Outils de base en analyse numérique

Sébastien Charnoz & Adrian Daerr

Université Paris 7 Denis Diderot
CEA Saclay

En analyse numérique, un certain nombre de calculs sont faits de manière répétitive.
Exemple :

Résoudre un système linéaire,
Inverser une matrice,
Calculer le déterminant, valeurs propres, vecteurs propres

Interpoler une fonction
Extrapoler une fonction
Décomposer une fonction sur une base de fonctions (ex: TF)

« fitter » des points de mesure avec des fonctions connues
Etc…

Des algorithmes performants existent souvent, mais qui ont chacun des spécificités. En fonction de nos besoins, il faudra choisir l'algorithme adapté

La plupart de ces outils « de base » sont souvent déjà programmés
dans des librairies de calculs mathématiques (ex: NAG) .

De manière générale, il n'est jamais bon de l'utiliser comme une « boîte noire » :
En fonction du problème, en choisissant l'algorithme le plus adapté, on peut
gagner du temps de calcul , ou mieux : éviter des instabilités numériques.

Exemple 1: Si vous avez besoins de calculer l'inverse d'une matrice, et que sont déterminant est très proche de 0 (mais non exactement 0), le calcul sera très sensibles aux erreurs de troncatures de la machine. Dans ce cas il faut préférer une inversion itérative, plutôt qu'une inversion exacte…

Exemple 2 : Si vous souhaitez décomposer une fonction sur une base de fonctions,
ll vaut mieux bien connaître cette base, sinon vous ne saurez pas vraiment
ce que vous faites… Souvent, une telle décomposition engendre des instabilités
dans le schémas numérique en raison de la discrétisation etc…

Dans ce chapitre, nous aborderons 3 points :

Systèmes linéaires

Calcul matriciel, inversions etc…

Interpolation et Extrapolation de fonctions

Différentes méthodes, décomposition sur une base, moindres carrés etc..

Générateurs de nombres pseudo-aléatoires

Qu’est ce qu’un bon générateur ? Comment les construire ? etc…

LES SYSTEMES LINEAIRES

En analyse numérique, résoudre un système linéaires, c'est résoudre

$$Ax = b$$

Où A est une matrice (à priori) inversible , b est un vecteur, et X est le vecteur inconnu.

Les questions attenantes à ce problème sont :

Calculer **A^{-1}, Det(A), valeurs propres de A, vecteurs propres de A**

Une dizaine de méthodes existent, mais certains outils sont commun à de nombreuses méthodes… (Pivot de Gauss, Décomposition en matrice triangulaire, Méthode de substitution)

Un tel système est facilement soluble dans le cas où la matrice **A** est triangulaire…

Pourquoi ?

Par ce qu'une simple substitution itérative permet de résoudre facilement le système
Ex :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Matrice NxN
avec N=4

On a alors :

$$x_4 = b_4 / a_{44}$$

$$x_3 = \frac{1}{a_{33}} \left(b_3 - x_4 a_{34} \right)$$

$$etc...$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

D'une manière générale, on peut trouver toutes les valeurs de X_i par la méthode itérative suivante (**substitution arrière)**

$$x_N = b_n / a_{nn}$$

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{j=i+1}^{N} a_{ij} x_j \right]$$

On calcule X_n, puis X_{n-1}, X_{n-2} … X_1

Une méthode similaire marche aussi pour un matrice triangulaire inférieure

$$\begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

$$x_1 = b_1 / a_{11}$$

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right]$$

On calcule X_1, puis X_2, X_3 … X_N

Substitution avant

Dans de nombreuses méthodes, l'objectif est de transformer **A** pour la rendre Triangulaire, inférieure ou supérieurs, et ensuite calculer les solutions en utilisant une substitution (avant ou arrière si A est inférieure ou supérieure).

2 grandes méthodes EXACTES:

ELIMINATION DE GAUSS JORDAN

Simple à comprendre, mais nécessite de modifier **b** en même temps que **A**
Ne donne pas directement $\mathbf{A}^{-1}$ et les vecteurs propres

FACTORISATION L U

Un peu plus subtile, ne modifie pas **b**, donc on peut utiliser la même décomposition pour tout vecteur **b**, donne **A-1** et les vecteurs propres

Gauss Jordan (pivot de Gauss)

Prenons l'exemple suivant :

$$\begin{cases} x + 2y + 2z = 2 & L_1 \\ x + 3y - 2z = -1 & L_2 \\ 3x + 5y + 8z = 8 & L_3 \end{cases}$$

On conserve la ligne L1, qui sert de pivot pour éliminer l'inconnue x des autres lignes; pour cela, on retire L1 à L2, et 3 fois L1 à L3. On obtient :

$$\begin{cases} x + 2y + 2z = 2 & L_1 \\ y - 4z = -3 & L_2 \leftarrow L_2 - L_1 \\ -y + 2z = 2 & L_3 \leftarrow L_3 - 3L_1 \end{cases}$$

On conserve alors la ligne L2 qui sert de pivot pour éliminer y de la troisième ligne; pour cela, on remplace la ligne L3 par L3-L2. On trouve :

$$\begin{cases} x + 2y + 2z &= 2 & L_1 \\ y - 4z &= -3 & L_2 \\ -2z &= -1 & L_3 \leftarrow L_3 + L_2 \end{cases}$$

On rèsoud finalement le système par substitution arrière
(car on résoud d'abord Z , puis Y, puis X)

Comment cela se passe avec des matrices ??

$$\begin{cases} x + 2y + 2z = 2 & L_1 \\ x + 3y - 2z = -1 & L_2 \\ 3x + 5y + 8z = 8 & L_3 \end{cases}$$

pivot

$$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & -2 \\ 3 & 5 & 8 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ 8 \end{pmatrix}$$

$$\begin{cases} x + 2y + 2z = 2 & L_1 \\ y - 4z = -3 & L_2 \leftarrow L_2 - L_1 \\ -y + 2z = 2 & L_3 \leftarrow L_3 - 3L_1 \end{cases}$$

pivot

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & -4 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ 2 \end{pmatrix}$$

$$\begin{cases} x + 2y + 2z = 2 & L_1 \\ y - 4z = -3 & L_2 \\ -2z = -1 & L_3 \leftarrow L_3 + L_2 \end{cases}$$

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & -4 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ -1 \end{pmatrix}$$

Le pivot parcours la diagonale

Pivot de Gauss
4 principes fondamentaux

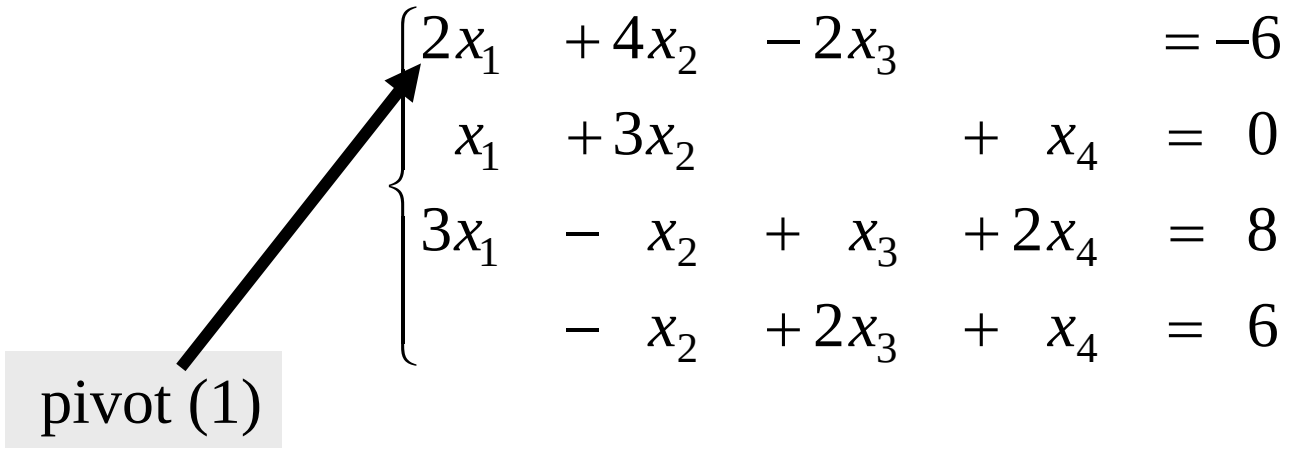
On ne change pas la solution lorsque l'on :

1. permute 2 lignes
2. permute 2 colonnes
3. divise par un même terme non nul les éléments d'une ligne
4. ajoute ou retranche à une ligne un certain nombre de fois une autre ligne

Stratégie : Transformer le système linéaire en un système équivalent … facile à résoudre

Triangulaire !

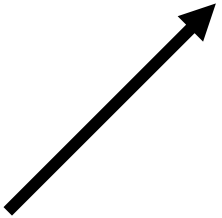
Pivot de Gauss : un autre exemple



Attention aux valeurs nulles du pivot

Pivot de Gauss : un exemple

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = -6 \\ x_1 + 3x_2 + x_4 = 0 \\ 3x_1 - x_2 + x_3 + 2x_4 = 8 \\ - x_2 + 2x_3 + x_4 = 6 \end{cases}$$



L2 = L2-L1 a_{21}/pivot (1)

Pivot de Gauss : un exemple

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = -6 \\ x_1 + 3x_2 + x_4 = 0 \\ 3x_1 - x_2 + x_3 + 2x_4 = 8 \\ -x_2 + 2x_3 + x_4 = 6 \end{cases}$$

$L2 = L2 - L1\ a_{21}/\text{pivot}\ (1)$

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = -6 \\ 0 + x_2 + x_3 + x_4 = 3 \\ 3x_1 - x_2 + x_3 + 2x_4 = 8 \\ -x_2 + 2x_3 + x_4 = 6 \end{cases}$$

Pivot de Gauss : un exemple

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = -6 \\ x_1 + 3x_2 + x_4 = 0 \\ 3x_1 - x_2 + x_3 + 2x_4 = 8 \\ -x_2 + 2x_3 + x_4 = 6 \end{cases}$$

L3 = L3-L1 a_{31}/pivot (1)

$$\begin{cases} 2x_1 + 4x_2 - 2x_3 = -6 \\ 0 + x_2 + x_3 + x_4 = 3 \\ 0 - 7x_2 + 4x_3 + 2x_4 = 17 \\ -x_2 + 2x_3 + x_4 = 6 \end{cases}$$

Le première variable à été éliminée de toutes les équations sauf une

L’algorithme du pivot de Gauss

Triangularisation

$A\,x = b$

On voit que si $a_{kk} \sim 0$, on introduit une instabilité dans le système

pour $k = 1$ jusqu'à $n-1$
    $pivot \leftarrow a_{kk}$ (* stratégie de pivot *)
    si $pivot \neq 0$ alors
        pour $i = k+1$ jusqu'à n
            $b_i \leftarrow b_i - \frac{a_{ik}}{pivot} b_k$
            pour $j = k+1$ jusqu'à n
                $a_{ij} \leftarrow a_{ij} - \frac{a_{ik}}{pivot} a_{kj}$
            fait
        fait
    sinon "problème"
fait

Représentation Matricielle du Pivot de Gauss pour une matrice NxN

$$\text{pour} \quad i = k+1,\ldots,n \quad \begin{cases} a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} & \text{pour } j = k+1,\ldots,n \\ b_i^{(k+1)} \leftarrow b_i^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)} \end{cases}$$

$$\text{matriciellement} : A^{(k+1)} = M^{(k)} A^{(k)}; \quad b^{(k+1)} = M^{(k)} b^{(k)};$$

À chaque étape a_{kk} est le pivot.

Remarques

Choix du pivot : minimiser les erreurs d'arrondis

si un pivot est nul, on permute deux lignes

si tous les pivots restant sont nuls ⇨ la matrice est singulière

(*i.e.* le système d'équations n'admet pas de solution unique)

pour minimiser les erreurs d'arrondis :

on choisi le plus grand pivot possible (en valeur absolue)

et donc on permute les lignes (voir les colonnes associées)

c'est la stratégie du pivot maximal (partiel (lignes) ou total)

déterminant d'une matrice = produit des pivots

Problème de la méthode du pivot de Gauss seule :

Nécessite de modifier **b** ,

Donc pour résoudre **Ax=b** et **Ax'=b'** il faut recommencer la procédure deux fois

Il faudrait une méthode qui ne modifie pas **b**

=> **Décomposition LU**

Décomposition LU

Supposons que nous sommes capables d'écrire la matrice **A** sous la forme d'une produit de deux matrices triangulaires L (lower) et U (upper)

L x U = A

$$\begin{pmatrix} l_{11} & & & 0 \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ & & u_{33} & u_{34} \\ 0 & & & u_{44} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Triang. Inférieure (lower)

Triang. Supérieure (upper)

Alors résoudre **Ax=b** peut aussi s'écrire

Ax=(L U) x = L (U x) = b

Que l'on peut décomposer en deux étapes, avec une variable intermédiaire : **y**

Ax=b ⇔ (LU)x=b ⇔ L (Ux) = b ⇔ L y=b avec y=Ux

1. Résoudre **L y =b**
2. Résoudre **U x =y**

Intérêt de la méthode :

Si on connaît **L** et **U** les étapes (1) et (2) se résolvent simplement
En appliquant pour (1) une substitution avant (L est **triangulaire** inf.)
En appliquant pour (2) une susbstitution arrière (U est triangulaire sup.)

De plus on peut calculer simplement l'inverse de A, on ne touche pas à b

On peut montrer qu'en temps normal il existe une infinité de décomposition LU Si A est inversible.

Cependant il n'existe qu'une seule décomposition telle que L ait pour éléments diagonaux des « 1 » uniquement:

Pour Calculer L et U facilement on se sert de **la méthode du pivot de Gauss**, en utilisant une matrice intermédiaire un peu spéciale, : la **matrice augmentée**

Exemple : Soit à Résoudre :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{22} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$ où

$$\begin{pmatrix} 1 & 2 & -1 \\ 4 & 3 & 1 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

On peut écrire ce système manière plus compacte avec la **matrice augmentée** (qui n'est plus carré…)

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{22} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \Rightarrow \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{22} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & -1 \\ 4 & 3 & 1 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & 3 & 1 & 3 \\ 2 & 2 & 3 & 5 \end{pmatrix}$$

Matrices augmentées

On peut garder la trace des différentes étapes du pivot de gauss en considérant
La matrice augmentée

1ere étape : Pivot avec a_{11} : La 2eme ligne =L2- 4/1 x L1

$$\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & 3 & 1 & 3 \\ 2 & 2 & 3 & 5 \end{pmatrix} => \begin{pmatrix} 1 & 2 & -1 & 2 \\ 4-4 & 3-8 & 1+4 & 3-8 \\ 2 & 2 & 3 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 & 2 \\ 0 & -5 & 5 & -5 \\ 2 & 2 & 3 & 5 \end{pmatrix}$$

Nouvelle matrice A — Nouveau vecteur b

Idée géniale : Au lieu de garder le 0 dans la nouvelle matrice augmentée
On conserve dans cette case le coefficient par lequel on a multiplié L1

On a changé le 0
En 4

$$\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & -5 & 5 & -5 \\ 2 & 2 & 3 & 5 \end{pmatrix}$$

Augmentée : $\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & -5 & 5 & -5 \\ 2 & 2 & 3 & 5 \end{pmatrix}$

Maintenant on echange L3=L3-2/1 L1

Nouvelle augmentée $\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & -5 & 5 & -5 \\ 2 & -2 & 5 & 1 \end{pmatrix}$

Au lieu du 0 ici, on a mis le « 2 » par lequel on avait multiplié L1

On pivote maintenant sur la ligne 2, le pivot vaut « -5 » (a_{22})

L2 ne change pas, et L3=L3 - (-2/-5) L3

Au lieu des 0 on a conservé les coefs. multiplicateurs

$\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & -5 & 5 & -5 \\ 2 & 2/5 & 3 & 3 \end{pmatrix}$

$$\begin{pmatrix} 1 & 2 & -1 & 2 \\ 4 & -5 & 5 & -5 \\ 2 & 2/5 & 3 & 3 \end{pmatrix}$$

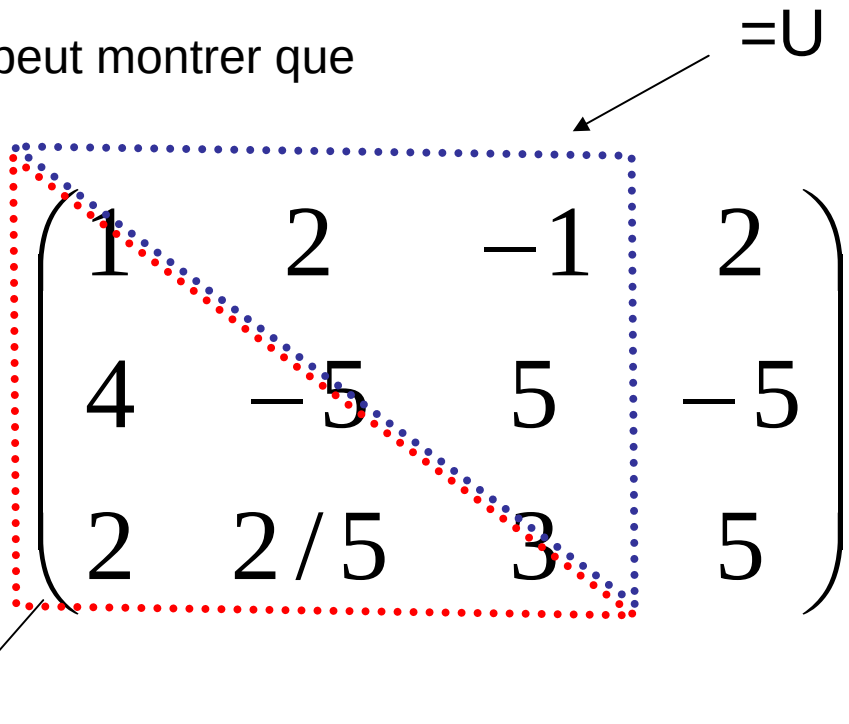
Augmentée finale.
On en déduite la forme triangulaire de A et la nouvelle forme de b

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -5 & 5 \\ 0 & 0 & 3 \end{pmatrix} \quad \text{et b} = \begin{pmatrix} 2 \\ -5 \\ 3 \end{pmatrix}$$

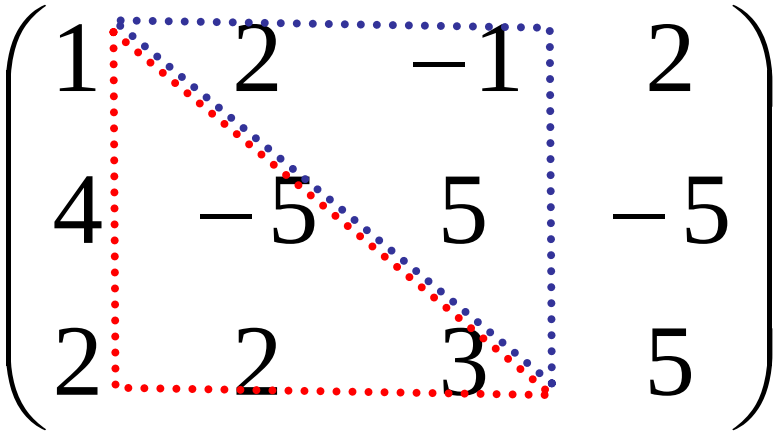
Et on trouve X par substitution classiquement

Alors, pourquoi tout cela, le rapport avec L et U?

Et bien on peut montrer que



Plus précisement, on divise chaque colonne par l'élément diagonal pour mettre des 1 sur la diagonale de L



$$L = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 2 & -2/25 & 1 \end{pmatrix} \quad \text{et} \quad U = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -5 & 5 \\ 0 & 0 & 3 \end{pmatrix}$$

Vous pourrez vérifier que **LxU=A** !!

ALGORITHME DE CROUT ($r_{ij}=u_{ij}$ et $l_{ij}=l_{ij}$ avec nos notations)

```
pour j de 1 à n faire
   pour i de 1 à j faire          // Calcul des r_{i,j}   Matrice U
      r_{i,j} ← a_{i,j}
      pour k de 1 à i − 1 faire
         r_{i,j} ← r_{i,j} − l_{i,k} r_{k,j}
      fin pour
   fin pour      pour i de j + 1 à n faire          // Calcul des l_{i,j}
      l_{i,j} ← a_{i,j}
      pour k de 1 à j − 1 faire
         l_{i,j} ← l_{i,j} − l_{i,k} r_{k,j}
      fin pour
         l_{i,j} ← l_{i,j} / r_{j,j}
   fin pour
   fin pour
```

Matrice L

Une fois qu'on possède la décomposition LU, on obtient aisément :

Le déterminant :

$Det(A)=Det(L)*Det(U) = u_{11} x\ u_{22} x \ldots x\ U_{nn}$

La matrice inverse :

On résoud successivement pour toute dimension

A $x_j = e_j$ où e_j est le vecteur unitaire de la dimension j

=> En combinant tous les x_j on trouve successivement toutes les colonnes de la matrice A^{-1}

$$Ax_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad Ax_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad Ax_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \Rightarrow A^{-1} = \begin{pmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_3 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

Avantage de la méthode L-U : On ne calcule que L et U qu'une seule fois

METHODES ITERATIVES (Relaxation)

Pour les très grosses matrices on préférera les méthodes itératives :
Elle convergent vers la solution progressivement:

Idée : Résoudre une équation du type : F(x)=X
Avec une suite du type : $X^{k+1}=F(X^k)$
Si X^0 n'est pas trop loin de la solution, on X^k convergera vers la bonne valeur de X.

