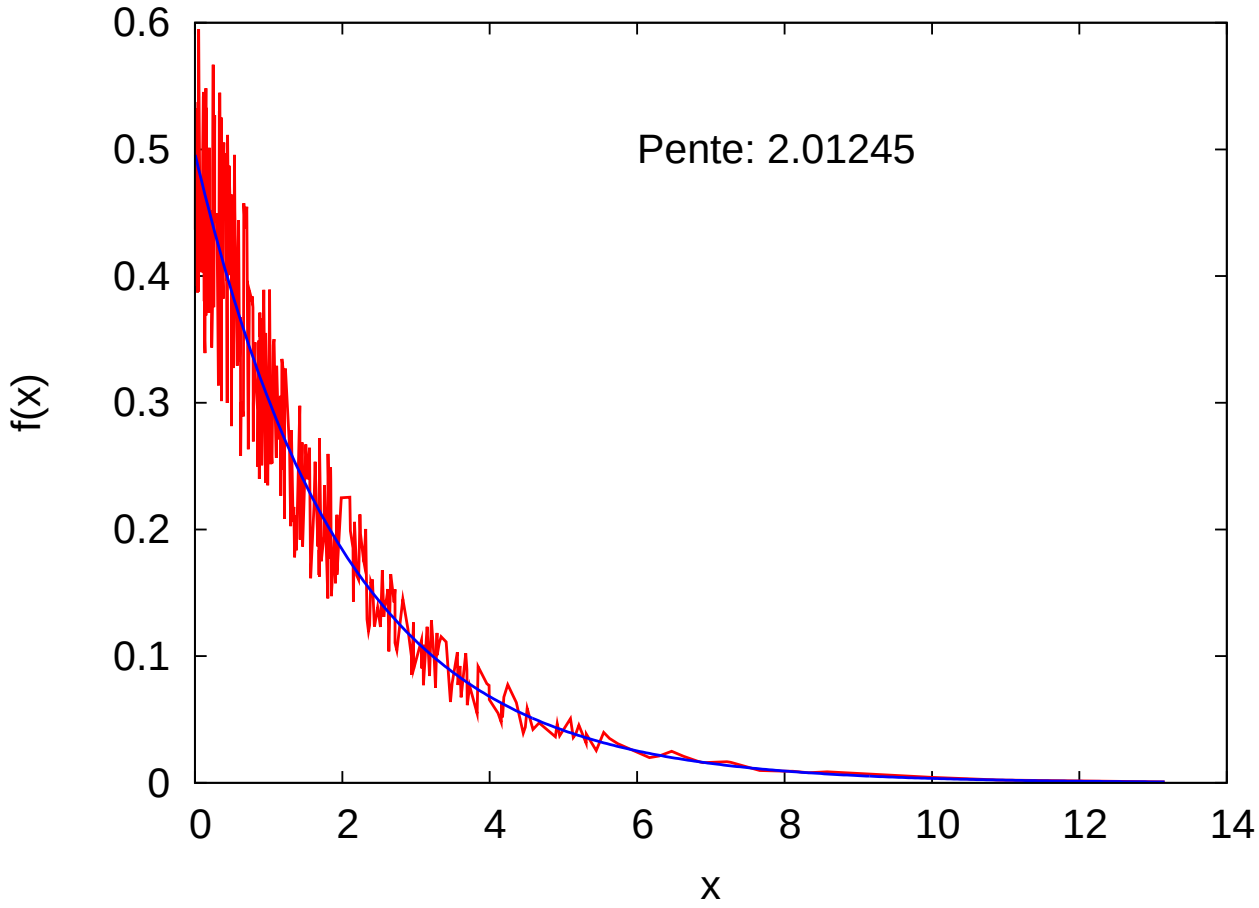
$$\chi^2 = \sum_i \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

Fits

- Si f dépend de p paramètres a_1, a_2, ... a_p, on cherche les valeurs des a_j qui minimisent χ^2.
  - On parle d'ajustement linéaire si f dépend **linéairement** des a_j. Dans ce cas, il existe une solution analytique.
  - Sinon, on utilise l'algorithme de Levenberg-Marquardt (par exemple dans gnuplot).
- Exemple :

