

# Outils informatiques en économie (4103010)

Manfred Gilli

Département d'économétrie  
Université de Genève

Eté 2008

<b>Introduction à Matlab</b>	<b>3</b>
Eléments de syntaxe . . . . .	3
Opérations éléments par éléments et script files . . . . .	4
Graphiques 2D $y = x^2 e^x$ . . . . .	5
Graphiques 3D $z = x^2 y^2$ . . . . .	6
<b>Introduction à la programmation structurée</b>	<b>7</b>
Notion de programme . . . . .	7
Programmation structurée (affectation, répétition, choix) . . . . .	10
Syntaxe Matlab pour répétition et choix . . . . .	15
<b>Exemples de programmes</b>	<b>16</b>
Moyenne éléments d'un vecteur et précision machine . . . . .	16
Maximum des éléments d'un vecteur . . . . .	17
Tri à bulles . . . . .	18
Simulation du lancer d'un dès . . . . .	19
<b>Programmation avec Matlab</b>	<b>20</b>
Réalisation d'une fonction . . . . .	21
Fonction <code>inline</code> . . . . .	23
Fonction <code>feval</code> . . . . .	24
Exemple de fonction ( <code>call</code> Européen) . . . . .	26
<b>Introduction à la simulation stochastique</b>	<b>27</b>
Quelques exemples . . . . .	27
Notion de variable aléatoire . . . . .	30
Génération d'une v.a. uniforme $[0, 1[$ . . . . .	36
Génération de v.a. discrètes . . . . .	41
Génération de v.a. continues . . . . .	47
<b>Finding the roots of <math>f(x) = 0</math></b>	<b>51</b>
Résolution graphique . . . . .	52
Résolution aléatoire . . . . .	53
Recoupement par intervalles (bracketing) . . . . .	55
Méthode de la bisection . . . . .	58
Itérations de point fixe . . . . .	62
Méthode de Newton . . . . .	71

## **Overview**

**Introduction à Matlab**

**Introduction à la programmation structurée**

**Exemples de programmes**

**Programmation avec Matlab**

**Introduction à la simulation stochastique**

**Finding the roots of  $f(x) = 0$**

OIEC (4103010) – M. Gilli

Eté 2008 – 2

**Éléments de syntaxe**

- Polycopié
- Syntaxe (éléments de)
  - Symboles [ ] ; , ... %
  - *variable = expression*
  - ans
- whos, help, lookfor
- Variables (matrices par défaut)
- Opérations avec vecteurs, matrices
  - $x = [8\ 3\ 7\ 9]$
  - $x(5) = 12$
  - $x(6) = x(1) + x(4)$
  - $A = [4\ 5; 7\ 3]; A(2,1) = A(2,1) - 3;$
  - Transposition  $A'$
  - Addition, soustraction, multiplication, division, puissance  $+ - * / \ \wedge$

OIEC (4103010) – M. Gilli

Eté 2008 – 3

**Opérations éléments par éléments et script files**

- Opérations éléments par éléments
  - Addition, soustraction (toujours élément par élément)
  - Multiplication  $.*$  division  $./$  puissance  $.^$
  - $x = [1\ 2\ 3] .* [4\ 5\ 6]$
  - $x = [4\ 12\ 9] ./ [2\ 4\ 3]$
  - $x = [1\ 2\ 3].^2$
  - $x = 2.^[1\ 2\ 3]$
- Matlab-files (Script)
  - Extension doit être .m
  - Variables définies globalement
  - Fichier de commandes (scripts)
  - Possibilité d'imbriquer des script
  - Edition, exécution, débogage **edit**
  - Exemple

OIEC (4103010) – M. Gilli

Eté 2008 – 4

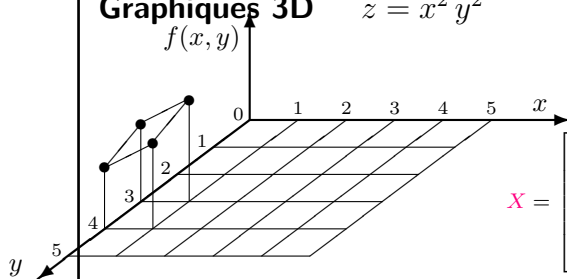
## Graphiques 2D $y = x^2 e^x$

```
x = linspace(-4,1);
y = x.^2 .* exp(x);
plot(x,y);
xlabel('x');
ylabel('y');
title('y = x^2 exp(x)');
grid on
ylim([-0.5 2.5]);
hold on
d = 2*x.*exp(x)+x.^2.*exp(x);
plot(x,d,'r:');
```

OIEC (4103010) – M. Gilli

Eté 2008 – 5

## Graphiques 3D $z = x^2 y^2$

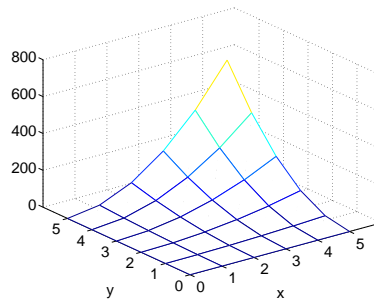


```
x = linspace(0,5,6);
y = linspace(0,5,6);
[X,Y] = meshgrid(x,y);
```

$$X = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

```
Z = X.^2 .* Y.^2;
mesh(Z);
mesh(x,y,Z);
```

○ mesh, surf, plot3, hist3, bar3, ...



OIEC (4103010) – M. Gilli

Eté 2008 – 6

### Notion de programme

Ordinateur effectue des opérations élémentaires :

- addition
- soustraction
- affectation
- comparaison

Programme :

- orchestre déroulement opérations dans le temps
- Le programme est un **objet statique**
- lors de son exécution il **évolue** de façon **dynamique** en fonction de l'état des variables.

OIEC (4103010) – M. Gilli

Eté 2008 – 7

### Problématique de la programmation

Problème :

- Comment concevoir un programme afin d'avoir une **vision** aussi **claire** que possible des situations dynamiques qu'il engendre lors de son exécution
- Objectif de la programmation est une **lisibilité** aussi claire que possible du programme, afin de pouvoir le communiquer entre "programmeurs" et surtout pour pouvoir fournir une démonstration plus ou moins rigoureuse de sa validité (faire la preuve qu'il fournira les résultats désirés).

**Question** : Comment atteindre au mieux ces objectifs ?

**Réponse** : Avec des techniques adéquates de programmation.

OIEC (4103010) – M. Gilli

Eté 2008 – 8

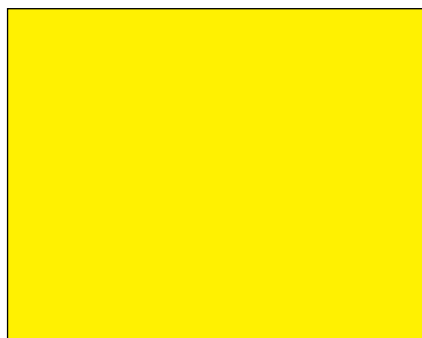
### Note historique

A l'origine les enchaînements étaient contrôlés avec des instructions de branchement (**GOTO**, **IF avec branchement**, etc.).

```

START
100 continue
  Lire des données
  :
  Calcul valeur de I
  if (I) 400, 300, 200
300 continue
  :
  Calcul valeur de K
  if (K > 0) GOTO 100
400 continue
  :
  GOTO 100
200 continue
  :
STOP

```



Spaghetti code!! Difficile d'imaginer les différents états dynamiques d'un tel programme. Les schémas fléchés (organigrammes) ne permettent pas de contourner cette difficulté.

