Numerical Methods in Finance

Manfred Gilli

Department of Econometrics
University of Geneva and
Swiss Finance Institute

Spring 2008

Introduction
- Outline
- References
- Problems
- Examples

Introduction à la programmation structurée
- Notion de programme
- Programmation structurée (affectation, repetition, choix)
- Syntaxe Matlab pour répétition et choix

Examples de programmes
- Moyenne éléments d'un vecteur et précision machine
- Maximum des éléments d'un vecteur
- Tri à bulles
- Simulation du lancer d'un dès

Introduction à Matlab
- Eléments de syntaxe
- Opérations éléments par éléments et script files
- Graphiques 2D $y = x^2\, e^x$
- Graphiques 3D $z = x^2\, y^2$

Programmation avec Matlab
- Réalisation d'une fonction
- Fonction `inline`
- Fonction `feval`
- Exemple de fonction (*call* Européen)

Computer arithmetic
- Representation of real numbers
- Machine precision
- Example of limitations of floating point arithmetic

Numerical differenciation
- Forward-difference
- Backward-difference
- Central-difference
- Approximating second derivatives
- Partial derivatives
- How to choose h
- An example

Measuring the error
- Absolute and relative error
- Combining absolute and relative error

Numerical instability and ill conditioning
- Numerically unstable algorithm
- Numerically stable algorithm
- Ill conditioned problem
- Condition of a matrix

Complexity of algorithms
- Order of complexity $\mathcal{O}(\cdot)$ and classification
- Evolution of performance

Equations differentielles
- Exemple de solution numérique
- Classification des equations différentielles
- Equation de Black-Scholes

Finite difference methods
- Definition of the grid
- Formalization of the system of finite difference equations

Finite difference solution
LU factorization
Comparison of FDM
Coordinate transformation

American options
Projected SOR (PSOR)
Explicit payout (EP)
Example

Monte-Carlo methods
Generation of uniform random variables $[0, 1[$
Simulation of asset price $S(t)$
Efficient simulation of price paths
Valuation of an European call
Confidence intervals
Computing confidence intervals with Matlab
Examples
Convergence rate
Variance reduction
Antithetic variates
"Crude" vs antithetic Monte Carlo
Quasi-Monte Carlo
Halton sequences
Box-Muller transformation
Example for Halton sequences

Lattice methods
Binomial trees
Multiplicative binomial tree algorithm
American put

Portfolio selection
Mean–variance model (Markowitz (1952))
Solving the QP with Matlab
Computing the efficient frontier
Portfolio with risk-free asset (cash)
Holding size constraints
Profile of efficient portfolios

Introduction

The aim is to provide an overview of the basic computational tools that are used by financial engineers. The course combines an introduction to pricing financial derivatives with finite differences, Monte Carlo simulation and lattice based methods and to portfolio selection models.
Emphasis is on practical implementations and numerical issues.

Outline

- Introduction to programming
  - Programming principles
  - Introduction to Matlab
  - Caveats of floating point computation
- Finite difference methods
  - Numerical approximation of derivatives
  - Coordinate transformations
  - Forward Euler scheme (explicit)
  - Backward Euler scheme (implicit)
  - Crank-Nicolson scheme
  - θ-scheme
  - Stability analysis

Outline (cont'd)

- Monte Carlo methods
  - Principle of Monte Carlo methods
  - Generation of random numbers
  - Evaluation of European-style options (crude MC)
  - Confidence intervals
  - Variance reduction techniques
  - Quasi-Monte Carlo methods
  - Examples of valuation of American and exotic options
- Lattice methods
  - The binomial method
  - Trinomial trees and finite difference methods
  - Implied trees and exotic options
- Portfolio selection
  - The mean-variance approach
  - Downside risk minimization

References

Brandimarte, P. (2002). *Numerical Methods in Finance. A MATLAB-Based Introduction*. Wiley Series in Probability and Statistics. Wiley.
Clewlow, L. and C. Strickland (1998). *Implementing Derivatives Models*. Wiley series in financial engineering. Wiley.
Higham, D. J. and N. J. Higham (2000). *Matlab Guide*. SIAM.
Hull, J.C. (1993). *Options, futures and other derivative securities*. Prentice-Hall.
Prisman, E. Z. (2000). *Pricing Derivative Securities: An Interactive Dynamic Environment with Maple V and Matlab*. Academic Press.
Tavella, D. and C. Randall (2000). *Pricing Financial Instruments : The Finite Difference Method*. Wiley series in Financial Engineering. Wiley. New York.
Wilmott, P. (1998). *Derivatives. The theory and Practice of Financial Engineering*. Wiley.
Wilmott, P., J. Dewynne and S. Howison (1993). *Option Pricing: Mathematical Models and Computation*. Oxford Financial Press.
Wilmott, P., S. Howison and J. Dewynne (1995). *The Mathematics of Financial Derivatives: A Student Introduction*. Cambridge University Press.

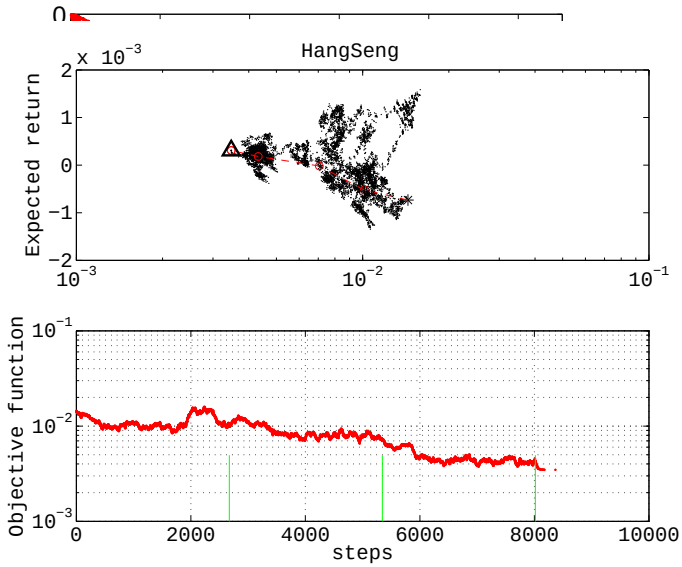
Problems

- Interest rate models
- Option pricing
- Free boundary problems
- Scenario simulation
- Portfolio selection
- Option replication and hedging
- Floating point computation
- Numerical approximation of derivatives
- Solution of $Ax = b$
- Random number generation
- Evaluation of integrals

Problems (cont'd)

- Binomial trees, Trinomial trees
- FDM explicit, implicit, ADI, Crank-Nicolson, θ-method
- FEM
- Spectral methods
- Projected-SOR
- Monte Carlo, (variance reduction)
- Quasi-Monte Carlo
- Static optimization, constraint, unconstraint, LP, QP
- Dynamic optimization, DP, stochastic control
- Neural networks
- Heuristic optimization

Examples


HangSeng
x 10^-3
Expected return
2
1
0
−1
−2
10^-3
10^-2
10^-1

Objective function
10^-1
10^-2
10^-3
0
2000
4000
6000
8000
10000
steps

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.

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).

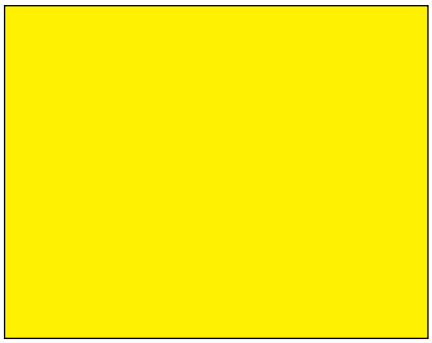
Question: Comment atteindre au mieux ces objectifs?

Réponse: Avec des techniques adéquates de programmation.

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.

Programmation structurée (affectation, repetition, 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

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
```


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.

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`.

Structure hierarchique

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.

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
```

Moyenne éléments d'un vecteur et précision machine

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

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

Calcul de la précision machine

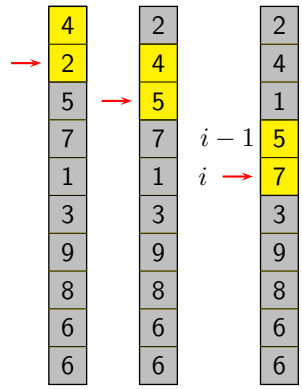
```
e = 1;
while 1 + e > 1
   e = e/2;
end
emach = 2*e;
```


Maximum des éléments d'un vecteur

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


Tri à bulles

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




```
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
```


Demonstration

`Bubblego`

Simulation du lancer d'un dès

Simuler n lancers d'un dès
(x_i, $i = 1, \ldots, n$ 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)
```


Demo

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

Eléments de syntaxe

- Polycopié
- Syntaxe (éléments de)
  - Symboles [ ] ; , ... %
  - *variable* = *expression*
  - `ans`
- `whos`, `help`, `lookfor`
- Variables (matrices par defaut)
- 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 + − ∗ / \ ^

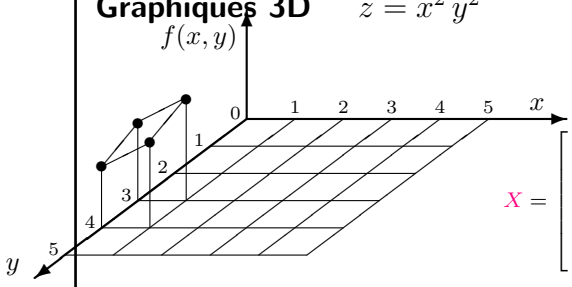
Opérations éléments par éléments et script files

- 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

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:');
```

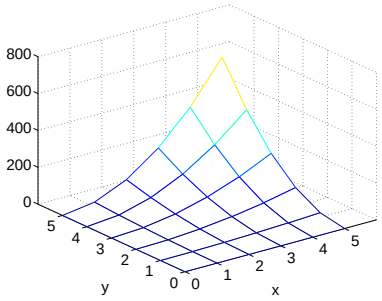

Graphiques 3D $z = x^2\, y^2$




```
x = linspace(0,5,6);
y = linspace(0,5,6);

[X,Y] = meshgrid(x,y);
```

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

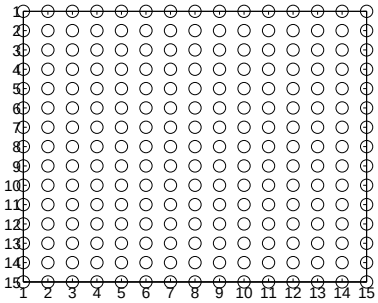
```
Z = X.^2 .* Y.^2;
mesh(Z);

mesh(x,y,Z);
```



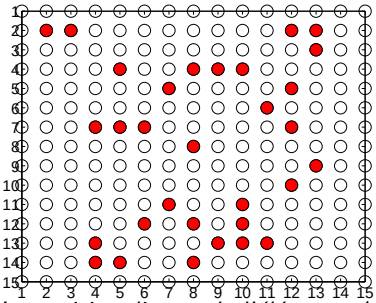
- `mesh`, `surf`, `plot3`, `hist3`, `bar3`, ...

Programmation d'un automate

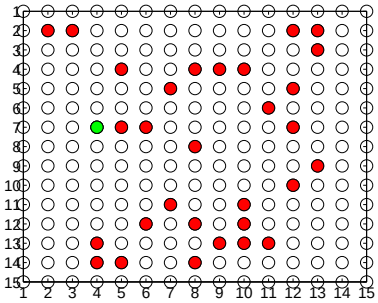
Soit une matrice A (vide)



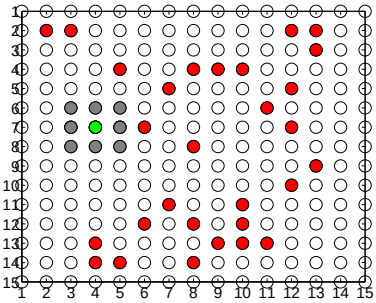
Initialisons quelques éléments avec la valeur 1



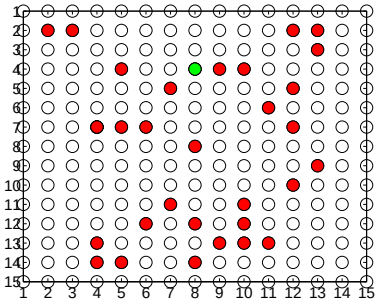
Considérons l'élément $A_{7,4}$



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

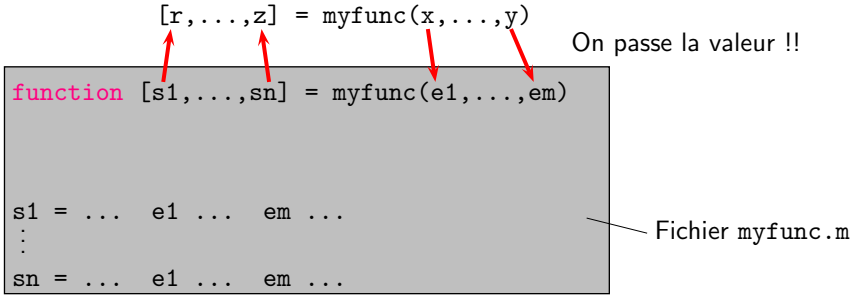


```
N = A(6,3) + A(6,4) + A(6,5) + A(7,3) + A(7,5) + ...
    A(8,3) + A(8,4) + A(8,5);
```



```
N = A(3,7) + A(3,8) + A(3,9) + A(4,7) + A(4,9) + ...
    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) + ...
    A(i+1,j-1) + A(i+1,j) + A(i+1,j+1);
```

Réalisation d'une fonction



- 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

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
```


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`.

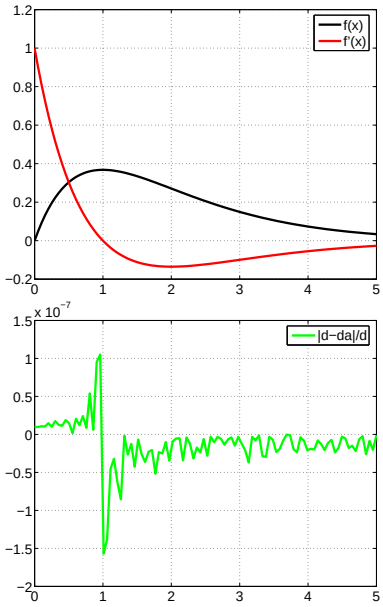
- `f = inline('expression')` si la fonction a un seul argument il n'y a pas d'ambiguïté.
  Ex.: $f(x) = x^2\, e^x$, `f = inline('x^2 * exp(x)'); f(0.75) = 1.1908`
- `f = inline('expression','arg1','arg2')` s'il y a plusieurs arguments
  Ex.: $f(x,y) = 2\sin(x)^2/\log(y)$, `f = inline('2*sin(x)^2 / log(y)','x','y');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

Fonction `feval`

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

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');
```



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;
```

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);
```

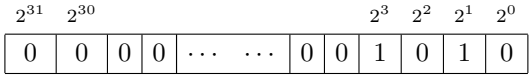
- Application:

```
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)
```

Two sources of errors appear systematically in numerical computation:

- *Truncation errors* due to the simplification of the mathematical model (e.g. replacement of a derivative by a finite difference, discretization);
- *Rounding errors* due to the fact that it is impossible to represent real numbers exactly in a computer.