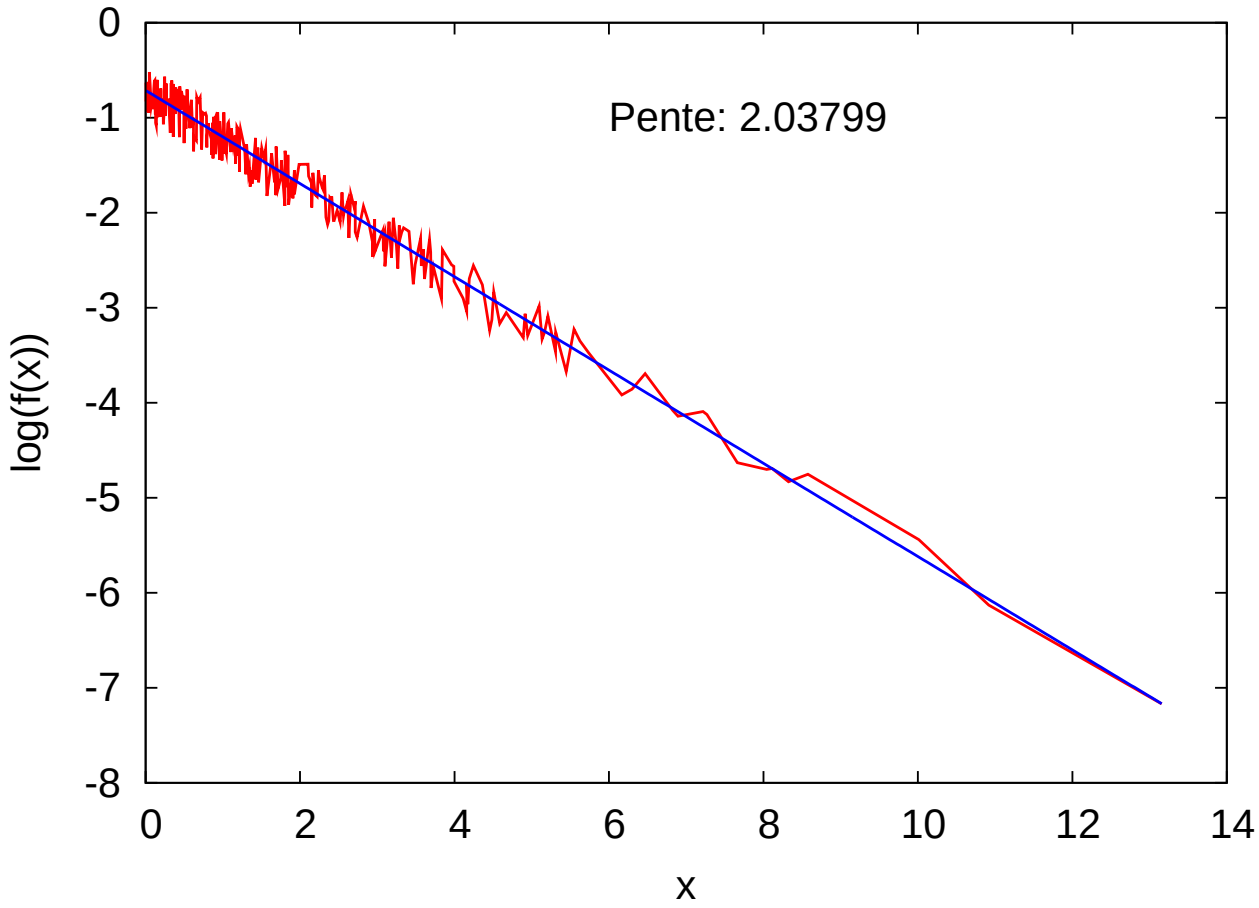
$$f(x) = \frac{1}{\lambda} \exp\left(-\frac{x}{\lambda}\right)$$

Fits




```
fit f(x) "d_fit.out" u 1:2 via a
```

Fits




```
fit log(f(x)) "d_fit.out" u 1:(log($2)) via a
```

Résolution de systèmes linéaires

Soit à résoudre :

$$A\,X = B$$

- Deux types de méthodes :

Résolution de systèmes linéaires

Soit à résoudre :

$$A\,X = B$$

- Deux types de méthodes :
  - ✦ Directes (Décomposition LU ou variantes)

Résolution de systèmes linéaires

Soit à résoudre :

$$A\,X = B$$

- Deux types de méthodes :
  - ✦ Directes (Décomposition LU ou variantes)
  - ✦ Itératives

Résolution de systèmes linéaires

Soit à résoudre :

$$A\,X = B$$

- Deux types de méthodes :
  - Directes (Décomposition LU ou variantes)
  - Itératives
- Dépend de la forme de la matrice A

Résolution de systèmes linéaires

Soit à résoudre :

$$A\,X = B$$

- Deux types de méthodes :
  - ✦ Directes (Décomposition LU ou variantes)
  - ✦ Itératives
- Dépend de la forme de la matrice A
- Difficultés à prévoir si A est “mal conditionnée”

Conditionnement d’une matrice A :

$$\kappa = \|A\| \cdot \left\|A^{-1}\right\| = \frac{\max\left(|\lambda_i|\right)}{\min\left(|\lambda_i|\right)}$$

$\{\lambda_i\}$ valeurs propres de A.