OIEC (4103010) – M. Gilli

Eté 2008 – 9

## Programmation structurée (affectation, répétition, choix)

- Après des échecs (catastrophes) un courant de pensée en faveur d'un autre "style" de programmation c'est développé
- Dijkstra (1968) : "GOTO Statement Considered Harmful"
- Wilkes (1968) : "The Outer and the Inner Syntax of a Programming Language"
- Ces deux articles sont à l'origine d'une véritable polémique qui mettait en cause l'utilisation des instructions de branchement dans les langages de programmation

Les seules structures de contrôle qui existent dans la programmation structurée sont :

- enchaînement
- répétition
- choix

OIEC (4103010) – M. Gilli

Été 2008 – 10

## Enchaînement

L'enchaînement consiste en une simple succession d'un nombre quelconque d'instructions d'affectation, de modification, de lecture, d'écriture :

```
lire y
x = sqrt(y)
:
écrire z
```

OIEC (4103010) – M. Gilli

Été 2008 – 11

## Répétition

La répétition d'un ensemble (block) d'instructions, appelée aussi boucle, peut prendre deux formes :

```
pour i dans l'ensemble I répéter
: (instructions)
fin

tant que condition vraie répéter
: (instructions)
fin
```

Pour la première forme, le nombre de répétitions pour la première forme est défini par le nombre d'éléments de l'ensemble *I*, alors que pour la seconde forme, la répétition se fait à l'infini si la condition reste vraie.

OIEC (4103010) – M. Gilli

Été 2008 – 12

## Choix

Les instructions qui permettent d'opérer un choix parmi différents blocks d'instructions possibles sont les suivantes :

```
si condition 1 vraie faire
  : (instructions)
sinon si condition 2 vraie faire
  : (instructions)
sinon si ...
  :
sinon
  : (instructions)
fin
```

On exécute les instructions qui suivent la première condition qui est vraie et on avance jusqu'à la fin. Si aucune condition n'est vraie, on exécute les instructions qui suivent sinon.

OIEC (4103010) – M. Gilli

Été 2008 – 13

## Structure hiérarchique

Tout programme est alors composé d'une succession de ces trois structures qui sont exécutées l'une après l'autre.

Par opposition à un programme contenant des instructions de branchement, un programme conçu avec les trois éléments de la programmation structurée, permettra une lecture hiérarchique de haut en bas ce qui facilite sa maîtrise intellectuelle.

Le recours aux organigrammes a d'ailleurs été abandonné.

OIEC (4103010) – M. Gilli

Été 2008 – 14

## Syntaxe Matlab pour répétition et choix

### ○ Nombre de répétitions fini

```
for v = expression
  instructions
end
```

### ○ Nombre de répétitions indéfini

```
while expression
  instructions
end
```

### ○ Choix simple

```
if expression
  instructions
end
```

### ○ Choix multiple

```
if expression1
  instructions
elseif expression2
  instructions
elseif expression3
  instructions
else
  instructions
end
```

OIEC (4103010) – M. Gilli

Été 2008 – 15

**Moyenne éléments d'un vecteur et précision machine**

$$\bar{x} = \frac{1}{n}(x_1 + x_2 + \dots + x_n) \equiv \frac{1}{n} \sum_{i=1}^n x_i$$

```
s = 0;
for i = 1:n
    s = s + x(i);
end
xbar = s/n;
```

Calcul de la précision machine

```
e = 1;
while 1 + e > 1
    e = e/2;
end
emach = 2*e;
```

OIEC (4103010) – M. Gilli

Été 2008 – 16

**Maximum des éléments d'un vecteur**

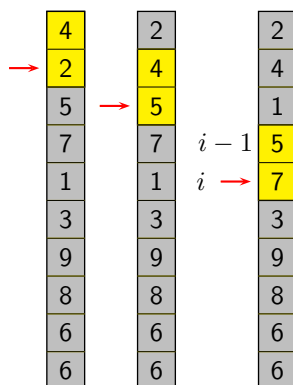
```
m = -realmax;
for i = 1:n
    if x(i) > m
        m = x(i);
    end
end
m
```

OIEC (4103010) – M. Gilli

Été 2008 – 17

**Tri à bulles**

Sorter les éléments d'un vecteur dans l'ordre croissant



OIEC (4103010) – M. Gilli

```
inversion = 1;
while inversion
    inversion = 0;
    for i = 2 :N
        if x(i-1) > x(i)
            inversion = 1;
            t = x(i);
            x(i) = x(i-1);
            x(i-1) = t;
        end
    end
end
```

Été 2008 – 18

**Demonstration**

Bubblego

OIEC (4103010) – M. Gilli

Été 2008 – note 1 of slide 18

## Simulation du lancer d'un dè

Simuler  $n$  lancers d'un dè  
( $x_i, i = 1, \dots, n$  contient le résultat du  $i$ me lancer)

```
x = zeros(1,n)
for i = 1:n
    u = rand;
    if u < 1/6
        x(i) = 1;
    elseif u < 2/6
        x(i) = 2;
    elseif u < 3/6
        x(i) = 3;
    elseif u < 4/6
        x(i) = 4;
    elseif u < 5/6
        x(i) = 5;
    else
        x(i) = 6;
    end
end
hist(x,6)
```

OIEC (4103010) – M. Gilli

Eté 2008 – 19

## Demo

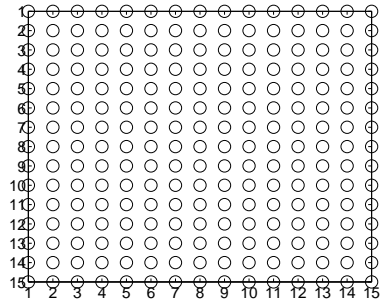
j16.m Montrer effet de l'initialisation de  $x$ .

OIEC (4103010) – M. Gilli

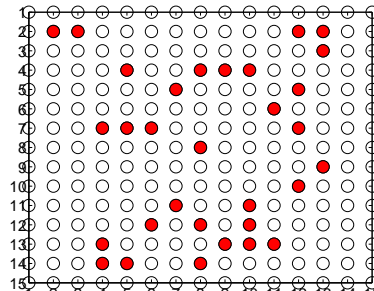
Eté 2008 – note 1 of slide 19

**Programmation d'un automate**

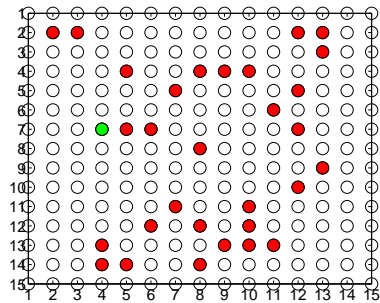
Soit une matrice  $A$  (vide)



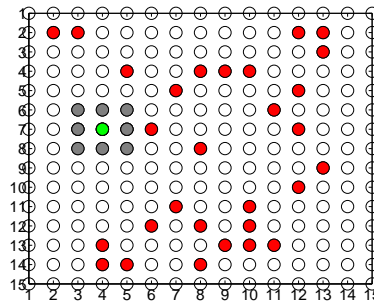
Initialisons quelques éléments avec la valeur 1



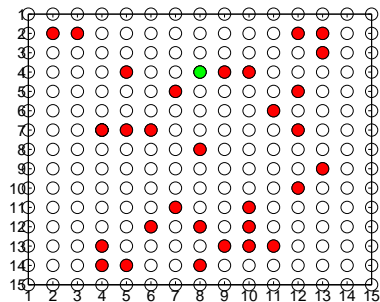
Considérons l'élément  $A_{7,4}$



Les voisins directs de l'élément  $A_{7,4}$



$$N = A(6,3) + A(6,4) + A(6,5) + A(7,3) + A(7,5) + \dots \\ A(8,3) + A(8,4) + A(8,5);$$



$$N = A(3,7) + A(3,8) + A(3,9) + A(4,7) + A(4,9) + \dots \\ A(5,7) + A(5,8) + A(5,9);$$

$i = 4; j = 8;$

$$N = A(i-1,j-1) + A(i-1,j) + A(i-1,j+1) + A(i,j-1) + A(i,j+1) + \dots \\ A(i+1,j-1) + A(i+1,j) + A(i+1,j+1);$$

## Réalisation d'une fonction

$[r, \dots, z] = \text{myfunc}(x, \dots, y)$  On passe la valeur !!

```
function [s1, ..., sn] = myfunc(e1, ..., em)

s1 = ... e1 ... em ...
⋮
sn = ... e1 ... em ...
```

Fichier myfunc.m

- Création d'un fichier avec extension .m (Nom fichier = nom fonction)
- Première ligne premier mot → **function**
- Arguments d'entrée et de sortie (l'ordre importe, pas les noms des variables)
- Toutes les variables, sauf les arguments de sortie sont locales

OIEC (4103010) – M. Gilli

Été 2008 – 21

## Réalisation d'une fonction

```
function N = NVoisins(i,j,M)
% NVoisins calcule le nombre de voisins non nuls de M(i,j)
% NVoisins(i,j,M) i indice de ligne et j indice de colonne
% Version 1.0 30-04-2006
N = M(i-1,j-1) + M(i-1,j) + M(i-1,j+1) + ...
    M(i ,j-1) + M(i ,j+1) + ...
    M(i+1,j-1) + M(i+1,j) + M(i+1,j+1);
```

Ex. : Chercher les nombre de voisins des *éléments intérieurs* d'une matrice  $A$

```
n = size(A,1);
NV = zeros(n,n);
for i = 2:n-1
    for j = 2:n-1
        NV(i,j) = NVoisins(i,j,A);
    end
end
```

OIEC (4103010) – M. Gilli

Été 2008 – 22

## Fonction inline

Lorsqu'il y a un seul argument de sortie et la structure de la fonction est simple il convient de réaliser la fonction avec **inline**.