The support for all information in a computer is a "word" constituted by a sequence of bits (generally 32), a bit having value 0 or 1.



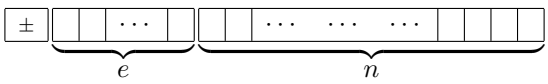
This sequence of bits is then interpreted as the binary representation of an integer. As a consequence integers can be represented exactly in a computer.
If all computations can be made with integers we face *integer arithmetic*.
Such computations are exact.

Representation of real numbers

In a computer, a real variable $x \neq 0$ is represented as

$$x = \pm n \times b^e$$

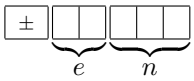
n is the mantissa (or fractional part), b the base and e the exponent
(base always 2).
(Ex.: $12.153 = .75956 \times 2^4$ or with base 10 we have 0.12153×10^2).
In practice we partition the word in three parts, one containing e the second n and the first bit from left indicates the sign.



Therefore, whatever size of the word, we dispose of a limited number of bits for the representation of the integers n et e.
As a result it will be impossible to represent real numbers exactly.

Representation of real numbers

To illustrate this fact, consider a word with 6 bits, with $t = 3$ the number of bits reserved for the mantissa, $s = 2$ bits reserved for the exponent and 1 bit for the sign.



$e =$

0	0	0
0	1	1
1	0	2
1	1	3

$n =$

1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Normalizing the mantissa, i.e. imposing the first bit from left being 1, we have $n \in \{4, 5, 6, 7\}$ and defining the *offset* of the exponent as $o = 2^{s-1} - 1$ we have

$$(0 - o) \leq e \leq (2^s - 1 - o),$$

i.e. $e \in \{-1, 0, 1, 2\}$. The real number x then varies between

$$(2^{t-1} \leq n \leq 2^t - 1) \times 2^e$$

Representation of real numbers

Adding 1 to the right of $(2^{t-1} \leq n \leq 2^t - 1) \times 2^e$ we get $n < 2^t$ and multiplying the whole expression by 2^{-t} we obtain the interval

$$(2^{-1} \leq n < 1) \times 2^{e-t}$$

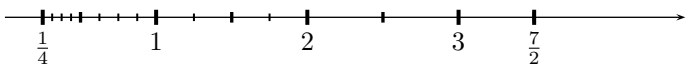
for the representation of real numbers.

Representation of real numbers

Set of real numbers $f = n \times 2^{e-t}$:

n	2^{e-t}			
	$e=-1$	$e=0$	$e=1$	$e=2$
	$1/16$	$1/8$	$1/4$	$1/2$
1 0 0 $=2^2=4$	$1/4$	$1/2$	1	2
1 0 1 $=2^2+2^0=5$	$5/16$	$5/8$	$5/4$	$5/2$
1 1 0 $=2^2+2^1=6$	$3/8$	$3/4$	$3/2$	3
1 1 1 $=2^2+2^1+2^0=7$	$7/16$	$7/8$	$7/4$	$7/2$

We also observe that the representable real numbers are not equally spaced.



Representation of real numbers

Matlab uses double precision (64 bits) with $t = 52$ bits for the mantissa
and $e \in [-1023, 1024]$ the range of integers for the exponent.

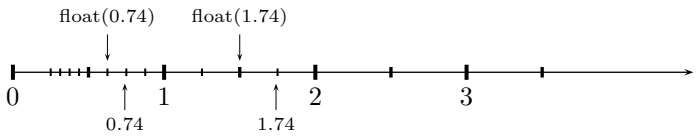
We then have $m \leq f \leq M$ with $m \approx 1.11 \times 10^{-308}$ and $M \approx 1.67 \times 10^{308}$.

If the result of an expression is inferior to m we get an "*underflow*"
if the result is superior to M we are in a situation of "*overflow*".

Representation of real numbers

The notation *float* $(\cdot)$ represents the result of a floating point computation.
A computer using the arithmetic of the preceding example ($t = 3$ and $s = 2$) would produce with "*chopping*"
the following numbers

float $(1.74) = 1.5$ and *float* $(0.74) = 0.625$



The result for *float* $(1.74 - 0.74) = 0.875$, illustrates that even if the exact result is in F its representation
might be different.

Machine precision

The *precision* of a floating point arithmetic is defined as the smallest positif number ϵ_{mach} such that

$$\textit{float}\ (1 + \epsilon_{\text{mach}}) > 1$$

The machine precision ϵ_{mach} can be computed with the following algorithm

1: $e = 1$
2: **while** $1 + e > 1$ **do**
3: $\quad e = e/2$
4: **end while**
5: $\epsilon_{\text{mach}} = 2e$

With Matlab we have $\epsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$.

Rounding errors and precision

Floating point computation is not associative.
We can observe that for $e = \epsilon_{\text{mach}}/2$ we obtain the following results:

$$\textit{float}\ ((1 + e) + e) = 1$$
$$\textit{float}\ (1 + (e + e)) > 1$$

Example of limitations of floating point arithmetic

Sometimes the approximation obtained with floating point arithmetic can be surprising. Consider the expression

$$\sum_{n=1}^{\infty} \frac{1}{n} = \infty$$

Computing this sum will, of course, produce a finite result. What might be surprising is that this sum is certainly inferior to 100. The problem is not generated by an "*underflow*" of $1/n$ or an "*overflow*" of the partial sum $\sum_{n=1}^{k} 1/n$ but by the fact that for n verifying

$$\frac{1/n}{\sum_{k=1}^{n-1} \frac{1}{k}} < \epsilon_{\text{mach}}$$

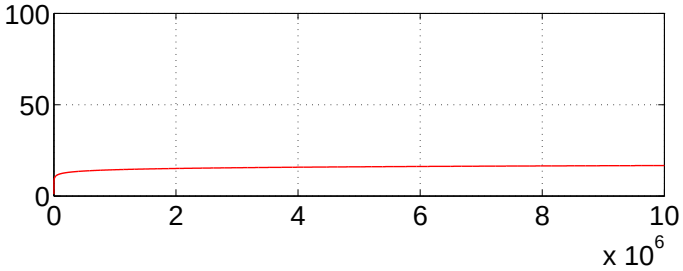
the sum remains constant.

Example of limitations of floating point arithmetic (contd.)

To show this we consider $e < \epsilon_{\text{mach}}$and we can write

$$\begin{aligned}
&\textit{float}\ (\quad 1 \quad + \quad e \quad) = 1 \\
&\textit{float}\ (\quad \textstyle\sum_{k=1}^{n-1} \frac{1}{k} \quad + \quad \frac{1}{n} \quad) = \textstyle\sum_{k=1}^{n-1} \frac{1}{k} \\
&\textit{float}\ (\quad 1 \quad + \quad \frac{1/n}{\sum_{k=1}^{n-1} \frac{1}{k}} \quad) = 1
\end{aligned}$$

We can easily experiment that $\sum_{k=1}^{n-1} \frac{1}{k} < 100$ for values of n which can be considered in practice



and given that $\epsilon_{\text{mach}} \approx 2 \times 10^{-16}$ we establish

$$\frac{1/n}{100} = 2 \times 10^{-16}$$

from which we conclude that n is of order 2×10^{14}. For a computer with a performance of 100 Mflops, the sum will stop to increase after $(2 \times 2 \times 10^{14})/10^{8}) = 4 \times 10^{6}$ seconds, i.e. 46 days of computing without exceeding the value of 100.

Approximation of derivatives

We want to replace derivatives (partial) by numerical approximations.
Given $f : \mathbb{R} \to \mathbb{R}$ a continuous function with f' continue. For $x \in \mathbb{R}$ we can write:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1}$$

If, instead of letting h go to zero, we consider "small" values for h, we get an approximation for $f'(x)$

Question: How to choose h to get a good approximation if we work with floating point arithmetic?

Approximation of derivatives (cont.d)

We consider f with derivatives of order $n+1$
For h finite, Taylor expansion writes

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + R_n(x+h)$$

with the remainder

$$R_n(x+h) = \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(\xi) \quad \text{avec} \quad \xi \in [x,\, x+h]$$

ξ not known !!

Thus Taylor series allow to approximate a function with a degree of precision which equals the remainder.

Forward-difference

Consider Taylors expansion up to order two

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(\xi) \quad \text{avec} \quad \xi \in [x,\, x+h]$$

from which we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(\xi) \tag{2}$$

Expression (2) decomposes $f'(x)$ in two parts,
the *approximation of the derivative* and the *truncation error*

This approximation is called *forward-difference*.

Backward-difference

Choosing h negativewe get

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{h}{2} f''(\xi) \tag{3}$$

This defines the *backward-difference* approximation.

Central-difference

Consider the difference $f(x+h) - f(x-h)$:

$$f(x+h) = f(x) + h\, f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(\xi_+) \quad \text{avec} \quad \xi_+ \in [x,\, x+h]$$
$$f(x-h) = f(x) - h\, f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(\xi_-) \quad \text{avec} \quad \xi_- \in [x-h,\, x]$$

We have

$$f(x+h) - f(x-h) = 2h\, f'(x) + \frac{h^3}{6}\Big(f'''(\xi_+) + f'''(\xi_-)\Big)$$

If f''' is continuous we can consider the mean $f'''(\xi)$, $\xi \in [x-h,\, x+h]$

$$\frac{h^3}{3}\Big(f'''(\xi_+) + f'''(\xi_-)\Big) = \frac{2\,h^3}{3} \underbrace{\frac{f'''(\xi_+) + f'''(\xi_-)}{2}}_{f'''(\xi)}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3} f'''(\xi)$$

Defines the approximation by *central-difference*

More precise as truncation error is of order $O(h^2)$

Approximating second derivatives

Developing the sum $f(x+h) + f(x-h)$

$$f(x+h) = f(x) + h\, f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(\xi_+) \quad \text{avec} \quad \xi_+ \in [x,\, x+h]$$
$$f(x-h) = f(x) - h\, f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(\xi_-) \quad \text{avec} \quad \xi_- \in [x-h,\, x]$$

we get

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{3} f''''(\xi) \quad \text{avec} \quad \xi \in [x-h,\, x+h]$$

There exist several different approximation schemes, also of higher order (see (Dennis and Schnabel, 1983))

Partial derivatives

Given a function of two variables $f(x, y)$ then the approximation, using central-difference, of the partial derivative with respect to x is

$$f_x = \frac{f(x + h_x, y) - f(x - h_x, y)}{2h_x}$$

Approximating with a central-difference the partial derivative of f_x with respect to y gives

$$\begin{aligned} f_{xy} &= \frac{\frac{f(x+h_x,y+h_y)-f(x-h_x,y+h_y)}{2h_x} - \frac{f(x+h_x,y-h_y)-f(x-h_x,y-h_y)}{2h_x}}{2h_y} \\ &= \frac{1}{4h_x h_y}\left(f(x + h_x, y + h_y) - f(x - h_x, y + h_y)\right. \\ &\qquad \left. - f(x + h_x, y - h_y) + f(x - h_x, y - h_y)\right) \end{aligned}$$

How to choose h

How to choose h if we use floating point arithmetic ?

To reduce the truncation error we would be tempted to choose h as small as possible. However we also must consider the rounding error !!

If h is too small, we have $\mathit{float}\,(x + h) = \mathit{float}\,(x)$
and even if $\mathit{float}\,(x + h) \neq \mathit{float}\,(x)$,
we can have $\mathit{float}\left(f(x + h)\right) = \mathit{float}\left(f(x)\right)$ if the function varies very slowly

Truncation error for forward-difference $-\frac{h}{2}f''(\xi)$

Denote

$$f'_h(x) = \frac{f(x + h) - f(x)}{h}$$

the approximation of the derivative corresponding to a given value of h

If we accept that in the truncation error $-\frac{h}{2}f''(\xi)$ we have

$|f''(t)| \leq M$ for all t in the domain of our application

we get

$$|f'_h(x) - f'(x)| \leq \frac{h}{2}M$$

bound for the truncation error, indicates that we can reach any precision given we choose h sufficiently small. However, in floating point arithmetic it is not possible to evaluate $f'_h(x)$ exactly (rounding errors !!)