Comment transformer : **AX=B** en un équation du type **X=F(X) ?** Où F est une opération linéaire ??

Idée : Décomposer **A = M-N**

Alors **AX=B** ⇔ **(M-N)X = B** ⇔ MX=NX+B ⇔

$$\mathbf{X = M^{-1} N X + M^{-1} B = P X + C}$$

$$\underbrace{\mathbf{M^{-1} N X + M^{-1} B}}_{\mathbf{F(x)}}$$

Donc si on arrive à trouver une décomposition de **A** avec une matrice M facilement inversible, alors on peut mettre en marche un algorithme itératif du type :

$$\mathbf{X_{k+1} = P X_k + C}$$

Décomposition E, D, F : idée : prendre pour M une simple matrice diagonale D

$$E = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ & \ddots & & & \vdots \\ & & \ddots & & \vdots \\ & -a_{i,j} & & \ddots & \vdots \\ & & & & 0 \end{pmatrix} \qquad D = \begin{pmatrix} & 0 & \cdots & \cdots & 0 \\ 0 & & \ddots & & \vdots \\ \vdots & \ddots & a_{i,i} & \ddots & \vdots \\ \vdots & & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & \end{pmatrix}$$

$$F = \begin{pmatrix} & & & 0 \\ & -a_{i,j} & & \ddots & \vdots \\ & & \ddots & & \vdots \\ & \ddots & & & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

Donc A = D-(E+F) ⇔ M=D et N=E+F

Dans ce cas $M^{-1}= D^{-1}$

$$M^{-1} = \begin{pmatrix} 1/a_{1,1} & & 0 \\ & \ddots & \\ 0 & & 1/a_{n,n} \end{pmatrix}.$$

La formule de récurrence : $\mathbf{X_{k+1}=M^{-1}\,N\,X_k + M^{-1}\,B}$

$$x_i = \frac{1}{-}$$

S'écrit :

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \cdot \left(b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)} \right), \qquad 1 \leq i \leq n$$

Où les x_i^k sont les composantes du vecteur $\mathbf{X^k}$

C'est la méthode de Jacobi

Méthode de Gauss Seidel :

Converge plus rapidement .

On s'inspire du calcul de Jacobi

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \cdot \left(b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)} \right), \qquad 1 \leq i \leq n$$

Mais on fait la première somme sur les coeffs déjà calculés de l'étape k+1

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \cdot \left(b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k)} \right), 1 \leq i \leq n.$$

Ca converge plus vite, mais difficilement parralélisable …

A doit être à « dominante diagonale » i.e ABS(diagonale)soit être > somme des v.abs des élements sur la ligne pour être *sûr* de la convergence….

Nous avons vu 2 familles de méthodes :

-Les méthodes exactes, type pivot de Gauss

Précises, peu rapides, sensibles aux fluctuations

- Les méthodes itératives (ou dites de « relaxation »)

Moins précises, plus rapides, plus stables…
Mais il est parfois difficile de prédire si elles convergent
Plus on fait d'itérations, plus la méthode est précise..
La matrice doit être à diagonale dominante…

Interpolations et Extrapolations de fonctions

Le problème :

On a un ensemble de n points : x_0 …. X_{n-1} et pour chaque point on connaît la valeur d'une fonction f inconnue (à-priori): $f(x_0)$ …. $f(x_{n-1})$
(rem :les $f(x_i)$ seront aussi notés f_i)

Question : Quelle est la valeur de f sur les points intermédiaire ?

*** Pour cela on doit *supposer* un modèle mathématique de f (polynome, somme de sinus etc…)**

* On doit aussi savoir :

1) Les $f(x_i)$ sont-elles des valeurs **exactes**
2) Les $f(x_i)$ sont-elles des valeurs **approchées (ex : pts de mesure)** ?

DONC : il n'y a pas de manière « idéale » d'interpoler/extrapoler une fonction car on a besoins **d'hypothèses** sur **f**.

$\Rightarrow$Différentes **hypothèses** donneront des interpolations/extrapolations **différentes** **…** eh oui…

Donc ce sera à nous de savoir, en fonctions de la physique du problème, quelle est la meilleur hypothèse sur la forme de f à effectuer.

•Si les $f(x_i)$ sont des valeurs **exactes** on fera une décomposition sur une base de fonctions (des polynômes le plus souvent)

•Si les f(xi) sont des valeurs **approchées** on essaiera d'adapter (de **fitter** en anglais) un modèle mathématique aux points de de mesure.

Les $f(x_i)$ sont des valeurs exactes : Décomposition sur une base de fonctions

On suppose que f est la combinaison linéaire d'une famille de fonctions φ

$$f(x) = \sum_i a_i \varphi_i(x)$$

f est toujours définie sur un domaine : $[x_{min}, x_{max}]$

1er question : que choisir comme base φ ?

2ème question : Comment calculer les a_i ?

Remarque : parfois on a pas besoin de connaître les a_i explicitement, Certains algorithmes fournissent les a_i , d'autres donnent directement le résultat $f(x_i)$…

1. **Décomposition polynomiale**

Supposons que f est polynomiale.
Avec N points on peut construire un polynôme d'ordre N-1 qui traverser les N points :

On suppose : $$f(x) = \sum_{i=0}^{N-1} a_i x^i$$

Pour calculer les coefficients a_i

On a n équations à n inconnues :

Les inconnues sont les coefficients a_i
Et les n équations sont : $f(x_i)=f_i$
où les f_i sont les valeurs de f aux points x_i .

=> Cela peut s'écrire matriciellement

$$\left.\begin{array}{c} a_0 + a_1 x_0^1 + ... + a_{n-1} x_0^{n-1} = f_0 \\ ... \\ a_0 + a_1 x_{n-1}^1 + ... + a_{n-1} x_{n-1}^{n-1} = f_{n-1} \end{array}\right\} \Leftrightarrow$$

N équations linéaires
à N inconnues.

Les ai sont les inconnues

Les x_i^j sont les coefficients

$$\begin{pmatrix} 1 & x_0^1 & ... & x_0^{n-1} \\ 1 & x_1^1 & ... & x_1^{n-1} \\ ... & ... & ... & ... \\ 1 & x_{n-1}^1 & ... & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ ... \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ ... \\ f_{n-1} \end{pmatrix}$$

M x A = f

Pour trouver **A** il suffit de calculer : $\mathbf{A = M^{-1} f}$

=> Il faut donc inverser une matrice de N points … cette méthode
est intéressante, marche mais elle est coûteuse temps de calcul
Car souvent on a **pas besoin connaître** les $\mathbf{a_i}$ **…** Seul f(x) nous interesse…

2. Décomposition en polynômes de Lagrange

Pour calculer une décomposition polynomiale sans calculer les $\mathbf{a}_i$ on peut utiliser les polynômes de **Lagrange** :

$$f(x) = \sum_{i=0}^{N-1} a_i l_i(x)$$

a_i : coefficients
$a_i = f(x_i)$

i[em] polynôme de Lagrange

$$l_i(x) = \prod_{j=0, j\neq i}^{n-1} \frac{(x - x_j)}{(x_i - x_j)}$$

Les $L_i(x)$ sont des polynômes d'ordre n-1 : ~ produits de $(x-x_j)$ pour j différent de i

Exemple :
Xi= [0 , 2 , 4, 6] et F_i=[0, 4, 0, 4]

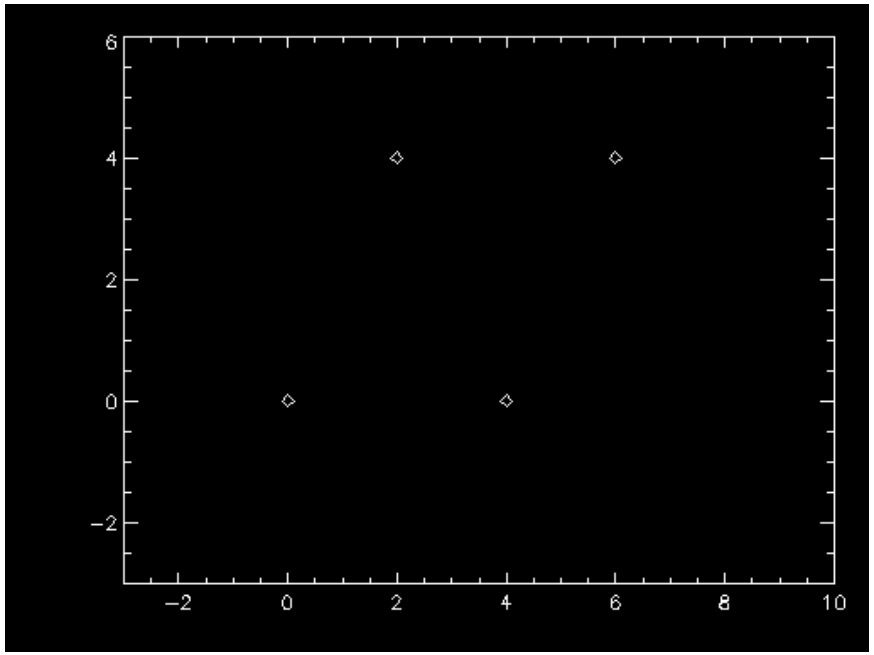
Calculer les polynomes de Lagrange ….

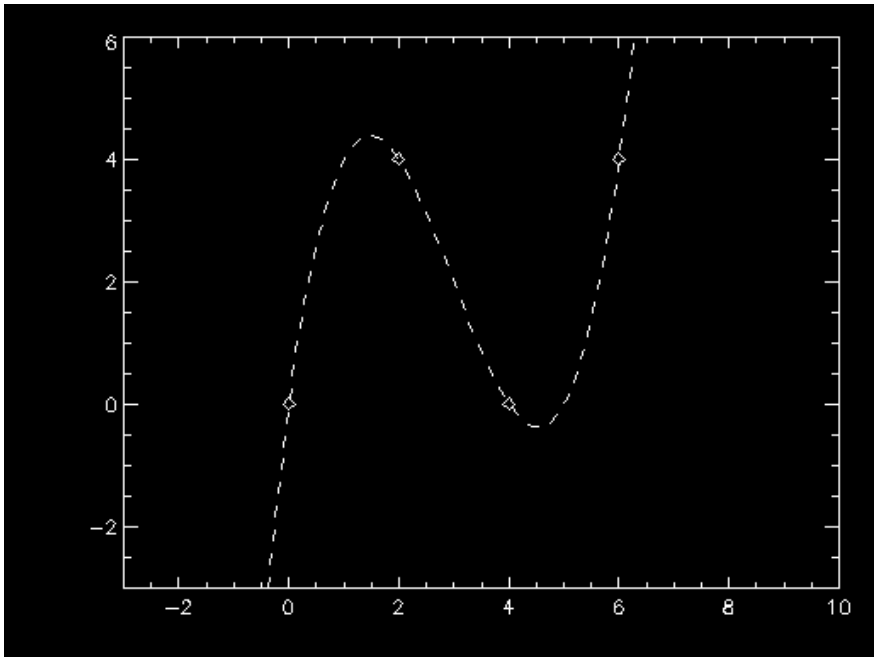
On considère les points $(0,0)$, $(2,4)$, $(4,0)$ et $(6,4)$.

$$l_0(x) = \frac{(x-2)(x-4)(x-6)}{(0-2)(0-4)(0-6)}, \quad l_1(x) = \frac{(x-0)(x-4)(x-6)}{(2-0)(2-4)(2-6)},$$

$$l_2(x) = \frac{(x-0)(x-2)(x-6)}{(4-0)(4-2)(4-6)}, \quad l_3(x) = \frac{(x-0)(x-2)(x-4)}{(6-0)(6-2)(6-4)}.$$

$$\begin{aligned} p(x) &= 0 \times l_0(x) + 4 \times l_1(x) + 0 \times l_2(x) + 4 \times l_3(x) \\ &= \frac{1}{3}x(x-4)(x-5). \end{aligned}$$

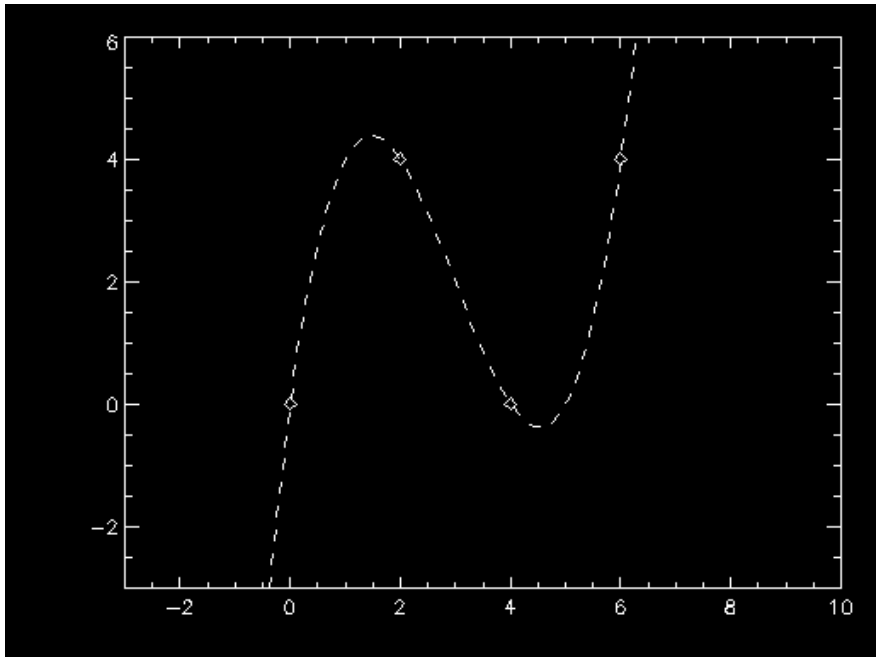




On a donc un polynôme d’ordre 3….

Les polynomes de Lagrange sont une méthode rapide de calculer des points sur la courbes… sans avoir à calculer explicitement les coefficients…

Si on a peu de points à calculer c’est interessant…
S’il y en a beaucoup, il est plus intéressant de calculer les coefficients (moins de calculs à effectuer).



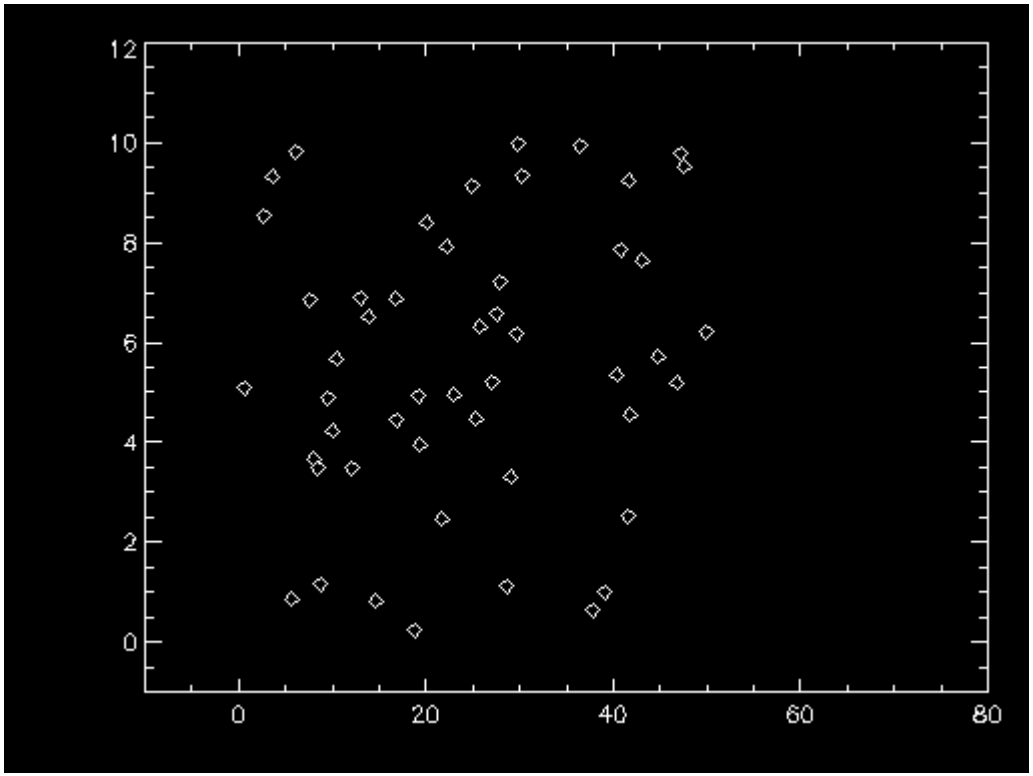
On voit que les méthodes polynomiales divergent toujours à l'infini :

⇒Pas un problème pour l'interpolation si N est faible

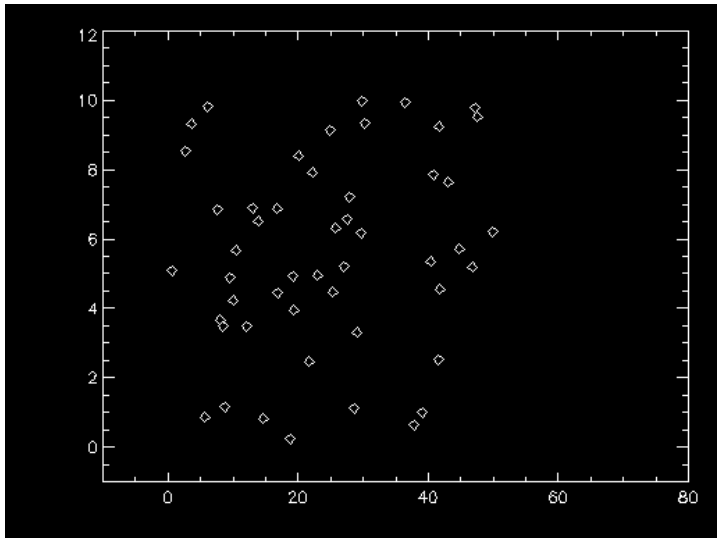
⇒ Peut être un problème pour l'extrapolation…