- $f = \text{inline}(\text{'expression'})$  si la fonction a un seul argument il n'y a pas d'ambiguïté.  
Ex. :  $f(x) = x^2 e^x$ ,  $f = \text{inline}(\text{'x^2 * exp(x)'}); f(0.75) = 1.1908$
- $f = \text{inline}(\text{'expression'}, \text{'arg1'}, \text{'arg2'})$  s'il y a plusieurs arguments  
Ex. :  $f(x,y) = 2 \sin(x)^2 / \log(y)$ ,  $f = \text{inline}(\text{'2*sin(x)^2 / log(y)', 'x', 'y'}); f(1,4) = 1.0215$
- S'il y a des paramètres on doit les nommer  $P_1, \dots, P_n$  et donner le nombre  
Ex. :  $f(x) = a x^c$ ,  $f = \text{inline}(\text{'P1*x^P2'}, 2);$   
Pour  $x = 2.7$ ,  $a = 2$  et  $c = 3$  on peut écrire  $f(2.7, 2, 3) = 39.3660$

OIEC (4103010) – M. Gilli

Été 2008 – 23

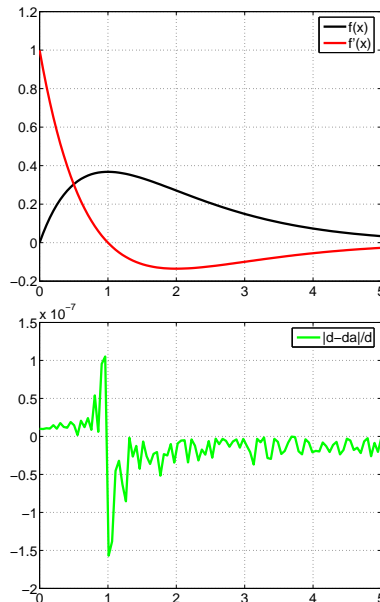
## Fonction feval

Dérivée de  $f(x)$  :  $\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$

En évaluant cette expression pour  $h$  donné on obtient une approximation numérique de  $f'(x)$ . Ex. :

$$f(x) = \frac{x}{e^x}, f'(x) = \frac{1-x}{e^x}$$

```
h = 1e-3;
f = inline('x./exp(x)');
x = linspace(0,5);
da = (f(x+h)-f(x)) / h
plot(x,f(x),'k'); hold on
plot(x,da,'r');
% Précision
d = inline('(1-x)./(exp(x))');
plot(x,abs(d(x)-da)./d(x),'g');
```



OIEC (4103010) – M. Gilli

Été 2008 – 24

## Fonction feval

On réalise une fonction qui calcule la dérivée numérique pour  $f$  quelconque

```
function da = Dnum00(f,x)
h = 1e-3;
da = ( f(x+h) - f(x) ) / h;
```

```
g = inline('2*x^2 + 3*exp(-x)');
d = Dnum00(g,0.1)
```

Si la fonction  $g$  a été définie avec `function` alors il faut utiliser `feval` :

```
function da = Dnum01(f,x)
h = 1e-3;
da = ( feval(f,x+h) - feval(f,x) ) / h;
```

OIEC (4103010) – M. Gilli

Été 2008 – 25

## Exemple de fonction (call Européen)

- Fonction  $\equiv$  fichier avec extension .m
- Arguments d'entrée et de sortie
- Toutes variables dans la fonction sont locales

```
function c = BScall(S,E,r,T,sigma)
% Call Européen avec Black-Scholes
% c = BScall(S,E,r,T,sigma) S sous-jacent ...
%
d1 = (log(S./E) + (r + sigma.^2 /2) .* T) ...
    ./ (sigma .* sqrt(T));
d2 = d1 - sigma .* sqrt(T);
Ee = E .* exp(-r .* T);
c = S .* normcdf(d1) - Ee .* normcdf(d2);
```

- Application :

```
vol = linspace(0.1,0.4);
P = 100; strike = 110; ret = 0.06; exp = 1;
call = BScall(P,strike,ret,exp,vol);
plot(vol,call)
```

**Quelques exemples**

- Fission nucléaire
- Automate cellulaire (Conway's Life Game)

OIEC (4103010) – M. Gilli

Eté 2008 – 27

**Demos**Fission Levy *et al.* (2000)

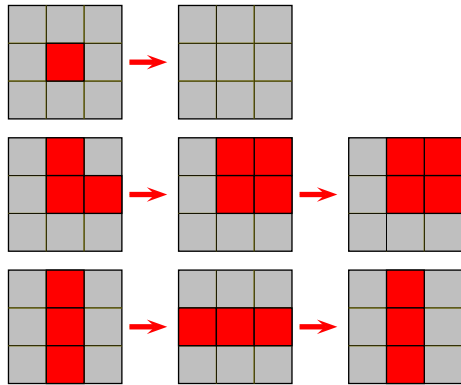
OIEC (4103010) – M. Gilli

Eté 2008 – note 1 of slide 27

**Conway's Life Game**

Cellules dispersées dans un plan :

- Une cellule **inactive** entouré de 3 cellules actives devient active (**naît**)
- Une cellule **active** entouré de 2 ou 3 cellules reste active (**survit**)
- Dans tous les autres cas la cellule **meurt** ou reste **inactive**



OIEC (4103010) – M. Gilli

Eté 2008 – 28

**Demos**

LifeGame

OIEC (4103010) – M. Gilli

Eté 2008 – note 1 of slide 28

**Outline**

Générer des événements aléatoires avec l'ordinateur. A la base de toute simulation est la variable aléatoire (v.a.) uniforme.

- Rappel notion variable aléatoire (v.a.)
- V.a. uniformément distribuée  $[0, 1]$
- Génération variables pseudo-aléatoires uniformes  $[0, 1]$
- Transformation en v.a. quelconque
- Commandes Matlab pour générer des v.a.
- Exemples
- Monte Carlo
- Bootstrap

OIEC (4103010) – M. Gilli

Eté 2008 – 29

## Notion de variable aléatoire

De manière générale il s'agit du résultat d'une :

- expérience (lancer monnaie, dés, ...)
- observation (âge, revenu d'une personne choisie au hasard)
- prix d'un actif financier
- etc.

Une v.a. (résultat) peut être :

- discrète (ensemble fini, dénombrable)
- continue (intervalle fini, infini)

OIEC (4103010) – M. Gilli

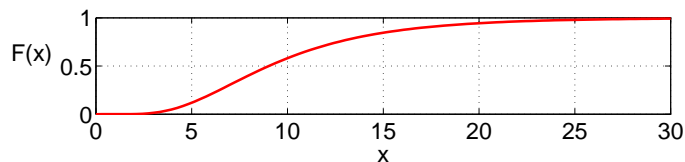
Eté 2008 – 30

## Fonction de distribution, probabilité, densité

On note  $X$  la variable aléatoire ( $X$  désigne l'ensemble des valeurs que peut prendre la v.a.)