How to choose h (cont'd)

Consider that the rounding error for evaluating $f(x)$ has the following bound

$$\left| \textit{float}\left(f(x) \right) - f(x) \right| \leq \epsilon$$

then the rounding error (for finite precision computations) for an approximation of type *forward-difference* is bounded by

$$\left| \textit{float}\left(f'_h(x) \right) - f'_h(x) \right| \leq \frac{2\epsilon}{h}$$

$$\begin{aligned} \textit{float}\left(f'_h(x) \right) &= \textit{float}\left(\frac{f(x+h) - f(x)}{h} \right) \\ &= \frac{\textit{float}\left(f(x+h) \right) - \textit{float}\left(f(x) \right)}{h} \\ &\quad \frac{\epsilon + \epsilon}{h} \end{aligned}$$

Finally, the upper bound for the sum of both sources of errors is

$$\left| \textit{float}\left(f'_h(x) \right) - f'(x) \right| \leq \frac{2\epsilon}{h} + \frac{M}{2} h$$

Necessitates compromise between reduction of the truncation error and reduction of rounding error
Consequence: The precision we can achieve is limited

h is chosen as power of 2 and therefore can be represented without rounding error

How to choose h (cont'd)

Minimization problem of $g(h) = \frac{2\epsilon}{h} + \frac{h}{2}M$

$$\min_h g(h) \Rightarrow g'(h) = 0$$

$$\begin{aligned} -\frac{2\epsilon}{h^2} + \frac{M}{2} &= 0 \\ h^2 &= \frac{4\epsilon}{M} \\ h &= 2\sqrt{\frac{\epsilon}{M}} \end{aligned}$$

Bound reaches its minimum for

$$h = 2\sqrt{\frac{\epsilon}{M}} \tag{4}$$

In practice ϵ corresponds to the machine precision ϵ_{mach} and if we admit that M is of order one, thus we set $M = 1$ and

$$h = 2\sqrt{\epsilon_{\text{mach}}}$$

With Matlab we have $\epsilon_{\text{mach}} \approx 2 \times 10^{-16}$ and $h \approx 10^{-8}$

How to choose h (cont'd)

Considering central-differences we have $g(h) = \frac{2\epsilon}{h} + \frac{h^2}{3}M$

$$\min_h g(h) \Rightarrow g'(h) = 0$$

$$\begin{aligned} -\frac{2\epsilon}{h^2} + \frac{2Mh}{3} &= 0 \\ h^3 &= \frac{6\epsilon}{2M} \\ h &= \left(\frac{3\epsilon}{M}\right)^{1/3} \end{aligned}$$

$h \approx 10^{-5}$

An example

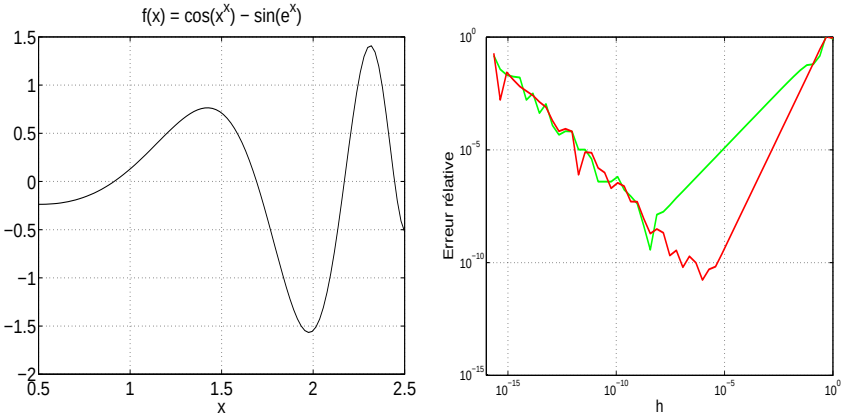
To illustrate the influence of the choice of h on the precision of the approximation of the derivative, consider the function

$$f(x) = \cos(x^x) - \sin(e^x)$$

and its derivative

$$f'(x) = -\sin(x^x)\, x^x \left(\log(x) + 1\right) - \cos(e^x)\, e^x.$$

How to choose h (cont'd)



Relative error for the approximation of the derivative for $x = 1.77$, as a function of h in the range of 2^0 et 2^{-52}, corresponding to ϵ_{mach} for Matlab.

Absolute and relative error

Measuring the error e between an approximated value $\hat{x}$ and an exact value x.

- Absolute error

$$|\hat{x} - x|$$

For $\hat{x} = 3$ and $x = 2$ absolute error is 1, cannot be considered as "small"
For $\hat{x} = 10^9 + 1$ and $x = 10^9$ can be considered "small" with respect to x

- Relative error

$$\frac{|\hat{x} - x|}{|x|}$$ Defined if $x \neq 0$ not defined if $x = 0$

If $x = 0$ the absolute error would do the job !!
The relative error for the previous example is 0.5 and 10^{-9} respectively ("small" for the second case)

Combining absolute and relative error

Combination between absolute and relative error

$$\frac{|\hat{x} - x|}{|x| + 1}$$

- Avoids problems if x is near zero
- Has properties of the relative error if $|x| \gg 1$
- Has properties of absolute error if $|x| \ll 1$

Numerical instability

If the "quality" of a solution is not acceptable it is important to distinguish between the following two situations:

- Errors are amplified by the algorithm → *numerical instability*
- Small perturbations of data generate large changes in the solution → *ill conditioned problem*

Numerically unstable algorithm

Example of a numerically unstable algorithm: solution of $ax^2 + bx + c = 0$
Analytical solutions are:

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}\,.$$

Algorithm: Transcription of analytical solution

1: $\Delta = \sqrt{b^2 - 4ac}$
2: $x_1 = (-b - \Delta)/(2a)$
3: $x_2 = (-b + \Delta)/(2a)$

For $a = 1$, $c = 2$ and a floating point arithmetic with 5 digits of precision the algorithm produces the following solutions for different values of b:

b	Δ	float(Δ)	float(x_2)	x_2	$\frac{\lvert \text{float}(x_2) - x_2 \rvert}{\lvert x_2 \rvert + 1}$
5.2123	4.378135	4.3781	−0.41708	−0.4170822	1.55×10^{-6}
121.23	121.197	121.20	−0.01500	−0.0164998	1.47×10^{-3}
1212.3	1212.2967	1212.3	0	−0.001649757	Catastrophic cancelation

Attention: *float* $(x_2) = (-b +$ *float* $(\Delta))/(2a)$

Numerically stable algorithm

Alternative algorithm, exploiting $x_1 x_2 = c/a$ (Viète theorem):

1: $\Delta = \sqrt{b^2 - 4ac}$
2: **if** $b < 0$ **then**
3: $\quad x_1 = (-b + \Delta)/(2a)$
4: **else**
5: $\quad x_1 = (-b - \Delta)/(2a)$
6: **end if**
7: $x_2 = c/(a\, x_1)$

b	float(Δ)	float(x_1)	float(x_2)	x_2
1212.3	1212.3	−1212.3	−0.0016498	−0.001649757

Avoids catastrophic cancelation
(loss of significant digits when subtracting two large numbers).

Ill conditioned problem

Measures the sensitivity of the solution of a linear system $Ax = b$ with respect to a perturbation of the elements of A

Example:

$$A = \begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \qquad b = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

The exact solution is $x = [1 \ -1]'$ (generally not known !!).
Matlab computes

$$x = A \backslash b = \begin{bmatrix} .99999999991008 \\ -.99999999987542 \end{bmatrix}$$

This is an ill conditioned problem !!!

Ill conditioned problem

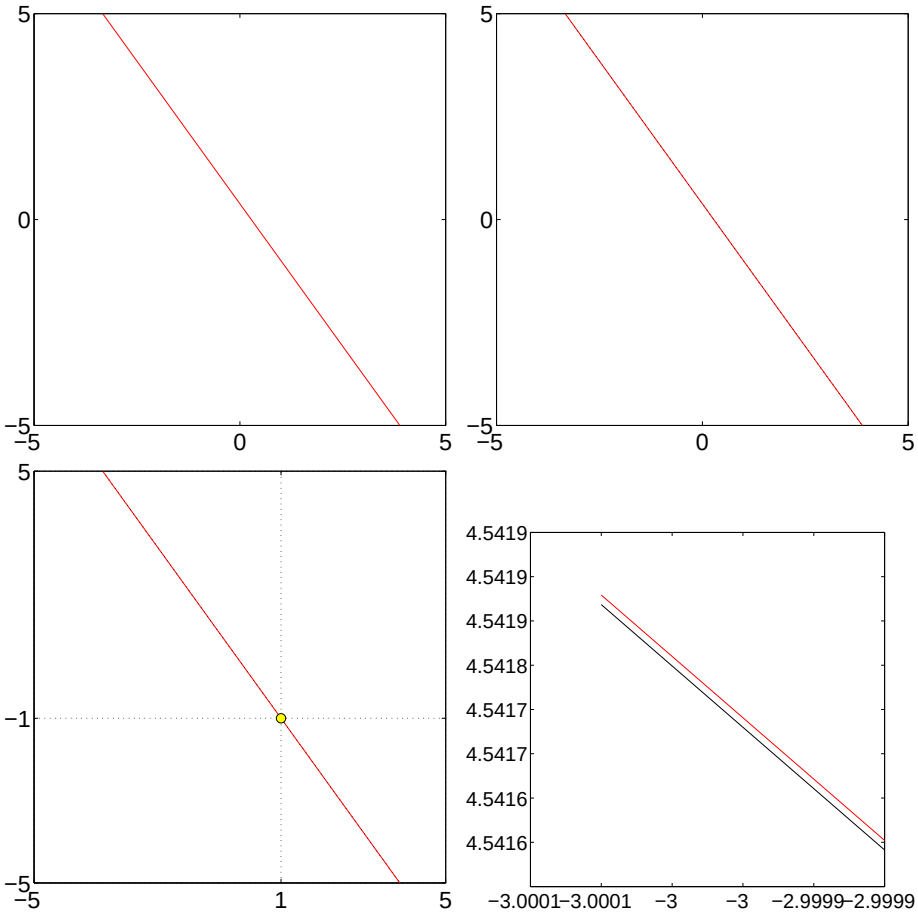
Consider the matrix

$$E = \begin{bmatrix} .001 & .001 \\ -.002 & -.001 \end{bmatrix}$$

and the perturbed system $(A + E)x_E = b$

The new solution is

$$x_E = \begin{bmatrix} -5.0000 \\ 7.3085 \end{bmatrix}$$



Condition of a matrix

The condition of a matrix $\kappa(A)$ is defined as

$$\kappa(A) = \|A^{-1}\| \, \|A\|$$

Matlab function `cond` computes the condition of a matrix.

For our example we have cond$(A) = 2.2 \times 10^6$

If cond$(A) > 1/$sqrt(eps), we should worry about our numerical results !!

For $eps = 2.2 \times 10^{-16}$, we have $1/$sqrt$(eps) = 6.7 \times 10^8$
(We loose half of the significant digits !!!)

Complexity of algorithms 66

- Many different algorithms to solve same problem
- How to compare them in order to choose the best
  Precise comparison generally very difficult → use notion of complexity
- Problem size → number of elements to be processed
  e.g. linear algebra algorithms: $Ax = b$ order n of A
  if $A \in \mathbb{R}^{n \times m}$ size given by n and m
  graph theory: $G = (V, E)$, $n = |V|$ and $m = |E|$

  Time necessary to execute the algorithm expressed as a function of problem size →
  independent from executing platform
- Operation count (flop): only 4 elementary operations $+ \; - \; \times \; \backslash$ considered
- Only order of function counting operations considered

  e.g. Complexity of algorithm with $\frac{n^3}{3} + n^2 + \frac{2}{3}n$ elementary operations is of order $O(n^3)$

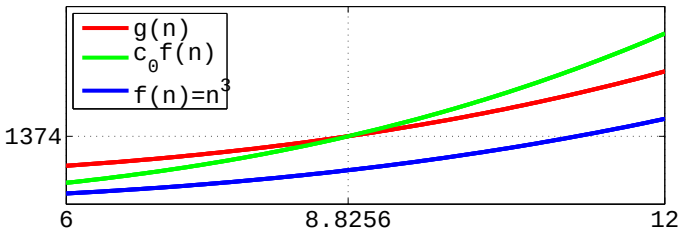
Order of complexity $\mathcal{O}(\cdot)$ and classification

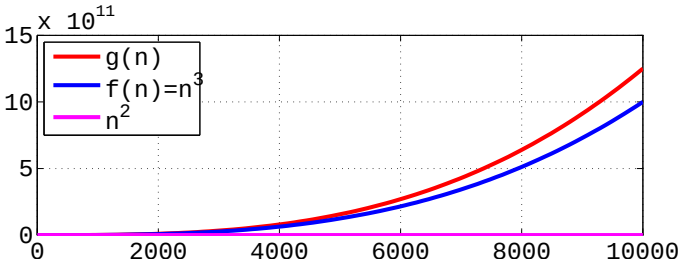
Definition: Function $g(n)$ is $O(f(n))$ if there exist constants c_o and n_o such that $g(n)$ is inferior to $c_o f(n)$ for all $n > n_0$

$g(n) = (5/4)\, n^3 + (1/5)\, n^2 + 500$
$g(n)$ is $\mathcal{O}(n^3)$ as

$$c_0\, f(n) > g(n) \quad \text{for} \quad n > n_0$$

$c_0 = 2$ and $n_0 = 9$





Algorithms are classified into 2 groups (according to the order of the function counting elementary operations):

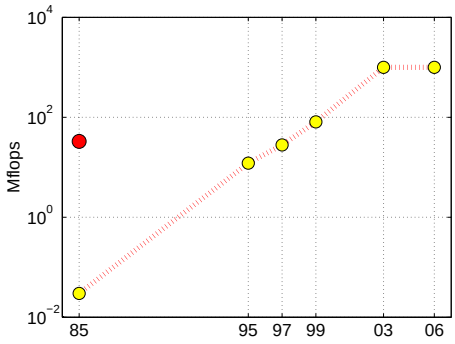
- *polynomial*
- *non-polynomial*

Only algorithms from the polynomial class can be considered as efficient !!

Performance of computer measured in `Mflops` (10^6 flops per second)

Evolution of performance

Machine	Mflops	Year
8086/87 à 8 MHz	0.03	1985
Cray X–MP	33	1985
Pentium 133 MHz	12	1995
Pentium II 233 MHz	28	1997
Pentium III 500 MHz	81	1999
Pentium IV 2.8 GHz	1000	2003
Intel U2500 1.2 GHz	1000	2006
Earth Simulator	30 TFlops	2003



`www.top500.org`

Fantastic increase of cheap computing power !!! We will make use of it !!!

Exemple de solution numérique

L'équation différentielle suivante

$$\frac{\partial f}{\partial x} = -\frac{f(x)}{x} \tag{5}$$

est une une équation différentielle ordinaire car f ne fait intervenir qu'une seule variable. Lorsqu'on résoud une équation différentielle, on cherche la fonction $f(x)$ qui satisfait (5). On vérifie que la solution générale est

$$f(x) = \frac{C}{x} \tag{6}$$

Si l'on dérive (6) on obtient $\frac{\partial f}{\partial x} = -\frac{C}{x^2}$ et

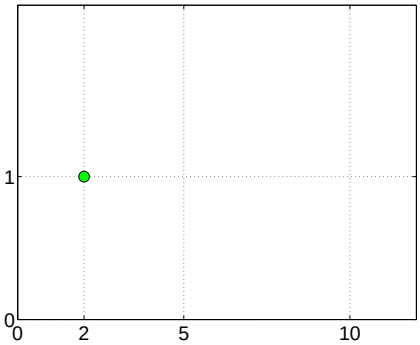
$$\frac{\partial f}{\partial x} = -\frac{f(x)}{x} = -\frac{\frac{C}{x}}{x} = -\frac{C}{x^2}$$

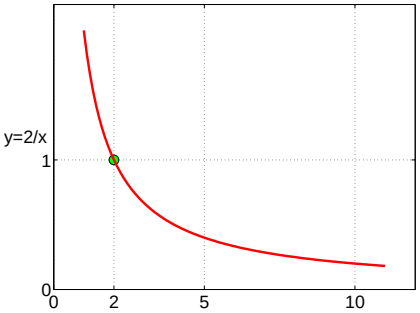
ce qui vérifie (5).