Décomposition LU

On écrit A sous la forme :

$$PA = LU$$

où P est une matrice de permutation, L est triangulaire inférieure, U triangulaire supérieure

Décomposition LU

On écrit A sous la forme :

$$PA = LU$$

où P est une matrice de permutation, L est triangulaire inférieure, U triangulaire supérieure
On pose :

$$UX = Y$$

On résout, par substitution directe $LY = PB$ puis, par substitution inverse $UX = Y$.

Décomposition LU

On écrit A sous la forme :

$$PA = LU$$

où P est une matrice de permutation, L est triangulaire inférieure, U triangulaire supérieure
On pose :

$$UX = Y$$

On résout, par substitution directe $LY = PB$ puis, par substitution inverse $UX = Y$.
La permutation P permet de garantir qu'il n'y a pas de division par 0 si $\det A \neq 0$ (pivot de Gauss).

Décomposition LU

En effet :

$$AX = B \Leftrightarrow PAX = PB \Leftrightarrow LUX = PB \Leftrightarrow LY = PB$$

Les premières lignes de cette dernière équation sont :

$$\begin{aligned} l_{11} \cdot y_1 &= b_1 \\ l_{21} \cdot y_1 + l_{22} \cdot y_2 &= b_2 \end{aligned}$$

Donc :

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j \right)$$

Puis, en partant de la dernière ligne de $UX = Y$:

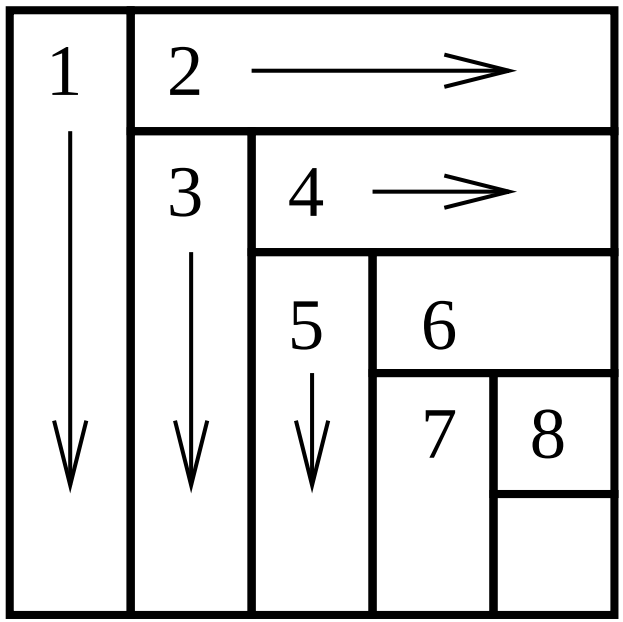
$$x_i = y_i - \sum_{j=i+1}^{n} u_{ij} \cdot x_j$$

Décomposition LU

On cherche L et U tels que :

$$a_{ij} = \sum_{k=1}^{n} l_{ik} \cdot u_{kj}$$

Avec : $l_{ik} = 0$ si $k > i$, $u_{kj} = 0$ si $k < j$ et $u_{jj} = 1$ (normalisation arbitraire). soit dans l'ordre :



Raffinement itératif

Le calcul numérique donne une solution $\bar{X}$ qui n'est pas la solution X recherchée. Mais, en repartant de la matrice A originale (avant décomposition), on a

$$A\,\bar{X} = \bar{B} \neq B$$

$$X = \bar{X} + \Delta X$$

On peut chercher à l'améliorer comme ceci :

$$A\,\left(\bar{X} + \Delta X\right) = B$$

$$A\,\Delta X = \left(B - \bar{B}\right)$$

Et on peut réutiliser la décomposition LU de A pour résoudre ce nouveau système et obtenir une correction à la solution initiale