- Fonction (cumulative) de distribution  $F$  :

$$F(x) = P\{X \leq x\} \quad x \in \mathbb{R}$$



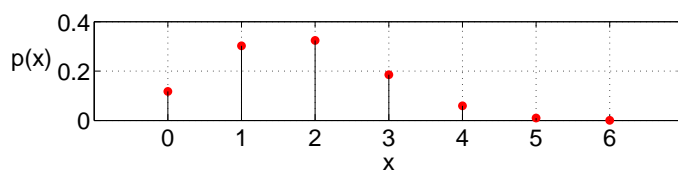
OIEC (4103010) – M. Gilli

Eté 2008 – 31

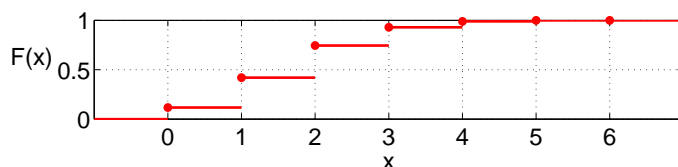
## Fonction de distribution, probabilité, densité

- Probabilité (v.a. discrète, nombre fini de valeurs possibles)

$$p(x) = P\{X = x\} \quad \sum_{i=1}^n p(x_i) = 1$$



$$F(x) = P\{X \leq x\} \quad x \in \mathbb{R}$$



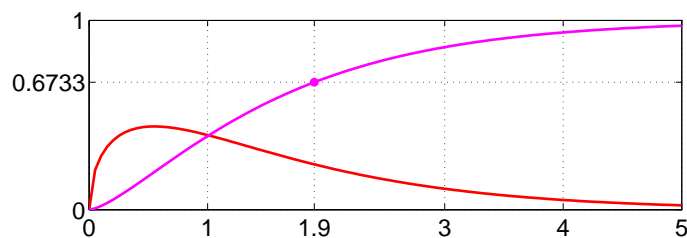
OIEC (4103010) – M. Gilli

Eté 2008 – 32

## Fonction de distribution, probabilité, densité

- Densité  $f(x)$  (v.a. continue)

$$F(a) = P\{X \in ]-\infty, a]\} = \int_{-\infty}^a f(x)dx$$



OIEC (4103010) – M. Gilli

Été 2008 – 33

## Quelques v.a. paramétrées

Discrètes :

- Bernouilli
- Binomiale
- Poisson
- Géométrique
- Binomiale négative
- Hypergéométrique
- ...

Continues :

- Uniforme
- Normale
- Lognormale
- Processus de Poisson et loi gamma
- Pareto
- ...

OIEC (4103010) – M. Gilli

Été 2008 – 34

## v.a. uniformément distribuée

$X$  est dit uniformément distribuée sur l'intervalle  $(a, b)$ ,  $a < b$ , si la densité est

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{si } a < x < b \\ 0 & \text{sinon} \end{cases}$$

On peut montrer qu'à partir d'une v.a. uniforme dans l'intervalle  $(0, 1)$  on peut construire une v.a. quelconque en procédant à des transformations appropriées.

Ex. : Bernouilli (résultat lancer pièce de monnaie)

OIEC (4103010) – M. Gilli

Été 2008 – 35

## Génération d'une v.a. uniforme $[0, 1[$

Exigences :

- Efficace (générer  $10^7$  valeurs/s)
- Reproductible!!!
- Bonne qualité (longue période, etc.)
- Portable (reproduire mêmes sequences sur machines différentes)

Solution :

- Modéliser une procédure complexe pour "assurer le caractère aléatoire". Mais absence de compréhension théorique du processus modélisé conduit souvent à des résultats désastreux!!
- Observer un phénomène physique aléatoire (émission de particules d'une source radioactive)
- Modèle mathématique  $\rightarrow$  v.a. pseudo-aléatoire

Il existe de nombreux modèles mathématiques (sujet de recherche très actif).

On présente la classe des générateurs *linéaires congruents*.

OIEC (4103010) – M. Gilli

Eté 2008 – 36

## Générateur linéaire congruent

- Fixer une valeur initiale  $x_0$ , puis calculer récursivement pour  $i \geq 1$

$$x_i = a x_{i-1} \pmod{M} \tag{1}$$

avec  $a$  et  $M$  des entiers donnés.

- $x_i \in \{0, 1, \dots, M-1\}$
- $x_i/M$  (*variable pseudo-aléatoire*), approche une v.a. uniforme dans l'intervalle  $[0, 1[$ .

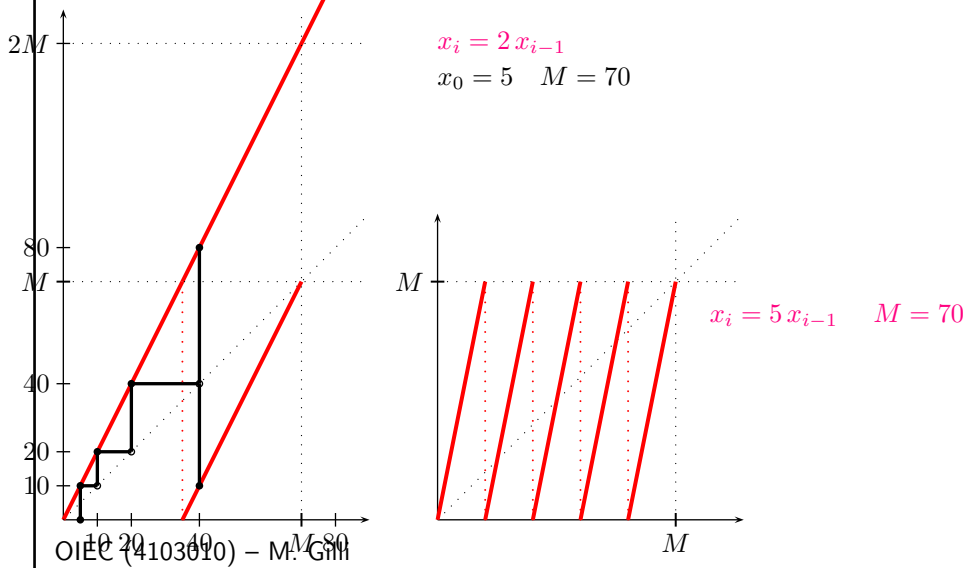
Exemple :  $x_i = 2 x_{i-1}$  avec  $x_0 = 5$ ,  $M = 70$

$i$	$x_i$	$x_i \pmod{70}$	$\frac{x_i \pmod{70}}{70}$
1	10	10	0.1429
2	20	20	0.2857
3	40	40	0.5714
4	80	10	0.1429
5	160	20	0.2857
...	...		

OIEC (4103010) – M. Gilli

Eté 2008 – 37

## Générateur linéaire congruent



Été 2008 – 38

## Générateur linéaire congruent

Propriétés en fonction de  $a$  et  $M$  :

- Longueur du cycle (très important !!)
- Couverture de l'intervalle (distance entre les valeurs générées, granularité).

Modulo fait évoluer la fonction sur un tore de longueur  $M$  et de circonférence  $M$ .

OIEC (4103010) – M. Gilli

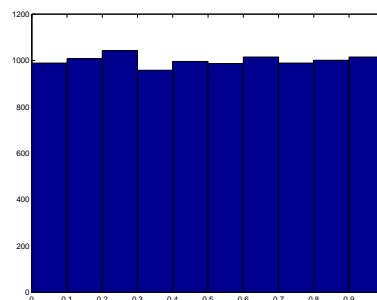
Été 2008 – 39

## Génération de v.a. uniforme (0, 1) avec Matlab

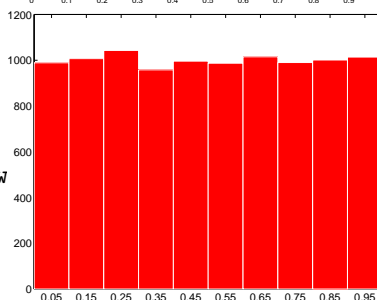
Période =  $2^{1492}$ , granularité (tous les  $F \in [0, 1]$ )

- `rand`
- `rand('seed', n)`  $n$  est un point de départ quelconque
- `rand(n, m)` génère une matrice  $n \times m$

```
y = rand(10000,1); hist(y);
```



```
x = 0.05:0.1:0.95
hist(y,x)
h = findobj(gca,'Type','patch');
set(h,'FaceColor','r','EdgeColor','w')
set(gca,'xtick',x);
set(gca,'FontSize',14);
```