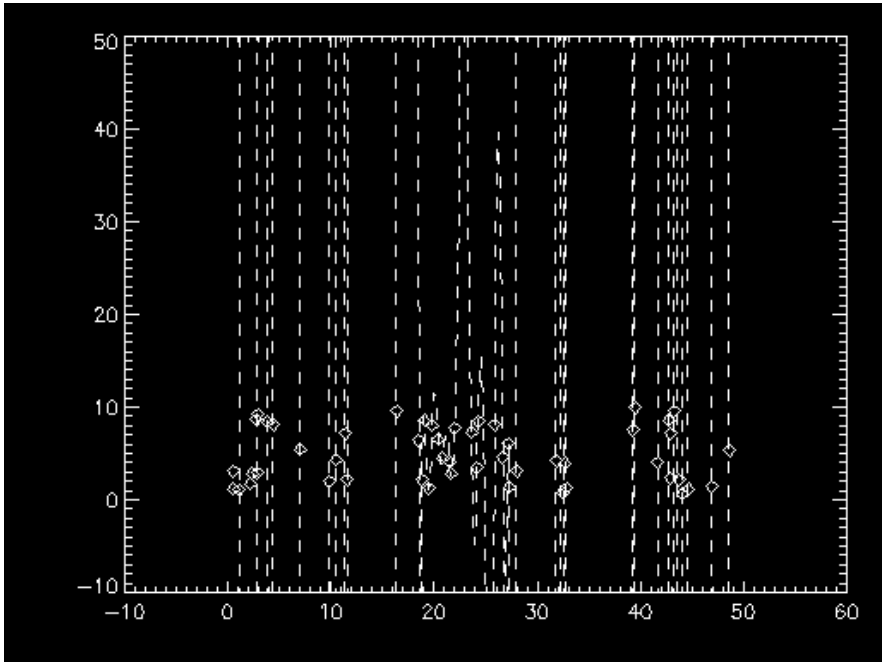
Mais c'est une méthode : INSTABLE

Autre exemple… avec 50 points



On interpole ces 50 points avec un polynome d’odre 49

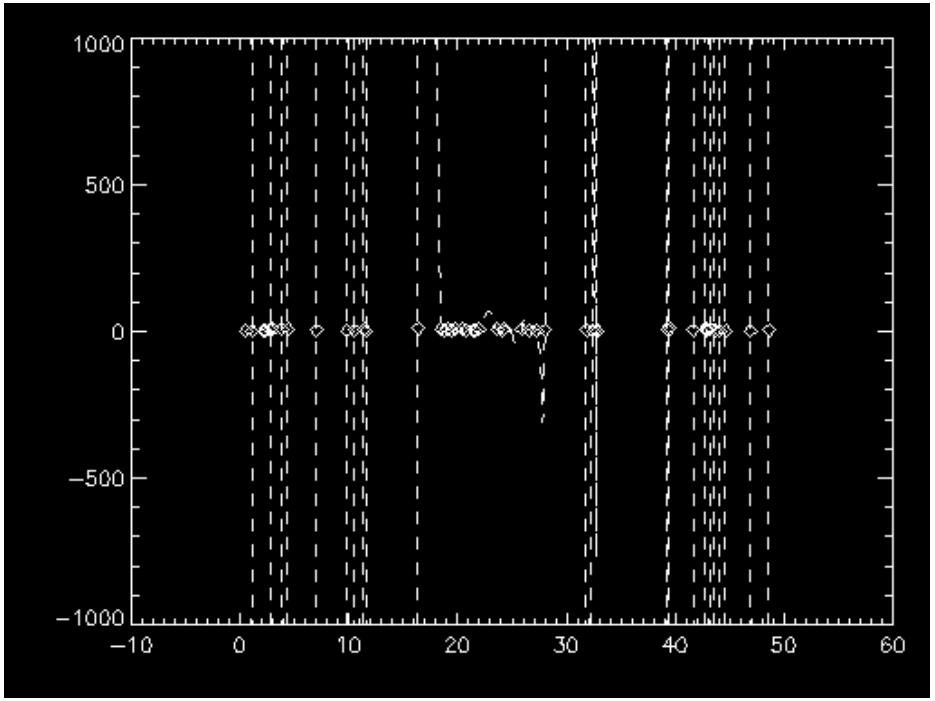




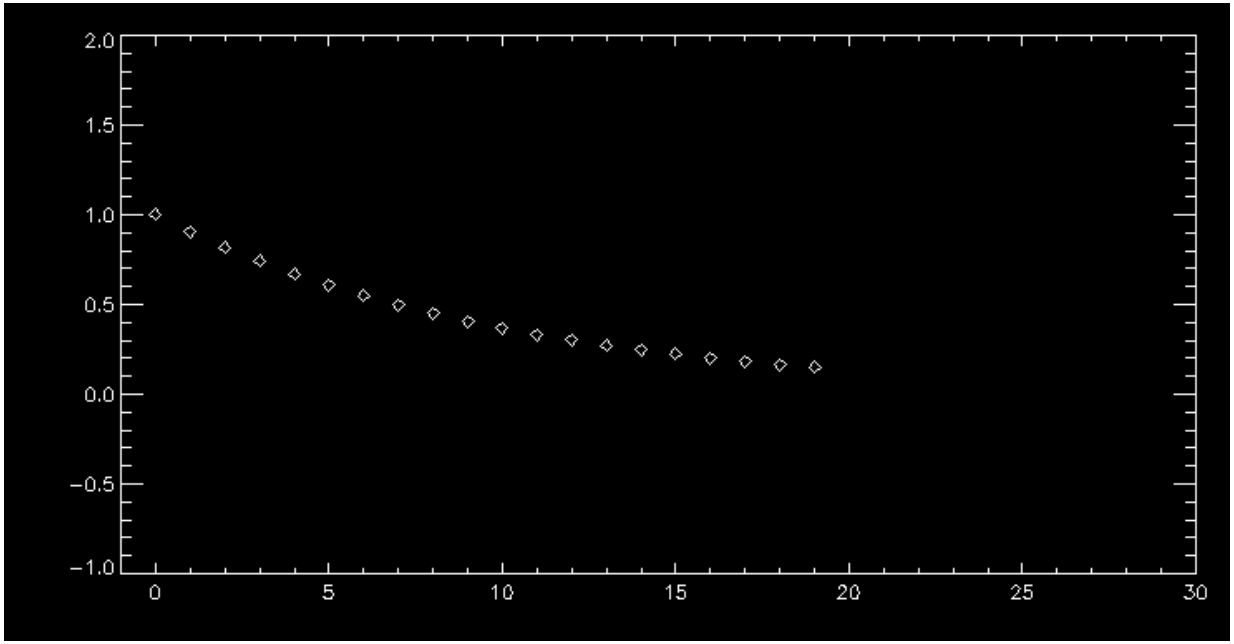
Notre polynôme
passe bien par tous les points
… mais entre ces points
Il a des variations IMPORTANTES

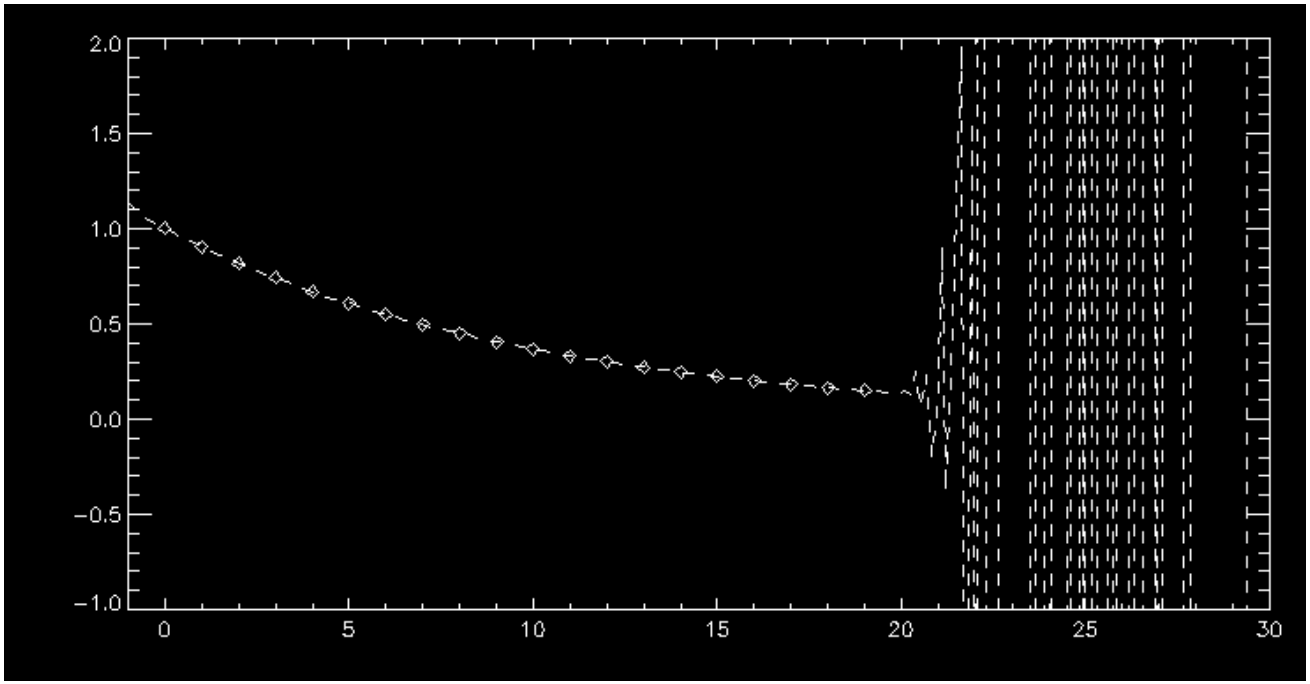
(ici : + ou – 10^{21} !!!!)

=> **TRES INSTABLE**
quand on a beaucoup de points…



Autre exemple (moins sauvage)





Interpolation
par polynômes
de Lagrange

L'interpolation polynomiale ne doit-être utilisée que :

-**Si on a peu de points (moins de 10)**
-**Si on est sur que les f_i sont des valeurs EXACTES (pas de pts de mesure)**

***Pour résoudre ce problème d'instabilité en conservant un approche polynomiale* :**

Méthode simple :
Ne pas utiliser tous les points, mais seulement les voisins

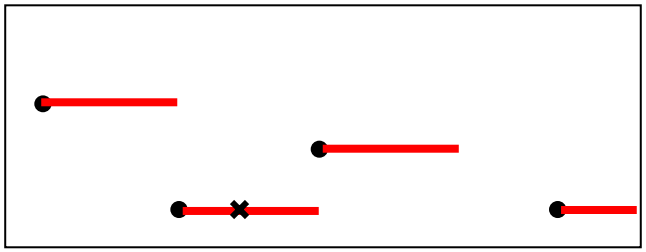
1 point : F est **constante par morceaux** (mais discontinue)

2 points : F est **affine par morceaux** (mais de non dérivable, classe C^0)

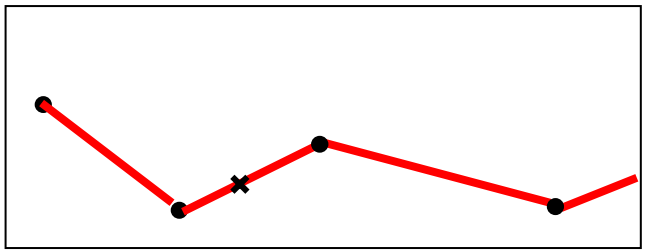
3 points : F est **parabolique par morceaux** (dérivable 1 fois, classe C^1)

Etc…

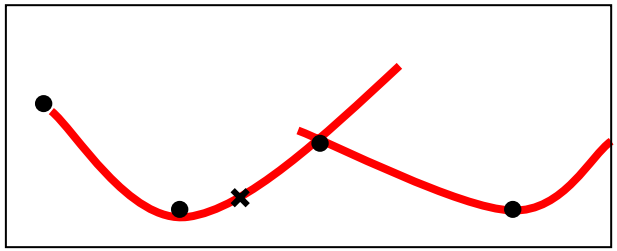
1 point : **constantes par morceaux**



2 points : **affine par morceaux**



3 points : **Parabolique par morceaux**



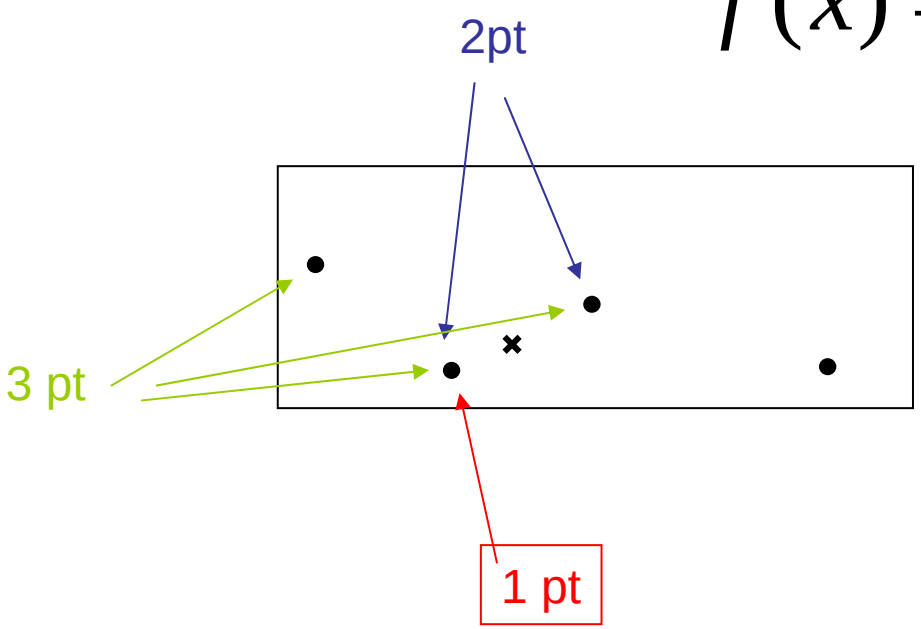
Ces méthodes sont plutôt stables… souvent très utiles… mais ont de mauvaises propriétés de dérivabilités…

Pour les calculer : On utilise les polynômes de Legendre.

On veut calculer f(x) en utilisant les **p** plus proches voisins : X_{j1} **…**X_{jp}

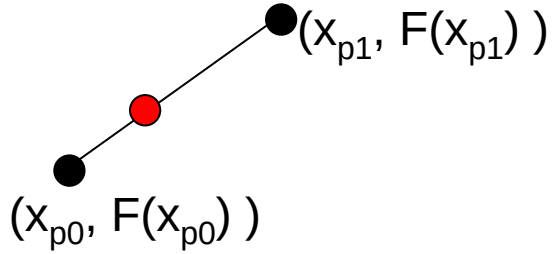
On a alors : $x_{j1} < x < X_{jp}$

$$f(x) = \sum_{i=j0}^{jp} a_i \prod_{j=j0,\, j\neq i}^{jp} \frac{(x - x_j)}{(x_i - x_j)}$$



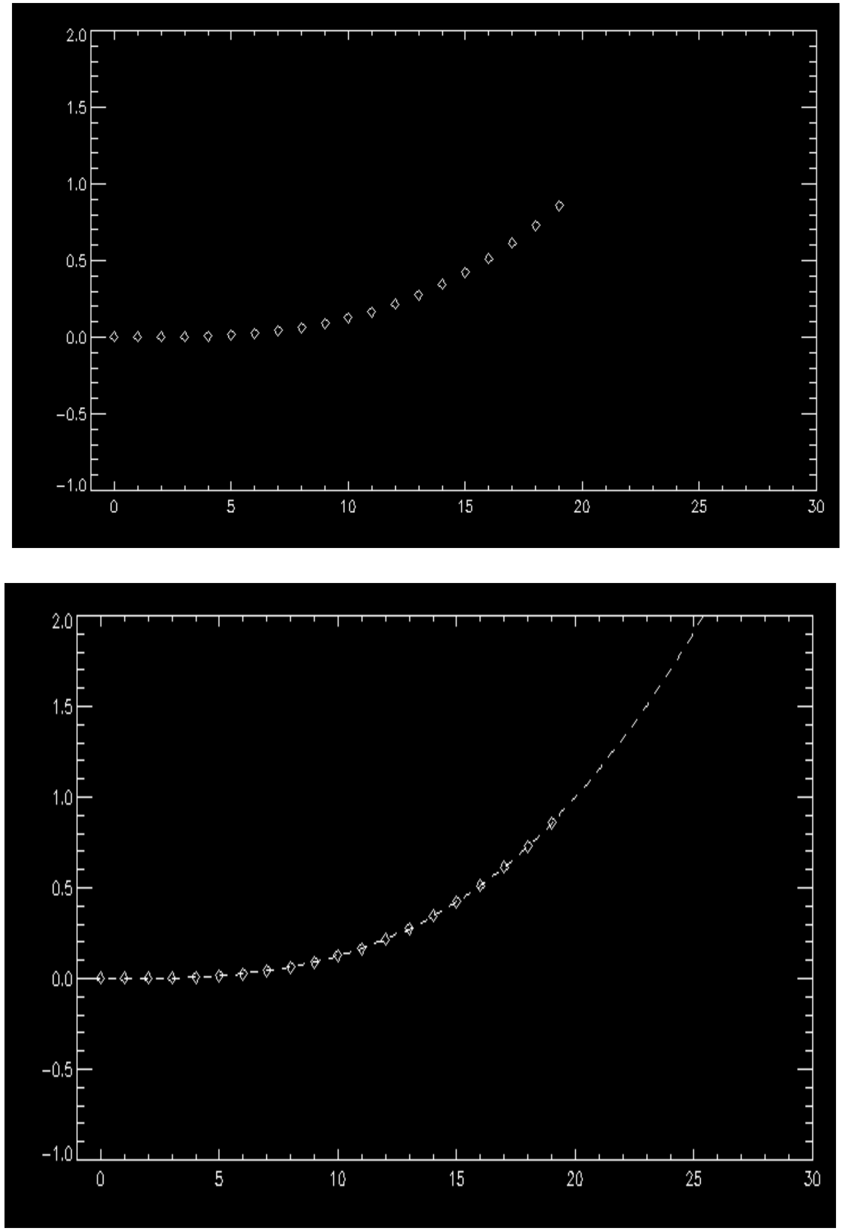
Dans le cas a 1 point : $F(x)=f(x_{p0})$ (X_{P0} est le pt immédiatement inférieur à X)

Dans le cas à 2 points : $$f(x) = \frac{(x - x_{p0})}{(x_{p1} - x_{p0})}(f(x_{p1}) - f(x_{p0}))$$

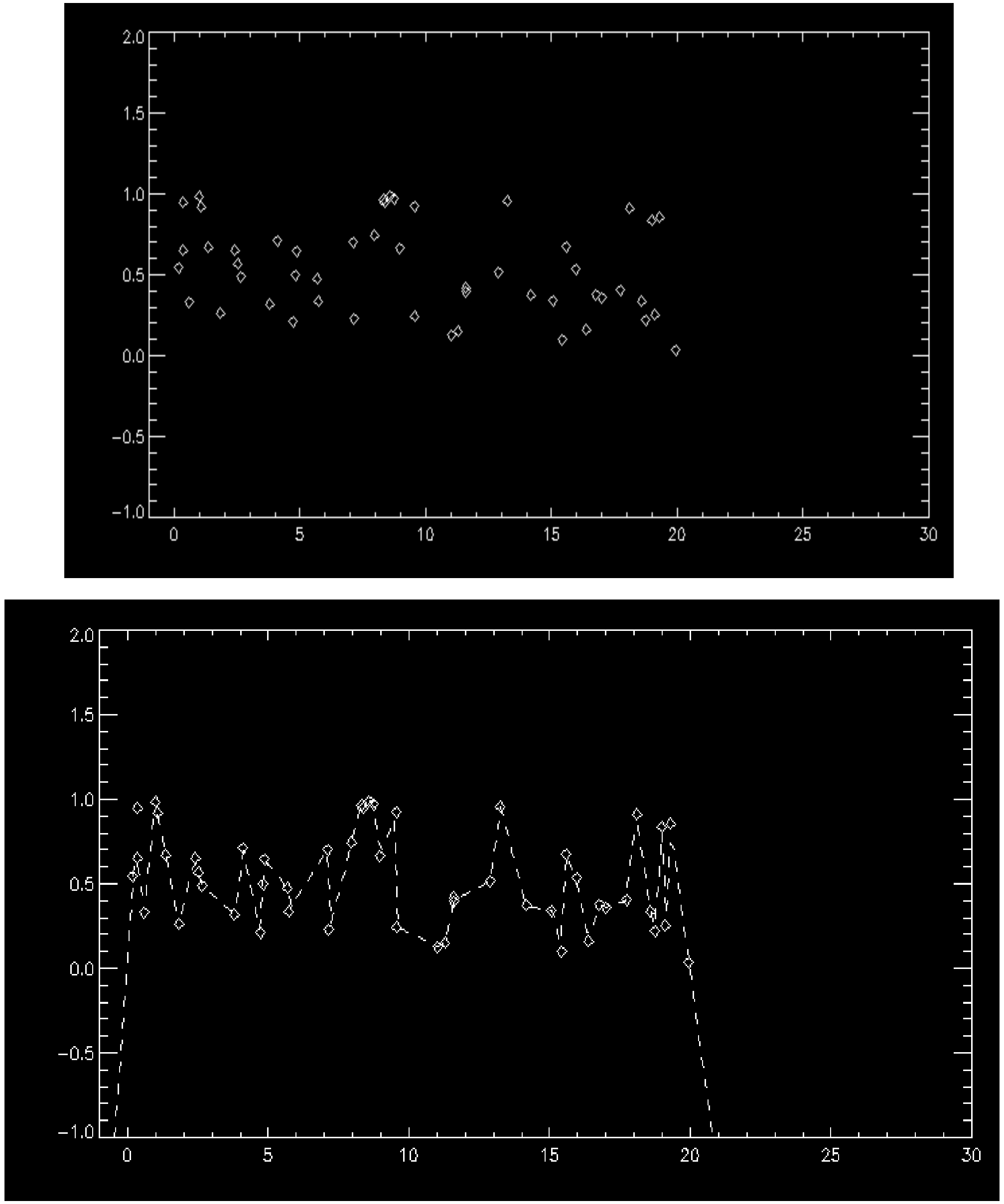


Simple interpolation linéaire

Une méthode d'interpolation populaire s'appelle « CUBIC SPLINE », elle utilise 4 points => F est de classe C^3 (polynome d'odre 3)



Interpolation parabolique (3 voisins).



Interpolation linéaire (2 voisins)

Les interpolations polynomiales sont simples à programmer
MAIS
Elles sont dangereusement instables.
Plus l'ordre est élevé plus l'instabilité sera grande.
=>
Il faut préférer les interpolations qui se basent sur les 2 ou 3 voisins (ordre faible)

Il existe encore d'autres méthodes d'interpolation exactes :

On peut aussi décomposer en fractions rationnelles

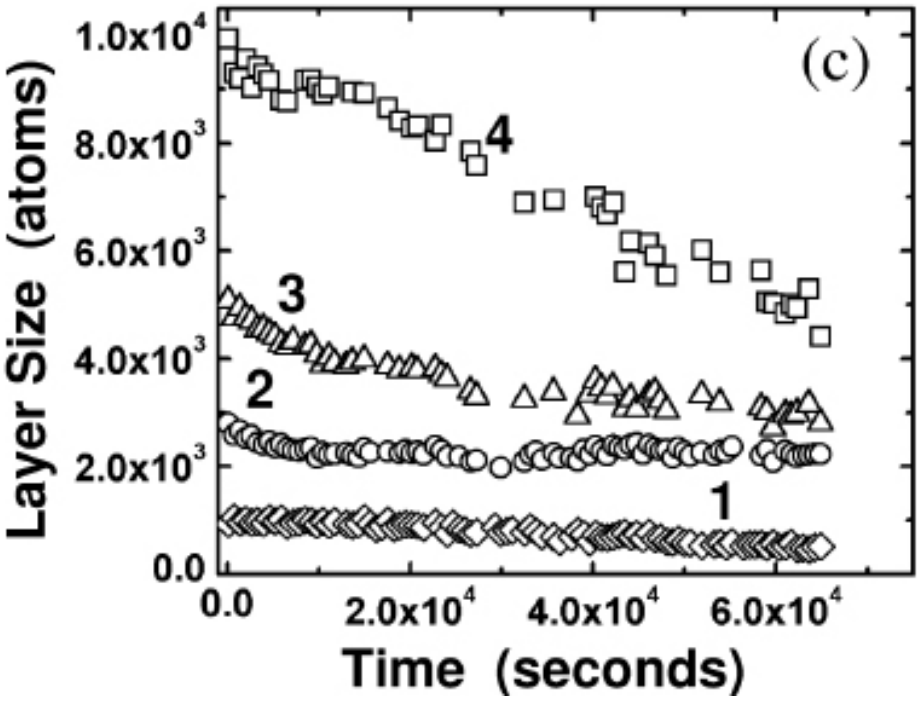
$$f(x) = \frac{p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1}}{q_0 + q_1 x + q_2 x^2 + \ldots + q_{n-1} x^{n-1}}$$

Les inconnues sont les p_i et les q_i .
Les fractions rationnelles sont moins instables que les polynomes mais moins faciles à coder.
Elles peuvent modéliser des fonctions avec des pôles (complexes) etc…

Pour plus d'info, lire « Numerical Recipes »

Les $f(x_i)$ sont des valeurs approchées : Adaptation d'un modèle de fonctions

Souvent en physique, on obtient des points de mesures $f(x_i)$ aux points x_i :
Ex :
- température d'un solide en fct de la température
-Nombre d'atomes radio-actifs en fonction du temps
Etc….