Exemple de solution numérique

Dans la solution (6) intervient une constante arbitraire C. Il y a un choix infini pour C dans (6) qui satisfait (5), étant donné que la constante C disparaît en dérivant. Pour trouver une solution particulière, il faut imposer quelques contraintes à (6), comme par exemple de passer par un point précis donné. On appelle ceci des conditions initiales.

Conditions initiales $x_0 = 2$ et $y_0 = 1$, d'où $1 = C/2$ et $C = 2$





Solution particulière: $y = C/x = 2/x$

Exemple de solution numérique

Trois approches peuvent être envisagées pour obtenir une solution d'une équation différentielle:

- Recherche d'une solution analytique
  (difficile, souvent elle n'existe pas)
- Recherche d'une solution analytique approchée
  (beaucoup d'efforts pour une approximation dont on mesure mal la précision)
- Approximation numérique
  (permet de s'attaquer à n'importe quelle équation différentielle; relativement facile à réaliser)

Une approximation numérique

En remplaçant la dérivée par une approximation, on peut écrire l'équation différentielle $\frac{\partial f}{\partial x} = -\frac{f(x)}{x}$ comme

$$\frac{f(x+\triangle_x) - f(x)}{\triangle_x} = -\frac{f(x)}{x}$$

d'où l'on tire

$$f(x+\triangle_x) = f(x)\left(1 - \frac{\triangle_x}{x}\right)$$

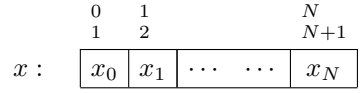
Ainsi, partant du point $x_0 = 2$, $y_0 = 1$, on peut déterminer numériquement la valeur de la fonction en $x_0 + \triangle_x$

Écrivons le programme qui calcule N approximations de la solution entre x_0 et x_N.

Une approximation numérique

Dans Matlab la première adresse d'un élément d'un vecteur est 1 (d'autres langages commencent avec 0). Ainsi `x(1)` adresse la variable x_0. Afin de rendre un programme plus lisible il serait souhaitable de pouvoir faire correspondre x_0 à l'adresse 0. On peut le réaliser avec l'artifice suivant:

- définir une variable `f7=1` (*offset*)
- décaler l'indice d'adressage avec `f7`
- `x(f7+i)` $i = 0, 1, \ldots, N$



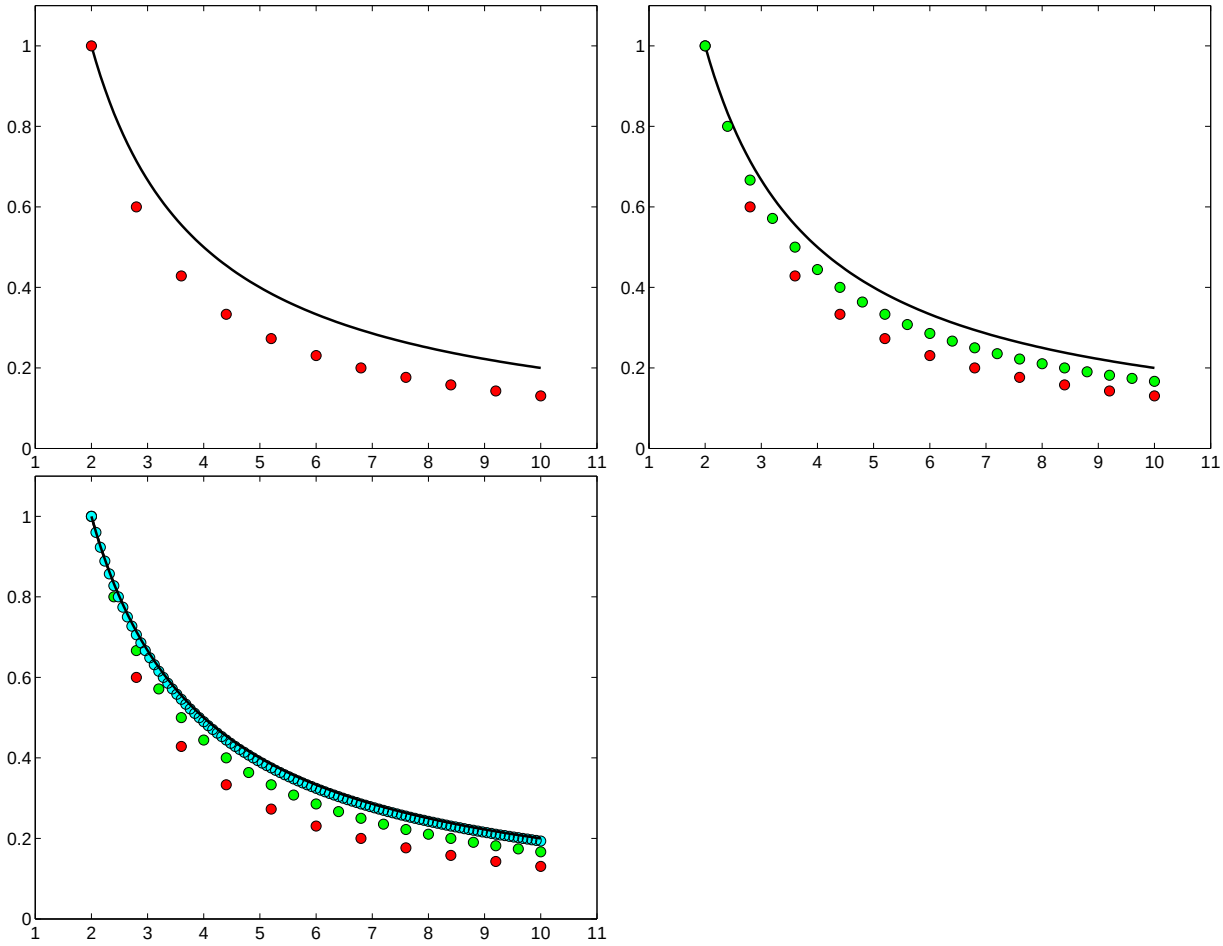

```
f7 =  1;          % Offset pour l'adressage
x0 =  2; y0 = 1; % Initial conditions
xN = 10; N  = 199;
dx = (xN - x0)/N;
x  = linspace(x0,xN,N+1);
y(f7+0) = y0;
for i = 1:N
    y(f7+i) = y(f7+i-1) * ( 1- dx/x(f7+i-1) );
end

plot(x,y,'ro'), hold on
z = linspace(x0,xN); f = 2./z;  plot(z,f,'b')
```


Note

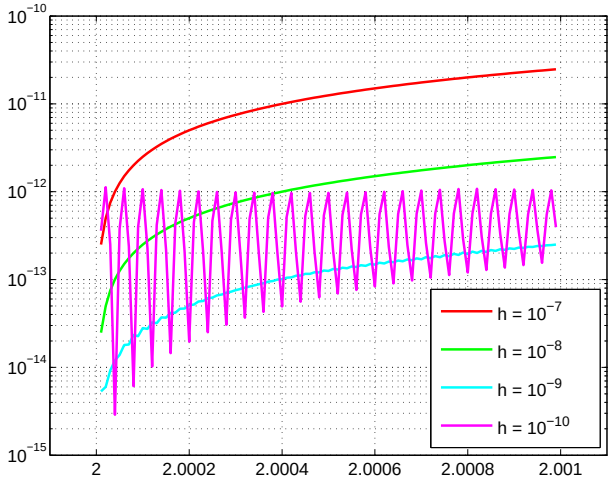
En exécutant ce code, on peut observer comment la précision des approximations dépend de $\triangle_x$.

Une approximation numérique

En exécutant ce code, on peut observer comment la précision des approximations dépend de $\triangle_x$.



Let's choose the optimal value for $\triangle_x$. (We have to reduce the interval !!!)



Note

Demonstration avec `ODEex1.m` comment la précision augmente lorsqu'on diminue $\triangle_x$ MAIS le nombre de steps devient prohibitif !!!

Une autre approximation

On peut aussi remplacer la dérivée de l'équation différentielle par l'approximation suivante

$$\frac{f(x+\triangle_x) - f(x-\triangle_x)}{2\,\triangle_x} = -\frac{f(x)}{x}$$

et on peut écrire

$$f(x) = \frac{x}{2\,\triangle_x}\Big(f(x-\triangle_x) - f(x+\triangle_x)\Big)$$

Pour pouvoir résoudre cette expression, il faut connaître deux points de la solution $f(x-\triangle_x)$ et $f(x+\triangle_x)$.
On a donc besoin davantage de conditions initiales.

Une autre approximation

Ecrivons l'approximation pour quatre points successifs:

$$f(x) = \frac{x}{2\,\triangle_x}\Big(f(x-\triangle_x) - f(x+\triangle_x)\Big)$$

$$\begin{aligned}
y_1 &= x_1(y_0 - y_2)/(2\,\triangle_x)\\
y_2 &= x_2(y_1 - y_3)/(2\,\triangle_x)\\
y_3 &= x_3(y_2 - y_4)/(2\,\triangle_x)\\
y_4 &= x_4(y_3 - y_5)/(2\,\triangle_x)
\end{aligned}$$

et l'on remarque qu'il s'agit d'un système linéaire

$$\begin{bmatrix} 1 & c_1 & & \\ -c_2 & 1 & c_2 & \\ & -c_3 & 1 & c_3 \\ & & -c_4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} c_1 y_0 \\ 0 \\ 0 \\ -c_4 y_5 \end{bmatrix}$$

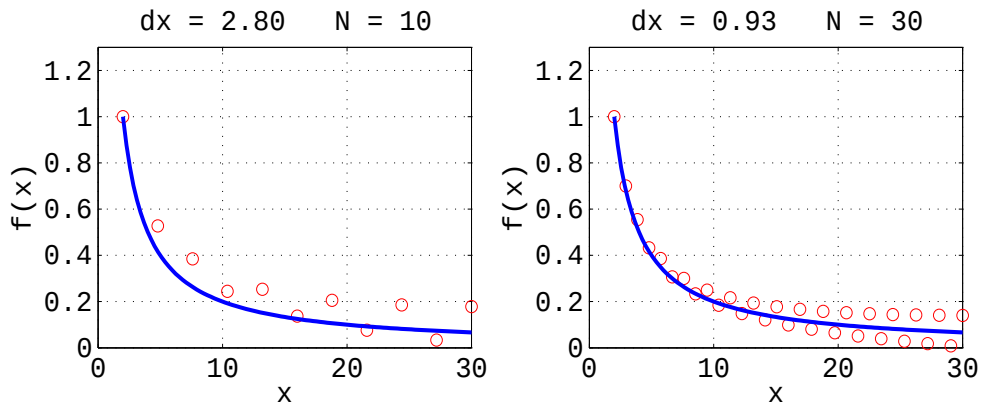
où $c_i = x_i/(2\,\triangle_x)$, et que l'on peut résoudre si l'on connaît y_0 et y_5.

Une autre approximation

Code pour construire et résoudre le système pour x_0, y_0, x_N, y_N et N données

Code /Teaching/MNE/Ex/ODEex2.m:

```
clear, close all
f7 = 1;
x0 = 2;  y0 = 1;
xN = 30; yN = 0; N  = 30;
dx = (xN - x0)/N;
x  = linspace(x0,xN,N+1);
y(f7+0) = y0;
x(f7+0) = x0;
for i = 1:N
    c(i) = x(f7+i)/(2*dx);
end
A = spdiags([[-c(2:N) 0]' ones(N,1) [0 c(1:N-1)]'],-1:1,N,N);
b = [c(1)*y0  zeros(1,N-2) -c(N)*yN]';
y(f7+(1:N)) = A\b;
figure
plot(x,y,'ro'), hold on
z = linspace(x0,xN); f = 2./z;  plot(z,f,'b')
%
%% -- Question 2
clear
f7 = 1;
x0 = 2;  y0 = 1;
xp = 20;
yN = 0; N  = 1e4;
for xN = [30 100 400 500 600 800 1000]
    dx = (xN - x0)/N;
    x  = linspace(x0,xN,N+1);
    y(f7+0) = y0;
    x(f7+0) = x0;
    for i = 1:N
        c(i) = x(f7+i)/(2*dx);
    end
    A = spdiags([[-c(2:N) 0]' ones(N,1) [0 c(1:N-1)]'],-1:1,N,N);
    b = [c(1)*y0  zeros(1,N-2) -c(N)*yN]';
    y(f7+(1:N)) = A\b;
    f = 2./xp;
    p = find(x < xp);
    p = p(end);
    e = abs(y(p) - f) ./ f;
    fprintf('\n Erreur relative maximale est %6.4f (xN = %i)\n',e,xN);
end
```



Précision de la méthode implicite en fonction du choix de $\triangle_x$.

Fonction `spdiags`

`spdiags(B,d,m,n)`

Dans la suite du cours on présentera en détail ces deux méthodes numériques dans le cadre des équations différentielles utilisées pour l'évaluation des options.

Classification des equations différentielles

- EDO Equations Différentielles Ordinaires (une seule variable indépendante $y = f(x)$)
- EDP Equations Différentielles Partielles (PDE) (plusieurs variables)

  Critères de classification:

  - degré le plus élevé de la dérivée
  - linéaire avec coefficients constants:
    $$u_{xx} + 3u_{xy} + u_{yy} + u_x - u = e^{x-y}$$
  - linéaire avec coefficients variables:
    $$\sin(xy)u_{xx} + 3x^2u_{xy} + u_{yy} + u_x - u = 0$$
  - non linéaire:
    $$u_{xx} + 3u_{xy} + u_{yy} + u_x^2 - u = e^{x-y}$$

Equations différentielles (2)

EDP linéaires à 2 dimensions et de degré 2:

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$$

Classées selon la valeur du discriminant:

$$b^2 - 4ac \begin{cases} > 0 & \text{équation hyperbolique} \\ = 0 & \text{équation parabolique} \\ < 0 & \text{équation elliptique.} \end{cases}$$

Avec coefficients variables on peut passer d'une situation à une autre !!

