Compléments en C++ & MATLAB

Sylvain Barthélémy, TAC

sylvain.barthelemy@tac-financial.com

2014

Plan du Cours

- Introduction
- Calcul numérique en C++
- Impact d'un choc sur la pauvreté avec MATLAB
- Un self organizing map sur des variables de logement
- Calcul de risque pays en C++
- Estimer un CAPM avec MATLAB
- Un réseau de neurones en C++
- Réseau de neurones sur les prix du logement en MATLAB
- Un algorithme génétique en C++
- Débogage en C++ d'un réseau de neurones

Introduction

Pourquoi MATLAB et C++ ?

- Un statisticien doit aujourd'hui connaitre un certain nombre de langages de programmation « fondamentaux ».
- Il faut aussi savoir évoluer avec l'informatique et pouvoir apprendre de nouveaux langages rapidement.
- L'apprentissage du C++, langage « de base », permet d'apprendre les rudiments de la programmation. Comme on apprend une grammaire.

Savoir programmer aujourd'hui

- On n'attend plus la même chose d'un programmeur qu'il y a 20 ans.
- Il doit avant tout savoir quoi chercher et quel langage est le plus adapté à la résolution de son problème.
- Connaitre la syntaxe « par cœur » est un moins important. En revanche, savoir assembler habilement des bouts de codes, éventuellement issus de plusieurs langages est essentiel.

Les alternatives ?

- Autres langages du même type: R, SAS, Ox,…
- Les programmes « presse-bouton »: RapidMiner, Tanagra, …
- Les « scripting languages » : Perl, Python, Ruby, PHP, Ruby, C#…
- Autres: Java, Fortran, …

Télécharger C++ et MATLAB

- Une version étudiant de MATLAB est téléchargeable sur

  http://www.mathworks.com/downloads/student_downloads

- Plusieurs versions de compilateur C++ disponibles:
  - GNU C++ sous Linux
  - Borland C++ sous Windows, téléchargeable gratuitement sur

    http://www.embarcadero.com/fr/products/cbuilder/free-compiler
  - Orwel Dev-C++ sous Windows (avec MingGW)

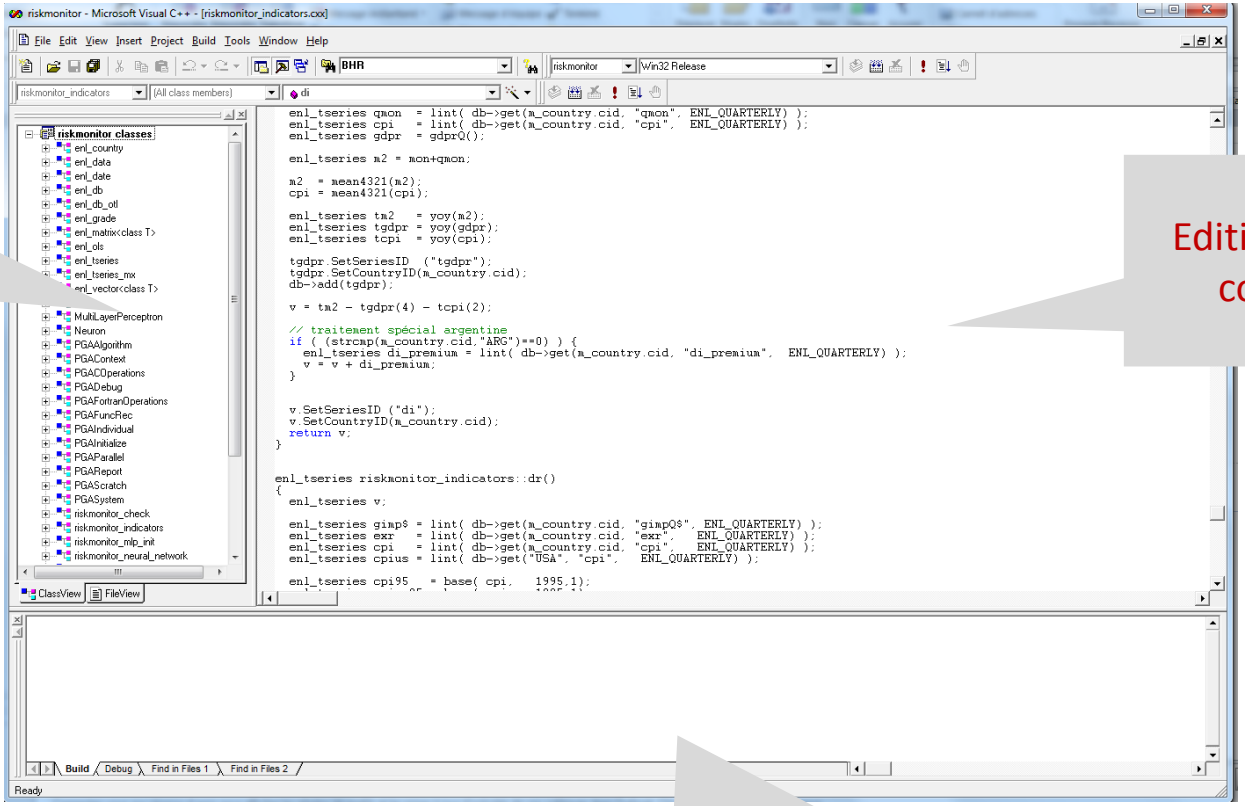
Différences C++ / MATLAB

C++	MATLAB
- Langage compilé - Norme ANSI, mais en pratique, des variantes entre compilateurs - Plusieurs version disponibles - Performances très élevées - Difficile d'accès et pas de version « all in one » - Objets mathématiques disponibles dans des librairies tierces - Contrôle total sur l'intégralité du code possible - …	- Langage interprété - Langage figé - Une seul version de MathWorks (quelques copies disponibles, comme octave par ex) - Performances moyennes - Facile d'accès et beaucoup de modules disponibles - Objets mathématiques intégrés au langage - Contrôle total impossible… - …

Installer C++?

- Au contraire de MATLAB, il n'y a pas de procédure universelle et vous devez choisir un compilateur, un éditeur et un débogueur.
- Deux grands types d'environnements:
  - Compilateur, éditeur, débogueur intégrés dans le même environnement (ex: version Microsoft). Simples d'utilisation, pas de makefile, debogage (plus) facile, …
  - Compilateur, éditeur, débogueur dans des modules séparés et pas forcément du même éditeur (par exemple sous Linux)

Environnement



Accès direct aux objets et fichiers

Edition du code

Résultat de la compilation, debug, sorties…

Calcul numérique en C++

Calcul numérique en C++

Beaucoup de librairies déjà disponibles:

- Numerical receipies: librairies C et C++.
- BLAS et Lapack: lib d'algèbre linéaire optimisées pour processeurs Intel ou AMD.
- Librairies de calcul parallèle: MPI, PVM,…
- Possibilité aujourd'hui de louer du temps de calcul sur des clouds (Sun, Amazon,…)

Calcul numérique en C++

- Exemple de programmation d'une librairie simple d'interpolation.
- Très utile en économie sur les séries temporelles avec quelques données manquantes.
- Trois méthodes implémentées: interpolation de Lagrange, méthode des différences et cubic splines.

Interpolation de Lagrange
18è siècle…

- Vous disposez de N+1 points initiaux.
- La méthode de Lagrange propose de choisir N points parmi les N+1 et de résoudre le système de N équations à N inconnues formé par les équation polynomiales passant par ces points.
- Si vous disposez de 4 points, vous rechercherez donc un polynôme de degré 3 passant par ces points.
- Difficulté: on ne choisi pas le degré du polynome.

Interpolation de Lagrange en C++

```
double lagrange(const double& u, const double* x, const double* y, const int& n)
{
  double interp = 0.0;

  for(int i=0; i < n; i++) {
    double p = 1.0;
    for(int j=0; j < n; j++) {
      if(i!=j) p *= (u-x[j])/(x[i]-x[j]);
    }
    interp += p * y[i];
  }

  return interp;
}
```

Méthode des différences

- Peut donner des résultats similaires à la méthode de Lagrange.
- Mais remplace avantageusement cette dernière lorsqu'il est possible d'ajuster un polynôme de degré inférieur sur les points à interpoler.

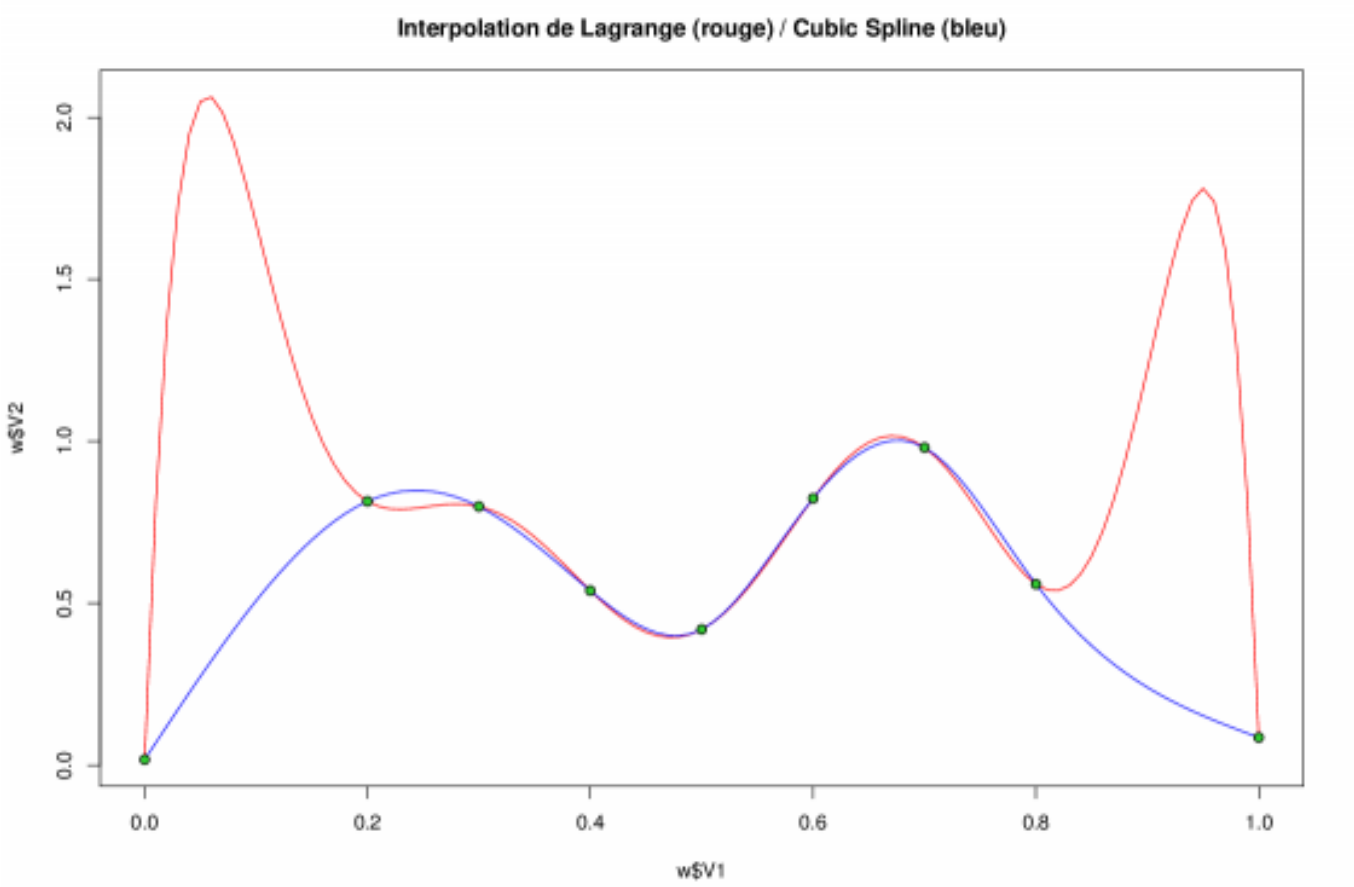
Méthode des différences en C++

```
void divdiff_coef(double* dd, const double* x, const double* y, const int& n)
{
  double temp1, temp2;
  for ( int i=0; i < n; i++ )
    dd[i] = y[i];
  for ( int j=1; j < n; j++ ) {
    temp1 = dd[j-1];
    for ( int k=j; k < n; k++ ) {
      temp2 = dd[k];
      dd[k] = (dd[k] - temp1)/(x[k] - x[k-j]);
      temp1 = temp2;
    }
  }
}

double divdiff(const double& u, const double* dd, const double* x, const int& n)
{
  double interp = 0.0;
   for ( int i=(n-1); i>=1; i-- )
      interp = (interp + dd[i]) * (u - x[i-1]);
   interp += dd[0];
  return interp;
}
```