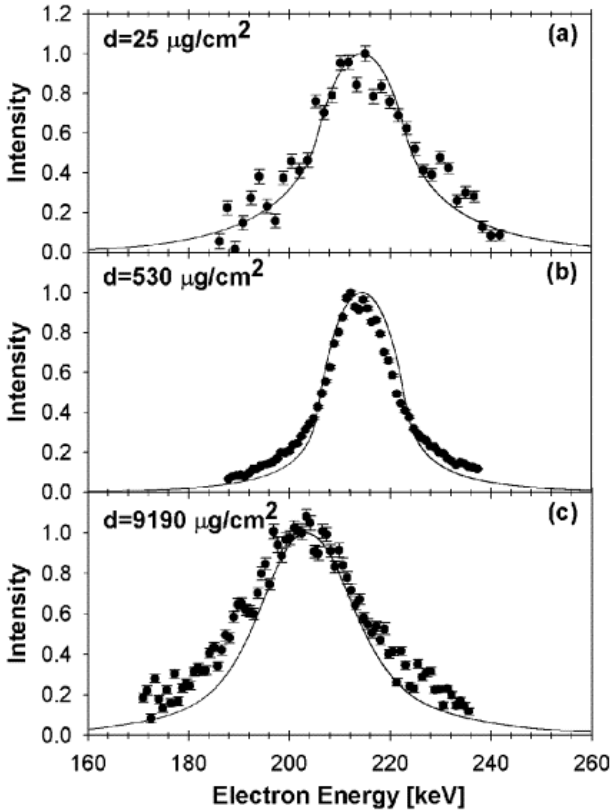
Exemple de mesure :
Nb d'atomes en fonction du temps

Si les f_i sont des points de mesures, ils sont soumis à des erreurs, des fluctuations (on appelle cela du bruit) qui rend une interpolation exacte INSTABLE et invariablement FAUSSE !!

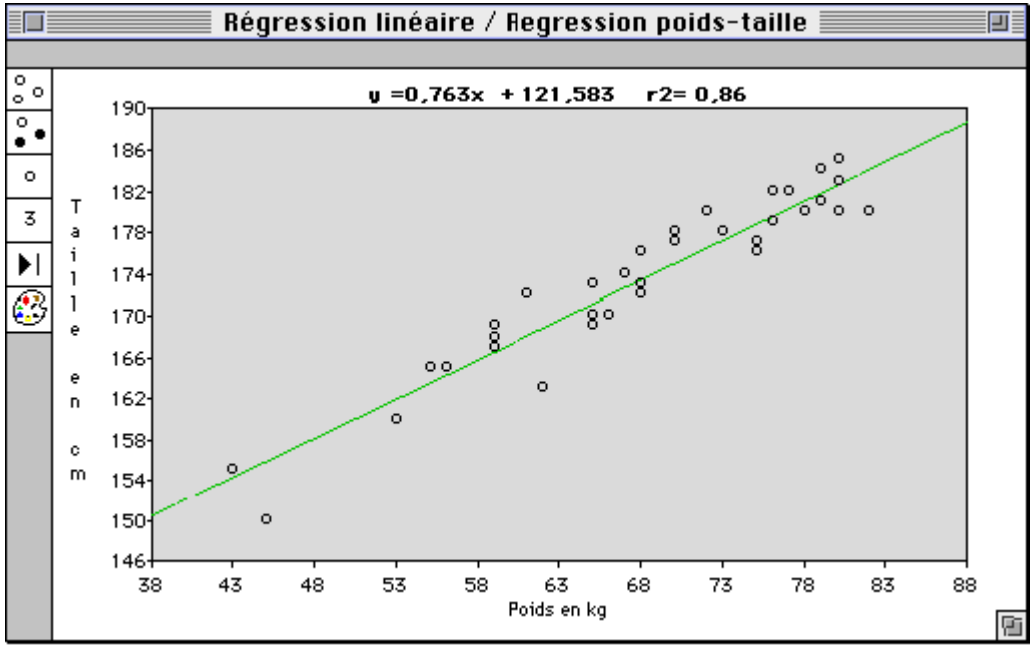
Dans ce genre de problème, typiquement on veut trouver
quelle est la meilleure fonction mathématique qui représente les données.

=> Elle ne passe pas exactement par les points, mais elle s'en approche.

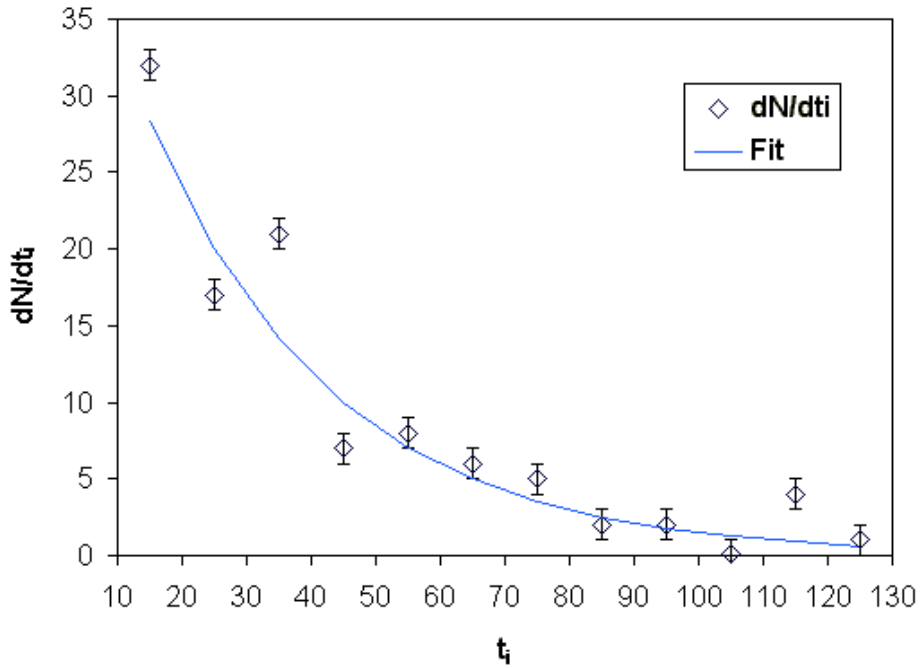
On appelle souvent cela « fitter » une fonction (en français « adapter »).



Ex : faire passer
une gaussienne par des points



Fitter un nuage de points par une droite *(Régression linéaire)*



Fitter un nuage par une exponentielle (decroissance radio-active)

Etc…

Le problème ici n’est donc pas d’interpoler MAIS
De trouver le meilleur modèle mathématique qui représente les données.

Donc **nécessité d’un modèle mathématique à priori !!!**

Ce modèle sera choisit en fonction :

-Soit de la physique du problème

-Soit de la « forme » des données expérimentales
(*Sachez que « l’œil humain » est un excellent outil pour analyser rapidement un ensemble de données … ne jamais négliger cela !!)*

On se choisit donc un modèle mathématique :

C’est un ensemble de fonctions qui dépendent d’un ou plusieurs paramètres.

EXEMPLES:

modèles	description	paramètres
$Y=aX+B$	Régression linéaire	a et b
$Y=a\ \mathbf{e}^{bx}$	exponentielle	a et b
$Y=a\cos(\omega x+\varphi)$	Sinus/Cosinus	a , ω , φ

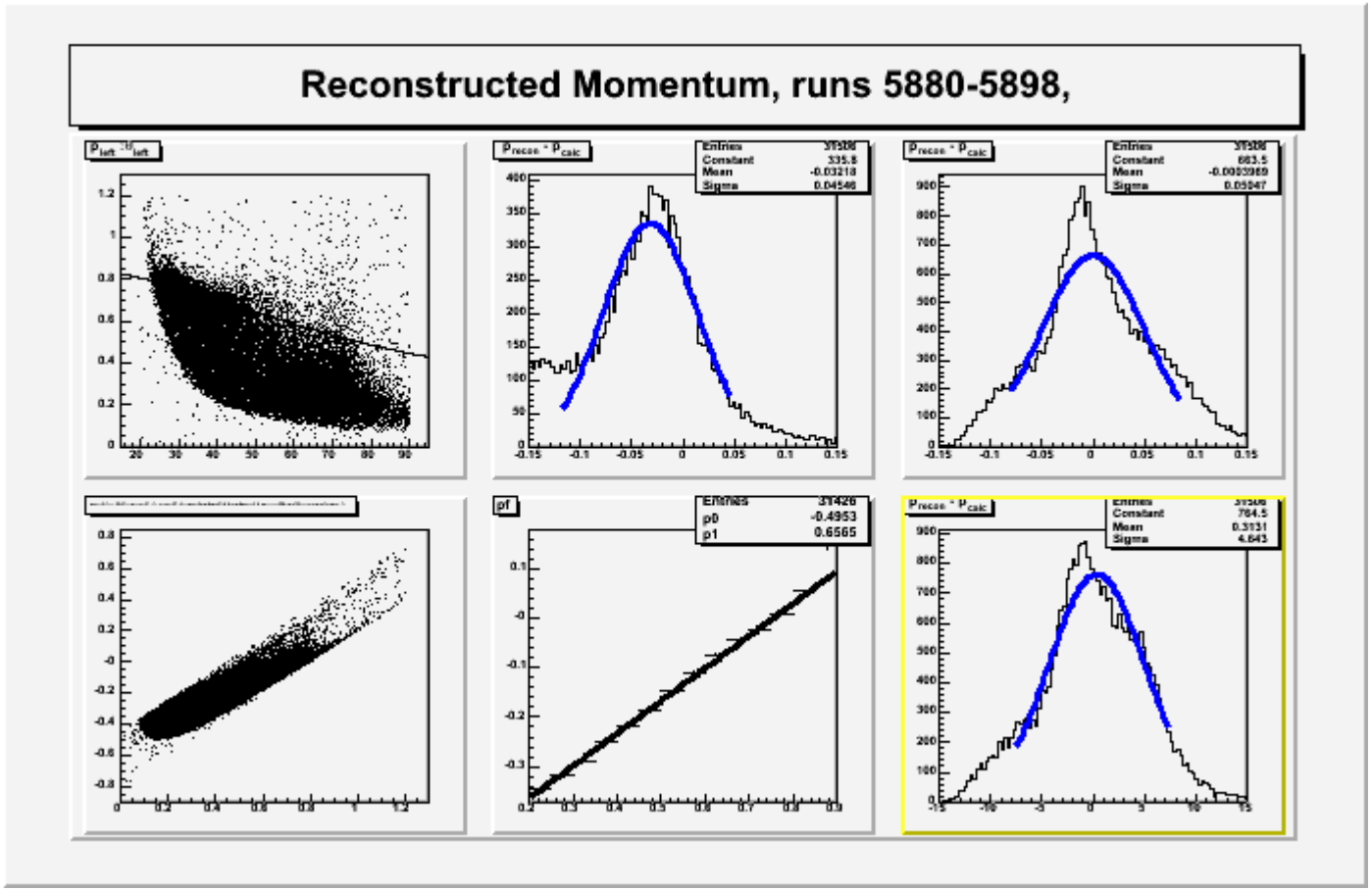
Etc …

Moins on a de paramètres libres, plus robuste sera le modèle (moins sujet aux instabilités) , mais le fit sera plus difficile

L’objectif n’est PAS de reproduire tous les points exactement (si c’est le cas, cf. Cours sur l’interpolation)

L’objectif EST de trouver la fonction qui passe au plus prés des points de mesure :
=> Minimiser la distance entre chaque point de la fonction et chaque pts de mesure

EXEMPLES



On doit donc se donner une MESURE de la distance entre les points et la fonction. On peut en imaginer de nombreuses ….

Une mesure courante : **MOINDRES CARRES (xhi 2, χ^2)**

$$\chi^2(a,b,c,...) = \sum_{\text{tt les pts}} \frac{(y^*_i - y_{a,b,c..}(x_i))^2}{\Delta_i^2}$$

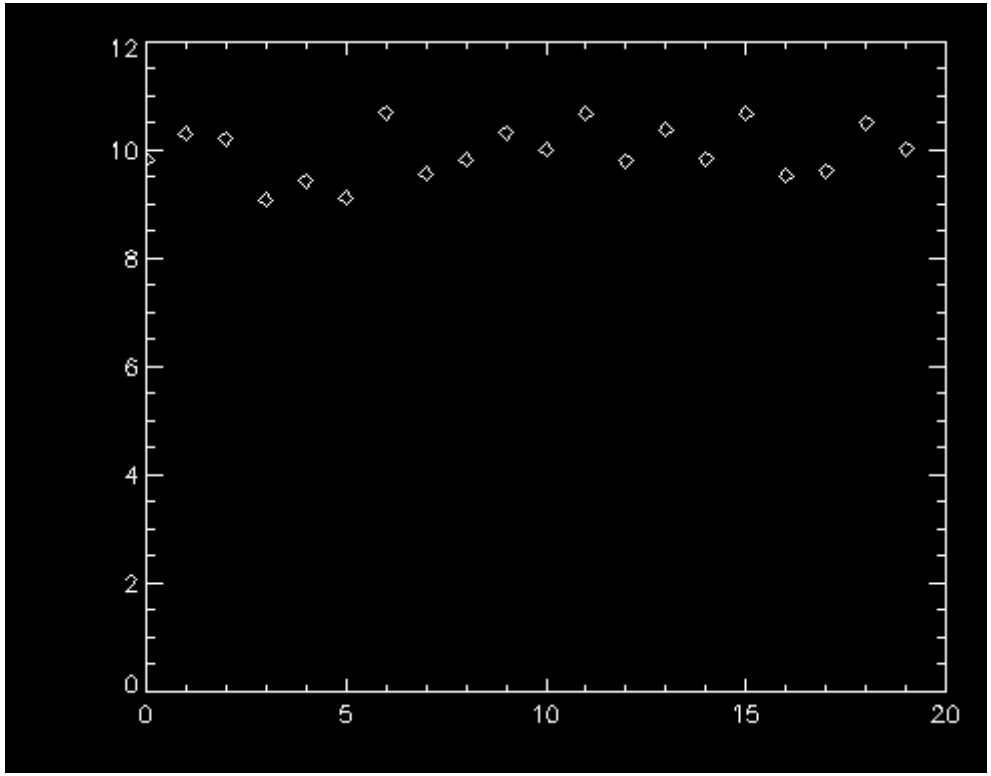
Y^*_i = pt de mesure numéro i (absice x_i)

$Y_{a,b,c..}(x_i)$= Modèle (qui dépend des paramètres a,b,c..) au point x_i

Δ_i = incertitude de mesure sur le point i

Donc le problème consiste à trouver les paramètres a,b,c etc… qui minimisent le Xhi2

En clair le xhi2 c'est : la somme des carrés des erreurs



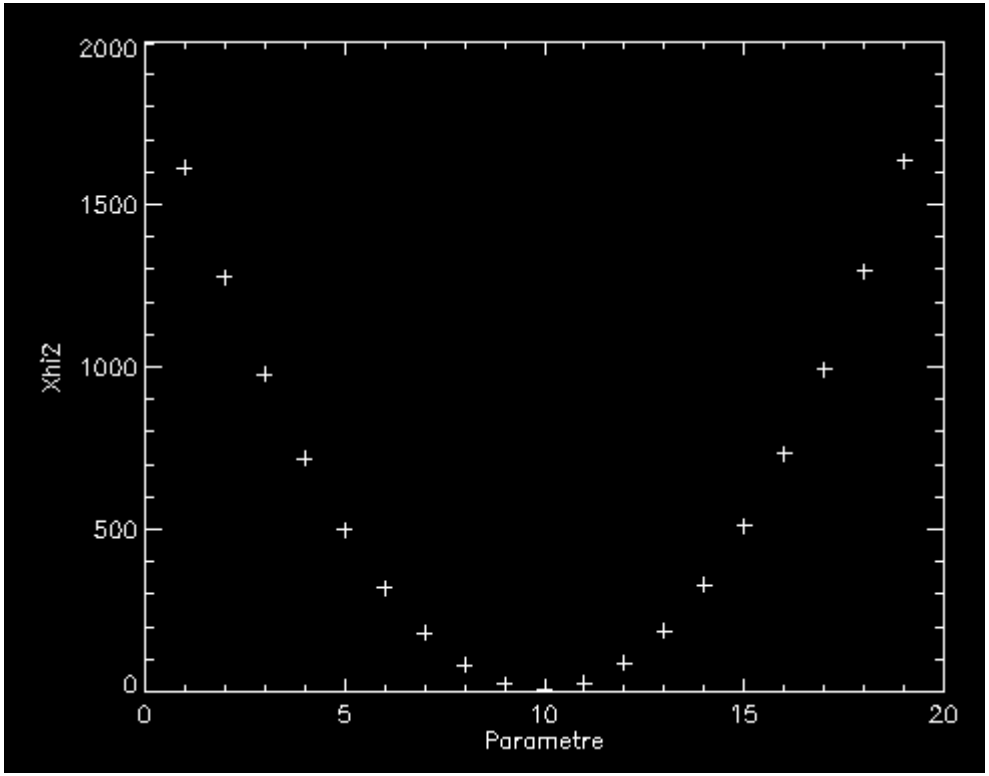
Considérons cet ensemble de mesures… Y^*_i pour i=1… n

A l'œil on voit que c'est
la fonction constante
Y(x)=10

On va utiliser un modèle
à 1 paramètre :

Y(x)=a

Quelle est la meilleure valeur de a ?



Ici nous avons tracé la valeur
du XHI2 en fonction du paramètre a

Nous voyons que le XHI2 est minimisé
pour a =10

C’est un exemple simple à 1 dimension
(1 paramètre)

En pratique souvent on utilise
2 ou 3 paramètres..

$\Rightarrow$Minimisation en plusieurs dimensions

Notez que la valeur du XHI2 nous
donne une estimation de la qualité
du fit.

Nous avons donc besoin d'un algorithme de minimisation pour trouver le meilleur ensemble de paramètres { a,b,c …} qui minimise le XHI2.

⇒Nous verrons de tels algorithmes plus tard dans le cours.