Equations différentielles (3)

- EDP hyperboliques (processus temporels), ex. mouvement d'une onde

$$u_{xx} - u_{tt} = 0$$

- EDP paraboliques (processus temporels de propagation)
  ex. propagation de la chaleur

$$u_t - u_{xx} = 0$$

- EDP elliptiques (systèmes en équilibre)
  ex. équation de Laplace

$$u_{xx} + u_{yy} = 0$$

Equations différentielles (4)

Trois composantes dans un problème d'EDP:

- l'EDP
- domaine spacial et temporel dans lequel l'EDP doit être satisfaite
- conditions initiales (au temps 0) conditions au bord du domaine.

Equation de Black-Scholes

$$V_{\mathrm{t}} + \frac{1}{2}\sigma^2 S^2 V_{\mathrm{SS}} + (r-q)S V_{\mathrm{S}} - r V = 0$$

V	valeur de l'option
S	prix du sous-jacent
σ	volatilité du sous-jacent
r	taux du marché
q	dividende

L'équation de Black-Scholes est:

- linéaire avec coefficients variables
- du premier ordre dans le temps
- du deuxième ordre dans l'espace
- EDP parabolique (rétrograde) ($b = 0$, $c = 0$)

Cond. terminales et cond. aux bords (*call*)

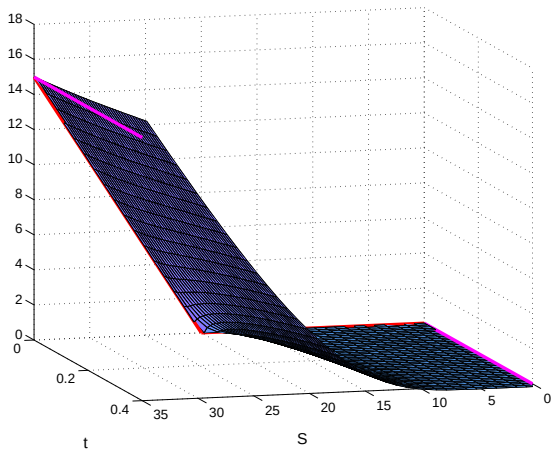
$$V(S,T) = (S-E)_+$$
$$V(0,t) = 0 \qquad \lim_{S\to\infty} V(S,t) = S$$

Cond. terminales et cond. aux bords (*put*)

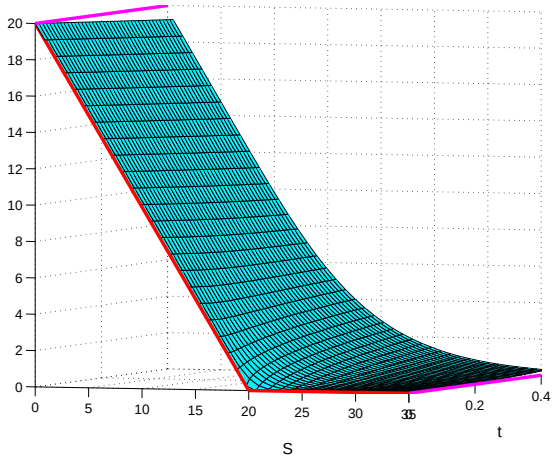
$$V(S,T) = (E-S)_+$$
$$V(0,t) = E\,e^{-r(T-t)} \qquad \lim_{S\to\infty} V(S,t) = 0$$

Equation de Black-Scholes (2)

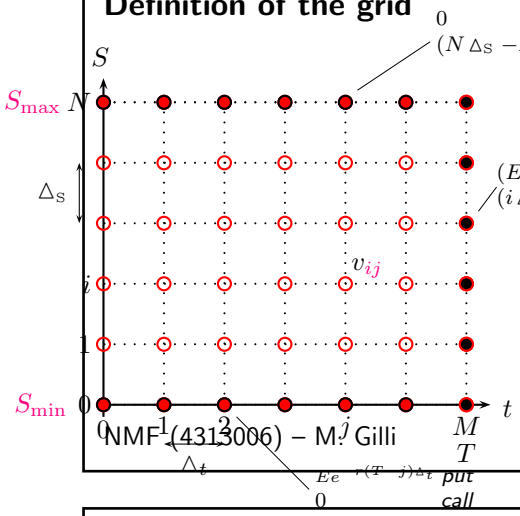
Call



Put



Definition of the grid



$\Delta_S = (S_{\max} - S_{\min})/N$

$\Delta_t = T/M$

Numerical approximations of $V(S,t)$ only computed at grid points

$S_i = S_{\min} + i\,\Delta_S \qquad i=0,\ldots,N$

$t_j = j\,\Delta_t \qquad i=0,\ldots,M$

$v_{ij} = V(S_{\min} + i\,\Delta_S, j\,\Delta_t) \qquad i=0,\ldots,N,\ j=0,\ldots,M$

denotes values of $V(S,t)$ at the $(N+1)\times(M+1)$ grid points

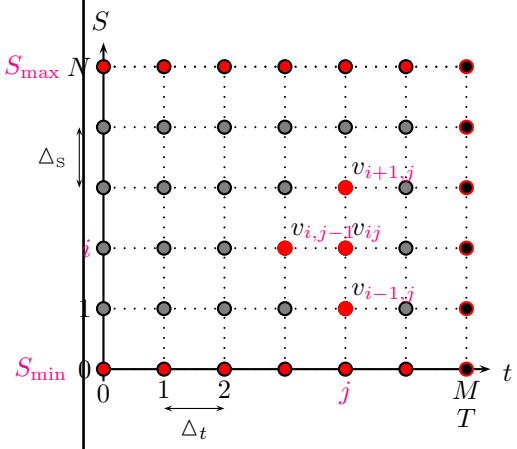
Finite difference solution for BS

Approximation of derivatives of $v_{ij} = V(S_{\min} + i\,\Delta_S, j\,\Delta_t)$:

$$V_t\,|_{t=j\Delta_t} \approx \frac{v_{ij} - v_{i,j-1}}{\Delta_t} \quad \textit{(backward difference)}$$

$$V_S\,|_{S=S_{\min}+i\Delta_S} \approx \frac{v_{i+1,j} - v_{i-1,j}}{2\,\Delta_S} \quad \textit{(central difference)}$$

$$V_{SS}\,|_{S=S_{\min}+i\Delta_S} \approx \frac{v_{i+1,j} - 2v_{ij} + v_{i-1,j}}{\Delta_S^2}$$



Four grid points are involved in the approximation of the derivative of v_{ij}

Finite difference solution for BS

We replace the derivatives by the approximations in the BS formula:

$$V_{\mathrm{t}} + \frac{1}{2}\sigma^2 S^2 V_{\mathrm{SS}} + (r-q)S V_{\mathrm{S}} - r V = 0$$

$$\frac{v_{ij} - v_{i,j-1}}{\triangle_t} + \frac{1}{2}\sigma^2 S_i^2 \frac{v_{i+1,j} - 2v_{ij} + v_{i-1,j}}{\triangle_S^2} + (r-q) S_i \frac{v_{i+1,j} - v_{i-1,j}}{2\triangle_S} - r\, v_{ij} = 0$$

Denote $a_i = \frac{\triangle_t \sigma^2 S_i^2}{2\triangle_S^2}$ and $b_i = \frac{\triangle_t (r-q) S_i}{2\triangle_S}$ and the $i = 1, \ldots, N-1$ equations of column j are:

$$v_{ij} - v_{i,j-1} + a_i\,(v_{i+1,j} - 2v_{ij} + v_{i-1,j}) + b_i\,(v_{i+1,j} - v_{i-1,j}) - \triangle_t\, r\, v_{ij} = 0$$

Factorising identical nodes:

$$v_{ij} - v_{i,j-1} + \underbrace{(a_i - b_i)}_{d_i} v_{i-1,j} + \underbrace{(-2a_i - \triangle_t\, r)}_{m_i} v_{ij} + \underbrace{(a_i + b_i)}_{u_i} v_{i+1,j} = 0$$

Formalization of the system of finite difference equations

Let us write the $i = 1, \ldots, N-1$ equations:

$$\begin{array}{rcrcrcrcrcl}
v_{1,j} & - & v_{1,j-1} & + & d_1\, v_{0,j} & + & m_1\, v_{1,j} & + & u_1\, v_{2,j} & = & 0 \\
v_{2,j} & - & v_{2,j-1} & + & d_2\, v_{1,j} & + & m_2\, v_{2,j} & + & u_2\, v_{3,j} & = & 0 \\
 & \vdots & & & \vdots & & \vdots & & \vdots & & \vdots \\
v_{i,j} & - & v_{i,j-1} & + & d_i\, v_{i-1,j} & + & m_i\, v_{i,j} & + & u_i\, v_{i+1,j} & = & 0 \\
 & \vdots & & & \vdots & & \vdots & & \vdots & & \vdots \\
v_{N-1,j} & - & v_{N-1,j-1} & + & d_{N-1}\, v_{N-2,j} & + & m_{N-1}\, v_{N-1,j} & + & u_{N-1}\, v_{N,j} & = & 0
\end{array}$$

in matrix notation: $$v_{1:N-1,j} - v_{1:N-1,j-1} + P\, v_{0:N,j} = 0 \qquad (7)$$

$$P = \begin{bmatrix}
d_1 & m_1 & u_1 & 0 & \cdots & \cdots & 0 \\
0 & d_2 & m_2 & u_2 & & & \vdots \\
\vdots & & d_3 & \ddots & \ddots & & \vdots \\
\vdots & & & \ddots & \ddots & u_{N-2} & 0 \\
0 & \cdots & \cdots & 0 & d_{N-1} & m_{N-1} & u_{N-1}
\end{bmatrix}$$

!! We change the notation for the indices of the vectors →

Finite difference solution for BS

We consider the following partition of the set $\overline{\Omega}$ of rows of the grid:

$$\overline{\Omega} = \Omega \cup \partial\Omega \quad \text{et} \quad \Omega \cap \partial\Omega = \emptyset$$

- Ω set of interior points $(i = 1, \ldots, N-1)$
- $\partial\Omega$ set of points lying on the border $(i = 0, N)$

We apply this partition to the columns of $P \rightarrow$ defines matrices A and B

$$P = \begin{bmatrix} d_1 & m_1 & u_1 & 0 & \cdots & \cdots & 0 \\ 0 & d_2 & m_2 & u_2 & & & \vdots \\ \vdots & & d_3 & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & u_{N-2} & 0 \\ 0 & \cdots & \cdots & 0 & d_{N-1} & m_{N-1} & u_{N-1} \end{bmatrix}$$

$$v_{\overline{\Omega}}^j \equiv v_{ij},\ i = 0, 1, \ldots, N$$
$$v_{\Omega}^j \equiv v_{ij},\ i = 1, \ldots, N-1$$
$$v_{\partial\Omega}^j \equiv v_{ij},\ i = 0, N$$

$$P\, v_{\overline{\Omega}}^j = A\, v_{\Omega}^j + B\, v_{\partial\Omega}^j$$

Columns

Columns of P correspond to the rows of the grid !!

Finite difference solution

Introducing the previous notation, equation

$$v_{1:N-1,j} - v_{1:N-1,j-1} + P\, v_{0:N,j} = 0$$

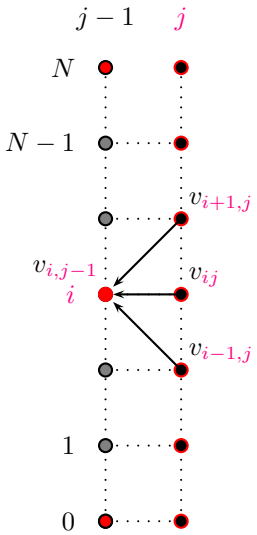
becomes

$$v_{\Omega}^j - v_{\Omega}^{j-1} + A\, v_{\Omega}^j + B\, v_{\partial\Omega}^j = 0 \qquad (8)$$

The solution v_{Ω}^0 is obtained by computing

$$v_{\Omega}^{j-1} = (I + A)\, v_{\Omega}^j + B\, v_{\partial\Omega}^j$$

for $j = M, M-1, \ldots, 1 \quad \rightarrow$ Explicit method



Finite difference solution for BS

If we approximate V_t with a forward difference:

$$V_t\,|_{t=j\triangle_t} \approx \frac{v_{i,j+1} - v_{ij}}{\triangle_t}$$

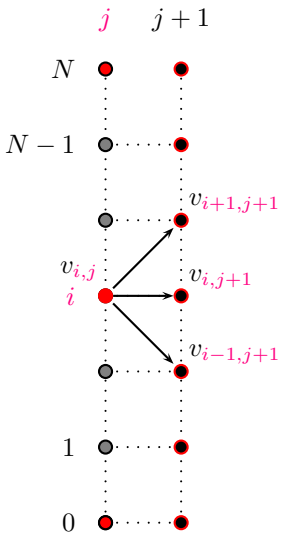
equation (8) becomes

$$v_\Omega^{j+1} - v_\Omega^j + A\,v_\Omega^j + B\,v_{\partial\Omega}^j = 0 \qquad (9)$$

and the solution v_Ω^0 is obtained by solving a linear system

$$(I - A)\,v_\Omega^j = v_\Omega^{j+1} + B\,v_{\partial\Omega}^j$$

for $j = M-1, \ldots, 1, 0$ $\rightarrow$ Implicit method



Finite difference solution for BS

Let us consider equation (9) at time step $j\ \triangle_t$

$$v_\Omega^{j+1} - v_\Omega^j + A\,v_\Omega^j + B\,v_{\partial\Omega}^j = 0$$

and equation (8) at time step $(j+1)\ \triangle_t$

$$v_\Omega^{j+1} - v_\Omega^j + A\,v_\Omega^{j+1} + B\,v_{\partial\Omega}^{j+1} = 0$$

Consider a linear combination of the two previous equations $\quad \theta \in [0,1]$

$$\theta\,(v_\Omega^{j+1} - v_\Omega^j + A\,v_\Omega^j + B\,v_{\partial\Omega}^j) + (1-\theta)\,(v_\Omega^{j+1} - v_\Omega^j + A\,v_\Omega^{j+1} + B\,v_{\partial\Omega}^{j+1}) = 0$$

and rearranging

$$A_m\,v_\Omega^j = A_p\,v_\Omega^{j+1} + \theta\,B\,v_{\partial\Omega}^j + (1-\theta)\,B\,v_{\partial\Omega}^{j+1} \qquad \theta\text{-method} \qquad (10)$$