OIEC (4103010) – M. Gilli

Été 2008 – 40

## Génération de v.a. discrètes

Comment générer des v.a. discrètes à partir de  $U(0, 1)$ ? On utilise la fonction de repartition de la v.a. discrète. Soit une v.a. discrète  $X$  :

$$P(X = x_j) = p_j \quad j = 0, 1, 2, \dots \quad \sum p_j = 1$$

$$F(x_k) = \sum_{j=0}^k p_j$$

$$p_i = F(x_i) - F(x_{i-1})$$

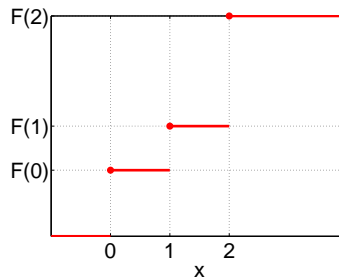
On génère  $u \in U(0, 1)$ ,  
si  $F(x_{i-1}) \leq u < F(x_i)$   
on a généré  $x_i$ . Ex. :

$$X = \{0, 1, 2\}$$

$$P(X = 0) = p_0 = 0.3$$

$$P(X = 1) = p_1 = 0.2$$

$$P(X = 2) = p_2 = 0.5$$



OIEC (4103010) – M. Gilli

Été 2008 – 41

## Génération de v.a. discrètes

Génération d'une v.a. discrète  $X$ ,  $x_i$ ,  $p_i$ ,  $i = 1, 2, \dots, n$  donnés.

$i = 1$ ,  $F = p_i$ , done = 0, générer  $u \in U(0, 1)$

**while** ~done **do**

**if**  $u < F$  **then**

$X = x_i$ , done = 1

**else**

$i = i + 1$ ,  $F = F + p_i$

**end if**

**end while**

Remarque : On peut procéder plus efficacement en procédant par ordre décroissant de la probabilité.

OIEC (4103010) – M. Gilli

Été 2008 – 42

## Génération de v.a. discrètes

Ex. : Génération d'une v.a. de Poisson

$$p_i = P(X = i) = e^{-\lambda} \frac{\lambda^i}{i!} \quad i = 0, 1, 2, \dots$$

$$p_{i+1} = \frac{\lambda}{i+1} p_i$$

$i = 0$ ,  $p = \exp(-\lambda)$ ,  $F = p$ , générer  $u \in U(0, 1)$ , done = 0

**while** ~done **do**

**if**  $u < F$  **then**

$X = i$ , done = 1

**else**

$p = \lambda p / (i + 1)$ ,  $F = F + p$ ,  $i = i + 1$

**end if**

**end while**

OIEC (4103010) – M. Gilli

Été 2008 – 43

## Génération de v.a. discrètes

```
function x = vaPoisson1(lambda)
i = 0;
p = exp(-lambda);
F = p;
u = rand;
done = 0;
while ~done
    if u < F
        x = i;
        done = 1;
    else
        p = lambda*p / (i+1);
        F = F + p;
        i = i + 1;
    end
end
end
```

```
function x = vaPoisson(lambda,n)
x = zeros(n,1);
for k = 1:n
    i = 0;
    p = exp(-lambda);
    F = p;
    u = rand;
    done = 0;
    while ~done
        if u < F
            x(k) = i;
            done = 1;
        else
            p = lambda*p / (i+1);
            F = F + p;
            i = i + 1;
        end
    end
end
end
```

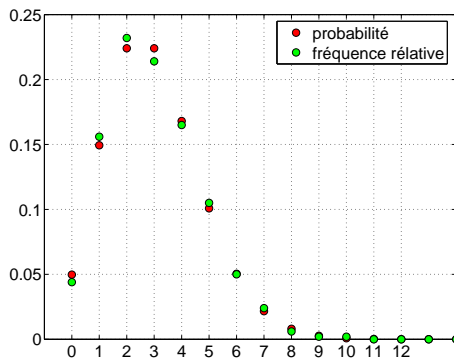
OIEC (4103010) – M. Gilli

Eté 2008 – 44

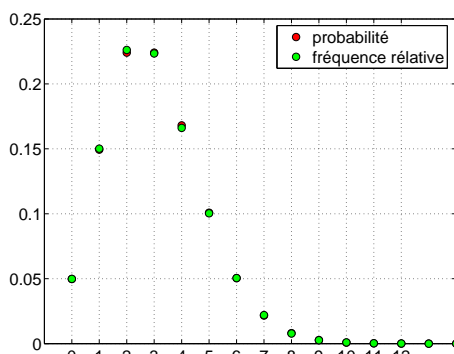
## Exemple de génération de v.a de Poisson

Avec Matlab on utilise `poissrnd(n,m)` (plus efficace)

Génération de 1000 v.a. de Poisson avec  $\lambda = 3$   
 Comparaison probabilité avec fréquence relative



Génération de 100 000 v.a. de Poisson avec  $\lambda = 3$



Question : Combien de variables faut-il générer pour avoir au moins une réalisation de  $x = 15$  (avec une probabilité donnée)

OIEC (4103010) – M. Gilli

Eté 2008 – 45

## Génération d'une v.a. binomiale

$X \sim B(n, p)$

$$P(X = i) = C_n^i p^i (1-p)^{n-i}$$
$$P(X = i + 1) = \frac{n-i}{i+1} \frac{p}{1-p} P(X = i)$$

Une autre façon consiste à générer  $n$  v.a. de Bernouilli  $B(p)$  et de faire la somme.

Génération d'une v.a. de Bernouilli  $B(p)$  :

Générer  $u \in U(0, 1)$

**if**  $u < p$  **then**

$X = 1$

**else**

$X = 0$

**end if**

OIEC (4103010) – M. Gilli

Été 2008 – 46

## Génération de v.a. continues

○ Génération de v.a. continues normales  $N(0, 1)$

○ Transformation de Box-Muller

A partir de deux variables uniformes  $u_1$  et  $u_2$  on construit deux v.a.  $X$  et  $Y$  normales  $N(0, 1)$

1: Générer  $u_1, u_2, \in U(0, 1)$

2:  $v = \sqrt{-2 \log(u_1)}$

3:  $X = v \cos(2\pi u_2)$

4:  $Y = v \sin(2\pi u_2)$

○ Avec Matlab on utilise la fonction `randn`, `randn(n,m)`

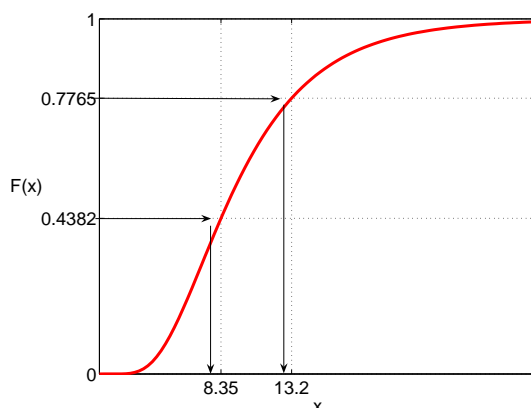
OIEC (4103010) – M. Gilli

Été 2008 – 47

## Génération de v.a. continues

○ Génération de v.a. continues quelconques

Transformation inverse de la fonction de répartition  $F$



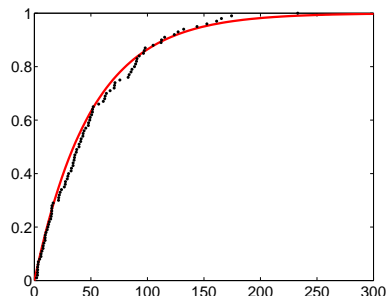
OIEC (4103010) – M. Gilli

Été 2008 – 48

## Exemple de génération de v.a. continues

- $F(x) = x^n$  avec  $0 < x < 1$   
 $u = F(x) = x^n \rightarrow x = u^{1/n}$
- $F(x) = 1 - e^{-\lambda x}$  avec  $0 < x < \infty$   
 $u = F(x) = 1 - e^{-\lambda x} \rightarrow e^{-\lambda x} = 1 - u$   
 $-\lambda x = \log(1 - u)$   
 $x = -\log(u)/\lambda$

```
F = inline('1-exp(-P1*x)',1);  
lambda = .02;  
y = linspace(0,6);  
plot(y,F(y,lambda),'r'); hold on  
n = 100;  
Fe = (1:n)'/n;  
u = rand(1,n);  
z = -log(u)/lambda;  
z = sort(z);  
plot(z,Fe,'k.')
```