CEPENDANT

Pour le cas d'une *régression linéaire* (fit par une droite, Y=aX+b) il existe un algorithme simple et analytique qui donne directement la meilleure valeur de a et b (2 paramètres) au sens des **moindres carrés.**

Cette méthode peut-être généralisée pour des fonctions polynomiales :
$Y=a_0+a_1X+a_2X^2+\ldots+a_nX^n$

Etude du XHI2

$$\chi^2(a,b,c,...) = \sum_{\text{tt les pts}} \frac{(y^*_i - y_{a,b,c..}(x_i))^2}{\Delta_i^2}$$

Le XHI2 est une fonction des paramètres a,b,c , notés maintenant :
a_1 , …, a_n

Si le XHI2 est minimal au point (a^*_1, …, a^*_n) alors toutes les dérivées partielle s'annulent à ce point là.

$$\left.\frac{\partial \chi^2}{\partial a_1}\right|_{a_1^*,\ldots,a_n^*} = 0$$

$$\left.\frac{\partial \chi^2}{\partial a_2}\right|_{a_1^*,\ldots,a_n^*} = 0$$

...

$$\left.\frac{\partial \chi^2}{\partial a_n}\right|_{a_1^*,\ldots,a_n^*} = 0$$

$$\chi^2(a,b,c,...) = \sum_{\text{tt les pts}} \frac{(y^*_i - y_{a,b,c..}(x_i))^2}{\Delta_i^2} = \sum_{\text{tt les pts}} \frac{(y^*_i - y_i)^2}{\Delta_i^2}$$

La distance

$$y_{a_0,a_1,..a_{n-1}} = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} \Leftrightarrow$$

$$y = \sum_{p=0}^{n-1} a_p x^p$$

La fonction. On cherche la meilleure combinaison de a_i pour minimiser la distance

On doit alors résoudre simultanement les n équations

$$\left.\frac{\partial \chi^2}{\partial a_1}\right|_{a_1^*,...,a_n^*} = 0$$

$$\left.\frac{\partial \chi^2}{\partial a_2}\right|_{a_1^*,...,a_n^*} = 0$$

...

$$\left.\frac{\partial \chi^2}{\partial a_n}\right|_{a_1^*,...,a_n^*} = 0$$

$$\chi^2(a,b,c,...) = \sum_{\text{tt les pts}} \frac{(y^*_i - y_i)^2}{\Delta_i^2} \Rightarrow$$

$$\chi^2 = \sum_{pts:i}^{Np} \left(\frac{y^*_i - \sum_{p=0}^{n-1} a_p x_i^p}{\Delta_i} \right)^2 \Rightarrow$$

Où les ($x_{i,}$, y^*_i) sont les points de mesure
Et les (x_i, , y_i) sont les points de la fonction « modèle »

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_i^{Np} \left(a_k x_i^{k-1} \right) \left(\frac{y^*_i - \sum_{p=0}^{n-1} a_p x_i^p}{\Delta_i} \right)$$

!! Attention aux indices :
k : numéro de la variable que l'on dérive
p: coefficient p du polynome

i: numéro du point de mesure

$$\sum_{i}^{Np}\left(a_k x_i^{k-1}\right)\left(\frac{y^*_i-\sum_{p=0}^{n-1}a_p x_i^p}{\Delta_i}\right)=0\Leftrightarrow$$

$$\frac{a_0}{\Delta_0}\sum_{i}^{Np}x_i^k+\frac{a_1}{\Delta_1}\sum_{i}^{N_p}x_i^{k+1}+\ldots+\frac{a_n}{\Delta_n}\sum_{i}^{N_p}x_i^{k+n-1}=\sum_{i}^{N_p}\frac{y_i^*}{\Delta_i}x_i^k$$

On obtient donc le système linaire, pour k=0, n-1

$$\begin{pmatrix} (n+1)/\Delta_0 & \left(\sum_{pts} x_i\right)/\Delta_0 & \dots & \left(\sum_{pts} x_i^{n-1}\right)/\Delta_0 \\ \left(\sum_{pts} x_i\right)/\Delta_1 & \left(\sum_{pts} x_i^{2}\right)/\Delta_1 & \dots & \left(\sum_{pts} x_i^{1+n-1}\right)/\Delta_1 \\ \dots & \dots & \dots & \dots \\ \left(\sum_{pts} x_i^{n-1}\right)/\Delta_n & \left(\sum_{pts} x_i^{1+n-1}\right)/\Delta_n & \dots & \left(\sum_{pts} x_i^{2n-2}\right)/\Delta_n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} \sum_{pts} y_i \\ \sum_{pts} y_i x_i \\ \dots \\ \sum_{pts} y_i x_i^{n-1} \end{pmatrix}$$

M x A = Y
=> $A=M^{-1}\,Y$

CAS PARTICULIER : Régression linéaire

$Y=a_0 + a_1 X$, donc n=1 et k=0,1

Et on pose $\Delta=1$ (même incertitude de mesure sur tous les points)

Le système devient : *Ordre du polynôme*

$$\begin{pmatrix} n+1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \end{pmatrix}$$

ou encore

$$\begin{pmatrix} 1 & \overline{x} \\ \overline{x} & \overline{x^2} \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \overline{y} \\ \overline{yx} \end{pmatrix},$$

où $\overline{x}$, $\overline{x^2}$, $\overline{y}$ et $\overline{yx}$ désignent les moyennes de x_i, x_i^2, y_i et $y_i x_i$.

On obtient alors les formules

$$a_0 = \frac{\overline{y} \cdot \overline{x^2} - \overline{x} \cdot \overline{yx}}{\overline{x^2} - (\overline{x})^2} \qquad a_1 = \frac{\overline{yx} - \overline{x} \cdot \overline{y}}{\overline{x^2} - (\overline{x})^2}$$

La Valeur du XHI2 est interessante : elle donne une indication
sur la qualité du fit :
Distance moyenne ~ (XHI2/n points)$^{1/2}$ en unités de Δ_i

=> Fit « idéal » : XHI2~n points (chaque points est « fitté en moyenne à 1 Δ)

Si XHI2 >> n points : mauvais fit ou barres d'erreur surestimées

Si XHI2 << n points : trop bon fit : Vous avez sûrement surestime les barres d'erreurs

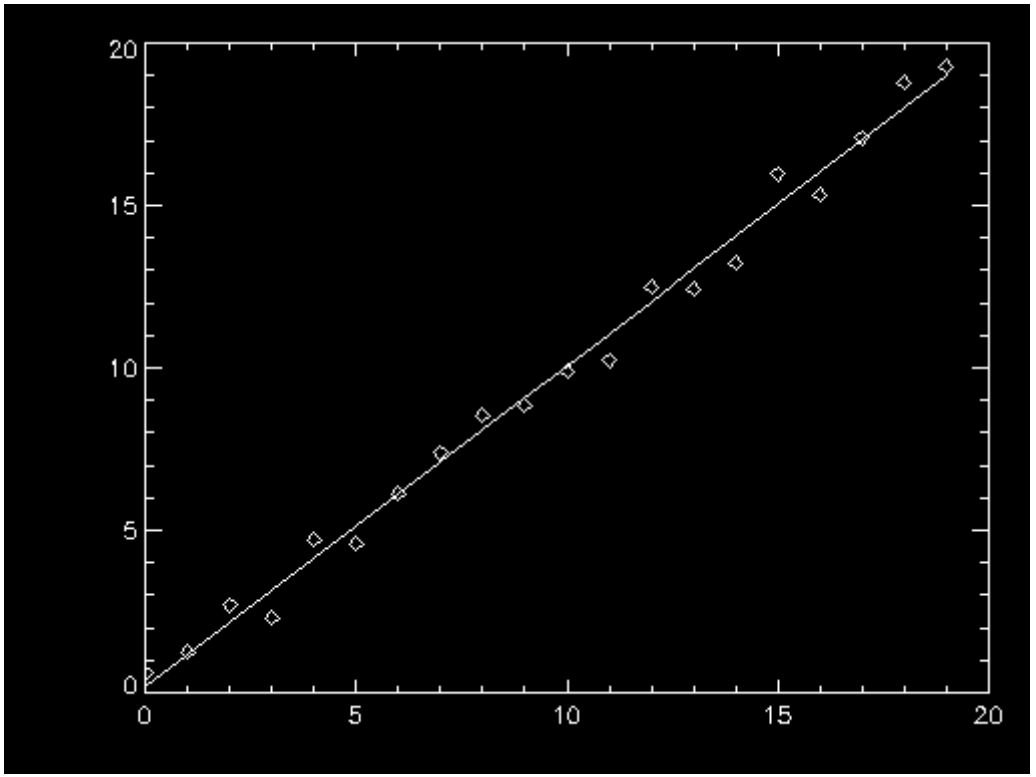
SOUVENT, quand on a besoin rapidement d'un fit , on met toutes les barres a 1
Mais dans ce cas on a pas d'estimation robuste de la qualité du fit

Exemple 1 : Fit par une droite

Ici tous les Δ_i =0.5

Valeur du
χ^2 = 25.2

$\Rightarrow$La deviation moyenne est
~ sqrt(25.2/n points)
~ 1.2 Delta

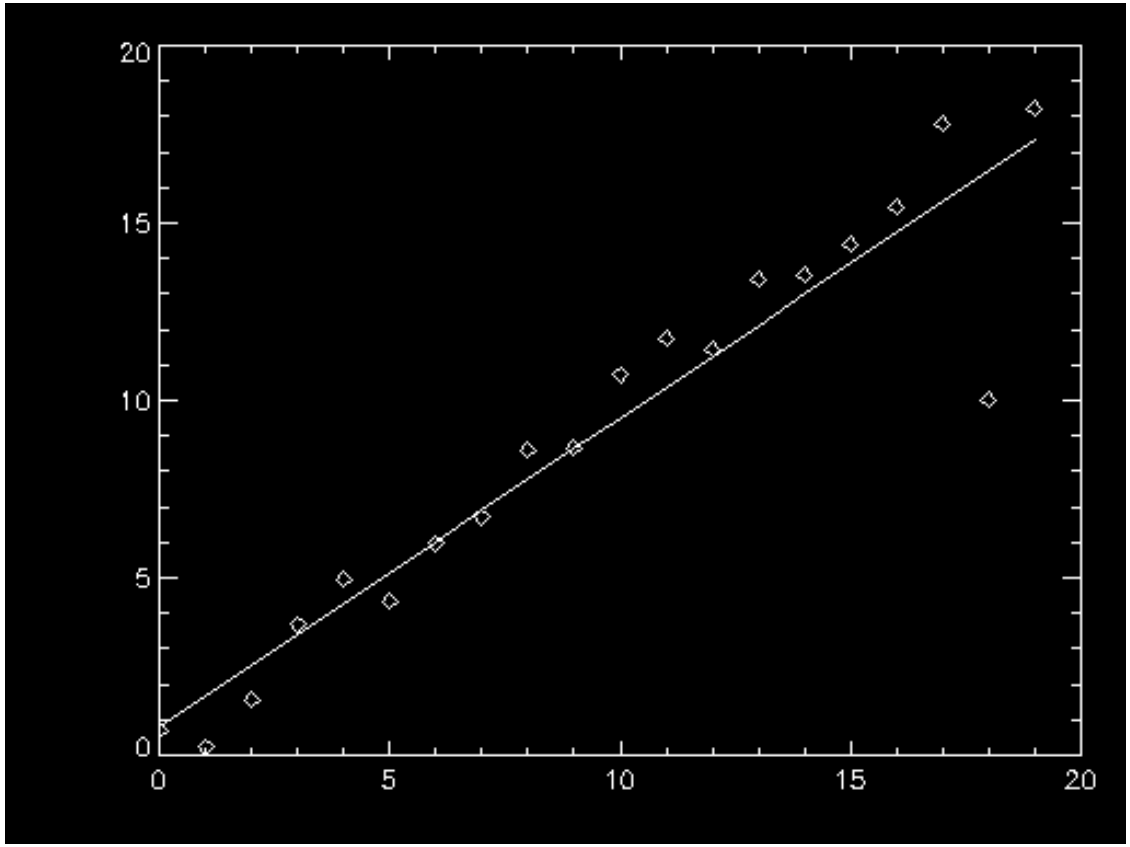


Exemple 2 : Fit par une droite avec un point aberrant

Δ=0.5 pour tous les pts

La droite a été
« tirée » vers le bas

XHI2=234 >> nb de pts
$\Rightarrow$Mauvais fit

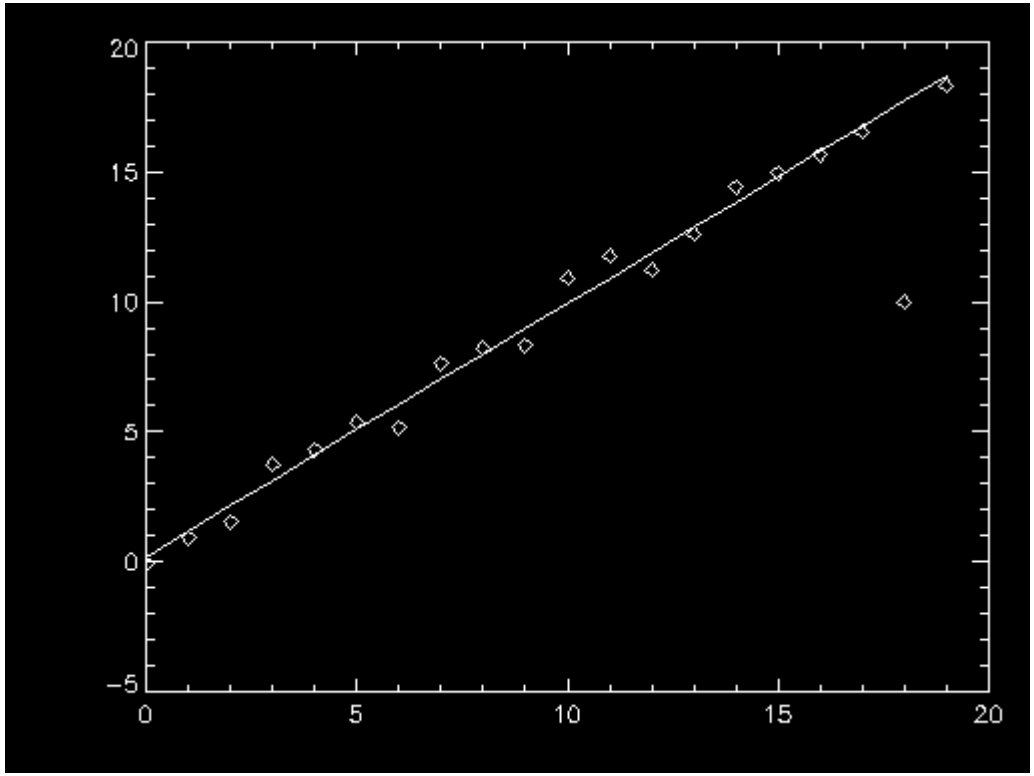


Solution ?? En fait la barre d'erreur sur le dernier pt peut etre
placée à beaucoup plus , disons Δ=10. pour ce pts, et 0.5 pour les autres

Δ=0.5 pour tous les pts et
Δ=10 pour le pt abérrant

$\Rightarrow$On retrouve une solution acceptable

XHI2=22 (Ok)



Conclusion: même si souvent on oublie de considérer les barres d'erreur de mesure peut être très utile pour s'assurer de la qualité du fit

Exemple 3 :

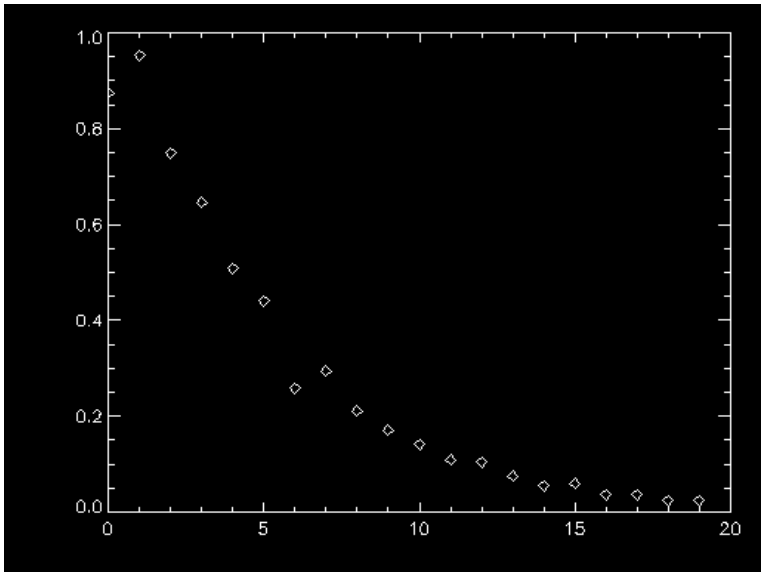
Fit par une fonction exponentielle : $Y=a\ e^{bx}$ les pts de mesure (x_i , y_i^*)

$\Rightarrow$On utilise la regression linéaire sur le LOG des points car :

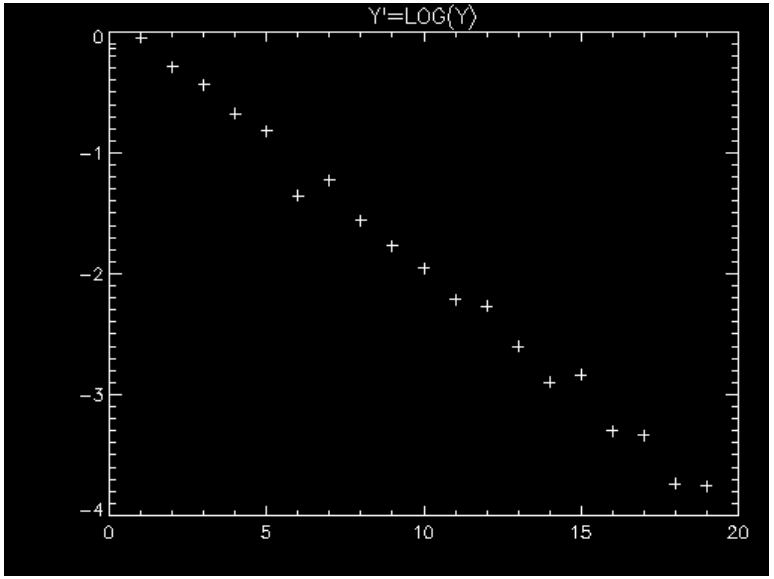
Si

les (x_i , y_i^*) sont représentables par $Y=a\ e^{bx}$ =>

Les (xi, $Ln(y_i^*)$) sont représentables par Y'=Ln(a)+ b X (c'est une droite)



=>
Y'=LOG(Y)



En résumé:

Faites attention quand vous calculez un fit : ayez toujours une estimation de sa qualité en considérant la valeur du XHI2 . Pour cela vous devez toujours connaître la qualité de vos points de mesure (barre d'erreur).

Si votre modèle de fonction est polynomial, vous pouvez trouver de manière analytique la meilleure solution en passant par une simple inversion de matrice.

Si vous voulez fitter par une fonction plus compliquée …
Il faudra faire une VRAIE minimisation du XHI2 avec un algorithme de minimisation itératif (Amoeba, Simplex etc…) … plus tard dans le cours

Les générateurs de nombres pseudo-aléatoires

Pour beaucoup d'applications en physique et en mathématique il est utile (voire nécessaire) d'avoir un générateur de nombres aléatoires :

Exemples :
Simulation d'une source radioactive
Simulation du mouvement brownien
Estimation d'une intégrale complexe en N dimensions (N>3)
Simulation d'un gaz
Etc …

On ne peut PAS réellement créer des nombres aléatoires de manière algorithmique car tous les outils sont détérministes.

Nous construirons des séries de nombres que nous qualifieront de « pseudo-aléatoires » si elles répondent à certaines tests statistiques i.e si elle a les mêmes propriété statistique qu'une *vraie* série aléatoire.

Toutes les séries pseudo-aléatoires dépendent d'un chiffre, appelé « graîne » (« seed ») . On pourra regénéer la même série de chiffre en partant de la même « graîne » =>
reproductibilité de la série (indispensable pour tester les programmes)

DISTRIBUTION UNIFORMES

Dans de nombreux langages de programmation, des générateurs de nombres pseudo-aléatoires sont implémentés.

Il s'appellent souvent : RAN, RANDOM , RANDOMU etc…

Un appel typique : X=RANDOM(*seed*) où *seed* est un nombre appelé « graîne »

La majorité des fonctions RANDOM sont appelé :

Générateurs congruentiels

Se sont des séries du type :

$$U_{j+1} = a \cdot U_j \pmod{m} \qquad \text{Ou} \qquad U_{j+1} = a \cdot U_j + c \pmod{m}$$

$$U_{j+1} = a \cdot U_j \pmod{m} \qquad \text{Ou} \qquad U_{j+1} = a \cdot U_j + c \pmod{m}$$

m= grand nombre ~
maximum représentable par la machine (~2^{32} = 4.3 10^9)

U_0 = graine

a, c : paramètres du générateur

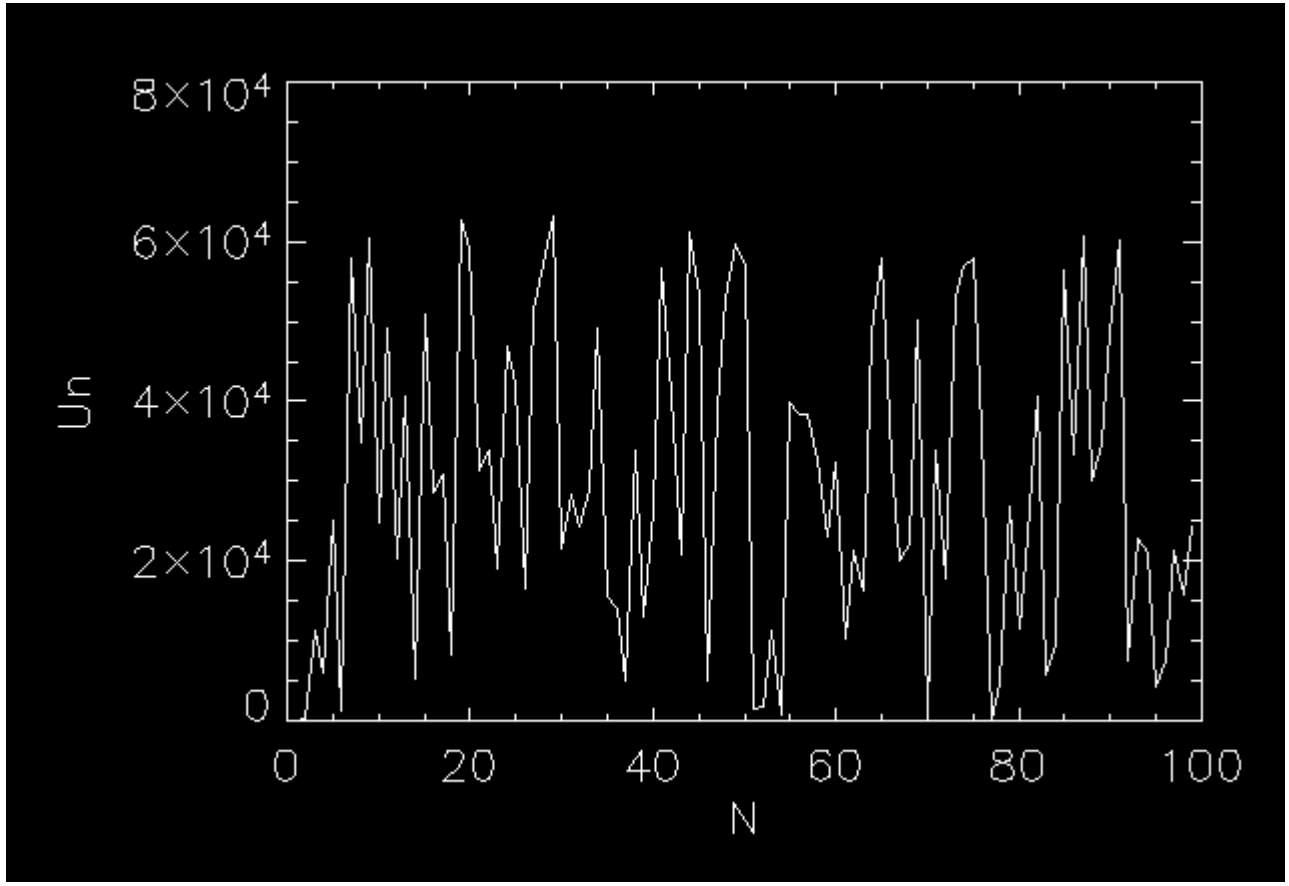
Donc une série de nombres pseudo-aléatoires est une suite
de chiffre calculés itérativement, avec U_0= graine.
Donc si on utilise la même graîne => on obtient la même suite

La « qualité » de la suite dépend d'un choix *Judicieux* de a et de c.