Méthode des cubic splines

- A la différence de la méthode précédente, le cubic spline n'est pas une méthode d'interpolation polynomiale à proprement parler
- Nous cherchons plutôt à lisser une série de points, un peu comme on le ferai avec une moyenne mobile, et non pas à trouver quel polynôme représente le mieux ces points.
- Effectuer une interpolation par un cubic spline revient à résoudre un système tri-diagonal. Comme pour la méthode des différences, on recherche tout d'abord les valeurs des coefficients avant de calculer les points interpolés.
- Portage en C++ adapté des Numerical Recipies in C

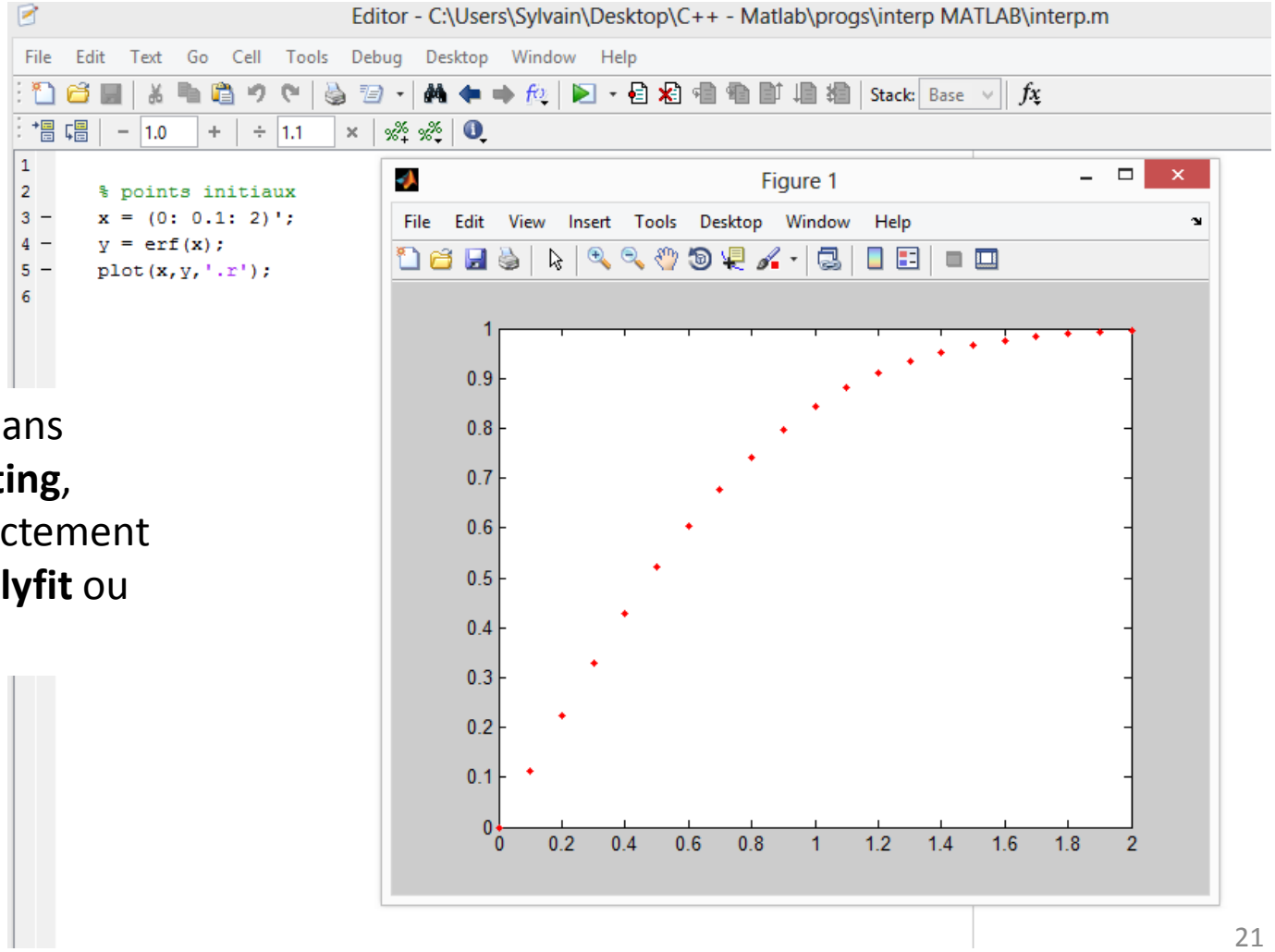
Comparaison des méthodes



Structure d’une librairie d’interpolation?

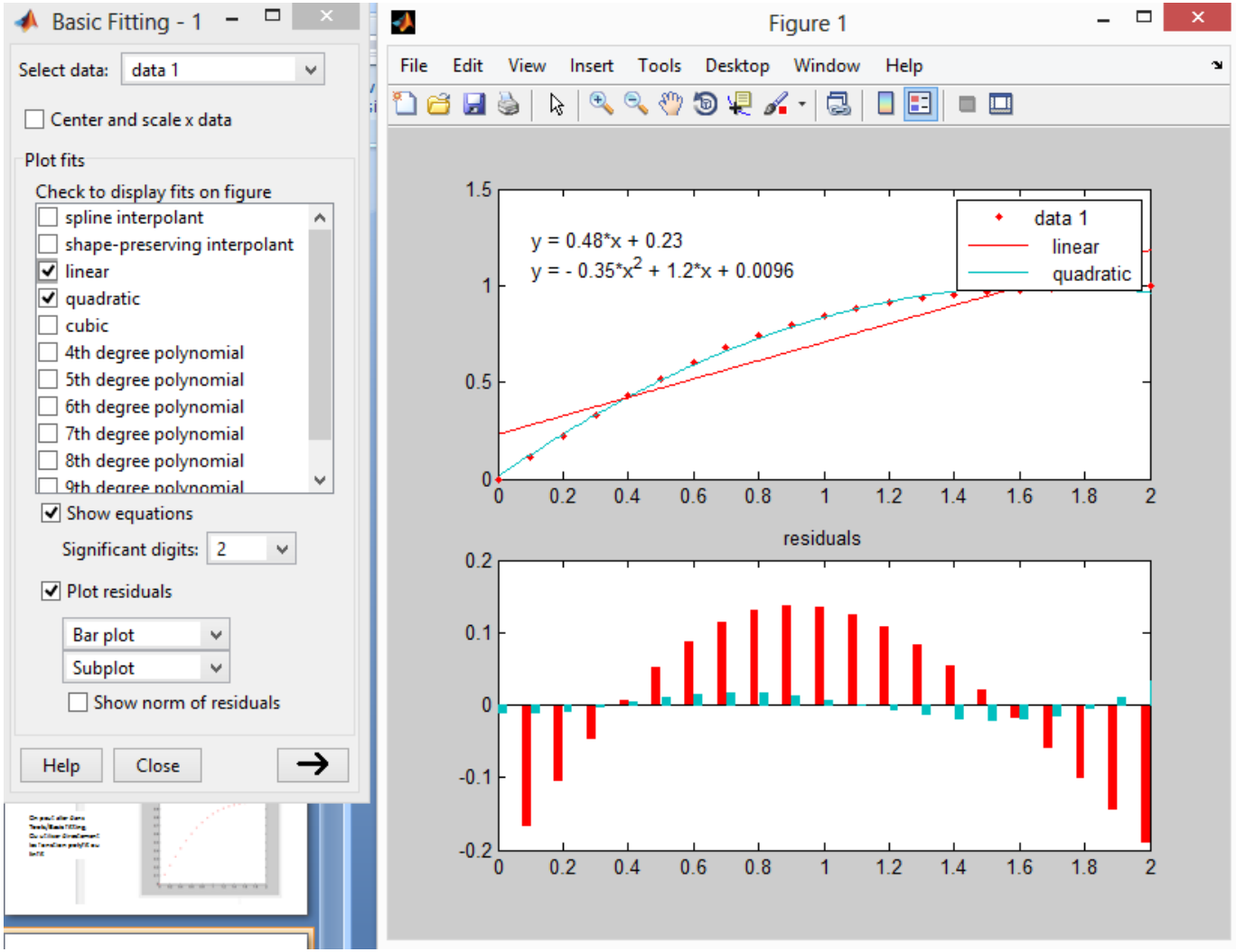
- Présentation du corps de la librairie.
- Le fichier d’entête
- L’objet enl_interp
- Comment l’utiliser dans un autre programme?

Et en MATLAB ?



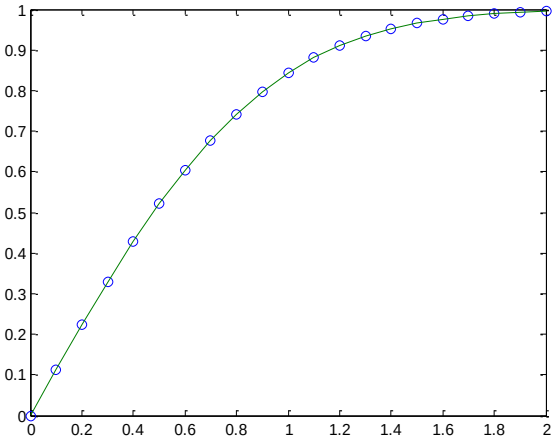
On peut aller dans **Tools/Basic fitting**,
Ou utiliser directement les fonction **polyfit** ou **linfit**

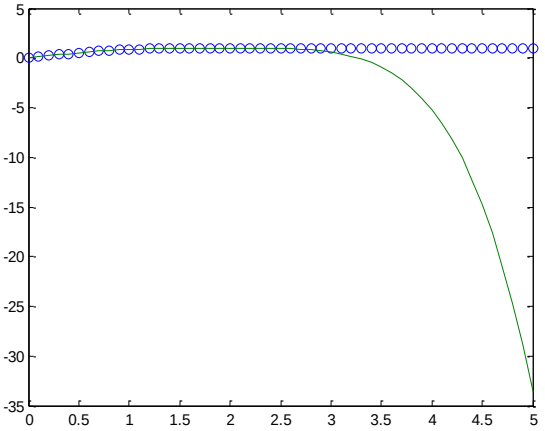
Et en MATLAB ?