OIEC (4103010) – M. Gilli

Été 2008 – 49

## Remark

However we can also use  $x = -\log(u)/\lambda$  as this is the complement of  $1 - u$  and is also  $U(0, 1)$

OIEC (4103010) – M. Gilli

Été 2008 – note 1 of slide 49

## Fonction de distribution empirique

Fonction cumulative de probabilité qui assigne la probabilité  $1/n$  à chaque observation dans l'échantillon de taille  $n$

Soit  $X_1, \dots, X_n$  des variables aléatoires indépendantes, alors la fonction de distribution empirique  $F_n(x)$  de l'échantillon  $X_1, \dots, X_n$  est donné par

$$\begin{aligned} F_n(x) &= \frac{\text{nombre d'éléments dans l'échantillon } \leq x}{n} \\ &= \frac{1}{n} \sum_{i=1}^n I_{\{x_i \leq x\}} \end{aligned}$$

OIEC (4103010) – M. Gilli

Été 2008 – 50

x. : Générer une v.a.  $\sim N(25, 9)$

### Zéros d'une fonction

Il s'agit de trouver la valeur de  $x$  qui satisfait l'équation

$$f(x) = 0$$

La solution est appelée **zéro de la fonction** (il peut y avoir plusieurs solutions).

Ex. :  $1 - e^{-x} = 0.5$  Pour retrouver la formulation  $f(x) = 0$  on ramène tous les termes à gauche du signe de l'égalité

$$1 - e^{-x} - 0.5 = 0$$

Dans ce cas il est facile de trouver la solution analytique

$$x = -\log(0.5)$$

Souvent dans la pratique, soit il n'existe pas de solution analytique à ce problème, soit son obtention est très laborieuse

→ On recourt à la **solution numérique**

OIEC (4103010) – M. Gilli

Été 2008 – 51

### Résolution graphique

Soit la fonction

$$\left(1 + \frac{1}{n-1}\right)^x = x^n \quad n > 1 \quad \text{et} \quad x \in [x_L, x_U]$$

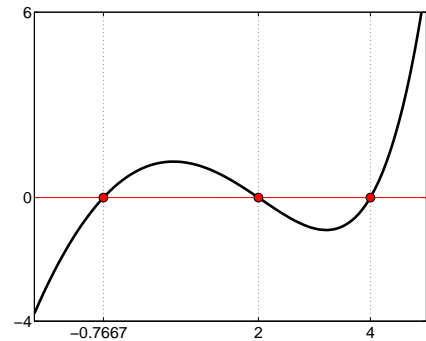
Graphique de la fonction :

```
f = inline('(1 + 1/(P1-1)).^x - x.^P1',1);
xL = -2; xU = 5;
x = linspace(xL,xU);
plot(x,f(x,2));
```

P1 correspond au paramètre  $n = 2$ .

On a utilisé les opérations élément par élément pour pouvoir évaluer  $f(x)$  avec  $x$  un vecteur.

OIEC (4103010) – M. Gilli



Été 2008 – 52

### Résolution aléatoire

Il s'agit de générer aléatoirement des valeurs pour  $x$  et de calculer la valeur de  $f(x)$  correspondante. Si  $X$  est l'ensemble des valeurs générées la solution  $x_{sol}$  est définie comme

$$x_{sol} = \operatorname{argmin}_{x \in X} |f(x)|$$

Pour la génération des  $x$  on se sert de la fonction Matlab rand qui génère des variables pseudo-aléatoires uniformément distribuées dans l'intervalle  $[0, 1[$ .

$x_L, x_U$  et la v.a. uniforme  $u$  données on génère  $x$  comme

$$x = x_L + (x_U - x_L)u$$

OIEC (4103010) – M. Gilli

Été 2008 – 53

## Résolution aléatoire (cont'd)

Nous avons choisi  $x_{\min} = -10$  et  $x_{\max} = 10$  et avons évalué la fonction  $f(x)$  pour  $R = 100\,000$  valeurs aléatoires de  $x$ . Le code Matlab est le suivant :

```
n = 2; xmin = -10; xmax = 10;
R = 100000;
x = xmin + (xmax - xmin)*rand(R,1);
z = f(x,n);
[sol,xind] = min(abs(z));
fprintf('\n f(%8.6f) = %8.6f\n',x(xind),sol);
```

On a trouvé la solution suivante  $f(-0.766735) = 0.000136$   
(depend de la séquence de v.a. générée!!)

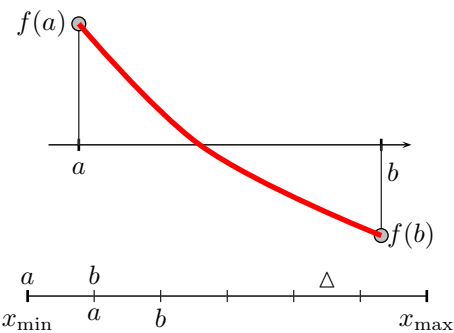
OIEC (4103010) – M. Gilli

Été 2008 – 54

## Recoupelement par intervalles (bracketing)

Il s'agit de construire des intervalles susceptibles de contenir un zéro. Ultérieurement on raffiner la recherche du zéro à l'intérieur de l'intervalle.

On subdivise un domaine donné en intervalles réguliers et on examine si la fonction traverse l'axe des  $x$  en analysant le signe de la fonctions évaluée aux bords de l'intervalle.



```
1:  $\Delta = (x_{\max} - x_{\min})/n$ 
2:  $a = x_{\min}$ 
3: for  $i = 1 : n$  do
4:    $b = a + \Delta$ 
5:   if  $\text{sign } f(a) \neq \text{sign } f(b)$  then
6:      $[a, b]$  peut contenir un zéro,
       imprimer
7:   end if
8:    $a = b$ 
9: end for
```

OIEC (4103010) – M. Gilli

Été 2008 – 55

## Matlab code pour bracketing

```
function ab = bracketing(f,xmin,xmax,n)
k = 0;
delta = (xmax - xmin)/n;
a = xmin;
fa = feval(f,a);
for i = 1:n
    b = a + delta;;
    fb = feval(f,b);
    if sign(fa)~=sign(fb)
        k = k + 1;
        ab(k,:) = [a b];
    end
    a = b;
    fa = fb;
end
```

OIEC (4103010) – M. Gilli

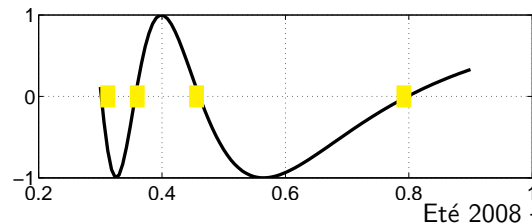
Été 2008 – 56

## Matlab code pour bracketing

Recherche des intervalles pour la fonction  $g(x) = \cos(1/x^2)$  dans le domaine  $x \in [0.3, 0.9]$  et  $n = 25$ .

```
f = inline('cos(1./x.^2)');
iv = bracketing(g,0.3,0.9,25);
x = linspace(0.3,0.9);
subplot(211)
plot(x,f(x),'k','LineWidth',3), hold on
for i = 1:size(iv,1)
    plot(iv(i,:),[0 0],'LineWidth',20,'Color',[1 239/255 0])
end
```

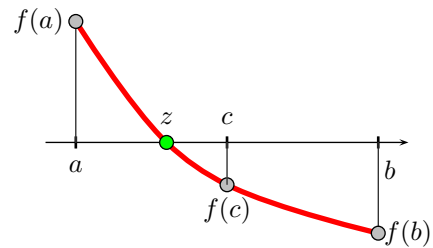
```
iv =
    0.3000    0.3240
    0.3480    0.3720
    0.4440    0.4680
    0.7800    0.8040
```



OIEC (4103010) – M. Gilli

Été 2008 – 57

## Méthode de la bisection



Recherche le zéro d'une fonction dans un intervalle  $[a, b]$  donné.