Certains couples (a,c) sont connus pour être bons, d'autres mauvais….

Exemple : $U_{j+1}= a\ U_j+c \pmod M$

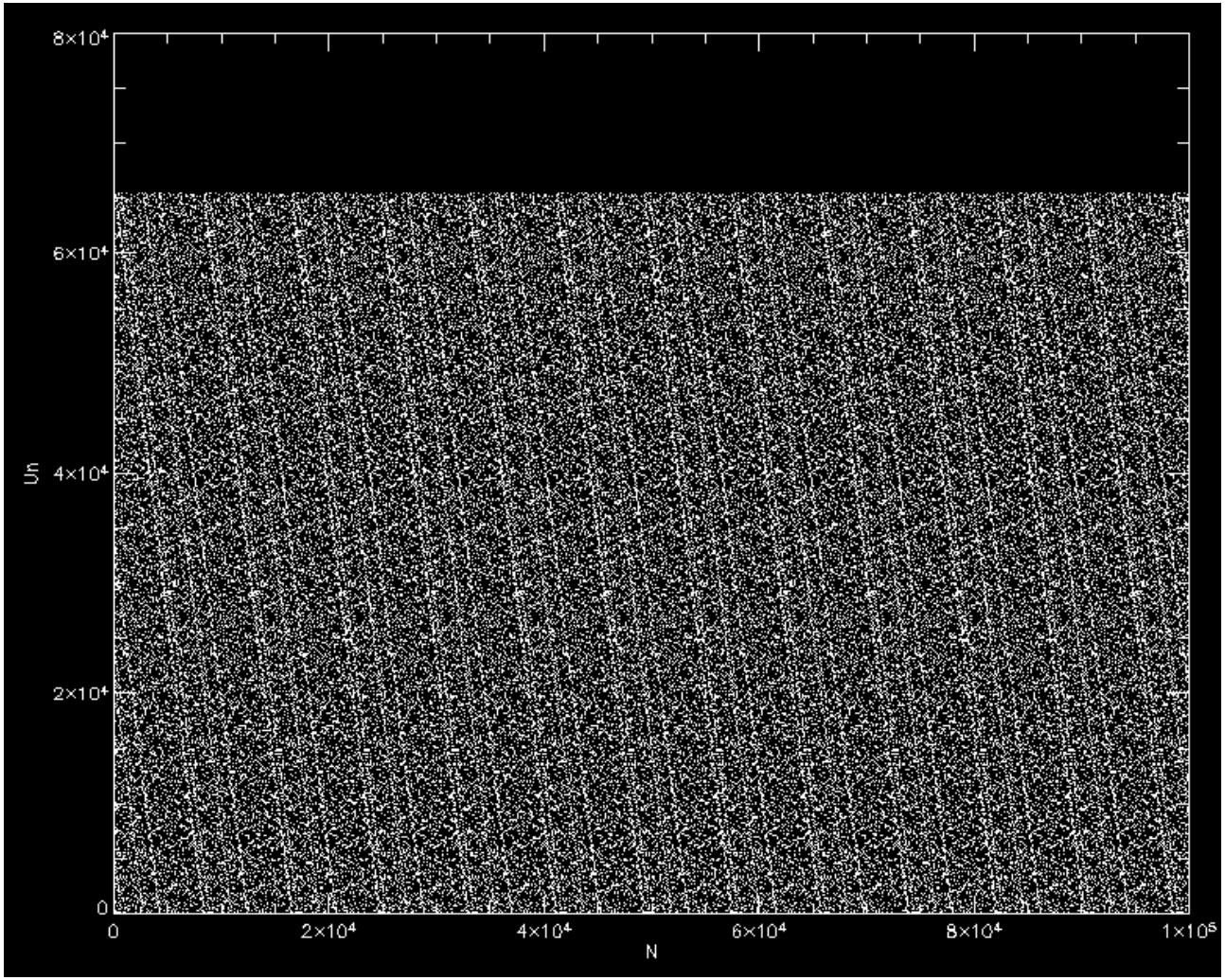
Avec a=47, b=5, m=65536



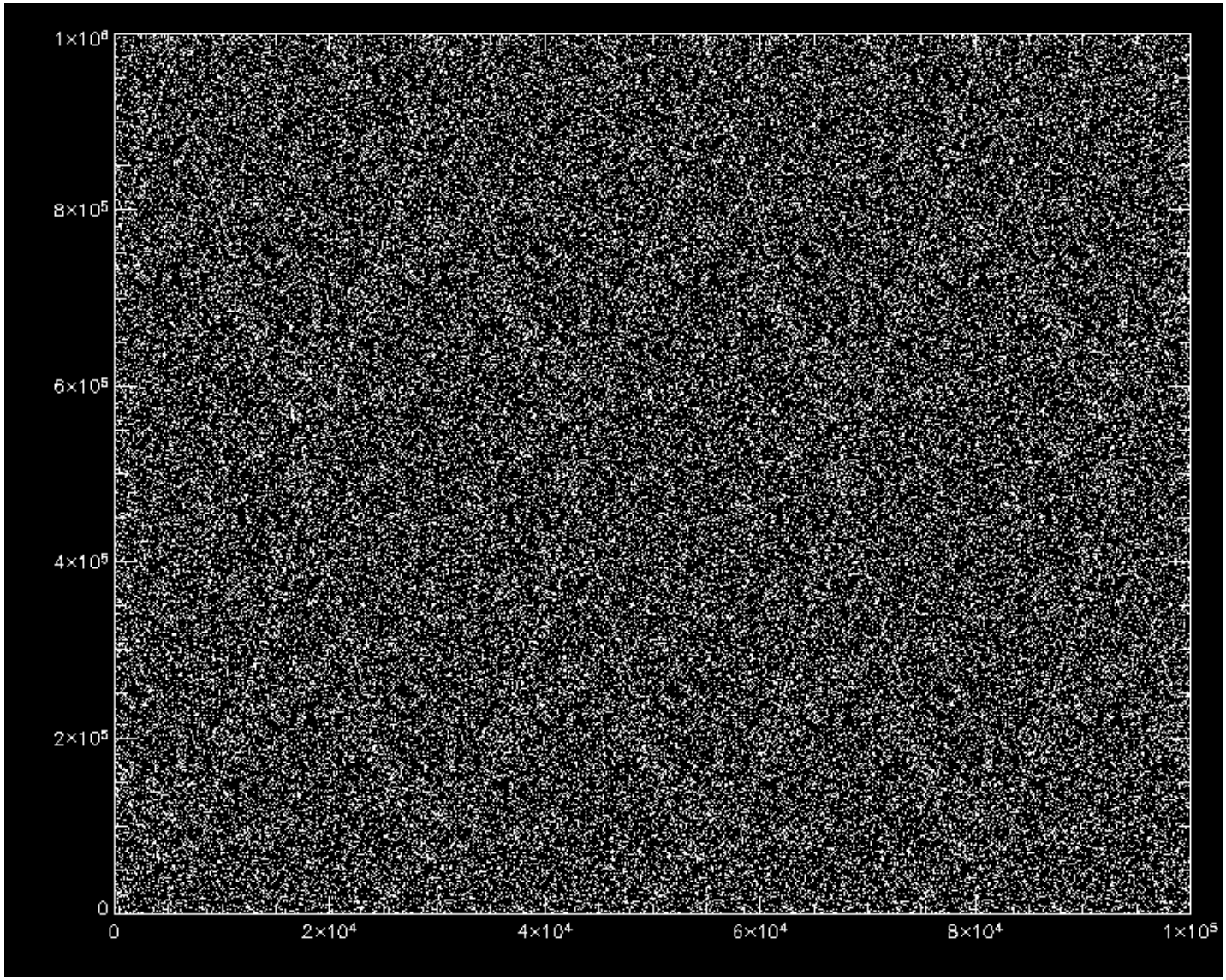
Si on veut une série entre 0 et 1 on divise par m

Pas mal comme résultat … mais si on tire 10^5 nombres les pb apparaissent :

Tirage de 1 million de chiffres pseudo-aléatoires



On voit clairement apparaître des « corrélations » entre les nombres,
Période tous les ~ 65000



Même générateur, mais avec $m=10^6$ => des corrélations existent toujours
MAIS plus longues : tous les 10^6 typiquement

Donc il faut bien choisir les valeurs de a, c et m

Il en existe beaucoup dans la littérature

$U_{j+1}=(U_j\, a + c) \bmod (m)$

Constants for Quick and Dirty Random Number Generators

overflow at	im	ia	ic
	6075	106	1283
2^{20}			
	7875	211	1663
2^{21}			
	7875	421	1663
2^{22}			
	6075	1366	1283
	6655	936	1399
	11979	430	2531
2^{23}			
	14406	967	3041
	29282	419	6173
	53125	171	11213
2^{24}			
	12960	1741	2731
	14000	1541	2957
	21870	1291	4621
	31104	625	6571
	139968	205	29573
2^{25}			
	29282	1255	6173
	81000	421	17117
	134456	281	28411
2^{26}			

overflow at	im	ia	ic
	86436	1093	18257
	121500	1021	25673
	259200	421	54773
2^{27}			
	117128	1277	24749
	121500	2041	25673
	312500	741	66037
2^{28}			
	145800	3661	30809
	175000	2661	36979
	233280	1861	49297
	244944	1597	51749
2^{29}			
	139968	3877	29573
	214326	3613	45289
	714025	1366	150889
2^{30}			
	134456	8121	28411
	259200	7141	54773
2^{31}			
	233280	9301	49297
	714025	4096	150889
2^{32}			

Les générateurs congruentiels sont faciles à coder et rapides (important si on a besoin de 10^9 nombres par exemples)…

MAIS

Ils finissent toujours par se répéter (i.e modulo m), les corrélations sont inévitables

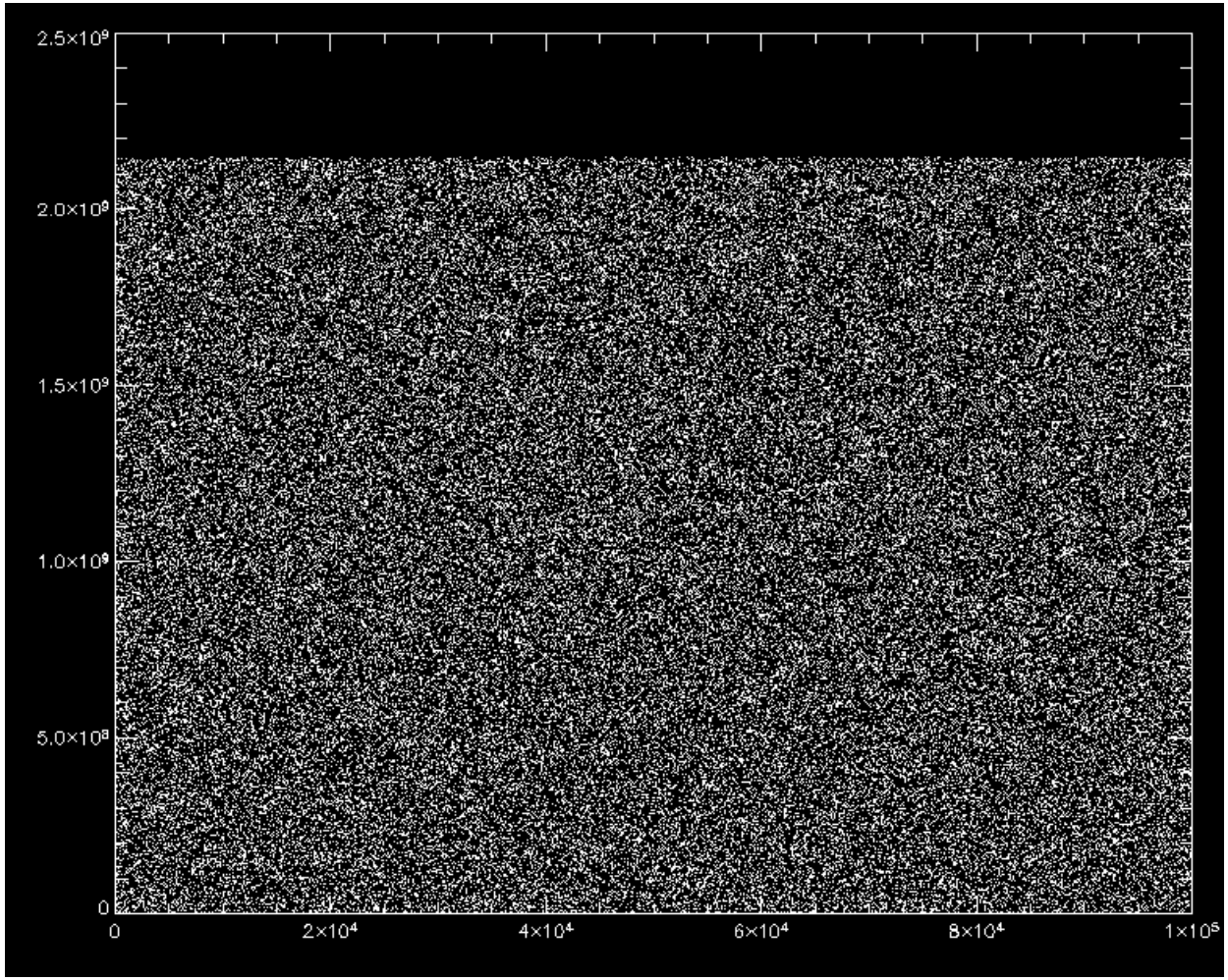
Donc il faut jamais utiliser des séries de N nombres où N est comparable à m

Il faut savoir aussi que le « hasard » est toujours plus important sur les premières décimales (essayez pour voir)=>

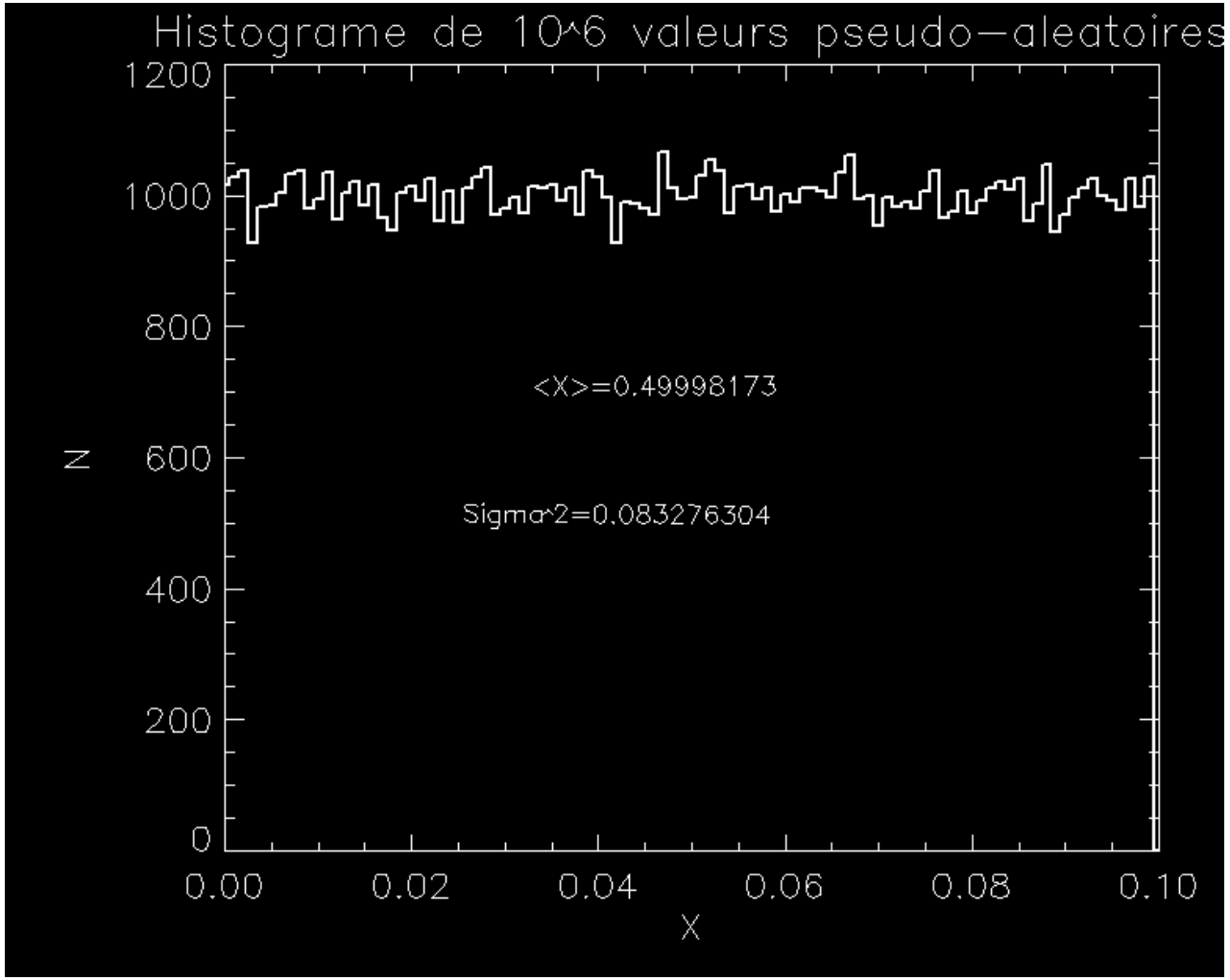
Si vous avez besoin de 2 nombres aléatoires, Tirez en 2, et NE FAITES PAS « je casse le nombre en plusieurs bouts pour en faire deux… »

Un générateur relativement simple et performant est :

$A=7^5 = 16087$
$M=2^{32}-1 = 2147483647$
$C=0$



Pour 10^6 chiffres … ça à l'air OK



Tirage entre 0 et 1

Valeurs théoriques : <X>=0.5

Variance (Sigma^2)=1./12.=0.083333

A la vue des 2 premiers moments notre distribution est assez fidèlement uniforme..

Pour diminuer encore les corrélations, diverses techniques existent,
Par exemple on tire au hasard « l'indice » du chiffre numéro j …
(« shuffling »)…

On peut aussi utiliser des suites de Fibonacci (demandent + de mémoire)

$U_{j+1} = (aU_{j-1} + bU_{J-1} + c) \bmod (m)$ etc…

Les générateurs congruentiels permettent de tirer une série de nombre avec un distribution ~ uniforme entre 0 et m-1

Si vous avez besoin d'une série entre 0 et 1, divisez les résultat par m.

EN PRATIQUE

On préférera utiliser des algorithmes préprogrammés en SE RENSEIGNANT sur leur robustesse !!!

Tirez au hasard 10^7 chiffres et affichez le résultat… déjà à l'œil vous aurez une idée s'il y a des corrélations évidentes ou non

Méfiez vous si vous avez besoin d'une LONGUE séquence ROBUSTE.. Des corrélations cachées peuvent vraiment affecter la validité de votre résultat !!

ENFIN

Si vous voulez du VRAI hasard, c'est possible.. Sur le web !!
http://www.random.org/

Où vous aurez des séries VRAIMENT aléatoires, tirées de systèmes physiques…

Fabrication de suites pseudo-aléatoires non-uniformes

Nous savons fabriquer une distribution uniforme entre a et b : $p(x) = \frac{1}{b-a}$

Mais de nombreux problèmes nécessitent des distributions non-uniformes :

exemples

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$ Gaussienne (Normale)

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$ Poisson

P(x)= densité de probabilité

$$p(x) = \frac{a}{\pi(a^2 + x^2)}$$ Cauchy / Lorentzienne

$$p(x) = \frac{x}{\sigma^2} e^{\frac{-x^2}{2\sigma^2}}$$ Raleygh etc….