Basic Fitting - 1
Select data: data 1
Center and scale x data
Plot fits
Check to display fits on figure
spline interpolant
shape-preserving interpolant
linear
quadratic
cubic
4th degree polynomial
5th degree polynomial
6th degree polynomial
7th degree polynomial
8th degree polynomial
9th degree polynomial
Show equations
Significant digits: 2
Plot residuals
Bar plot
Subplot
Show norm of residuals
Help
Close
Figure 1
File Edit View Insert Tools Desktop Window Help
y = 0.48*x + 0.23
y = - 0.35*x^2 + 1.2*x + 0.0096
data 1
linear
quadratic
residuals

Et en MATLAB ?

```
% points initiaux
x = (0: 0.1: 2)';
y = erf(x);
plot(x,y,'.r');

% un polyfit semble bien marcher
p = polyfit(x,y,6);
f = polyval(p,x);
plot(x,y,'o',x,f,'-');

% attention... si on va plus loin !
x = (0: 0.1: 5)';
y = erf(x);
f = polyval(p,x);
plot(x,y,'o',x,f,'-');
```





Impact d’un choc sur la pauvreté avec MATLAB

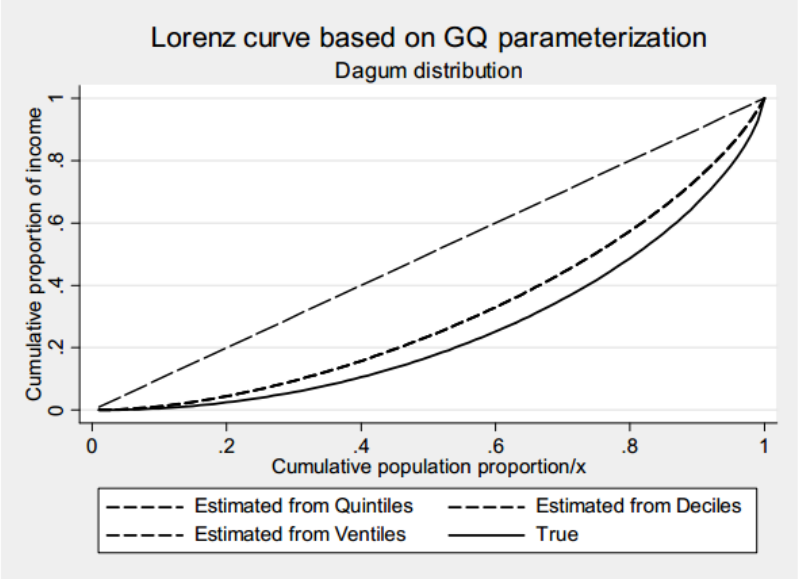
Le programme Povcal

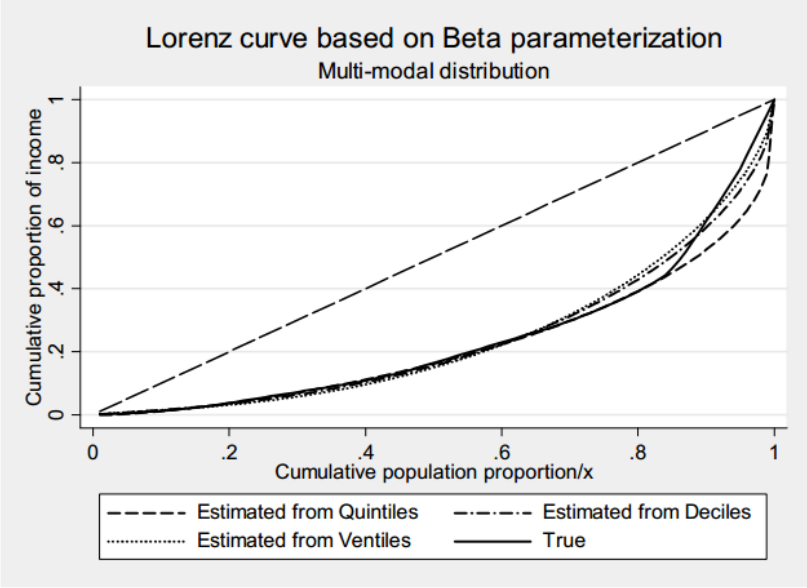
- Calculs effectués par la Banque Mondiale dans le cadre d'un programme de réduction de la pauvreté dans les pays en voie de développement.
- A donné lieu à la création d'un site web (PovcalNet) permettant de retrouver des informations sur les inégalités et les distributions de revenus.

Distributions de revenus avec MATLAB

- Estimation des courbes de Lorenz sur la base de deux méthodes:
  - GQ de Villesenor & Arnold (1989)
  - Betas de Kakwani (1980)
- Les données utilisées sont les parts de revenus, les revenus moyens des quantiles de population, les parts de populations dans certains quantiles,…
- Le programme produit: indice de gini, courbe de Lorenz, poverty headcount, poverty gap et élasticité de ces mesures au revenu moyen.

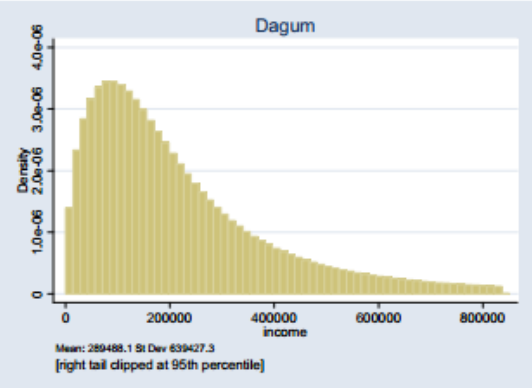
Estimation des courbes de Lorenz





Distributions de revenus théoriques

Dagum distribution



Multimodal (notional) distribution

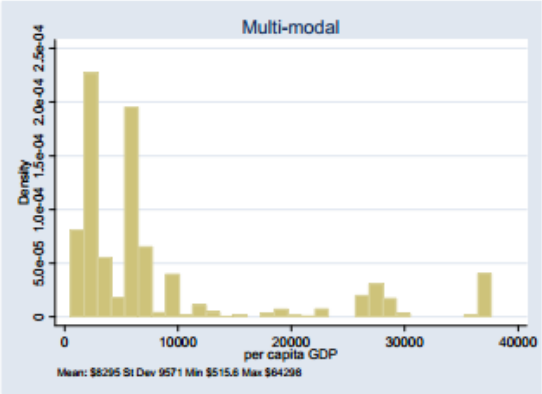
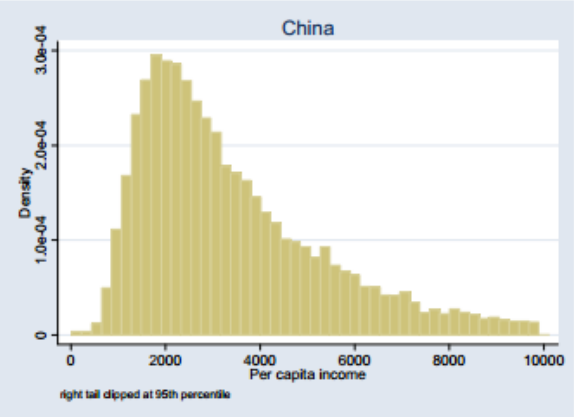
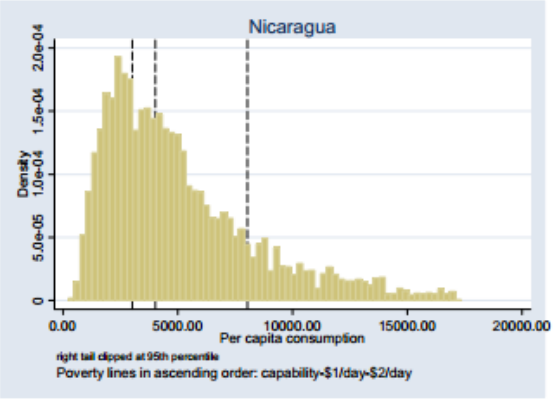


Figure 2. Household survey distributions used for deterministic comparisons

China (1995)

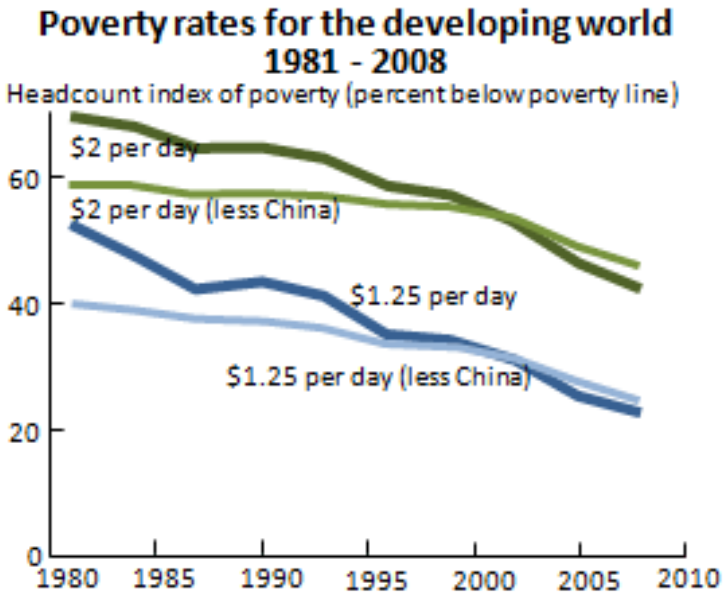


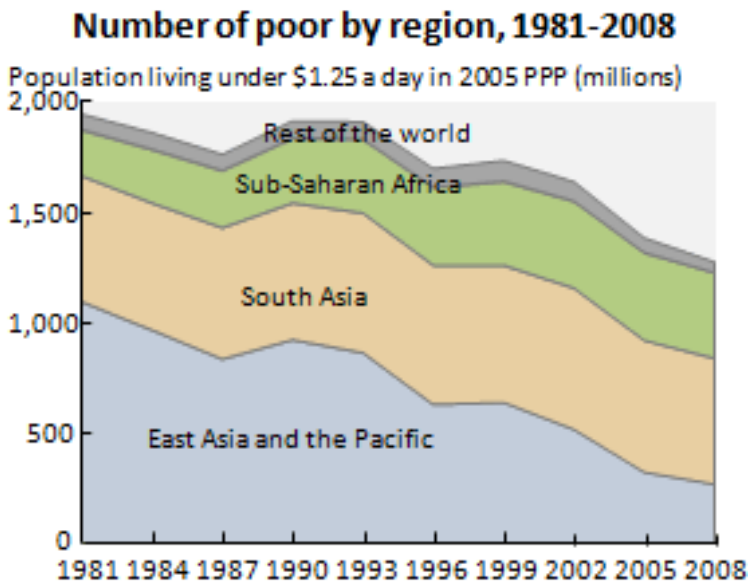
Nicaragua (1998)



Lois généralement Dagum plutôt que Pareto ou Lognormales

Distributions de revenus





SimSIP Poverty

- Combinaison de Povcal avec SimSIP et SimSIP Poverty, programmes de la Banque Mondiale en Excel et MATLAB sur les tableau entrées/sorties.
- Permet par exemple d'estimer l'impact probable d'un choc exogènes (ex: choc prix du pétrole= sur les inégalités/la pauvreté dans un pays ou un secteur donné.
- Téléchargeable gratuitement sur:

  http://simsip.org/IOs SAMs.html

SimSIP Poverty

- Combinaison de Povcal avec SimSIP et SimSIP Poverty, programmes de la Banque Mondiale en Excel et MATLAB sur les tableau entrées/sorties.
- Permet par exemple d'estimer l'impact probable d'un choc exogènes (ex: choc prix du pétrole= sur les inégalités/la pauvreté dans un pays ou un secteur donné.
- Téléchargeable gratuitement sur:

  http://simsip.org/IOs SAMs.html

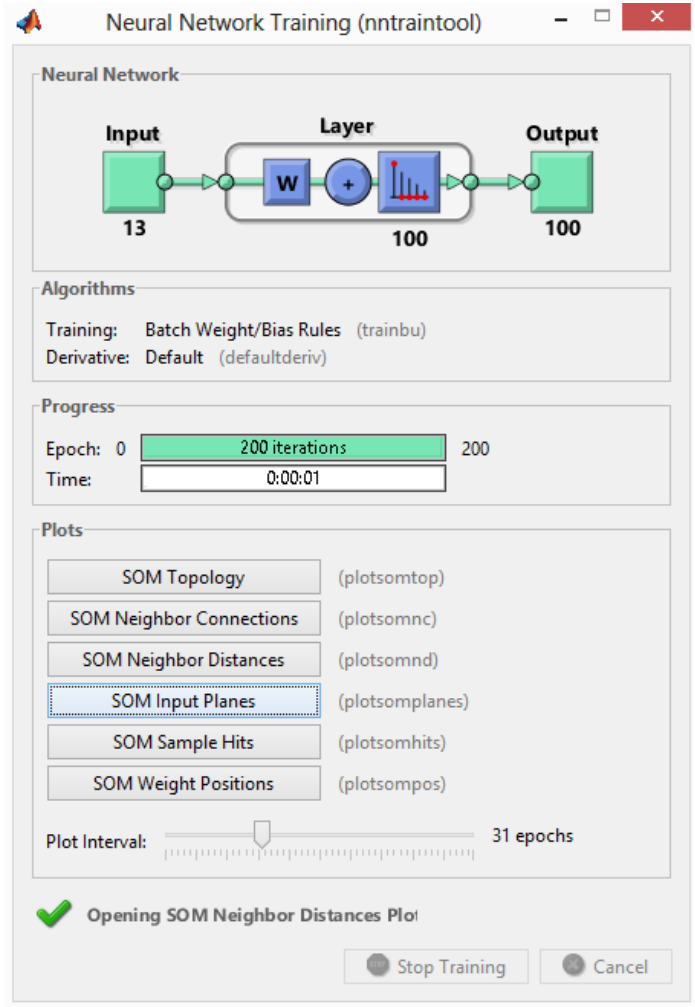
Un self organizing map sur des variables de logement