$A_m = I - \theta A$ and $A_p = I + (1-\theta)A$

Solution v_Ω^0 obtained by solving linear system (10) for $j = M-1, \ldots, 1, 0$

Finite difference solution for BS

The θ-method generalizes all methods:

$$A_m\, v_\Omega^j = A_p\, v_\Omega^{j+1} + \theta\, B\, v_{\partial\Omega}^j + (1-\theta)\, B\, v_{\partial\Omega}^{j+1}$$

$A_m = I - \theta A \qquad A_p = I + (1-\theta)A$

θ	Method	A_m	A_p
0	Explicit	I	$I + A$
1	Implicit	$I - A$	I
$\frac{1}{2}$	Crank-Nicolson	$I - \frac{1}{2}A$	$I + \frac{1}{2}A$

Algorithm 1 Computes the solution v_Ω^0 with the θ-method

1: Define initial and boundary conditions v_Ω^M, $v_{\partial\Omega}^{0:M-1}$ and θ
2: Compute A_m, A_p and B
3: **for** $j = M-1 : -1 : 0$ **do**
4: Solve $A_m\, v_\Omega^j = A_p\, v_\Omega^{j+1} + \theta\, B\, v_{\partial\Omega}^j + (1-\theta)\, B\, v_{\partial\Omega}^{j+1}$
5: **end for**

Implementation of θ-method with Matlab

Statement 2 of algorithm 1:

1: Define initial and boundary conditions v_Ω^M, $v_{\partial\Omega}^{0:M-1}$ and θ
2: Compute A_m, A_p and B
3: **for** $j = M-1 : -1 : 0$ **do**
4: Solve $A_m\, v_\Omega^j = A_p\, v_\Omega^{j+1} + \theta\, B\, v_{\partial\Omega}^j + (1-\theta)\, B\, v_{\partial\Omega}^{j+1}$
5: **end for**

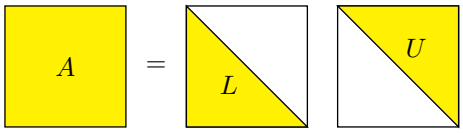
```
function [Am,Ap,B,S] = GridU1D(r,q,T,sigma, Smin,Smax,M,N,theta)
% Uniform grid in original variables
f7 = 1;
dt = T/M;
S  = linspace(Smin,Smax,N+1)';
dS = (Smax - Smin) / N;
a = (dt*sigma^2*S(f7+(1:N-1)).^2) ./ (2*dS^2);
b = (dt*(r-q)*S(f7+(1:N-1))) / (2*dS);
d = a - b;
m = - 2*a - dt*r;
u = a + b;
P  = spdiags([d m u], 0:2, N-1, N+1);
Am = speye(N-1) -    theta *P(:,f7+(1:N-1));
Ap = speye(N-1) + (1-theta)*P(:,f7+(1:N-1));
B  = P(:,f7+[0 N]);
```

LU factorization

Two broad categories of methods:

- Direct methods (resort to factorization)
- Iterative methods (stationary, non-stationary)

Direct methods resort to a factorization of matrix A (transform the system into one easier to solve)
LU factorization (method of choice if A has no particular structure nor quantification)



L Lower triangular matrix
U Upper triangular matrix

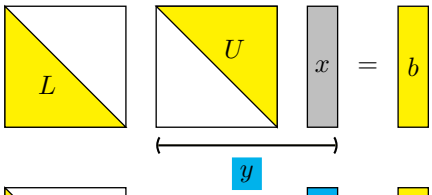
Operation count: $\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n + 1$

I

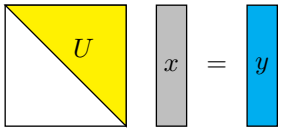
Ilustrate complexity with example $n^3 + n^2 + n$ for $n = 100, 1\,000, 10\,000$

Solution of $Ax = b$

In $Ax = b$ we replace A by its factorization



Solve $Ly = b$ (forward substitution)



Solve $Ux = y$ (backward substitution)

LU factorization with Matlab

- Matlab uses the syntax `x = A\b` to solve $Ax = b$ (no matter what A is)
- LU factorization Matlab: `[L,U] = lu(A);` But L is a permuted triangular matrix !!!
  We can not apply a forward substitution
- `[L,U,P] = lu(A);` produces a triangular matrix L
  P is a permutation matrix such that $LU = PA$

```
[L,U,P] = lu(A);
x = U \ ( L \ P*b );
```

If the system is solved only once, we simply code `x = A\b`
(see `ExSL0.m`)

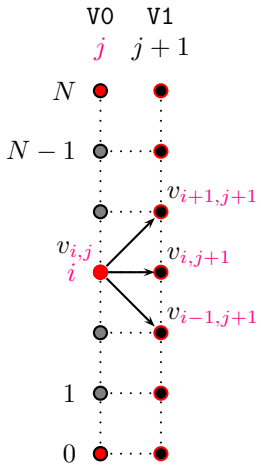
ExSL0.m

Generate A with Matlab and compute LU

Implementation of θ-method with Matlab

Statement 4 of the algorithm 1:

Solve $A_m \underbrace{v_\Omega^j}_{\texttt{V0(1:N-1)}} = \underbrace{A_p \underbrace{v_\Omega^{j+1}}_{\texttt{V1(1:N-1)}} + \theta B \underbrace{v_{\partial\Omega}^j}_{\texttt{V0([0 N])}} + (1-\theta) B \underbrace{v_{\partial\Omega}^{j+1}}_{\texttt{V1([0 N])}}}_{b}$




```
function P = FDM1DEuPut(S,E,r,T,sigma,q,Smin,Smax,M,N,theta)
% Finite difference method for European put
f7 = 1;
[Am,Ap,B,Svec] = GridU1D(r,q,T,sigma,Smin,Smax,M,N,theta);
V0 = max(E - Svec,0);
% Solve linear system for successive time steps
[L,U] = lu(Am);
for j = M-1:-1:0
    V1 = V0;
    V0(f7+0) = (E-Svec(f7+0))*exp(-r*(T-j*T/M)); % dt=T/M
    b = Ap*V1(f7+(1:N-1)) +  theta *B*V0(f7+[0 N])...
        + (1-theta)*B*V1(f7+[0 N]);
    V0(f7+(1:N-1)) = U\(L\b);
end
P = interp1(Svec,V0,S,'spline');
```

Example

```
>> P = BSput(50,50,0.10,5/12,0.40)
P =
    4.0760
>> P = FDM1DEuPut(50,50,0.10,5/12,0.40,0,0,150,100,500,1/2)
P =
    4.0760
>> P = FDM1DEuPut(50,50,0.10,5/12,0.40,0,0,150,100,500,0)
P =
 -2.9365e+154
>> P = FDM1DEuPut(50,50,0.10,5/12,0.40,0,0,150,100,500,1)
P =
    4.0695
>>
```


Example

```
function P = BSput(S,E,r,T,sigma)
% Evaluation d'un put Européen avec Black-Scholes
%
d1 = (log(S./E) + (r + sigma.^2 /2) .* T) ./ (sigma .* sqrt(T));
d2 = d1 - sigma .* sqrt(T);
Ee = E .* exp(-r .* T);
P  = Ee .* normcdf(-d2) - S .* normcdf(-d1);
```

```
>> P = FDM1DEuPut(50,50,0.10,5/12,0.40,0,0,150,1000,100,0)
P =
    4.0756
```

Need to discuss stability !!

Comparison of FDM

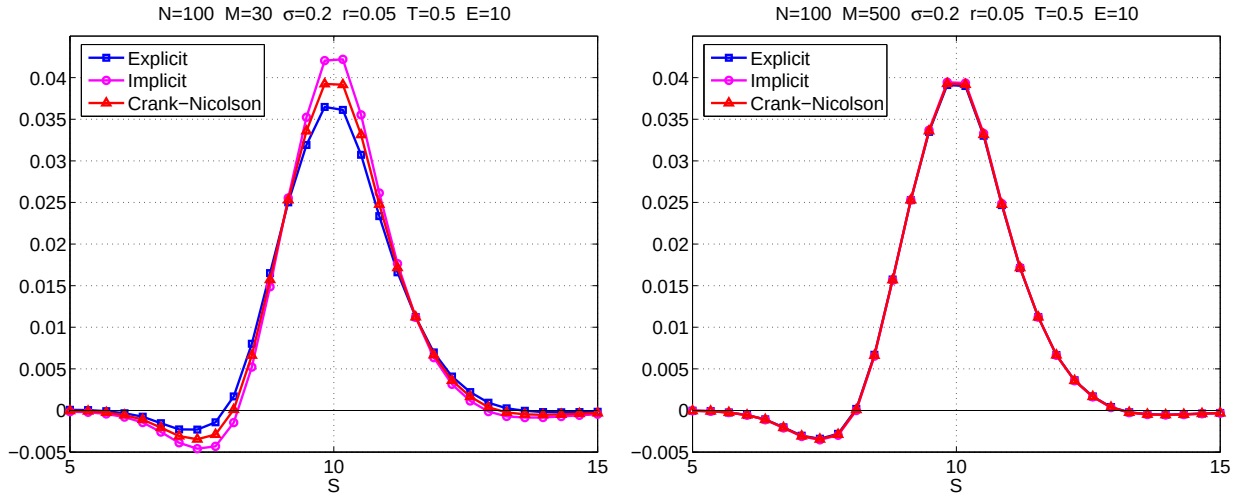
```
function CompareFDM(S,E,r,T,sigma,q, Smin,Smax,M,N)
Ex = FDM1DEuPut(S,E,r,T,sigma,q,Smin,Smax,M,N,0);
Im = FDM1DEuPut(S,E,r,T,sigma,q,Smin,Smax,M,N,1);
CN = FDM1DEuPut(S,E,r,T,sigma,q,Smin,Smax,M,N,1/2);
BS = BSPut(S,E,r,T,sigma);
Ex_err = BS - Ex;
Im_err = BS - Im;
CN_err = BS - CN;
figure
plot(S,Ex_err,'b.-','LineWidth',1),  hold on
plot(S,Im_err,'mo-','LineWidth',1)
plot(S,CN_err,'r^-','LineWidth',1)
plot([S(1) S(end)],[0 0],'k');
legend('Explicit','Implicit','Crank-Nicolson',2);
title(['N=',int2str(N),'  M=',int2str(M),'  sigma=',num2str(sigma,2),...
      '  r=',num2str(r,2),'  T=',num2str(T,1),'  E=',int2str(E)]);
xlabel('S'); ylim([-.005 .045]); grid on
```


Comparison of FDM

```
% goCompareFDM.m  (without transformation)
clear, close all
E = 10; r = 0.05; T = 6/12; sigma = .2; q = 0;
Smin = 0; Smax = 100;   S = linspace(5,15,30);
%
M = 30;  N = 100;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 500;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 30; N = 200;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 30; N = 200; sigma = 0.4;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 30; N = 310; sigma = 0.2;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 30; N = 700; sigma = 0.2;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
M = 30; N = 700; sigma = 0.5;
CompareFDM(S,E,r,T,sigma,q,Smin,Smax,M,N)
```

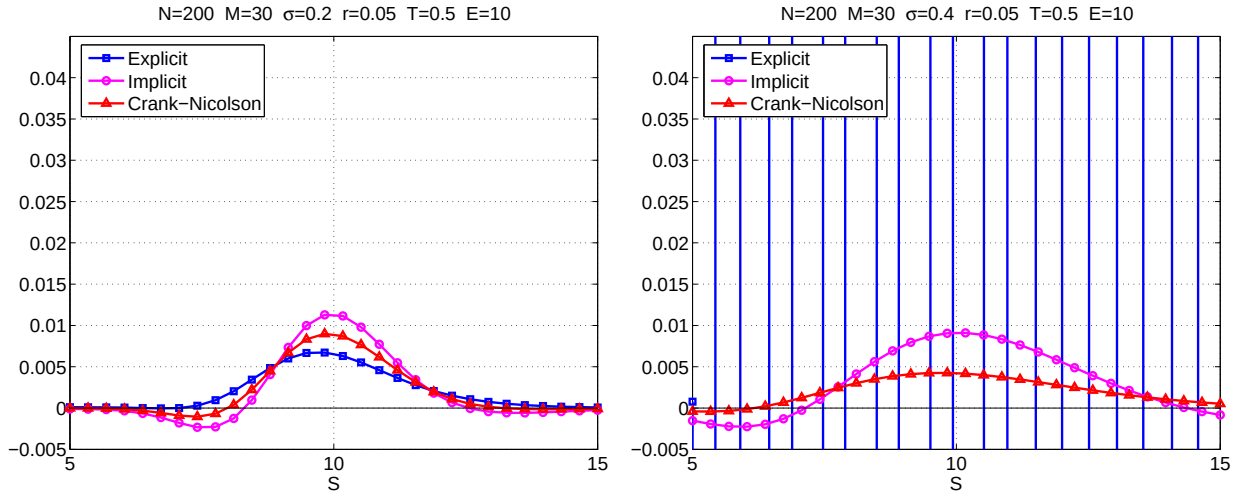
Comparison of FDM

Time discretization:



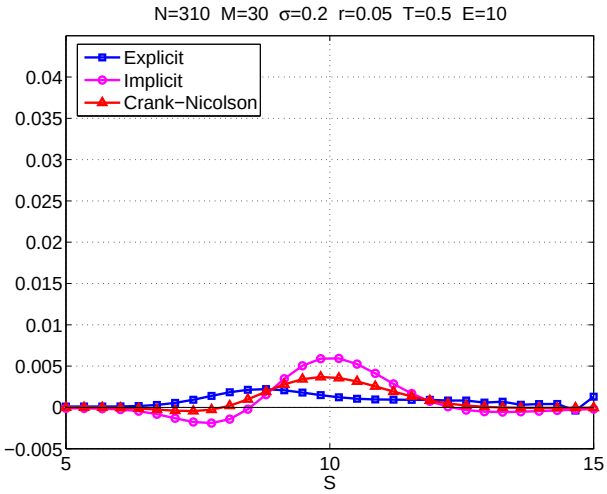
Comparison of FDM

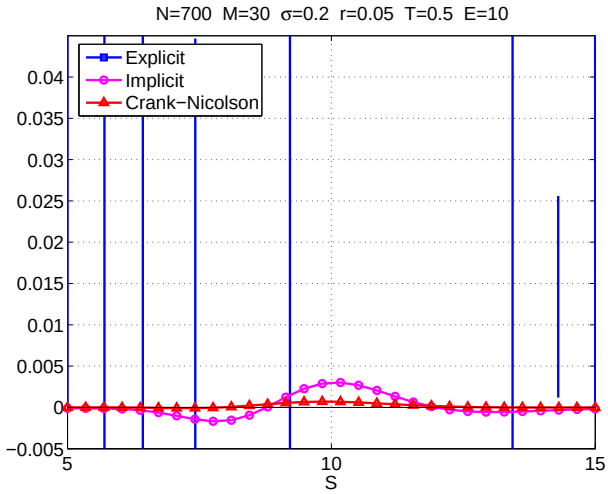
Increase space discretization with different volatilities:



Comparison of FDM

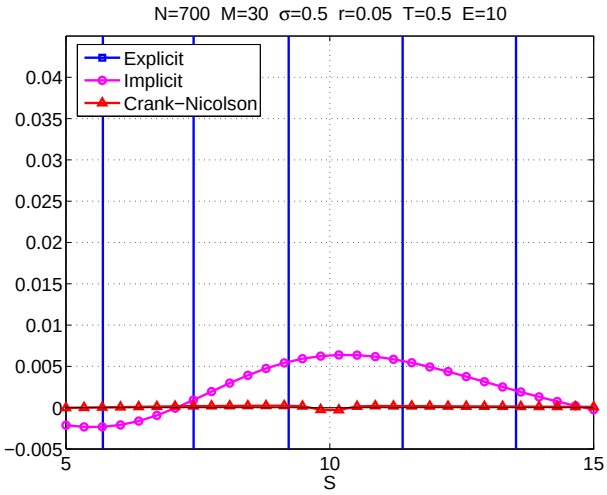
Increase space discretization:





Comparison of FDM

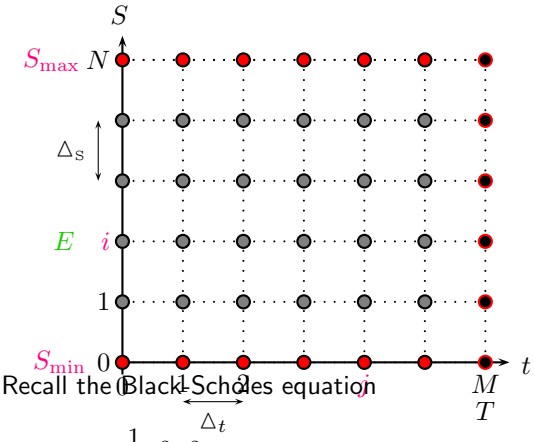
Higher volatility:



Conclusions:

- errors larger *at-the-money*
- errors depend on σ and r
- only Crank-Nicolson produces higher precision by increasing N (M constant)
- suggests variable grid (finer around E)

Coordinate transformation of space variable



Recall the Black-Scholes equation

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + (r-q)\, S\, V_S - r\, V = 0$$

consider the transformation

$$S = S(x) \tag{11}$$

and remplace (11) in the Black-Scholes equation.

Compute derivatives of V(S(x)) (with respect to x):

$$V_x = \frac{\partial V}{\partial S}\frac{\partial S}{\partial x} = V_S J(x) \quad \text{and} \quad V_S = \frac{V_x}{J(x)}$$

$$\frac{\partial}{\partial x}\left(V_S\right) = V_{SS} J(x) \quad \text{and} \quad V_{SS} = \frac{1}{J(x)}\frac{\partial}{\partial x}\underbrace{\left(\frac{V_x}{J(x)}\right)}_{V_S}$$

Coordinate transformation of space variable

Writing the Black-Scholes equation as a function of x

$$V_t + \frac{\sigma^2 S^2(x)}{2J(x)}\frac{\partial}{\partial x}\left(\frac{V_x}{J(x)}\right) + (r-q)\frac{S(x)}{J(x)}V_x - rV = 0$$

We use the following approximations:

$$\begin{aligned}
J(x) &\approx \frac{J\left(x+\frac{\triangle_x}{2}\right) + J\left(x-\frac{\triangle_x}{2}\right)}{2} \\
J\left(x+\tfrac{\triangle_x}{2}\right) &\approx \frac{S(x+\triangle_x) - S(x)}{\triangle_x} \\
V_x &\approx \frac{V(x+\triangle_x) - V(x-\triangle_x)}{2\,\triangle_x} \\
\frac{\partial}{\partial x}\left(\frac{V_x}{J(x)}\right) &\approx \left(\frac{V(x+\triangle_x) - V(x)}{J(x+\frac{\triangle_x}{2})\,\triangle_x^2} - \frac{V(x) - V(x-\triangle_x)}{J(x-\frac{\triangle_x}{2})\,\triangle_x^2}\right)
\end{aligned}$$

Coordinate transformation of space variable

Using a forward difference to approximate V_t we get:

$$\frac{v_{i,j+1} - v_{ij}}{\triangle_t} + \frac{\sigma^2 S_i^2}{2J_i}\left(\frac{v_{i+1,j} - v_{ij}}{\triangle_x^2\, J_{i+\frac{1}{2}}} - \frac{v_{ij} - v_{i-1,j}}{\triangle_x^2\, J_{i-\frac{1}{2}}}\right) + (r-q)\frac{S_i}{J_i}\left(\frac{v_{i+1,j} - v_{i-1,j}}{2\,\triangle_x}\right) - r\, v_{ij} = 0$$

Substituting

$J_{i+\frac{1}{2}} = \frac{S_{i+1}-S_i}{\triangle_x}$, $J_{i-\frac{1}{2}} = \frac{S_i - S_{i-1}}{\triangle_x}$ and $J_i = \frac{S_{i+1}-S_{i-1}}{2\triangle_x}$

Coordinate transformation of space variable

we obtain

$$v_{i,j+1} - v_{ij} + \frac{\sigma^2 S_i^2\, \triangle_t}{S_{i+1} - S_{i-1}}\left(\frac{v_{i+1,j} - v_{ij}}{S_{i+1} - S_i} - \frac{v_{ij} - v_{i-1,j}}{S_i - S_{i-1}}\right) + (r-q)\frac{S_i\, \triangle_t}{S_{i+1} - S_{i-1}}(v_{i+1,j} - v_{i-1,j}) - \triangle_t\, r\, v_{ij} = 0$$

We denote

$a_i = \frac{S_i \triangle_t}{S_{i+1}-S_{i-1}}$, $b_i = \frac{\sigma^2 S_i a_i}{S_{i+1}-S_i}$, $c_i = \frac{\sigma^2 S_i a_i}{S_i - S_{i-1}}$ and $e_i = (r-q)a_i$

we get the $i = 1, 2, \ldots, N-1$ equations

$$v_{i,j+1} - v_{ij} + b_i(v_{i+1,j} - v_{ij}) - c_i(v_{ij} - v_{i-1,j}) + e_i(v_{i+1,j} - v_{i-1,j}) - \triangle_t\, r\, v_{ij} = 0$$

Collecting identical terms

$$v_{i,j+1} - v_{ij} + \underbrace{(c_i - e_i)}_{d_i} v_{i-1,j} + \underbrace{(-b_i - c_i - \triangle_t\, r)}_{m_i} v_{ij} + \underbrace{(b_i + e_i)}_{u_i} v_{i+1,j} = 0$$

$$v_{i,j+1} - v_{ij} + \underbrace{(c_i - e_i)}_{d_i} v_{i-1,j} + \underbrace{(-b_i - c_i - \triangle_t\, r)}_{m_i} v_{ij} + \underbrace{(b_i + e_i)}_{u_i} v_{i+1,j} = 0$$

Coordinate transformation of space variable

In matrix notation

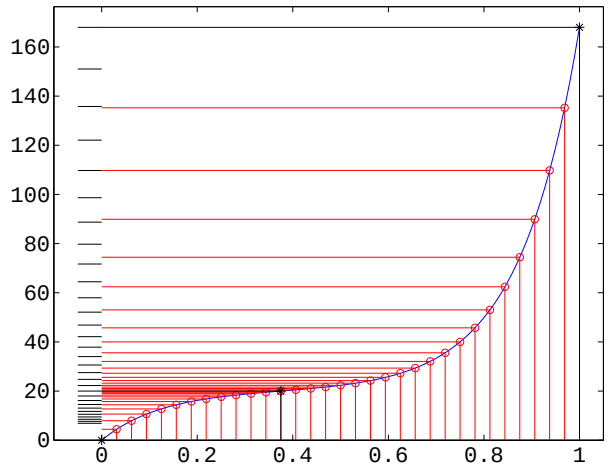
$$v_\Omega^{j+1} - v_\Omega^j + P\, v_{\overline{\Omega}}^j = 0$$

$$P = \begin{bmatrix} d_1 & m_1 & u_1 & 0 & \cdots & \cdots & 0 \\ 0 & d_2 & m_2 & u_2 & & & \vdots \\ \vdots & & d_3 & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & u_{N-2} & 0 \\ 0 & \cdots & \cdots & 0 & d_{N-1} & m_{N-1} & u_{N-1} \end{bmatrix}$$

We apply the same partition to P as done previously and compute the matrices A_m, A_p and B needed to implement the θ-method

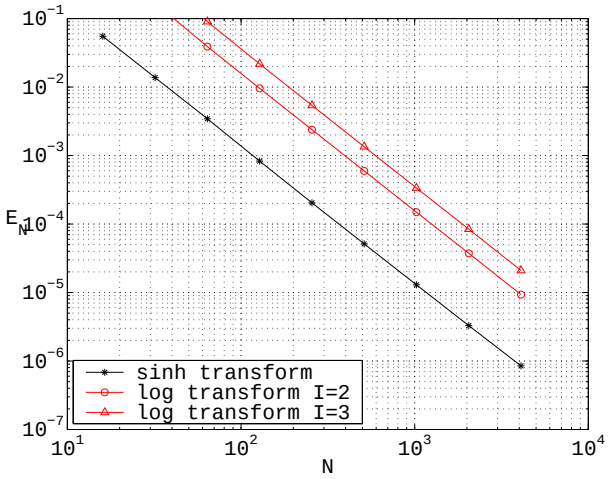
Choice of transformation $S = S(x)$

- $x = \log(S/E) \quad \rightarrow \quad S = Ee^x \quad x \in [-I,\, I]$ (log transform)
- $S = E\frac{e^{x-c}-e^{-(x-c)}}{2\lambda} + E \quad x \in [0,\, 1]$ (sinh transform)



Comparaison of transformations

Errors *at-the-money* for $\sinh$ and $\log$ transformation
($E = 40$, $r = 0.05$, $T = 0.25$, $\sigma = 0.40$, $q = 0$)



American options

Can be exercised at any time → price must satisfy the following PDE

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + (r-q)S V_S - r V \geq 0$$

and in order to exclude arbitrage opportunities

$$V(S,t) \geq (E-S)_+ \qquad V(S,t) \geq V(S,T) \tag{12}$$

Terminal and boundary conditions (*put*)

$$V(S,T) = (E-S)_+ \qquad V(0,t) = E, \quad \lim_{S\to\infty} V(S,t) = 0$$

We denote $S_f(t)$ the largest value of S, at time t, for which

$$V\big(S_f(t),t\big) = \Big(E - S_f(t)\Big)$$

$S_f(t)$ defines the *free* or *early exercise boundary*,
i.e. price of underlying at time t for which it is optimal to exercise the option

American options

Solution is formalized as

$$\mathcal{L}_{BS}(V)\ \Big(V(S,t) - V(S,T)\Big) = 0$$
$$\mathcal{L}_{BS}(V) \geq 0 \quad \text{et} \quad \Big(V(S,t) - V(S,T)\Big) \geq 0$$

$\mathcal{L}_{BS}(V) \stackrel{def}{=} V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + r S V_S - r V$ is the Black-Scholes differential operator

Constitutes a sequence of "*linear complementary problems*"

American options

Using the notation introduced for the formalization of the θ-method:

$$\left(A_m\, v_\Omega^j - b^j\right) \dot{\times} \left(v_\Omega^j - v_\Omega^M\right) = 0$$
$$A_m v_\Omega^j \geq b^j \qquad \text{and} \qquad v_\Omega^j \geq v_\Omega^M$$

$v_\Omega^M \equiv$ *payoff*
$A_m = I - \theta A \qquad A_p = I + (1-\theta)A$
$b^j = A_p\, v_\Omega^{j+1} + \theta\, B\, v_{\partial\Omega}^j + (1-\theta)\, B\, v_{\partial\Omega}^{j+1}$

The solution can be approached by first computing x the solution of

$$\underbrace{A_m\, v_\Omega^j}_{A\,x} = \underbrace{b^j}_{b} \quad \text{and } \textit{correct} \text{ it as} \quad v_\Omega^j = \max\left(x,\, v_\Omega^M\right)$$

Two main methods used for the *correction*:

- Projected successive overrelaxation (PSOR)
- Explicit payout

Explanation:

We must reason component-wise when explaining the $\max$ operation.