Problème : Nous voulons générer une série de N points pseudo-aléatoires qui sont distribués suivant une *densité de probabilité p(x)*

Pour cela nous allons utiliser un théorème fondamental de transformation des lois de probabilités :

Préliminaires : On considère une série de nombre aléatoires x_i entre a et b

p(x) dx est la probabilité qu'un nombre x_i soit entre x et x+dx

$$\Rightarrow \int_a^b p(x)dx = 1$$

« la probabilité d'être dans le domaine de définition [a,b] est égale à 1 » (Normalisation)

On définit C(x) la distribution cumulée : Probabilité d'être < x :

$$C(x) = \int_a^x p(x)dx$$

On a donc C(b)=1

Exemple : p(x)=1/(b-a) => c(x)= (x-a)/(b-a)

Transformons les probabilités :

On considère une série de nombre $\{x_i\}$ distribués uniforméments

On considère maintenant la série des $\{ y_i \}$ où $y_i=f(x_i)$ est une fonction bijective sur [a,b
Les $\{ y_i \}$ sont aussi noté les $\{ f(x_i) \}$

Connaissant p(x) nous voulons connaître la distribution p(y) des $\{y_i\}$
Soit $x=Y^{-1}(y)$

$$\left|p(y)\,dy\right| = \left|p(x)\,dx\right| \Rightarrow p(y) = p(x)\left|\frac{dx}{dy}\right|$$

Avec $x=f^{-1}(y)$

Exemple : Prenons $y=-\ln(x)$, et p(x)=1. entre 0 et 1
Alors :
$P(y)=p(x)*|dx/dy|$ avec $-1/dx=dy \Rightarrow dx/dy=-1/x$

$\Rightarrow P(y)= 1. \ | 1/x |$ or $x=e^{-y}$

$\Rightarrow p(y)=e^{y}$ Distribution exponentielle !!!

Exemple :

$\{x_i\}$={5.740e-005, 0.965 , 0.1291, 0.0106, 0.1106, 0.3120, 0.825, 0.9401, 0.922, 0.217, 0.44616127, …….}

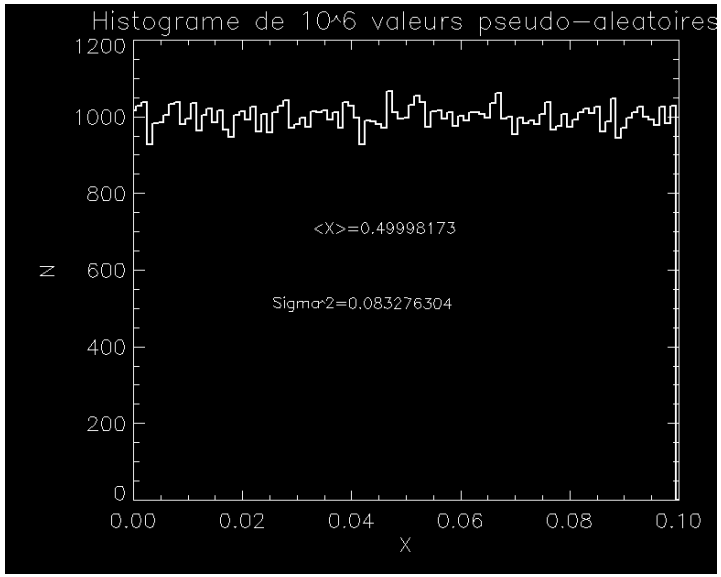
=>

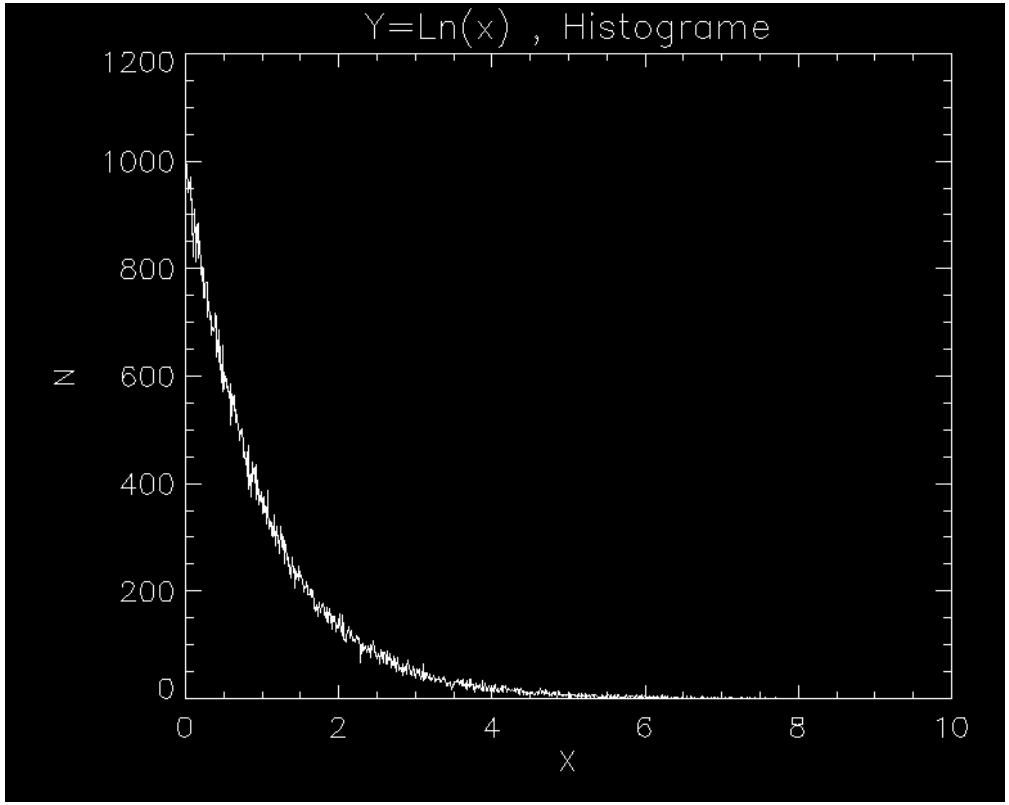
$\{y_i\}$={9.763, 0.03437, 2.0466, 4.541, 2.201, 1.16455, 0.1919, 0.0617, 0.080, 1.5274, 0.8070 ……}

$\{x_i\}$ distrib. uniforme entre 0 et 1

=>

{ $-\ln(x_i)$} : distrib. exponentielle





Donc en utilisant la formule de transformation des distributions

$$p(y) = p(x)\left|\frac{dx}{dy}\right| \text{ où } x = f^{-1}(y) \Leftrightarrow$$

$$p(y) = p\left[f^{-1}(y)\right]\cdot\left|\frac{df^{-1}(y)}{dy}\right|$$

On peut transformer notre distribution uniforme en n'importe quelle autre distribution

Sous réserve que l'on puisse calculer dx/dy et f^{-1} (ce qui n'est pas toujours les cas..)

De manière plus pratique : On peut utiliser faire intervenir la distrib. cumulative
Supposons que nous voulins obtenir une distribution p(y) à partir de nombres uniformément $\{x_i\}$ distribués entre 0 et 1, p(x)=1.

Comme la distrib. Cumulative $$C(y) = \int_{y\inf}^{y} p(y)dy$$

Si on pose : $y=f(x)=C^{-1}(x)$ <=> $x=f^{-1}(y)=C(y)$

Alors :

$$p(y) = p\left[f^{-1}(y)\right]\cdot\left|\frac{df^{-1}(y)}{dy}\right| <=>$$

$$p(y) = p(x)\cdot\left|\frac{dx}{dy}\right| <=>$$

$$p(y) = 1\cdot\left|\frac{1}{dC^{-1}/dx}\right| = \frac{dC}{dy} = p(y)$$

EN PRATIQUE :

Soit une série de nombres {xi} uniformément distribués entre 0 et 1
On veut construire une distribution $\{y_i\}$ qui suit la lois p(y)

Soit P(y) la distribution cumulative :

$$p(y) = \int_{y\,\mathrm{inf}}^{y} p(y)dy$$

On calcule alors $y_i = P^{-1}(x_i)$

Les y_i ont la bonne distribution

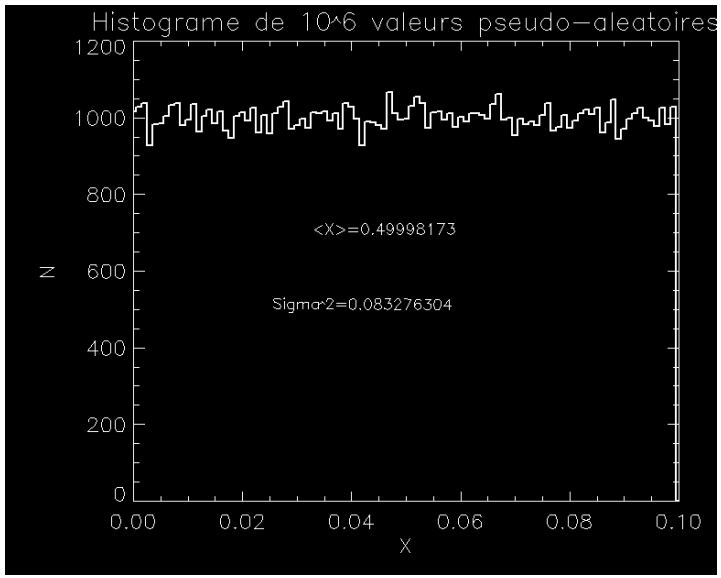
Exemple :

$$p(y) = \frac{1}{\pi(1+y^2)}$$ Lorentzienne

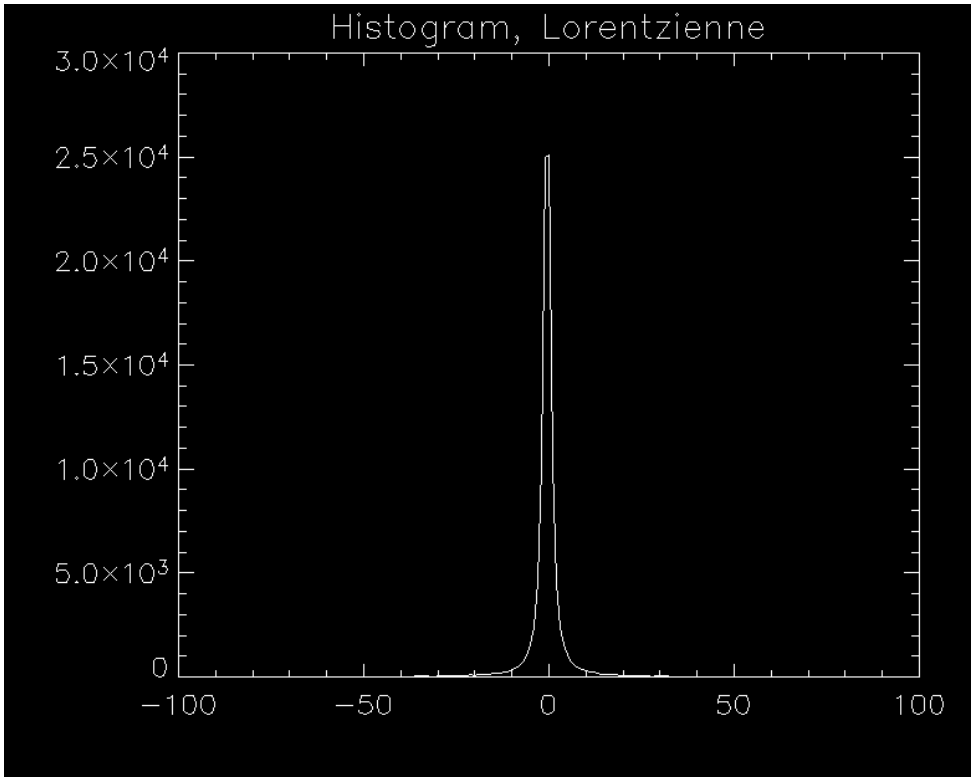
$$=> C(y) = \frac{1}{2} + \frac{\arctan(y)}{\pi}$$

$$=> C^{-1}(x) = \tan(\pi x - \pi/2)$$

On calcule des {xi} distribués uniformément
=>
On calcule les $y_i = C^{-1}(x_i)$ qui seront distribués selon une Lorentzienne

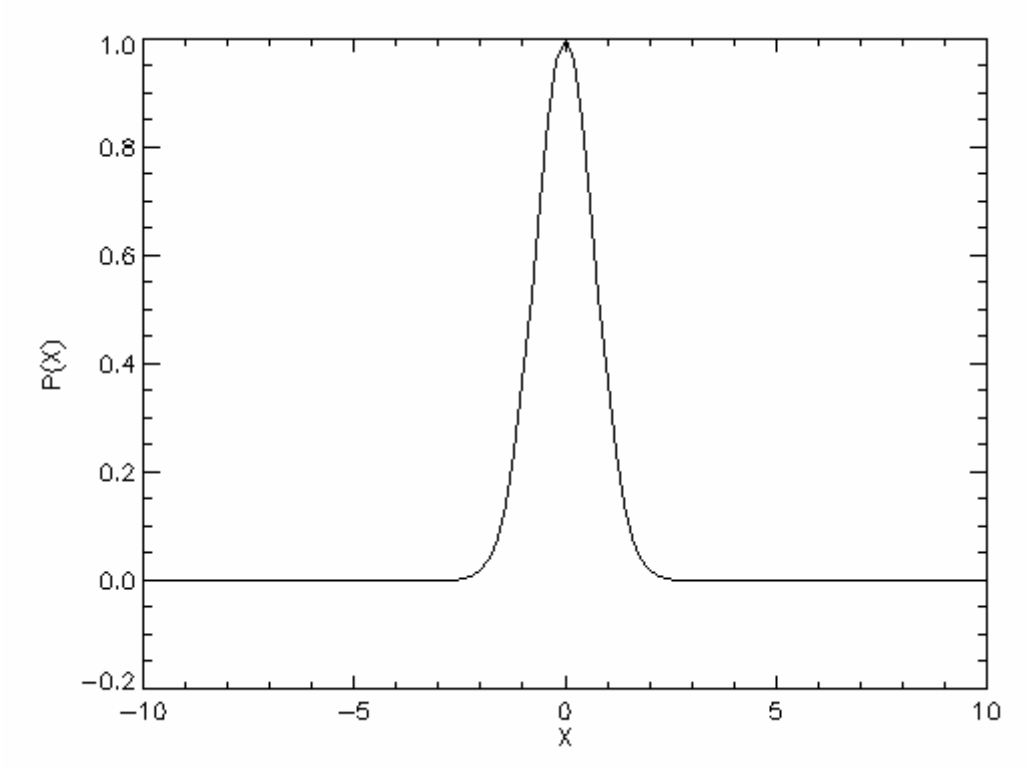

Histograme de 10^6 valeurs pseudo-aleatoires
1200
1000
800
600
400
200
0
N
<X>=0.49998173
Sigma^2=0.083276304
0.00
0.02
0.04
0.06
0.08
0.10
X

=>


Histogram, Lorentzienne
3.0×10⁴
2.5×10⁴
2.0×10⁴
1.5×10⁴
1.0×10⁴
5.0×10³
0
−100
−50
0
50
100

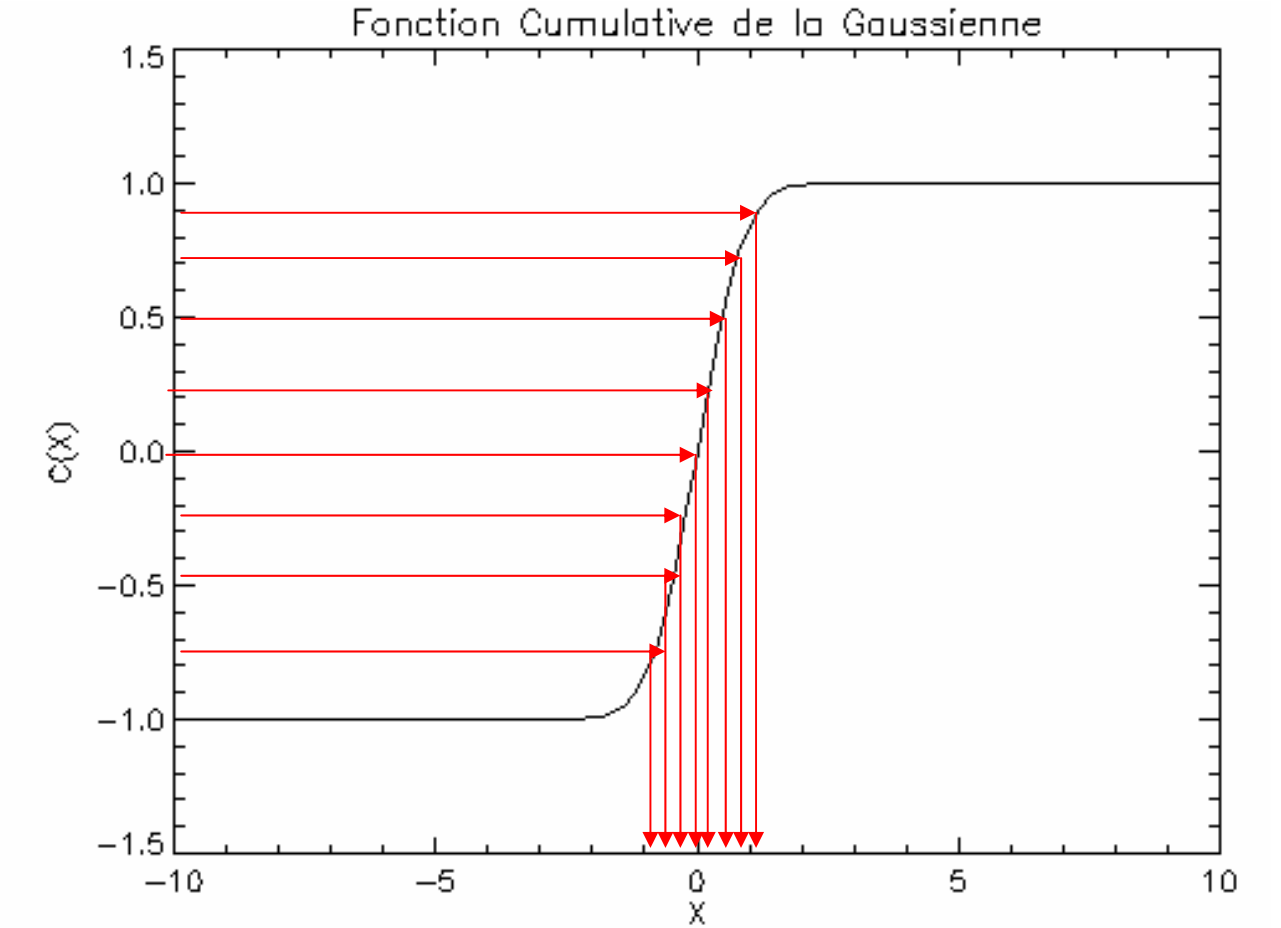
Approche Graphique: Soit C la distribution cumulative de p

P(x)= Gaussienne



Exemple de la Gaussienne … Mais attention il y aura un petit problème

On prend des points $\{x_i\}$ distribués uniformément entre -1 et 1



Sur l'axe de X on obtient les points $\{Y_i\}$ distribués selon une gaussienne

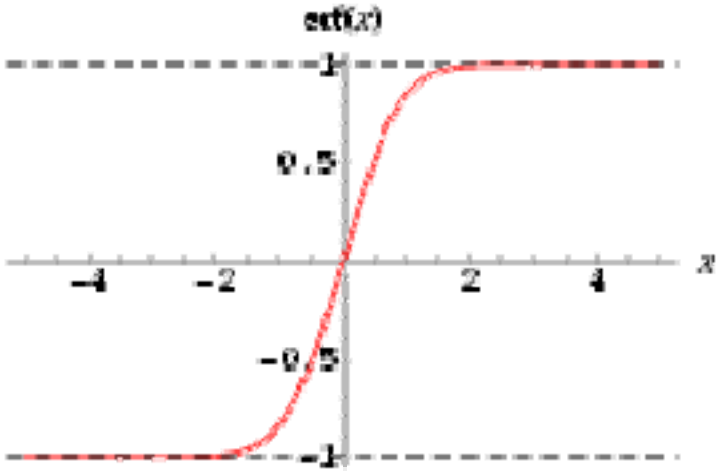
Graphiquement cela marche bien , mais analytiquement… problème !