Les variables (dataset MATLAB)

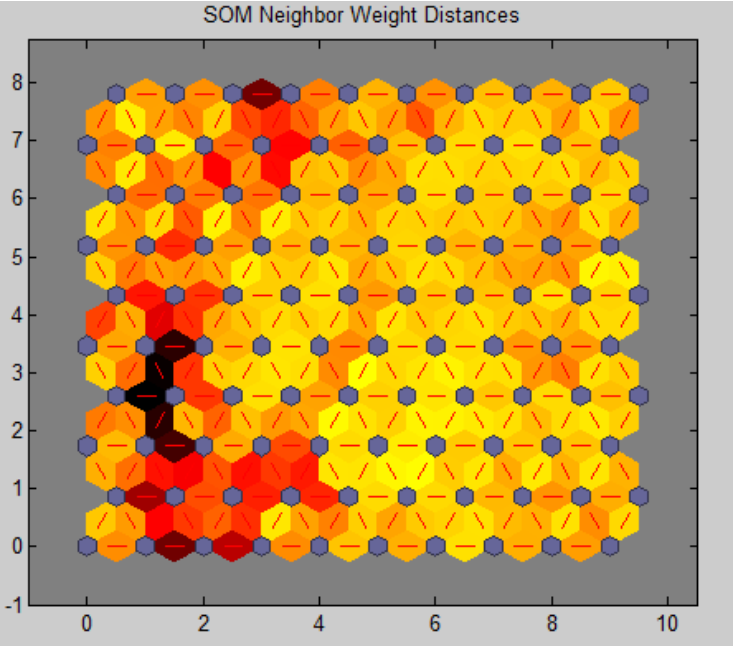
1. Per capita crime rate per town
2. Proportion of residential land zoned for lots over 25,000 sq. ft.
3. proportion of non-retail business acres per town
4. 1 if tract bounds Charles river, 0 otherwise
5. Nitric oxides concentration (parts per 10 million)
6. Average number of rooms per dwelling
7. Proportion of owner-occupied units built prior to 1940
8. Weighted distances to five Boston employment centres
9. Index of accessibility to radial highways
10. Full-value property-tax rate per $10,000
11. Pupil-teacher ratio by town
12. 1000(Bk - 0.63)^2, where Bk is the proportion of blacks by town
13. Percent lower status of the population

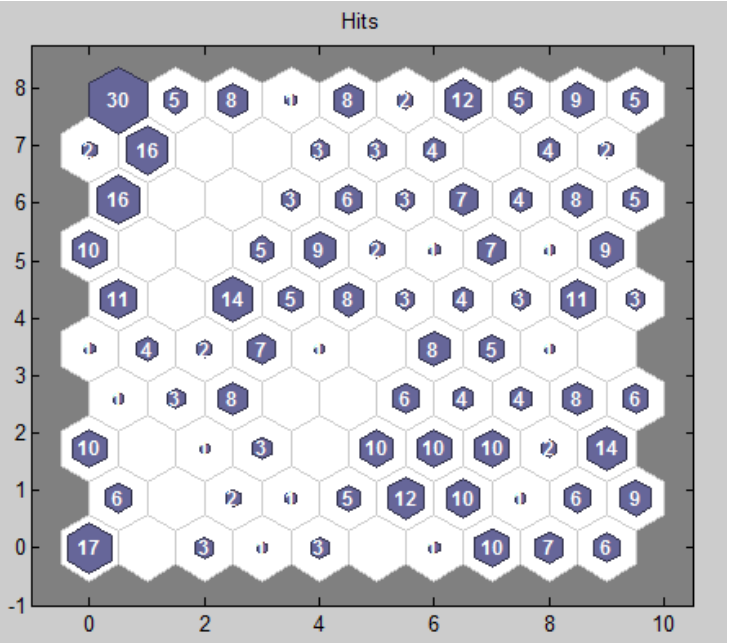
Le Self Organizing Map

```
% paramètres
load house_dataset;
inputs  = houseInputs;

% SOM
dimension1 = 10;
dimension2 = 10;
net = selforgmap([dimension1 dimension2]);
[net,tr] = train(net,inputs);
outputs = net(inputs);
```

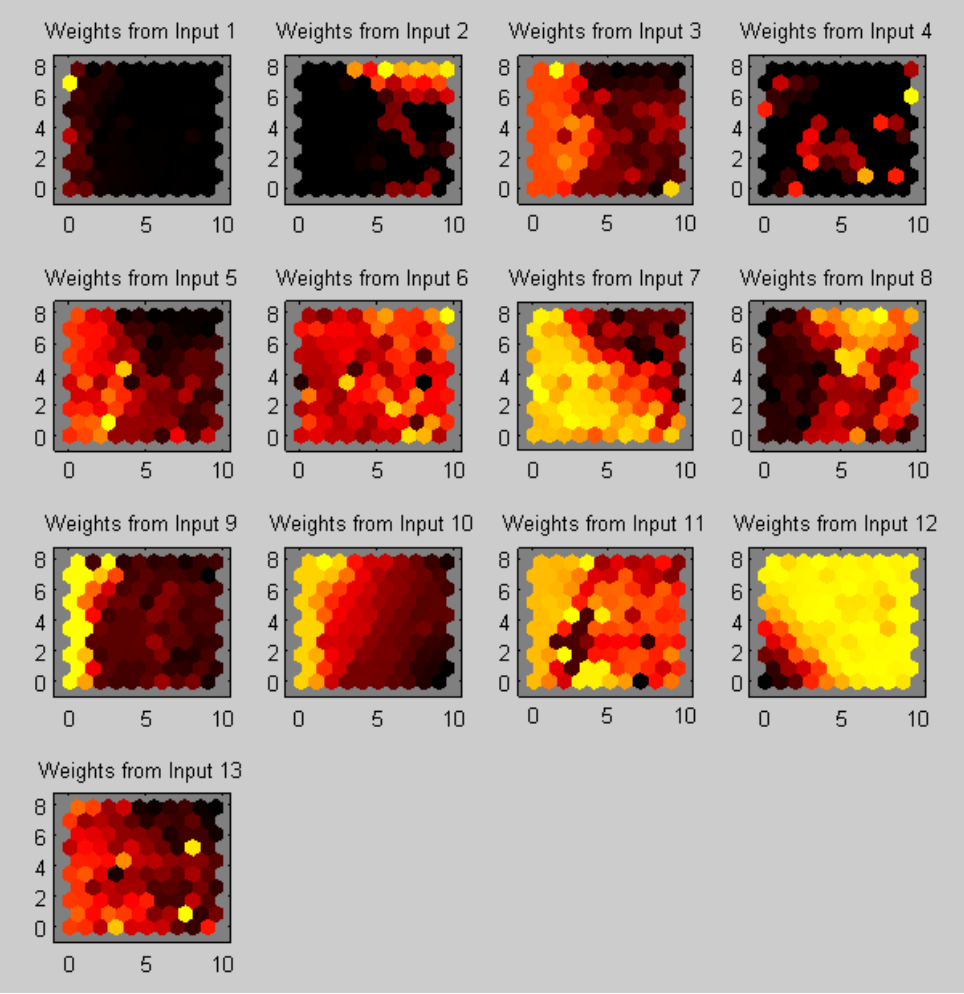


Les Résultats: distances & hitmap





Les Résultats: les « plans » d'entrées

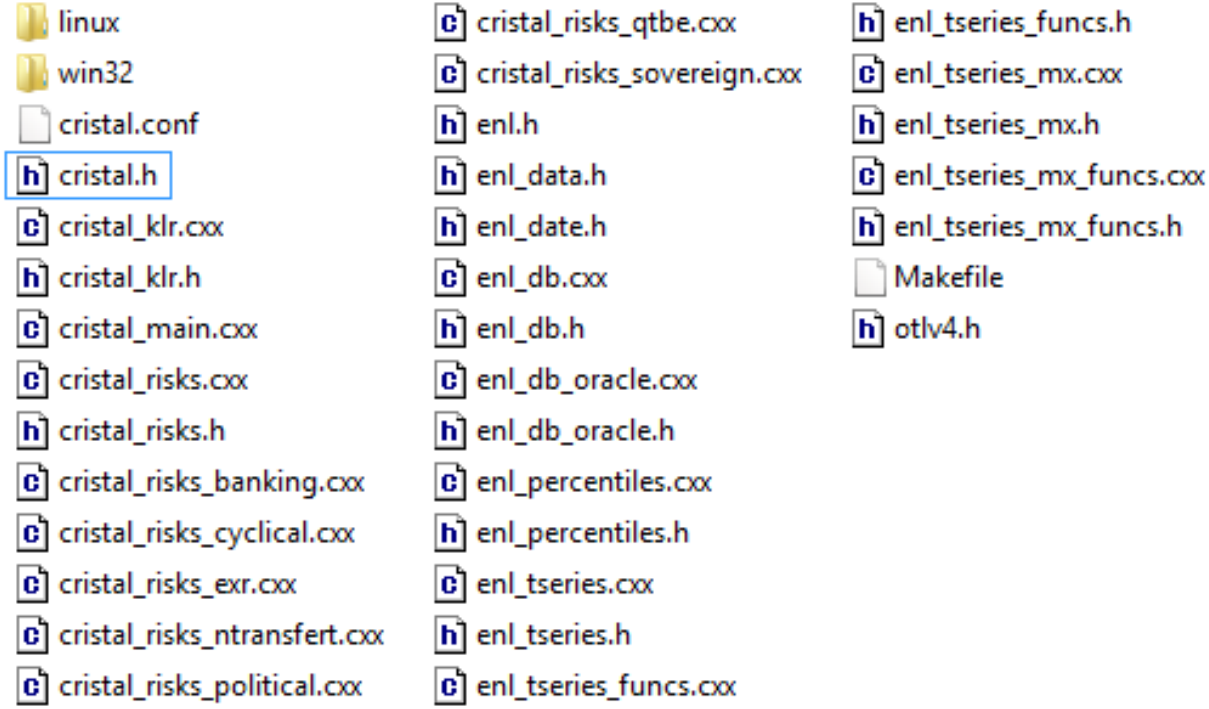


Calcul de risque pays en C++

Calcul de risque pays en C++

- Programme réalisé en C++.
- Projet d'intégration du programme directement dans le système informatique d'un client.
- Version de développement sous Linux. Mise en production sous IBM/Aix avec une connexion Oracle pour le stockage.
- Structure et mode de fonctionnement…

Calcul de risque pays en C++



Utilisation de la surcharge d'opérateurs pour simplifier l'utilisation de séries temporelles

```
// --- opérateurs d'affectation
enl_tseries& operator=(const enl_tseries& v);
enl_tseries& operator=(const enl_data& t);

void resize (const long& r);

// --- opérateurs arithmétiques enl_tseries
enl_tseries operator+(const enl_tseries& v) const;
enl_tseries operator-(const enl_tseries& v) const;
enl_tseries operator*(const enl_tseries& v) const;
enl_tseries operator/(const enl_tseries& v) const;

enl_tseries operator+= (const enl_tseries& v);
enl_tseries operator-= (const enl_tseries& v);
enl_tseries operator*= (const enl_tseries& v);
enl_tseries operator/= (const enl_tseries& v);

// --- opérateurs arithmétiques double
enl_tseries operator+= (const double& d);
enl_tseries operator-= (const double& d);
enl_tseries operator*= (const double& d);
enl_tseries operator/= (const double& d);

friend enl_tseries operator+ (const enl_tseries& v, const double& d);
friend enl_tseries operator- (const enl_tseries& v, const double& d);
friend enl_tseries operator* (const enl_tseries& v, const double& d);
friend enl_tseries operator/ (const enl_tseries& v, const double& d);

friend enl_tseries operator+ (const double& d, const enl_tseries& v);
friend enl_tseries operator- (const double& d, const enl_tseries& v);
friend enl_tseries operator* (const double& d, const enl_tseries& v);
friend enl_tseries operator/ (const double& d, const enl_tseries& v);

// --- opérateurs d'accès
enl_tseries operator()(const long n) const;
enl_tseries forward(const long n) const;
enl_data&       operator[](const long n);
const enl_data& operator[](const long n) const;
```