On considère un intervalle qui contient le zéro, on le divise en deux et on cherche le demi-intervalle qui contient le zéro.

La procédure est répétée jusqu'à ce que l'on atteigne un intervalle suffisamment petit.

$$c = \frac{a+b}{2} \quad \equiv a + \frac{b-a}{2} \quad (\text{numériquement plus stable})$$

OIEC (4103010) – M. Gilli

Été 2008 – 58

## Algorithme de la bisection

$\eta$  tolérance d'erreur donnée

- 1: Vérifier que  $f(a) \times f(b) < 0$
- 2: **while**  $|a - b| > \eta$  **do**
- 3:    $c = a + (b - a)/2$
- 4:   **if**  $f(c) \times f(a) < 0$  **then**
- 5:      $b = c$      ( $z$  est à gauche de  $c$ )
- 6:   **else**
- 7:      $a = c$      ( $z$  est à droite de  $c$ )
- 8:   **end if**
- 9: **end while**

OIEC (4103010) – M. Gilli

Été 2008 – 59

## Code Matlab de la bisection

```
function c = bisection(f,a,b,tol)
% Recherche zero d'une fonction avec methode de la bisection
% On cherche zero dans intervalle [a, b] avec tolerance tol
% La fonction doit etre de signe oppose en a et b
if nargin == 3, tol = 1e-8; end
fa = feval(f,a); fb = feval(f,b);
if sign(fa) == sign(fb), error('f pas de signe oppose en a et b'); end
fait = 0;
while abs(b-a) > 2*tol & ~fait
    c = a + (b - a) / 2;      % Chercher centre intervalle
    fc = feval(f,c);        % Evaluer f au centre
    if sign(fa) ~= sign(fc)  % zero a gauche de c
        b = c; fb = fc;
    elseif sign(fc) ~= sign(fb) % zero a droite de c
        a = c; fa = fc;
    else
        fait = 1;          % On tombe exactement sur zero
    end
end
end
```

OIEC (4103010) – M. Gilli

Été 2008 – 60

## Application de la bisection

```
g = inline('cos(1./x.^2)');
for i = 1:size(iv,1)
    z(i) = bisection(g,iv(i,1),iv(i,2));
end

z =
    0.3016    0.3568    0.4607    0.7979

iv =
    0.3000    0.3240
    0.3480    0.3720
    0.4440    0.4680
    0.7800    0.8040
```

OIEC (4103010) – M. Gilli

Eté 2008 – 61

## Itérations de point fixe

Soit  $f(x) = 0$ , on isole un terme contenant  $x$  de la sorte à pouvoir écrire

$$x_{\text{new}} = g(x_{\text{old}}) \quad \mathbf{g} \text{ est appelée fonction d'itération}$$

L'itération du point fixe est définie par l'algorithme :

- 1: Initialiser  $x^{(0)}$
- 2: **for**  $k = 1, 2, \dots$  jusqu'à convergence **do**
- 3:  $x^{(k)} = g(x^{(k-1)})$
- 4: **end for**

- o Notion de convergence doit être approfondie
- o Les valeurs successives de  $x$  ne doivent pas être conservées dans un vecteur

```
while ~converged
    x1 = g(x0)
    x0 = x1
end
```

- o Le fait que la solution  $x_{\text{sol}}$  vérifie  $x_{\text{sol}} = g(x_{\text{sol}}) \rightarrow$  on appelle  $x_{\text{sol}}$  **point fixe**
- o Choix de la fonction  $g(x)$  est déterminant pour la convergence de la méthode

OIEC (4103010) – M. Gilli

Eté 2008 – 62

## Code Matlab pour la méthode du point fixe

```
function x1 = FPI(f,x1,tol)
% FPI(f,x0) Iterations du point fixe
if nargin == 2, tol = 1e-8; end
it = 0; itmax = 100; x0 = realmax;
while ~converged1(x0,x1,tol)
    x0 = x1;
    x1 = feval(f,x0);
    it = it + 1;
    if it > itmax, error('Maxit in FPI'); end
end

function rep = converged1(x0,x1,tol)
rep = abs(x1-x0)/(abs(x0)+1) < tol;
```

OIEC (4103010) – M. Gilli

Eté 2008 – 63

## Application de la méthode du point fixe

Soit  $f(x) = x - x^{4/5} - 2 = 0$  et les 2 fonctions d'itération

$$g_1(x) = x^{4/5} + 2 \quad g_2(x) = (x - 2)^{5/4}$$

```
g1 = inline('x^(4/5)+2');
FPI(g1,1)
```

```
ans =
    6.4338
```

```
g2 = inline('(x-2)^(5/4)');
FPI(g2,8)
```

```
??? Error using ==> fpi
Maxit in FPI
```

OIEC (4103010) – M. Gilli

Eté 2008 – 64

## Application de la méthode du point fixe (cont'd)

Soit  $h(x) = x - 2x^{4/5} + 2 = 0$  et les 2 fonctions d'itération

$$g_1(x) = 2x^{4/5} - 2 \quad g_2(x) = \left(\frac{x+2}{2}\right)^{5/4}$$

```
h = inline('x - 2*x.^(4/5) + 2');
bracketing(h,0,50,30)
ans =
    3.3333    5.0000
   18.3333   20.0000

g2 = inline('((x+2)/2).^(5/4)');
FPI(g2,18.5)
ans =
    19.7603
FPI(g2,3,5)
??? Error using ==> fpi
Maxit in FPI

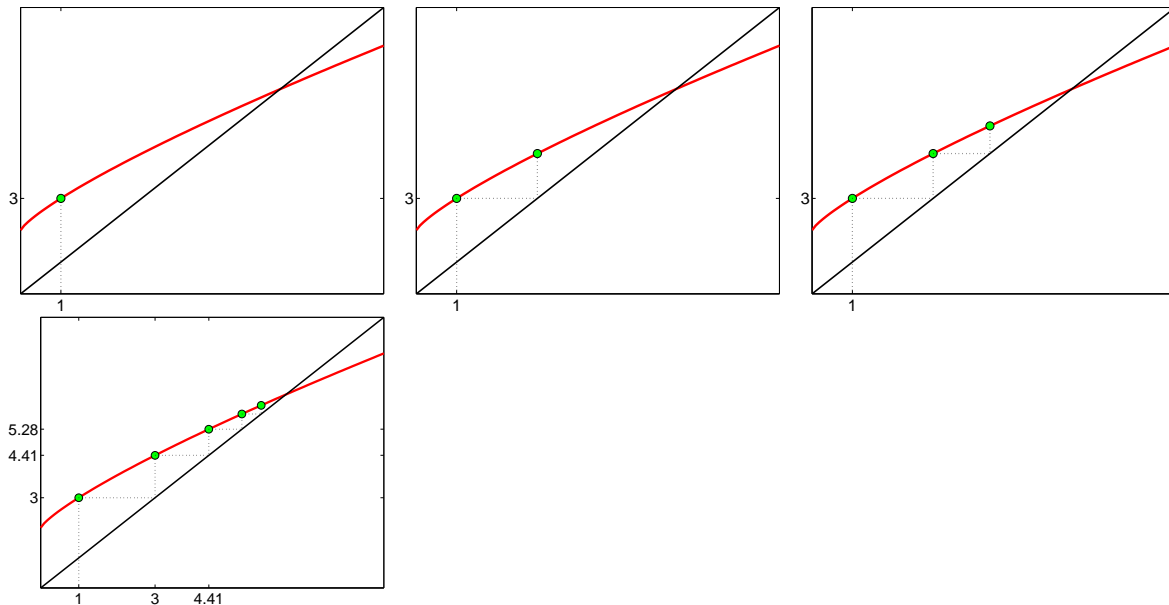
g1 = inline('2*x.^(4/5) - 2');
FPI(g1,18.5)
ans =
    19.7603
```