Tout cela est très bien … mais pour de nombreuses fonctions on ne sait pas calculer $C^{-1}(x)$

Exemple : Pour la Gaussienne :

$$p(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} \Rightarrow$$

$$C(y) = \int_{-\infty}^{y} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} dy \propto ERF(y)$$



Où Erf est la fonction « Erreur »…
Non analytique… donc non inversible analytiquement …

Autre exemple : Distribution Gamma:

$$p(x) = \frac{x^{a-1}e^{-x}}{\Gamma(a)}$$

On n'a pas d'expression de C(x) analytique … encore moins de C^{-1}

Etc…

Dans d’autres cas , on connaît C mais on n’est pas capable de calculer C^{-1}

Exemple $C(x)=a0+a1\ X + a2\ X^2 + a3\ X^3 + \ldots + aN\ X^N$

On polynôme d’ordre elevé (> 5) n’est pas analytiquement inversible…
Et même un polynôme d’ordre 3 est difficilement inversible

Dans ces deux cas là on utilisera une méthode de REJECTION

Que nous n’étudierons pas cette année…

Pour le cas particulier de la distribution Gaussienne, il existe un algorithme courant … qui genère deux variables aléatoires.*

Méthode de : *Box-Muller*

Soient x_1 et x_2 deux variables uniformes entre 0 et 1

Considérons les transformation suivantes :

$$y_1 = \sqrt{-2\ln(x_1)}\cos(2\pi x_2)$$

$$y_2 = \sqrt{-2\ln(x_1)}\sin(2\pi x_2)$$

On peut montrer que Y1 et Y2 sont distribués selon un distribution gaussienne de variance = 1

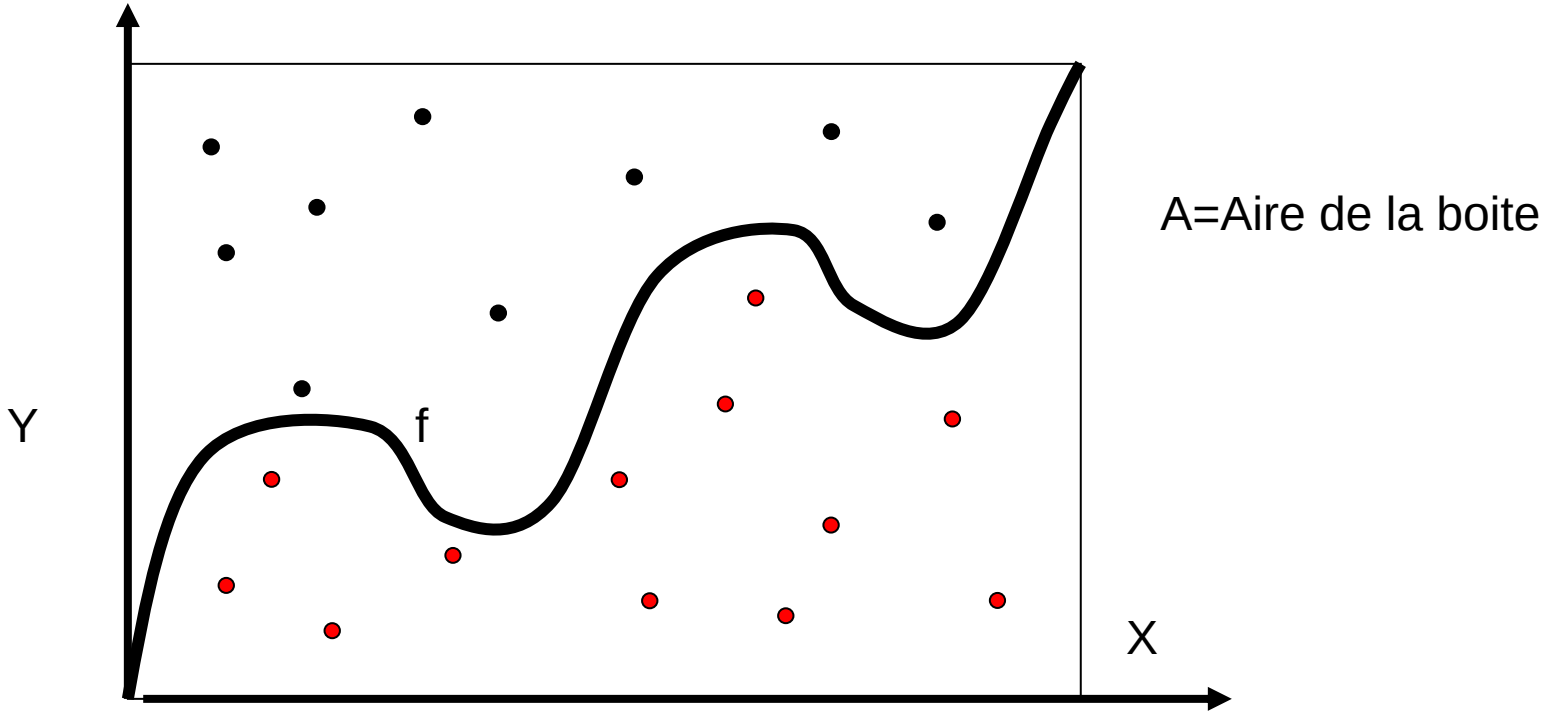
=> La norme aussi : $(Y1^2+Y2^2)^{1/2}$ => utile si on a besoin d'un seul chiffre

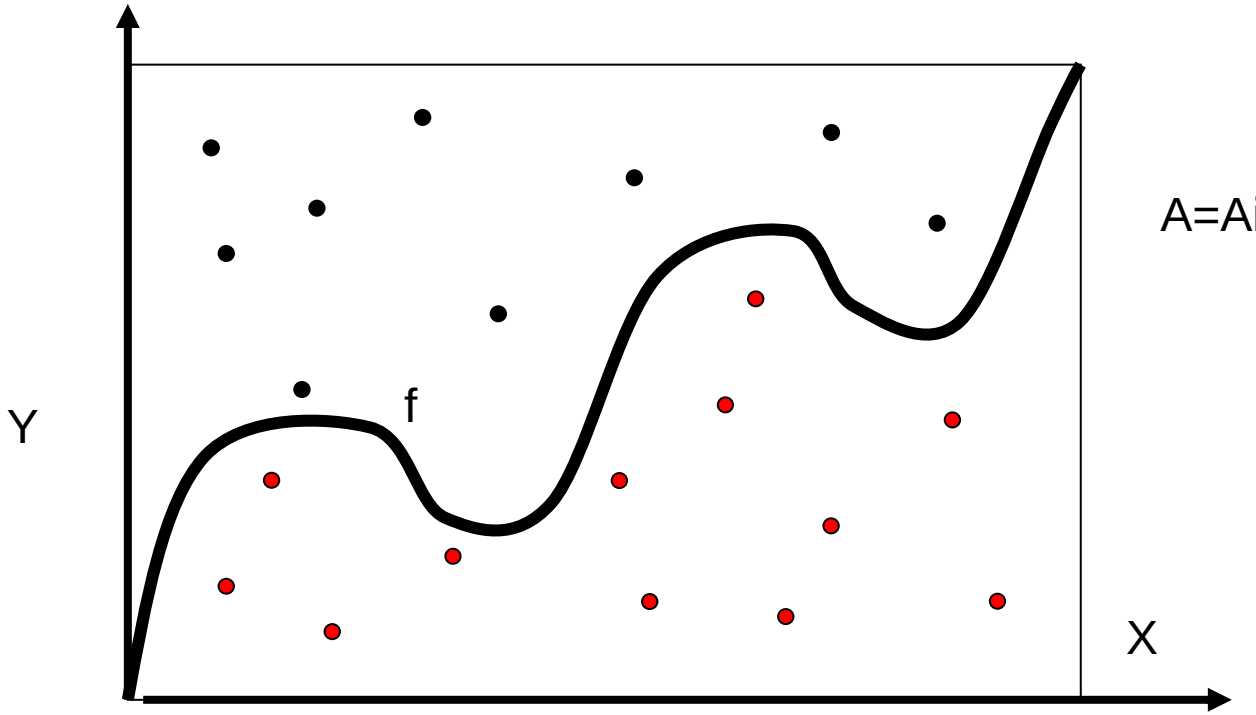
INTEGRATION MONTE-CARLO

Les nombres aléatoires servent aussi à intégrer des fonctions ….

L'intégration « Monte Carlo » est en général intéressante pour une intégrale multi-dimensionnelle (N>~3)

Idée de base : Si on tire N points au hasard le nombre de points sous la courbe est proportionnelle à l'aire de celle -ci





A=Aire de la boite

L'intégrale de f à l'intérieure de la boite (entre X_{min} et X_{max}) est

$$\int_{X\,\min}^{X\,\max} f(x)dx \approx \frac{N_{sous\,F}}{N_{total}} \times A$$

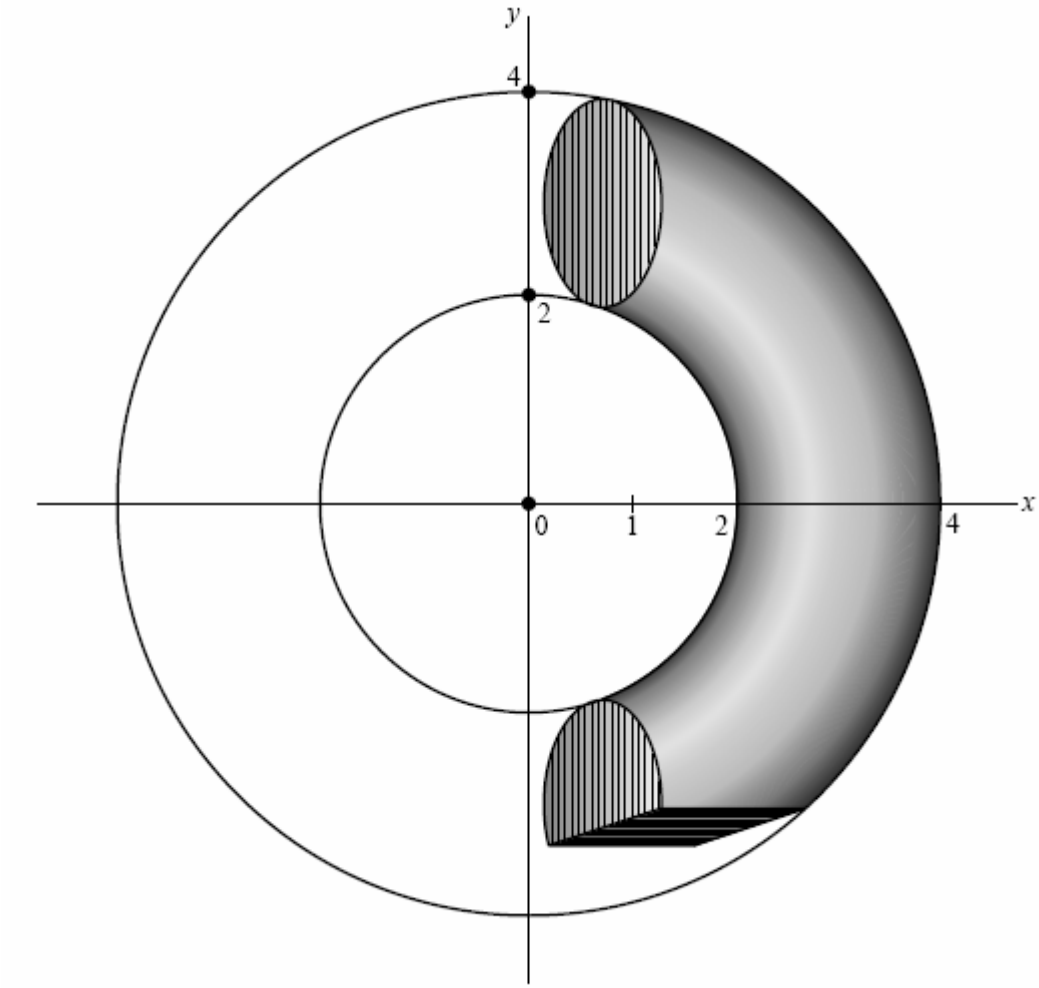
En fait l'intégration Monté Carlo est aussi intéressante si le domaine d'intégration est difficile à calculer (la fonction peut-être très simple):

Exemple :

$$z^2 + \left(\sqrt{x^2 + y^2} - 3\right)^2$$

$$x \geq 1$$

$$y \geq -3$$



F(x)=1

On veut calculer le volume :

$$V = \iiint_v dxdydz$$

On peut donc considérer un volume simple qui entoure le morceau de tore et compter le nombre de points qui tombent à l'intérieure…

MAIS cette méthode est très inefficace : il faudra tirer beaucoup de points pour converger vers le résultat…

Voici une méthode plus efficace :

On veut Calculer

$$I = \int_A f(x)dx$$

Sur un domaine compliqué A, x et dx sont multi-dimensionnels

On se donne un domaine plus simple, D qui englobe A
Soit V le volume de D

On réécrit l'intégrale :

$$I = \int_A f(x)dx$$

Comme : $$I = \int_A f(x)dx = \int_D f(x)g(x)dx$$

$$où \begin{cases} g(x) = 1 \text{ si x} \in \text{A} \\ g(x) = 0 \text{ si x} \notin \text{A} \end{cases}$$

Que l'on peut encore réécrire :

$$I = \int_D f(x)g(x)dx = \frac{1}{V}\int_D f(x)g(x)Vdx$$

Où V est le volume du domaine D.

Pourquoi cette transformation ?

Écrit sous cette forme :

$$I = \int_A f(x)dx = \frac{1}{V}\int_D f(x)g(x)Vdx$$

On voit que I est la valeur moyenne de h(x)=f(x)g(x)V sur le domaine D

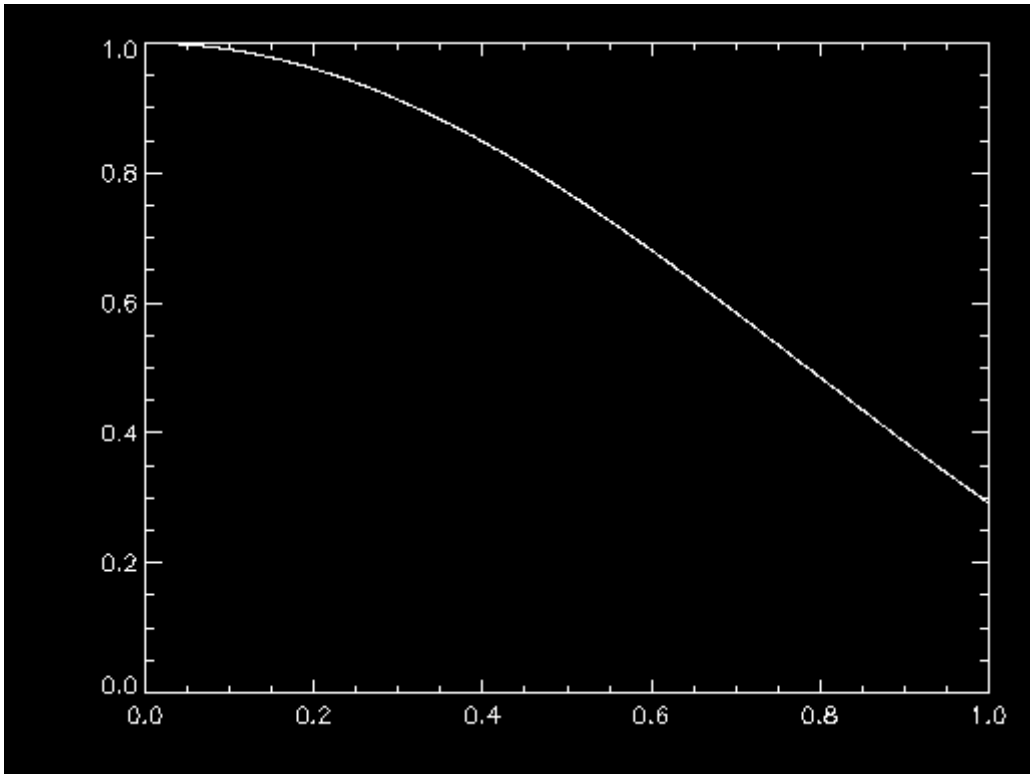
Donc maintenant si $\{x_i\}$ sont des points uniformément distribués dans D
Alors

$$I \approx \frac{1}{N}\sum_i h(x_i) = \frac{V}{N}\sum_i f(x_i)g(x_i)$$

Voilà un algorithme qui converge beaucoup plus rapidement que de tirer des points sous la courbe.
CEPENDANT : pour le calcule de g(x) , on doit toujours tester si x appartient au domaine

Exemple :

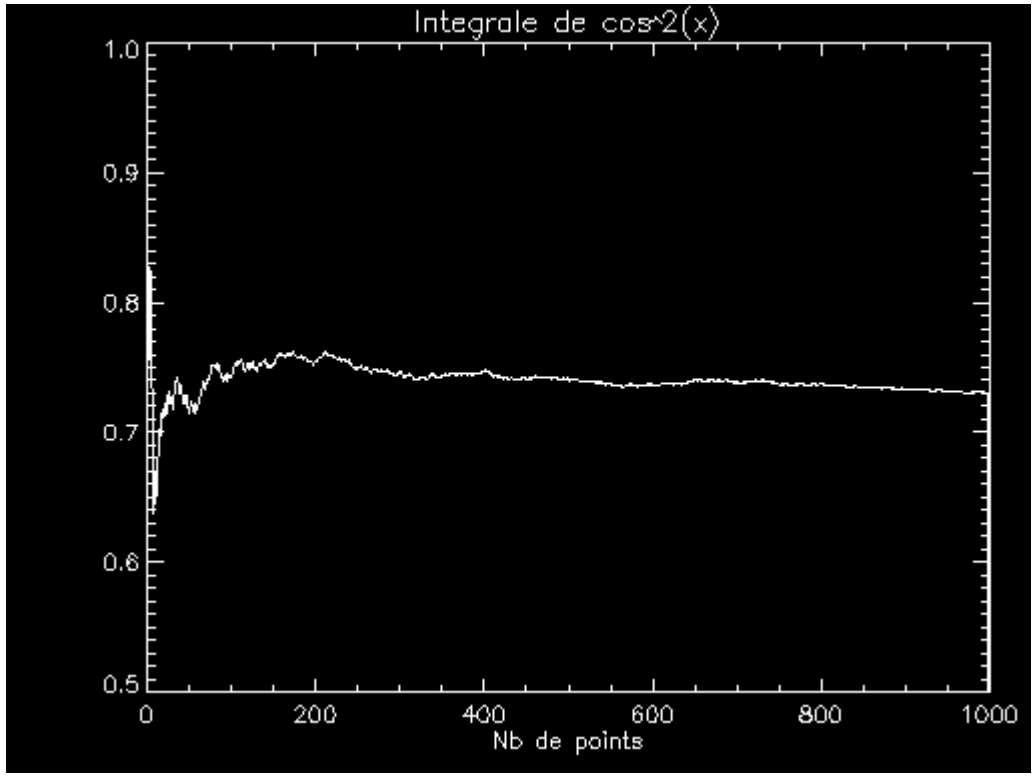
Intégrons $f(x)=\cos(x)^2$ sur l'intervalle 0-1



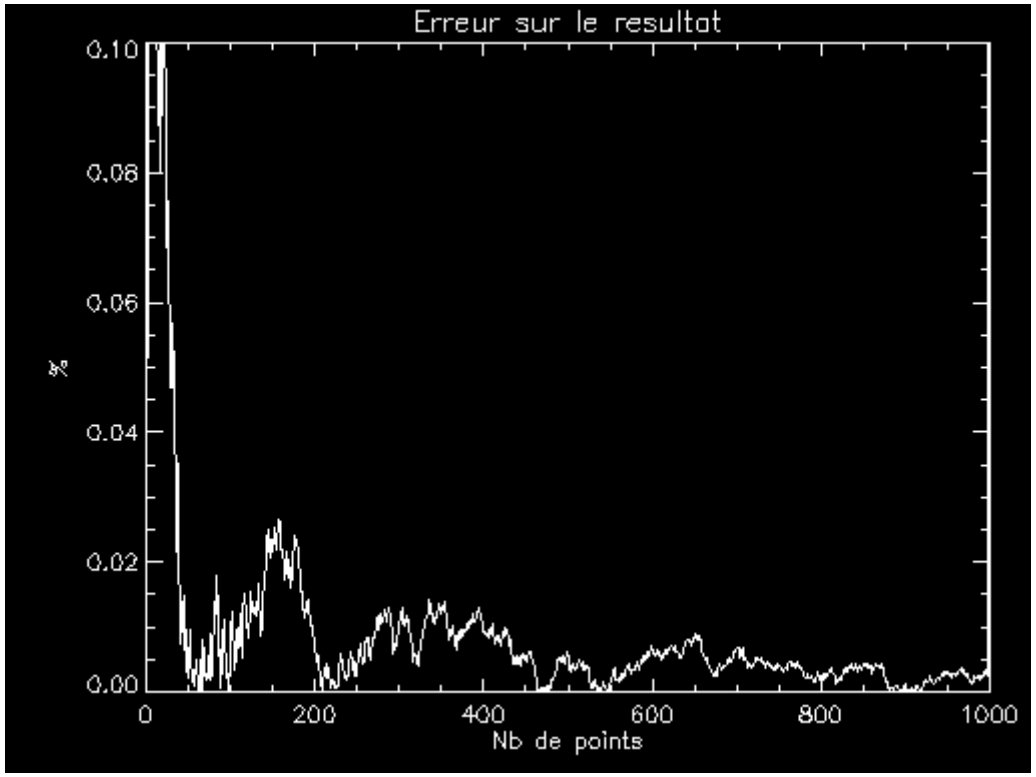
Le résultat exacte est : 0.72729516…..

$$I \approx \frac{V}{N} \sum_i f(x) g(x_i)$$

V = longueur de l'intervalle : 1
g(x) =1 si on tire uniquement des points entre 0 et 1
F(x)=cos(x)^2



L'erreur sur le résultat diminue avec le nb de points



Voici l'erreur sur le résultat en fonction du nombre de points tirés

On peut montrer que souvent , l'erreur décroit en $\frac{1}{\sqrt{N}}$

Donc pour avoir une erreur 2 fois plus faible , il faut 4 fois plus de points !!!

=> pour 50 points l'erreur (relative) est environ 0.05
⇒pour avoir une erreur de seulement 0.025 il faut environ 200 points
⇒Et pour erreur~0.01 il fait environ 800 points …..

En conclusion :

L’intégration par méthode Monte-Carlo permet d’intégrer des fonctions
-Complexes
- Sur des domaines compliquées
- A plusieurs dimensions.