Utilisation de la surcharge d'opérateurs pour simplifier l'utilisation de séries temporelles

```
// --- les seuils des centiles
std::vector<enl_grade> vp;
vp.push_back(enl_grade( 0.0,0.0));
vp.push_back(enl_grade(40.0,1.0));
vp.push_back(enl_grade(50.0,2.0));
vp.push_back(enl_grade(60.0,3.0));
vp.push_back(enl_grade(70.0,4.0));
vp.push_back(enl_grade(80.0,5.0));

// --- les données brutes
enl_tseries_mx a = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx b = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx c = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx d = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx e = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx f = db->getmx(cid, count, "xx",    ENL_QUARTERLY);
enl_tseries_mx g = db->getmx(cid, count, "xx",    ENL_QUARTERLY);

// --- les indicateurs
enl_tseries_mx i01 = 100.0 * e/c;
enl_tseries_mx i02 = 100.0 * e/d(1);

enl_tseries_mx cp        = c/(a+b) * 100.0;
enl_tseries_mx tcam_prix = pow(g  /g  (1), 3.0) - 1.0;
enl_tseries_mx tcam_cp   = pow(cp /cp (8), 0.5) - 1.0;
enl_tseries_mx tcam_i02  = pow(i02/i02(8), 0.5) - 1.0;

enl_tseries_mx i03 = f - 100.0*tcam_prix;
enl_tseries_mx i04 = 100.0 * tcam_cp;
enl_tseries_mx i05 = 100.0 * tcam_i02;

// --- conversion en percentiles
i01 = percentiles(i01, vp);
i02 = percentiles(i02, vp);
i03 = percentiles(i03, vp);
i04 = percentiles(i04, vp);
i05 = percentiles(i05, vp);
```

Utilisation d'OTL pour lire et écrire les données sous Oracle

```
  // --- prépare le where
  sprintf(szWhere,
      "countryID='%s' AND seriesID='%s' AND updateID=%i AND dataFreq=%i",
      cid, sid, m_updateID, freq);

  // --- compte le nombre de lignes
  long ndata;
  sprintf(sql, "SELECT count(*) FROM data WHERE %s", szWhere);
  otl_stream stcount(50,sql,db);
  stcount >> ndata;
  if(ndata==0) return tsempty;

  // --- lecture des lignes
//  sprintf(sql,"SELECT dataYear,dataSubp,seriesBaseYear,recodID,dataValue FROM data,series");
//  sprintf(sql, "%s WHERE (data.seriesID=series.seriesID) AND %s ", sql, szWhere);
//  sprintf(sql, "%s ORDER BY dataYear,dataSubp", sql);

  sprintf(sql, "SELECT dataYear,dataSubp,recodID,dataValue FROM data WHERE %s ORDER BY dataYear,dataSubp", szWhere);
  otl_stream strows(50,sql,db);

  // --- init du vecteur
  long byear;
  long bsubp;
  long recodID;
  strows >> byear;
  strows >> bsubp;
  strows >> recodID;
```

Un CAPM avec MATLAB

Retour rapide sur le CAPM

Le CAPM (MEDAF en fr) permet d'estimer le rendement attendu d'un actif en fonction du risque systématique , Sharpe (1964), Lintner (1965),...

Modèle estimé sur la base de

- $E(Ri) = Rf + beta * [E(Rm) - Rf]$

E(Ri) rendement anticipé de l'actif i, Rf taux sans risque, beta sensibilité des rendements de l'actif aux rendements de marché et E(Rm) return anticipé du marché.

Retour rapide sur le CAPM

Un CAPM peut par exemple être calculé sur la base de performances passées.

Le beta est défini comme le rapport entre la covariance entre les rendements de l'actif i et le marché et la variance du marché:

- $beta = cov(Ri, Rm)/Var(Rm)$

L'actif « bouge » avec le marché pour un beta égal à 1 et est plus sensible que le marché si le beta est supérieur à 1.

Estimation univariée sous MATLAB

```
load CAPMuniverse;

[NumPoints, NumAssets] = size(Data);
NumAssets = NumAssets - 2;

fprintf('\n\nCAPM/OLS estimated from %s to %s\n', ...
        datestr(Dates(1),1), datestr(Dates(end),1));
fprintf('%4s   %-9s %-9s\n', 'Asset', 'Alpha', 'Beta');

for i = 1:NumAssets
    rif = Data(:,i) - Data(:,14);
    des = zeros(NumPoints,2);
    des(:,1) = 1.0;
    des(:,2) = Data(:,13) - Data(:,14);
    [Param,Covar] = mvnrmle(rif,des,1);
    fprintf('%4s %9.4f %9.4f\n', Assets{i}, Param(1), Param(2));
end;
```

Estimation univariée sous MATLAB

CAPM/OLS estimated from 03-Jan-2000 to 07-Nov-2005

Asset	Alpha	Beta
AAPL	0.0012	1.2294
AMZN	0.0006	1.3661
CSCO	-0.0002	1.5653
DELL	-0.0000	1.2594
EBAY	0.0014	1.3441
GOOG	0.0046	0.3742
HPQ	0.0001	1.3745
IBM	-0.0000	1.0807
INTC	0.0001	1.6002
MSFT	-0.0002	1.1765
ORCL	0.0000	1.5010
YHOO	0.0001	1.6543

Estimation multivariée sous MATLAB

```
% estimation sur données groupées (valeurs techno)
NumParams = 2 * NumAssets;
rif = zeros(NumPoints,NumAssets);
des = cell(NumPoints,1);
tmp = zeros(NumAssets,NumParams);

for k = 1:NumPoints
    for i = 1:NumAssets
        rif(k,i) = Data(k,i) - Data(k,14);
        tmp(i,2*i-1) = 1;
        tmp(i,2*i) = Data(k,13) - Data(k,14);
    end
    des{k} = tmp;
end

load CAPMgroupparam;

[Param,Covar] = ecmmvnrmle(rif,des,[], [], [], Param0, Covar0);
Alpha = Param(1:2:end-1);
Beta  = Param(2:2:end);

fprintf('\n\nCAPM/Grouped Multivariate Normal estimated from %s to %s\n', ...
        datestr(Dates(1),1), datestr(Dates(end),1));
fprintf('%4s   %-9s %-9s\n', 'Asset', 'Alpha', 'Beta');
for i = 1:NumAssets
    fprintf('%4s %9.4f %9.4f\n', Assets{i}, Alpha(i), Beta(i));
end;
```

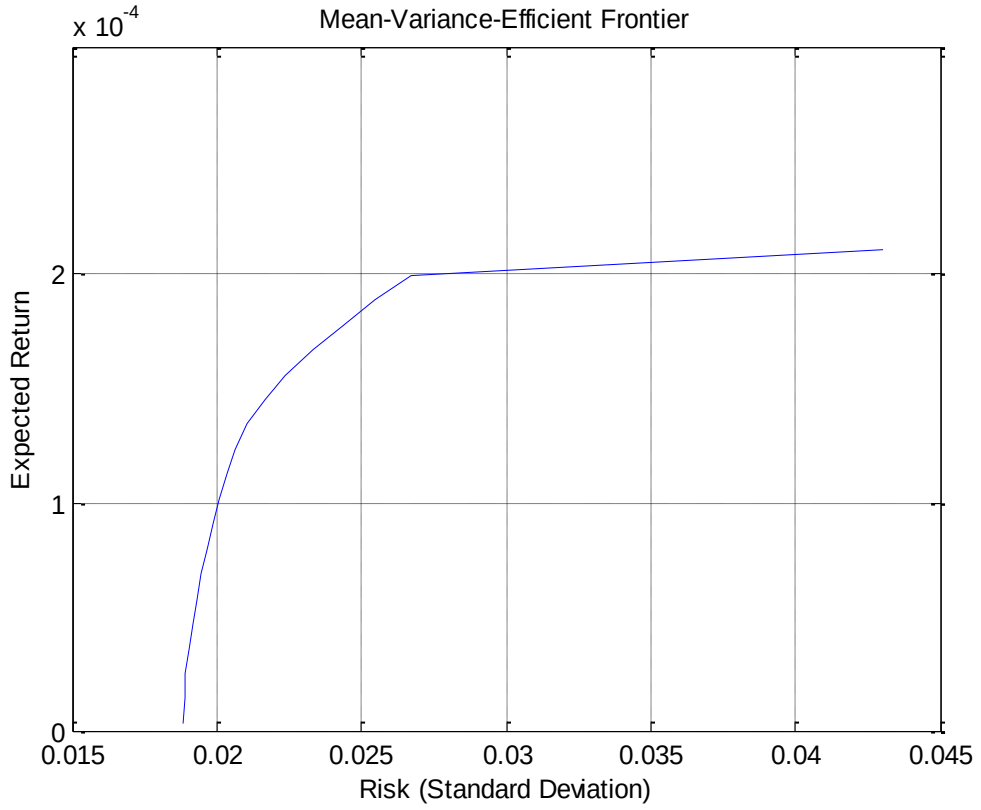
Estimation multivariée sous MATLAB

CAPM/Grouped Multivariate Normal estimated from 03-Jan-2000 to 07-Nov-2005

Asset	Alpha	Beta
AAPL	0.0012	1.2294
AMZN	0.0007	1.3673
CSCO	-0.0002	1.5653
DELL	-0.0000	1.2594
EBAY	0.0014	1.3441
GOOG	0.0041	0.6173
HPQ	0.0001	1.3745
IBM	-0.0000	1.0807
INTC	0.0001	1.6002
MSFT	-0.0002	1.1765
ORCL	0.0000	1.5010
YHOO	0.0001	1.6543

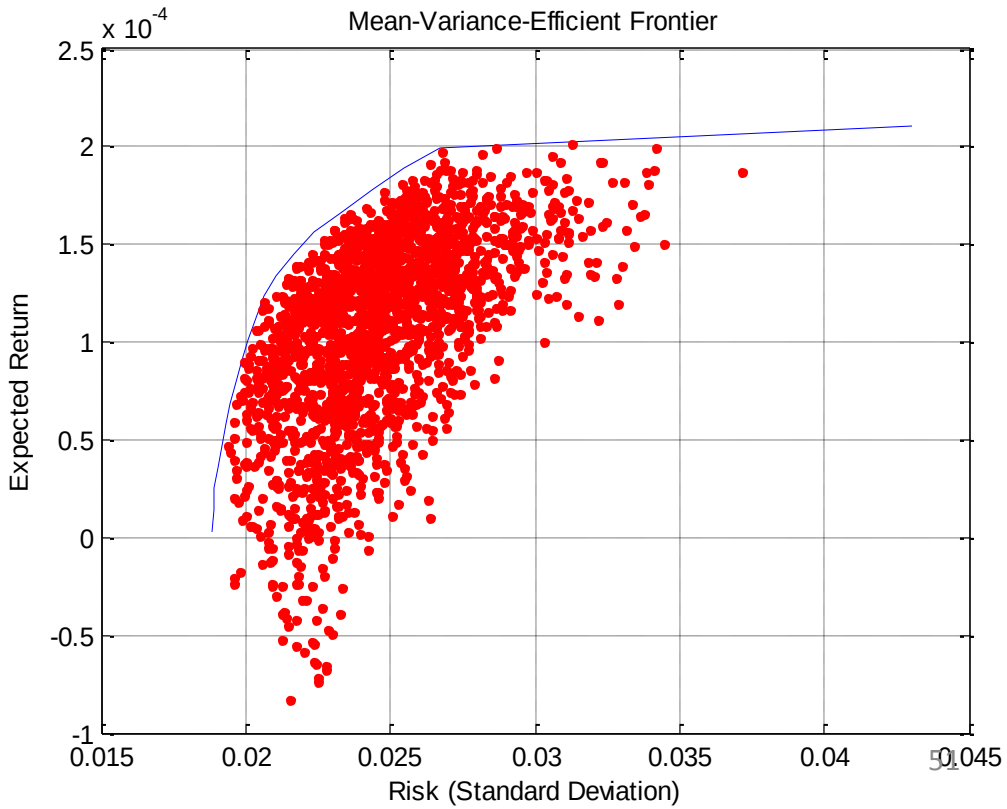
Frontière d'efficience

```
% frontières d'efficience
port = Data(:,7:12);
[NumPoints,NumAssets]=size(port);
returns = mean(port);
cova = cov(port);

portopt(returns,cova,20);
```