OIEC (4103010) – M. Gilli

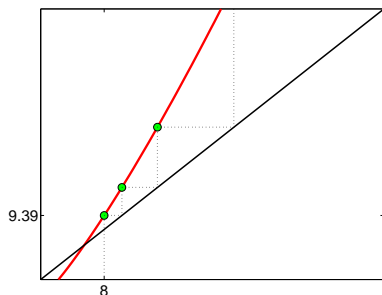
Été 2008 – 65

## Convergence de la méthode de point fixe (illustrations)

Soit  $f(x) = x - x^{4/5} - 2 = 0$  et la fonctions d'itération  $g_1(x) = x^{4/5} + 2$



Mais avec la fonctions d'itération  $g_2(x) = (x - 2)^{5/4}$

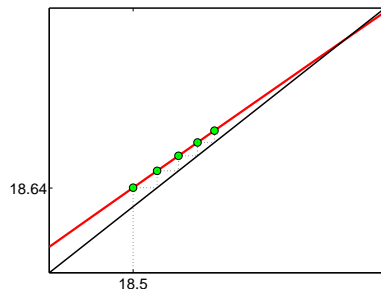


OIEC (4103010) – M. Gilli

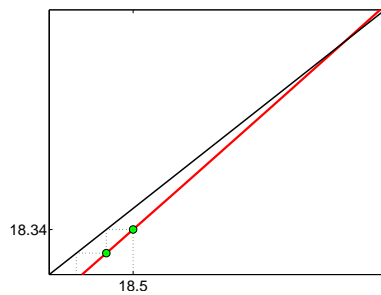
Été 2008 – 66

### Convergence de la méthode de point fixe (illustrations)

Soit  $f(x) = x - 2x^{4/5} + 2 = 0$  et la fonctions d'itération  $g_1(x) = 2x^{4/5} - 2$



Mais avec la fonctions d'itération  $g_2(x) = \left(\frac{x+2}{2}\right)^{5/4}$

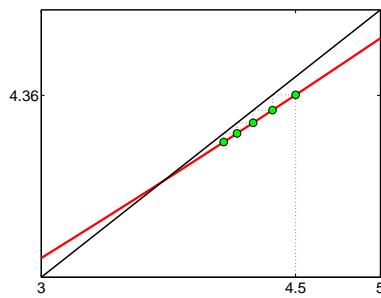


OIEC (4103010) – M. Gilli

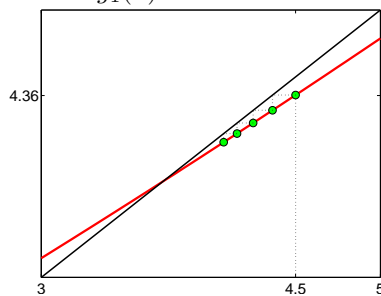
Été 2008 – 67

### Convergence de la méthode de point fixe (illustrations)

Mais dans l'intervalle  $x \in [3, 5]$  avec  $g_2(x) = \left(\frac{x+2}{2}\right)^{5/4}$



et avec  $g_1(x) = 2x^{4/5} - 2$  on converge



OIEC (4103010) – M. Gilli

Été 2008 – 68

## Convergence de la méthode de point fixe

Les itérations de point fixe convergent pour  $x \in [a, b]$  si l'intervalle contient un zéro et si

$$|g'(x)| < 1 \quad \text{pour } x \in [a, b]$$

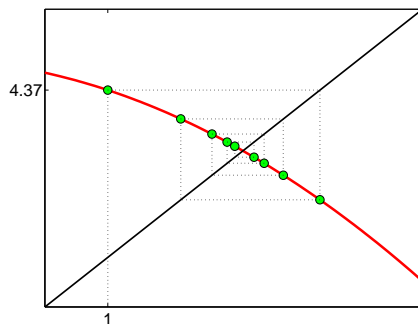
Si  $-1 < g'(x) < 0$  les itérations oscillent autour de la solution et si  $0 < g'(x) < 1$  les itérations convergent de façon monotone

OIEC (4103010) – M. Gilli

Été 2008 – 69

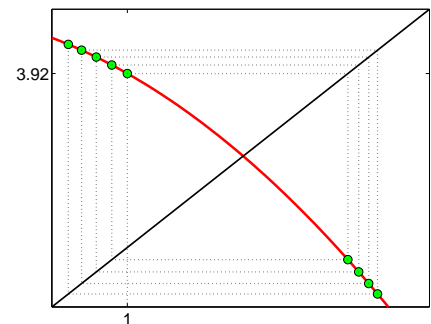
## Convergence de la méthode de point fixe (illustrations)

$-1 < g'(x) < 0$



OIEC (4103010) – M. Gilli

$g'(x) < -1$



Été 2008 – 70

## Remarques :

Suivant la valeur de la dérivée une même fonction d'itération peut converger pour certains intervalles et diverger pour d'autres!!!

OIEC (4103010) – M. Gilli

Été 2008 – note 1 of slide 70

## Méthode de Newton

Dérivée à partir de l'expansion de la fonction  $f$  en série de Taylor :

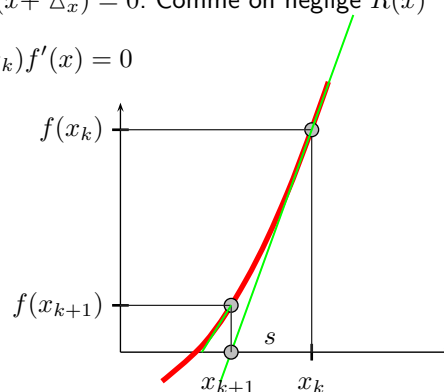
$$f(x + \Delta x) = f(x) + \Delta x f'(x) + R(x)$$

On cherche le pas  $\Delta x$  pour lequel  $f(x + \Delta x) = 0$ . Comme on néglige  $R(x)$

$$f(x_{k+1}) \approx f(x_k) + (x_{k+1} - x_k)f'(x_k) = 0$$

d'où

$$x_{k+1} = x_k - \underbrace{\frac{f(x_k)}{f'(x_k)}}_s$$



OIEC (4103010) – M. Gilli

Été 2008 – 71

## Algorithme de Newton

```
1: Initialiser  $x_0$ 
2: for  $k = 0, 1, 2, \dots$  jusqu'à convergence do
3:    $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ 
4: end for
```

Créer une fonction qui évalue  $f$  et sa dérivée (2 arguments de sortie)

```
function x1 = Newton(f,x1,tol)
% Newton(f,x0) Methode de Newton
if nargin == 2, tol = 1e-8; end
it = 0; itmax = 10; x0 = realmax;
while ~converged1(x0,x1,tol)
    x0 = x1;
    [fx,dfx] = feval(f,x0);
    x1 = x0 - fx/dfx;
    it = it + 1;
    if it > itmax, error('Maxit in Newton'); end
end
```

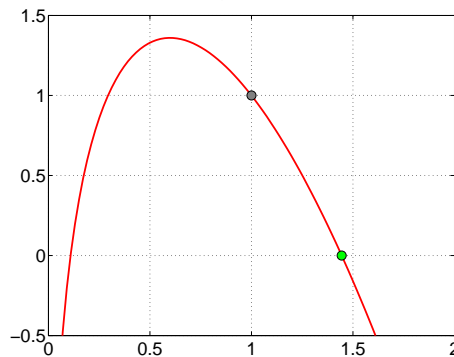
OIEC (4103010) – M. Gilli

Été 2008 – 72

## Exemple d'application de l'algorithme de Newton

```
function [f,d] = myf(x)
f = exp(-x).*log(x)-x.^2+2;
d = -exp(-x).*log(x)+exp(-x)./x-2*x;
```

```
x1 = Newton0('myf',1)
```



2 Newton avec dérivée approché numériquement → quasi-Newton

OIEC (4103010) – M. Gilli

Été 2008 – 73

## NewtonG

NewtonG donne une illustration graphique de l'algorithme de Newton. Voir aussi ZeroFunctionSlides

OIEC (4103010) – M. Gilli

Été 2008 – note 1 of slide 73

## Remarques sur la recherche du zéro d'une fonction

- Intervalle
- Point de départ
- Robustesse

OIEC (4103010) – M. Gilli

Été 2008 – 74

## References

Levy, M., H. Levy and S. Solomon (2000). *Microscopic Simulation of Financial Markets*. Academic Press.

OIEC (4103010) – M. Gilli

Été 2008 – 75