Frontière d'efficience et portefeuilles aléatoires

```
% quelques portefeuilles aléatoires
weights = exprnd(1,2000,NumAssets);
total = sum(weights,2);
total = total(:,ones(NumAssets,1));
weights = weights ./ total;
hold on
[portRisk, portReturn] = portstats(returns,cova, weights);
plot(portRisk,portReturn,'.r');
hold off
```



Un Réseau de Neurones en C++

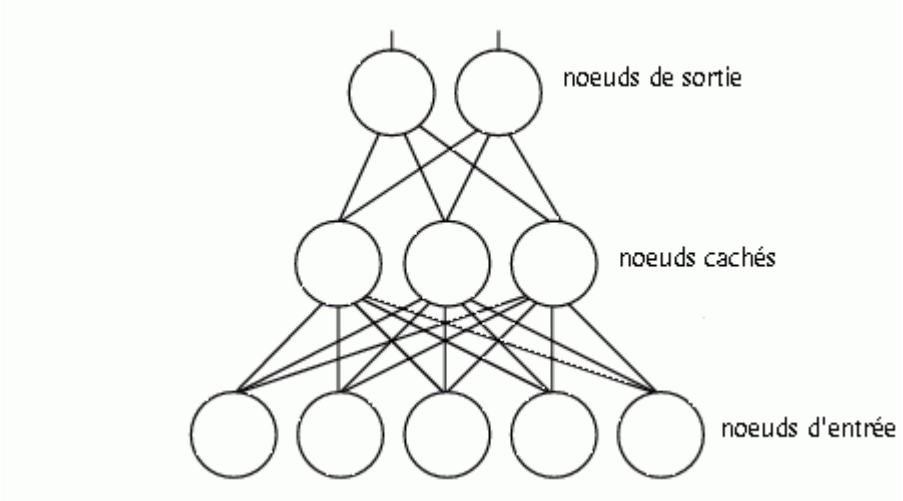
Rappels sur les RN

- **Les Perceptrons Multicouches:** structure relativement simple : une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées. Chaque neurone n'est relié qu'aux neurones des couches précédentes, mais à tous les neurones de la couche précédente.
- **Vector Quantization.** (apprentissage non supervisé). Introduits par Grossberg (1976), la quantification vectorielle est une méthode généralement qualifiée d'estimateur de densité non supervisé. Elle permet de retrouver des groupes sur un ensemble de données.
- **Self Organizing Map.** (apprentissage non supervisé). Les SOM sont issus des travaux de Fausett (1994) et Kohonen (1995). Ils permettent de cartographier en deux dimension et de distinguer des groupes dans des ensembles de données.
- **Learning Vector Quantization.** (apprentissage supervisé). Les réseaux utilisant la méthode LVQ ont été proposés par Kohonen (1988). Des trois types de réseaux présentés ici, la LVQ est la seule méthode qui soit réellement adaptée à la classification de données par "recherche du plus proche voisin".
- **Les Réseaux de Hopfield.** Ces réseaux sont des réseaux récursifs, un peu plus complexes que les perceptrons multicouches. Chaque cellule est connectée à toutes les autres et les changements de valeurs de cellules s'enchainent en cascade jusqu'à un état stable. Ces réseaux sont bien adaptés à la reconnaissance de formes.
- **Autres:** higher order, deep neural networks, …

Rappels sur les RN

- Ici nous parlons d'un perceptron à trois couches.
- En entrée les variables exogènes et en sortie les endogènes.
- Les neurones sont connectés entre eux par des sommes pondérées et à chaque étage, nous avons une fonction d'activation.
- Il faut ajouter un algorithme d'apprentissage (ici rétropropagation).

Le perceptron à 3 couches



L’algorithme

- On initialise les poids sur -0.5 , 0.5
- Normalisation des données d’entrée
- Lecture aléatoire des données (rotation)
- A chaque itération
  1. *Calcul des sorties (propagations avant)*
  2. *Ajuster les poids*
  3. *Répéter l’algorithme à partir de la lecture aléatoire de données jusqu’à max d’itérations ou erreurs min.*

La règle du « delta »

On ajuste les **poids** selon la formule

- $w(n) = w(n-1) + tx * x(n) * g(n)$

Et le **gradiant** est défini comme

- $g(n) = x(n) * [1 - x(n)] * e(n)$ (n. sortie)
- $g(n) = x(n) * [1 - x(n)] * \sum w(i)g(i)$ (autres)

tx étant le taux d'apprentissage

Structure d'un projet en C++

- Le corps du programme proprement dit (les fichiers .cc ou .cpp)
- Les en-têtes (headers)
- Les librairies
- Le makefile, configure,…

- *Quelle proposition d'architecture pour ce projet ?*

Les Objets (le header)

```
struct Neuron {
  double  x;      /* sortie */
  double  e;      /* erreur */
  double* w;      /* poids  */
};

struct Layer {
  int     nNumNeurons;
  Neuron* pNeurons;
};
```

Les conteneurs de base du projet

Les Objets (le header)

```
class MultiLayerPerceptron {

  int    nNumLayers;
  Layer* pLayers;

  double dMSE;
  double dMAE;

  void RandomWeights();

  void SetInputSignal (double* input);
  void GetOutputSignal(double* output);

  void SaveWeights();
  void RestoreWeights();

  void PropagateSignal();
  void ComputeOutputError(double* target);
  void BackPropagateError();
  void AdjustWeights();

  void Simulate(double* input, double* output, double* target, bool training);
…
```

*Les variables et fonctions privées de l'objet **MultiLayerPerceptron***

Les Objets (le header)

```
…

public:

  double dEta;
  double dAlpha;
  double dGain;
  double dAvgTestError;

  MultiLayerPerceptron(int nl, int npl[]);
  ~MultiLayerPerceptron();

  int Train(const char* fnames);
  int Test (const char* fname);
  int Evaluate();

  void Run(const char* fname, const int& maxiter);

};
```

*Les variables et fonctions publiques de l'objet **MultiLayerPerceptron***

Le corps du programme

Les librairies « standard »

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
```

Quelques fonctions…

```
void InitializeRandoms() { srand(4711); }

int RandomEqualINT(int Low, int High) {
  return rand() % (High-Low+1) + Low;
}
double RandomEqualREAL(double Low, double High) {
  return ((double) rand() / RAND_MAX) * (High-Low) + Low;
}
```

Constructeur & Destructeur

```
MultiLayerPerceptron::MultiLayerPerceptron(int nl, int npl[]) :
  nNumLayers(0),
  pLayers(0),
  dEta(0.25),
  dAlpha(0.9),
  dGain(1.0),
  dMSE(0.0),
  dMAE(0.0),
  dAvgTestError(0.0)
{
  int i,j;

  /* --- création des couches */
  nNumLayers = nl;
  pLayers    = new Layer[nl];

  /* --- init des couches */
  for ( i = 0; i < nl; i++ )
    {

      /* --- création des neurones */
      pLayers[i].nNumNeurons = npl[i];
      pLayers[i].pNeurons    = new Neuron[ npl[i] ];

      /* --- init des neurones */
      for( j = 0; j < npl[i]; j++ )
    {
      pLayers[i].pNeurons[j].x  = 1.0;
      pLayers[i].pNeurons[j].e  = 0.0;
      if(i>0)
        {
          pLayers[i].pNeurons[j].w     = new double[ npl[i-1] ];
          pLayers[i].pNeurons[j].dw    = new double[ npl[i-1] ];
          pLayers[i].pNeurons[j].wsave = new double[ npl[i-1] ];
        }
      else
        {
          pLayers[i].pNeurons[j].w     = NULL;
          pLayers[i].pNeurons[j].dw    = NULL;
          pLayers[i].pNeurons[j].wsave = NULL;
        }
    }

    }


}
```

```
MultiLayerPerceptron::~MultiLayerPerceptron()
{
  int i,j;
  for( i = 0; i < nNumLayers; i++ )
    {
      if ( pLayers[i].pNeurons )
    {
      for( j = 0; j < pLayers[i].nNumNeurons; j++ )
        {
          if ( pLayers[i].pNeurons[j].w )
        delete[] pLayers[i].pNeurons[j].w;
          if ( pLayers[i].pNeurons[j].dw )
        delete[] pLayers[i].pNeurons[j].dw;
          if ( pLayers[i].pNeurons[j].wsave )
        delete[] pLayers[i].pNeurons[j].wsave;
        }
    }
      delete[] pLayers[i].pNeurons;
    }
  delete[] pLayers;
}
```

Propagation du signal d'entrée

```
void MultiLayerPerceptron::PropagateSignal()
{
  int i,j,k;

  /* --- la boucle commence avec la seconde couche */
  for( i = 1; i < nNumLayers; i++ )
    {
      for( j = 0; j < pLayers[i].nNumNeurons; j++ )
    {
      /* --- calcul de la somme pondérée en entrée */
      double sum = 0.0;
      for ( k = 0; k < pLayers[i-1].nNumNeurons; k++ )
        {
          double out = pLayers[i-1].pNeurons[k].x;
          double w   = pLayers[i  ].pNeurons[j].w[k];
          sum += w * out;
        }
      /* --- application de la fonction d'activation (sigmoid) */
      pLayers[i].pNeurons[j].x = 1.0 / (1.0 + exp(-dGain * sum));
    }
    }
}
```

Rétro-propagation des erreurs
la règle du « delta »

```
void MultiLayerPerceptron::ComputeOutputError(double* target)
{
  int  i;
  dMSE = 0.0;
  dMAE = 0.0;
  for( i = 0; i < pLayers[nNumLayers-1].nNumNeurons; i++)
    {
      double x = pLayers[nNumLayers-1].pNeurons[i].x;
      double d = target[i] - x;
      pLayers[nNumLayers-1].pNeurons[i].e = dGain * x * (1.0 - x) * d;
      dMSE += (d * d);
      dMAE += fabs(d);
    }
  /* --- erreur quadratique moyenne */
  dMSE /= (double)pLayers[nNumLayers-1].nNumNeurons;
  /* --- erreur absolue moyenne */
  dMAE /= (double)pLayers[nNumLayers-1].nNumNeurons;
}
```

```
void MultiLayerPerceptron::BackPropagateError()
{
  int i,j,k;
  /* --- la boucle commence à l'avant dernière couche */
  for( i = (nNumLayers-2); i >= 0; i-- )
    {
      /* --- couche inférieure */
      for( j = 0; j < pLayers[i].nNumNeurons; j++ )
    {
      double x = pLayers[i].pNeurons[j].x;
      double E = 0.0;
      /* --- couche supérieure */
      for ( k = 0; k < pLayers[i+1].nNumNeurons; k++ )
        {
          E += pLayers[i+1].pNeurons[k].w[j] * pLayers[i+1].pNeurons[k].e;
        }
      pLayers[i].pNeurons[j].e = dGain * x * (1.0 - x) * E;
    }
    }
}
```

Ajustement des poids

```
void MultiLayerPerceptron::AdjustWeights()
{
  int i,j,k;
  /* --- la boucle commence avec la seconde couche */
  for( i = 1; i < nNumLayers; i++ )
    {
      for( j = 0; j < pLayers[i].nNumNeurons; j++ )
    {
      for ( k = 0; k < pLayers[i-1].nNumNeurons; k++ )
        {
          double x  = pLayers[i-1].pNeurons[k].x;
          double e  = pLayers[i  ].pNeurons[j].e;
          double dw = pLayers[i  ].pNeurons[j].dw[k];
          pLayers[i].pNeurons[j].w [k] += dEta * x * e + dAlpha * dw;
          pLayers[i].pNeurons[j].dw[k]  = dEta * x * e;
        }
    }
    }
}
```

Lecture d'un fichier d'apprentissage et lancement

```
int MultiLayerPerceptron::Train(const char* fname)
{
  int count = 0;
  int nbi   = 0;
  int nbt   = 0;
  double* input  = NULL;
  double* output = NULL;
  double* target = NULL;
  FILE*   fp = NULL;

  fp = fopen(fname,"r");
  if(!fp) return 0;

  input  = new double[pLayers[0].nNumNeurons];
  output = new double[pLayers[nNumLayers-1].nNumNeurons];
  target = new double[pLayers[nNumLayers-1].nNumNeurons];

  if(!input) return 0;
  if(!output) return 0;
  if(!target) return 0;


  while( !feof(fp) )
    {
      double dNumber;
      if( read_number(fp,&dNumber) )
    {
      /* --- on le transforme en input/target */
      if( nbi < pLayers[0].nNumNeurons )
        input[nbi++] = dNumber;
      else if( nbt < pLayers[nNumLayers-1].nNumNeurons )
        target[nbt++] = dNumber;

      /* --- on fait un apprentisage du réseau  avec cette ligne*/
      if( (nbi == pLayers[0].nNumNeurons) &&
          (nbt == pLayers[nNumLayers-1].nNumNeurons) )
        {
          Simulate(input, output, target, true);
          nbi = 0;
          nbt = 0;
          count++;
        }
    }
      else
    {
      break;
    }
    }

  if(fp) fclose(fp);

  if(input)  delete[] input;
  if(output) delete[] output;
  if(target) delete[] target;

  return count;
}
```

```
#include "mlp.h"

int main(int argc, char* argv[])
{

  int layers2[] = {1,5,1};
  MultiLayerPerceptron mlp2(3,layers2);
  mlp2.Run("sin.dat",500);

  return 0;
}
```

Réseau de neurones sur les prix du logement en MATLAB

Prix des logements

- On cherche ici a modéliser les prix des logements en fonction d'un certain nombre de paramètres (tirés de la base house_price de MATLAB).
- On modélise le lien entre ces paramètres et les prix des maisons à l'aide d'un perceptron.

Les variables

houseInputs

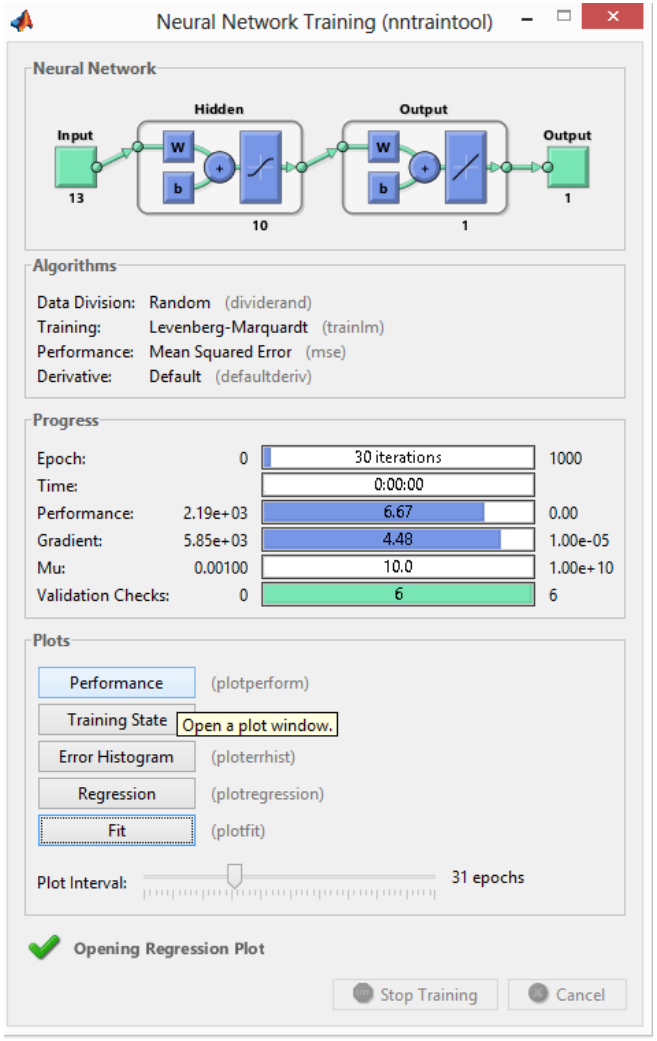
1. Per capita crime rate per town
2. Proportion of residential land zoned for lots over 25,000 sq. ft.
3. proportion of non-retail business acres per town
4. 1 if tract bounds Charles river, 0 otherwise
5. Nitric oxides concentration (parts per 10 million)
6. Average number of rooms per dwelling
7. Proportion of owner-occupied units built prior to 1940
8. Weighted distances to five Boston employment centres
9. Index of accessibility to radial highways
10. Full-value property-tax rate per $10,000
11. Pupil-teacher ratio by town
12. 1000(Bk - 0.63)^2, where Bk is the proportion of blacks by town
13. Percent lower status of the population

houseTargets - a 1x506 matrix of median values of owner-occupied homes in each neighborhood in 1000's of dollars.

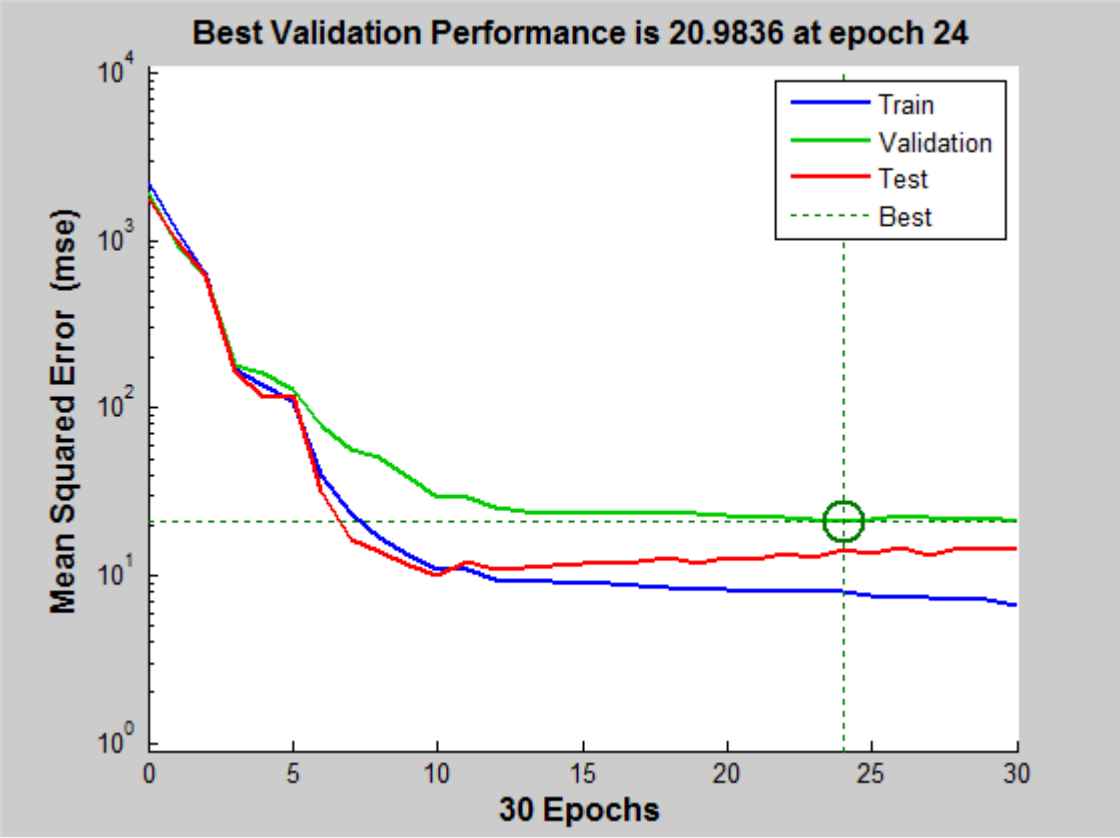
Le réseau de neurones
code MATLAB

```
% paramètres
load house_dataset;
inputs  = houseInputs;
targets = houseTargets;

hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio   = 15/100;
net.divideParam.testRatio  = 15/100;

% apprentissage
[net,tr] = train(net,inputs,targets);

% test
outputs = net(inputs);
errors  = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% graphs de performances
figure, plotperform(tr)
figure, plotregression(targets,outputs)
```

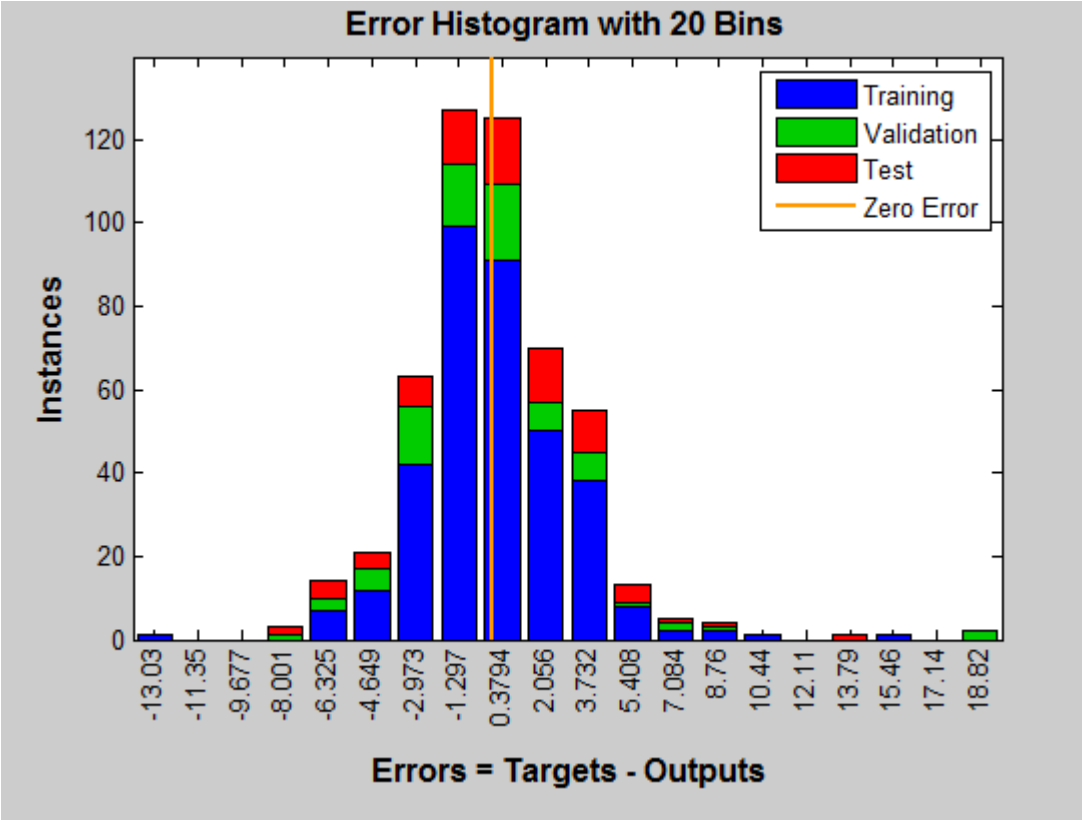
Le réseau de neurones
fenêtre principale



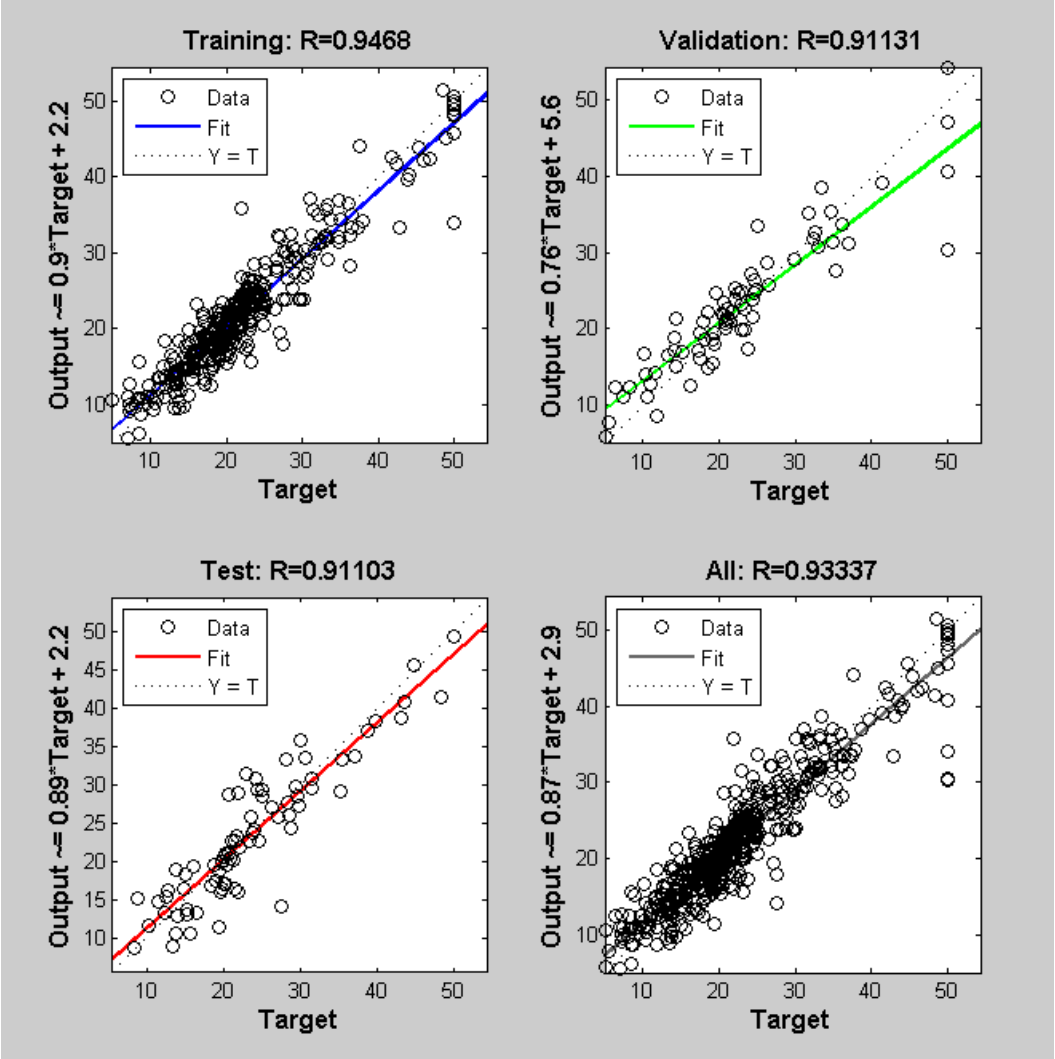
Performances
RMSE et itérations



Performances
histogramme des erreurs



Performances



Un algorithme génétique en C++

Les algorithmes génétiques

- Introduits par Holland en (1975) et plus largement développés en 1992 par Koza.
- Pour quels types de problèmes ?
- Pourquoi en C++ plutôt qu'en MATLAB ?

L’algorithme

Quatre grandes étapes:

- La sélection
- Le crossover
- La mutation
- L’évaluation

Implémentation en C++ ?

- Exemple d'utilisation d'une librairie tierce, déjà optimisée: la Parallel Genetic Algorithm Library (PGApack) de David Levine.
- Utilisable en Fortran ou en C (ou C++).
- Tourne en uni ou multi-proc (via MPI).
- Permet de traiter tous les types de données (binaires, entiers, chaines de caractères et réels).

Initialisation de PGApack

```
//lancement del'algo génétique
PGAContext* ctx;

int len = 44;
int pop = 50;
int rep = pop/4;

ctx = PGACreate (&argc,argv, PGA_DATATYPE_INTEGER, len, PGA_MAXIMIZE);
PGASetPopSize           (ctx,pop);
PGASetNumReplaceValue   (ctx,rep);
PGASetPopReplaceType    (ctx,PGA_POPREPL_BEST);
PGASetStoppingRuleType  (ctx,PGA_STOP_NOCHANGE);
PGASetMaxNoChangeValue  (ctx,20);
PGASetMutationType      (ctx,PGA_MUTATION_RANGE);
PGASetCrossoverType     (ctx,PGA_CROSSOVER_UNIFORM);
```

Codage du problème

```
// --- on défini les bornes
int *a=new int[len];
int *b=new int[len];
if( is3to5 == 0 ) {
    // --- les poids
    for(i=0;i<=19;i++)
    {
        a[i] = 20;
        b[i] = 50;
    }
    // --- les seuils
    for(i=20;i<=31;i++) {
        a[i] = -3;
        b[i] =  3;
    }
    // --- les courbures
    for(i=32;i<=43;i++) {
        a[i] = 10;
        b[i] = 30;
    }
} else {
    // --- les poids
    for(i=0;i<=19;i++)
    {
        a[i] = 10;
        b[i] = 50;
    }
    // --- les seuils
    for(i=20;i<=31;i++) {
        a[i] = -10;
        b[i] =  10;
    }
    // --- les courbures
    for(i=32;i<=43;i++) {
        a[i] = 10;
        b[i] = 40;
    }
}
```

Fonction d'évaluation

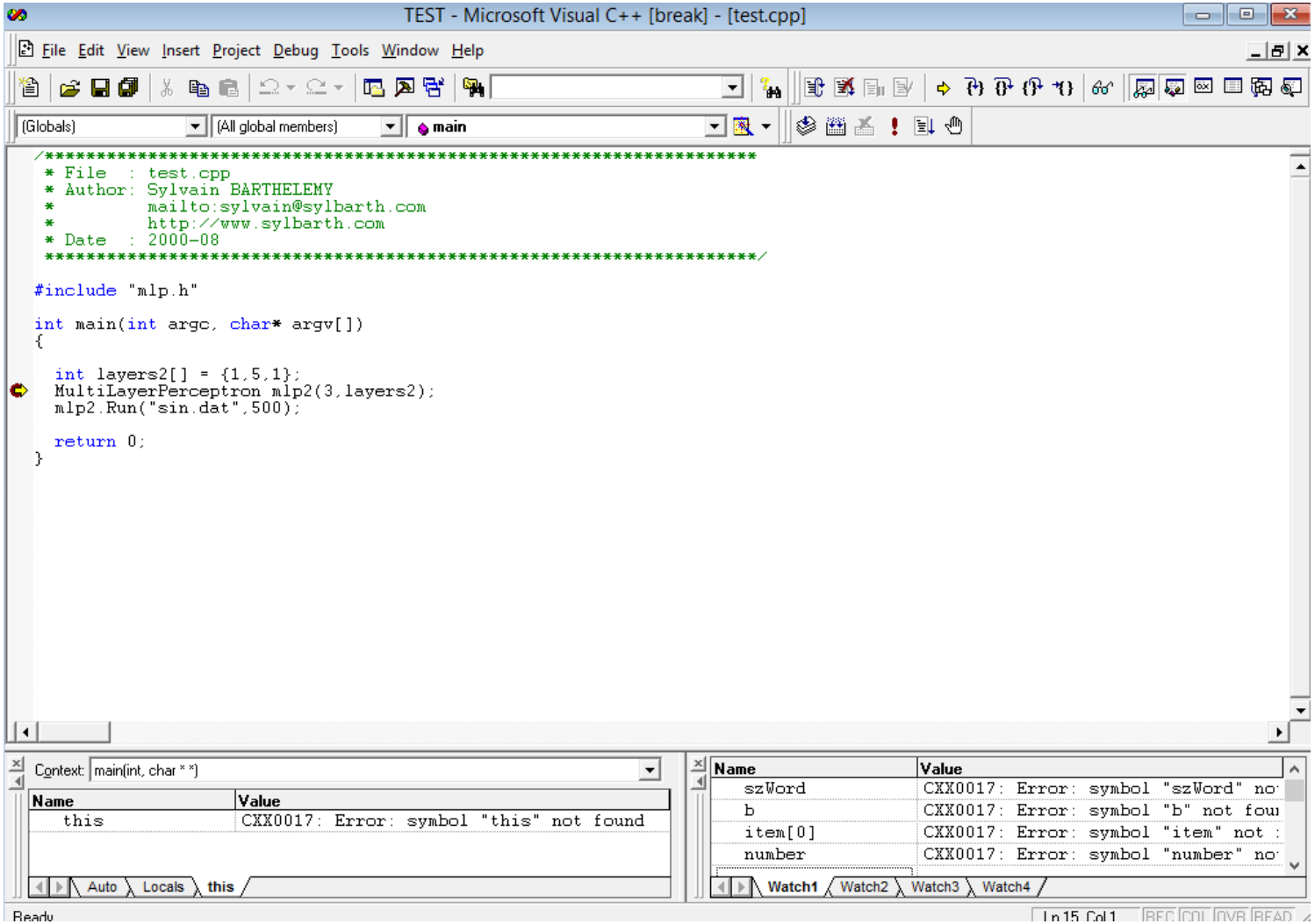
```
double evaluate(PGAContext *ctx,int p,int pop)
{
    riskmonitor_ratings_params param = readparams(ctx,p,pop);
    riskmonitor_ratings_evalratematrix evtot;
    for(int i=0; i<gCountriesCount; i++) {
        riskmonitor_ratings rm_rat(gCountries[i],gOdb);
        enl_tseries trigger = gOdb->get( gCountries[i].cid, gTriggerName, ENL_QUARTERLY );

        riskmonitor_ratings_evalratematrix ev = rm_rat.test_rating( trigger, &param, gHorizon );
        evtot.r025.ncri += ev.r025.ncri;
        evtot.r050.ncri += ev.r050.ncri;
        evtot.r075.ncri += ev.r075.ncri;
        evtot.r100.ncri += ev.r100.ncri;

        evtot.r025.nobs += ev.r025.nobs;
        evtot.r050.nobs += ev.r050.nobs;
        evtot.r075.nobs += ev.r075.nobs;
        evtot.r100.nobs += ev.r100.nobs;
    }

  double minObs025 = evtot.r100.nobs * 0.025;
  double minObs050 = evtot.r100.nobs * 0.05;
  double minObs075 = evtot.r100.nobs * 0.05;
  double minObs100 = evtot.r100.nobs * 0.025;
  double obs_r025 = evtot.r025.nobs;
  double obs_r050 = evtot.r050.nobs - evtot.r025.nobs;
  double obs_r075 = evtot.r075.nobs - evtot.r050.nobs;
  double obs_r100 = evtot.r100.nobs - evtot.r075.nobs ;

  double fac = 1 - fabs(evtot.r050.nobs - (evtot.r100.nobs-evtot.r050.nobs))/evtot.r100.nobs;
  return fac * ( evtot.r100.rate() - evtot.r025.rate() );

}
```

Résolution et écriture des résultats

```
PGASetIntegerInitRange(ctx, a, b);
PGASetUp(ctx);
delete a;
delete b;

PGAEvaluate(ctx, PGA_OLDPOP , evaluate, NULL);
PGAFitness (ctx, PGA_OLDPOP);
while( !PGADone(ctx,NULL) ){
    PGASelect                  (ctx, PGA_OLDPOP);
    PGARunMutationAndCrossover(ctx, PGA_OLDPOP, PGA_NEWPOP);
    PGAEvaluate                (ctx, PGA_NEWPOP, evaluate, NULL);
    PGAFitness                 (ctx, PGA_NEWPOP);
    PGAUpdateGeneration        (ctx, NULL);
    PGAPrintReport             (ctx,stdout,PGA_OLDPOP);
}

//ecriture des résultats
char file_name[1024];
sprintf(file_name,"c:/temp/ag_%s_%i.txt",triggerName, horizon);
FILE* fp = fopen(file_name,"wt");
if(fp) {
    fprintf(fp,"\n\nRésultat de l'apprentissage :\n");
    int best = PGAGetBestIndex(ctx,PGA_OLDPOP);
    PGAPrintIndividual(ctx,fp,best,PGA_OLDPOP);
    fclose(fp);
    save(group,ctx,best,PGA_OLDPOP);
}

PGADestroy(ctx);
```

Un exemple de débogage en C++

Exemple de débogage en C++ sur un réseau de neurones



Merci de votre attention