Projected SOR (PSOR)

- Solve $Ax = b$ with an iterative method
- If necessary truncate element in solution vector at every step

PSOR is a modification of Gauss-Seidel.
Gauss-Seidel for $Ax = b$:

$$A\,x = b \quad \equiv \quad \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{cases}$$

We explicit the diagonal element of each equation:

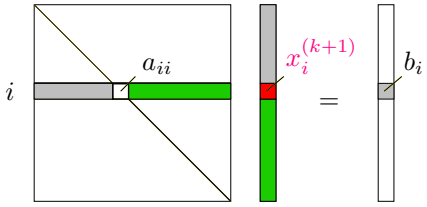
$$\begin{aligned} x_1 &= (b_1 \quad -a_{12}x_2 \quad -a_{13}x_3 \quad)/a_{11} \\ x_2 &= (b_2 \quad -a_{21}x_1 \quad -a_{23}x_3 \quad)/a_{22} \\ x_3 &= (b_3 \quad -a_{31}x_1 \quad -a_{32}x_2 \quad)/a_{33} \end{aligned}$$

Elements x_i on the right-hand side are unknown, assume an approximation $x^{(k)}$ for $x = A^{-1}b$, we then get a new approximation $x^{(k+1)}$ by computing

$$\begin{aligned} x_1^{(k+1)} &= (b_1 \quad -a_{12}\,x_2^{(k)} \quad -a_{13}\,x_3^{(k)} \quad)/a_{11} \\ x_2^{(k+1)} &= (b_2 \quad -a_{21}\,x_1^{(k+1)} \quad -a_{23}\,x_3^{(k)} \quad)/a_{22} \\ x_3^{(k+1)} &= (b_3 \quad -a_{31}\,x_1^{(k+1)} \quad -a_{32}\,x_2^{(k+1)} \quad)/a_{33} \end{aligned}$$

Implementation for Gauss-Seidel

1: Give initial solution $x^{(0)} \in \mathbb{R}^n$
2: **for** $k = 0, 1, 2, \ldots$ `until` convergence **do**
3: **for** $i = 1 : n$ **do**
4: $x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\, x_j^{(k)} \right) / a_{ii}$
5: **end for**
6: **end for**



Same array x can be used to store $x^{(k)}$ and $x^{(k+1)}$

PSOR for $Ax = b$

1: Give starting solution $x^{(0)} \in \mathbb{R}^n$
2: **for** $k = 0, 1, 2, \ldots$ `until` convergence **do**
3: **for** $i = 1 : n$ **do**
4: $x_i^{(k+1)} = \omega\, x_{\mathrm{GS},i}^{(k+1)} + (1 - \omega)\, x_i^{(k)}$
5: **end for**
6: **end for**

$x_{\mathrm{GS},i}^{(k+1)}$ Gauss-Seidel solution
$0 < \omega < 2$ relaxation parameter

Statement 4 can be rewritten

$$x_i^{(k+1)} = \omega \left(x_{\mathrm{GS},i}^{(k+1)} - x_i^{(k)} \right) + x_i^{(k)}$$

$$\frac{b_i - \sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\, x_j^{(k)}}{a_{ii}} - \frac{a_{ii}\, x_i^{(k)}}{a_{ii}}$$

$$\left(b_i - \sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}\, x_j^{(k)} \right) / a_{ii}$$

$$x_i^{(k+1)} = \omega \left(b_i - A_{i,1:n}\, x \right) / a_{ii} + x_i^{(k)}$$

PSOR for $Ax = b$

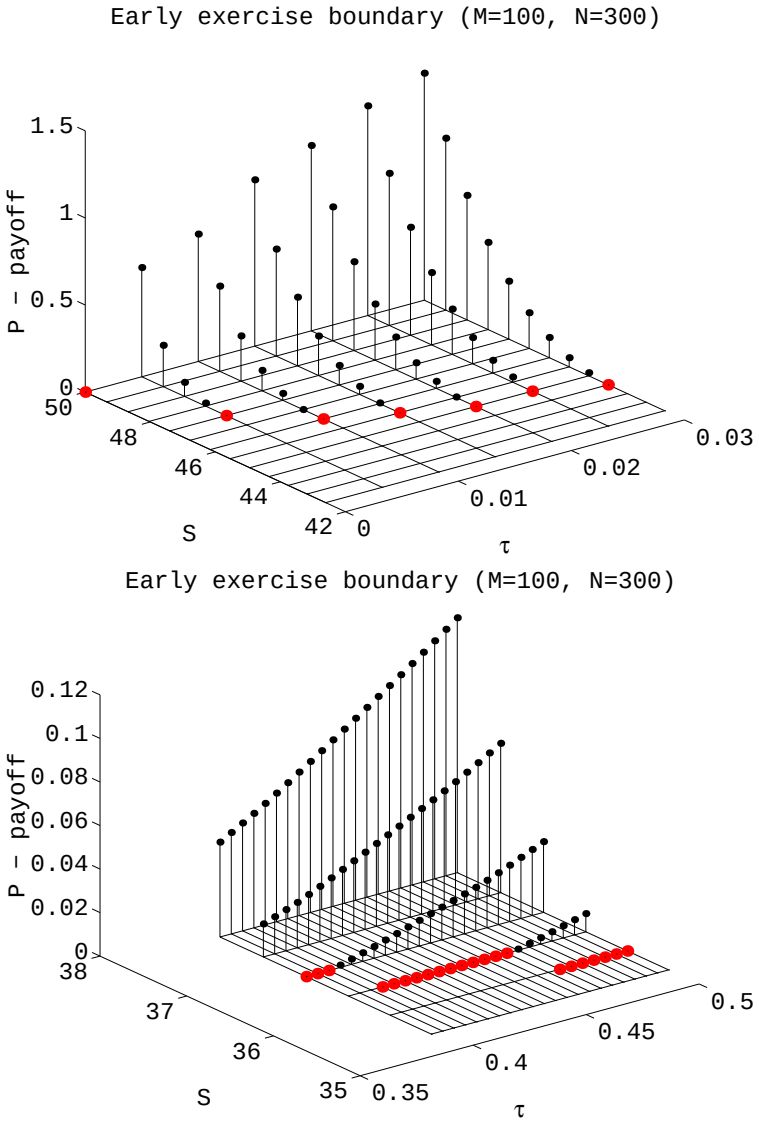
```
function x1 = PSOR(x1,A,b,payoff,omega,tol,maxit)
% Projected SOR for Ax = b
if nargin == 4, tol=1e-6; omega=1.2; maxit=100; end
it = 0; converged = 0; N = length(x1);
while ~converged
   x0 = x1;
   for i = 1:N-1
      x1(i) = omega*(b(i)-A(i,:)*x1)/A(i,i) + x1(i);
      x1(i) = max(payoff(i), x1(i));
   end
   converged = all((abs(x1-x0)./(abs(x0)+1)) < tol );
   it=it+1; if it>maxit, error('Maxit reached'), end
end
```


Explicit payout (EP)

- Solve linear system with a direct method
- Truncate the solution in order to satisfy (12) $V(S,t) \geq V(S,T)$

1: $v_\Omega^M = (E - S)_+$
2: **for** $j = M - 1 : -1 : 0$ **do**
3: $\quad b^j = A_p\, v_\Omega^{j+1} + \theta\, B\, v_{\partial\Omega}^j + (1-\theta)\, B\, v_{\partial\Omega}^{j+1}$
4: $\quad$ Solve $A_m\, v_\Omega^j = b^j$
5: $\quad v_\Omega^j = \max(v_\Omega^j,\, v_\Omega^M)$
6: **end for**

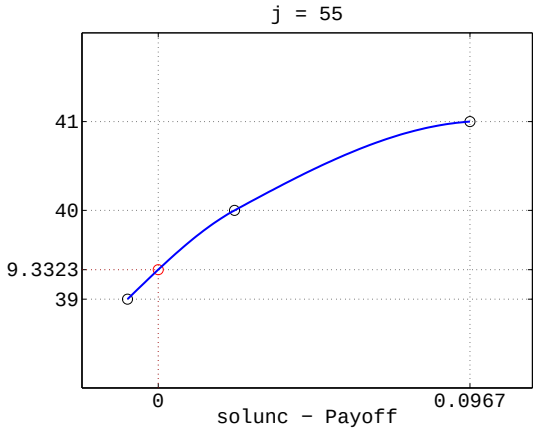
Computation of early exercise boundary



Comment

First panel shows the boundary for small values of τ (near expiration)
Second panel shows the boundary for larger values of τ (far from expiration). If we consider the truncated solution we will obtain a step function.

Computation of S_f by interpolation



Seek last negative element of `solunc` − `Payoff`, consider next following two elements, fit a polynomial, seek zero of polynomial

```
solunc = U\(L\b);
[V0(f7+(1:N-1)),Imax] = max([Payoff solunc],[],2);
p = find(Imax == 2);
i = p(1) - 1; % Grid line of last point below payoff
Sf(f7+j) = interp1(solunc(i:i+2) - Payoff(i:i+2), ...
                   Svec(f7+(i:i+2)),0,'cubic');
```

$$[\texttt{Payoff solunc}] = \begin{bmatrix} 20.0 & 15.1 \\ 19.0 & 15.6 \\ 18.0 & 15.8 \\ 17.0 & 16.0 \\ 16.0 & 16.1 \\ 15.0 & 16.2 \\ 14.0 & 16.5 \end{bmatrix} \qquad \texttt{Imax} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

Implementation with Matlab

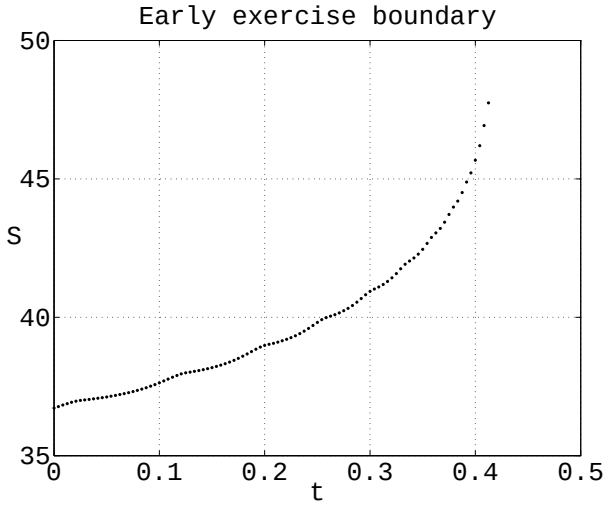
```
function [P,Sf] = FDM1DAmPut(S,E,r,T,sigma,q,Smin,Smax,M,N,theta,method)
% Finite difference method for American put
f7 = 1;
[Am,Ap,B,Svec] = GridU1D(r,q,T,sigma,Smin,Smax,M,N,theta);
V0 = max(E - Svec,0);   Payoff = V0(f7+(1:N-1));
Sf = repmat(NaN,M+1,1);
[L,U] = lu(Am);  % Solve linear system for successive time steps
for j = M-1:-1:0
    V1 = V0;
    V0(f7+0) = E;
    b = Ap*V1(f7+(1:N-1)) + theta *B*V0(f7+[0 N]) + (1-theta)*B*V1(f7+[0 N]);
    if strcmp(method,'PSOR')
        V0(f7+(1:N-1)) = PSOR(V1(f7+(1:N-1)),Am,b,Payoff);
    else % Explicit payout
        solunc = U\(L\b);
        [V0(f7+(1:N-1)),Imax] = max([Payoff solunc],[],2);
        p = find(Imax == 2);
        i = p(1) - 1; % Grid line of last point below payoff
        Sf(f7+j) = interp1(solunc(i:i+2)-Payoff(i:i+2),Svec(f7+(i:i+2)),0,'cubic');
    end
end
P = interp1(Svec,V0,S,'spline');
```


Example

```
S = 50; E = 50; r = 0.10; T = 5/12; sigma = 0.40; q = 0;
Smin = 0; Smax = 100; N = 100; M = 100; theta = 1/2;

tic
[P,Sf] = FDM1DAmPut(S,E,r,T,sigma,q,Smin,Smax,M,N,theta,'PSOR');
fprintf('\n PSOR:  P = %8.6f (%i sec)\n',P,ceil(toc));

 PSOR:   P = 4.279683  (2 sec)

tic
[P,Sf] = FDM1DAmPut(S,E,r,T,sigma,q,Smin,Smax,M,N,theta,'EPOUT');
fprintf('\n EPout: P = %8.6f (%i sec)\n',P,ceil(toc));

 EPout:  P = 4.277555  (1 sec)
```

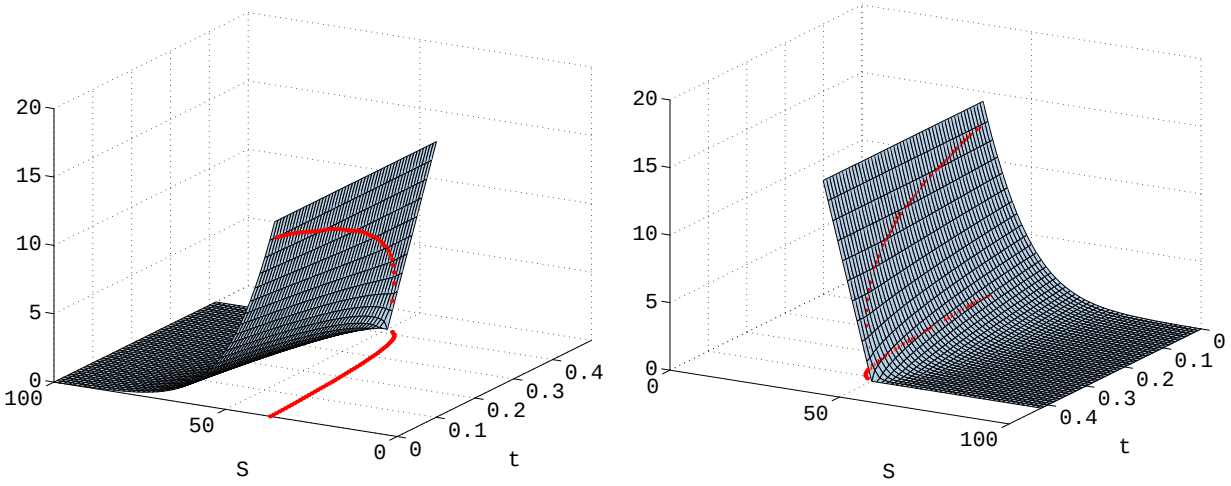
- PSOR executes slower
- Explicit payout $\neq$ PSOR

Example

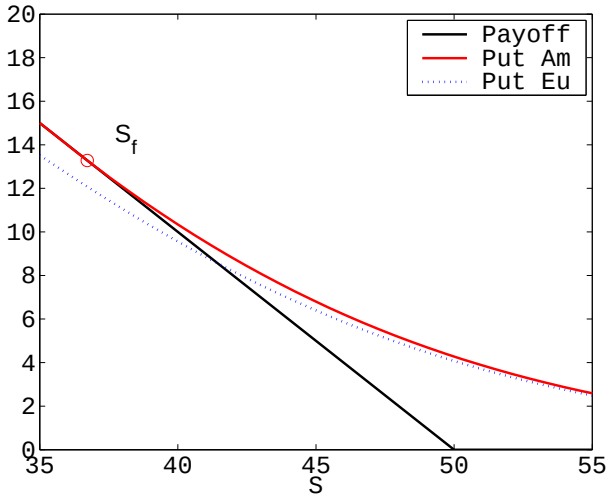


Option is exercised if S is below the boundary

Example



Example



Generation of uniform random variables $[0, 1[$

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*.

Générateur linéaire congruent

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

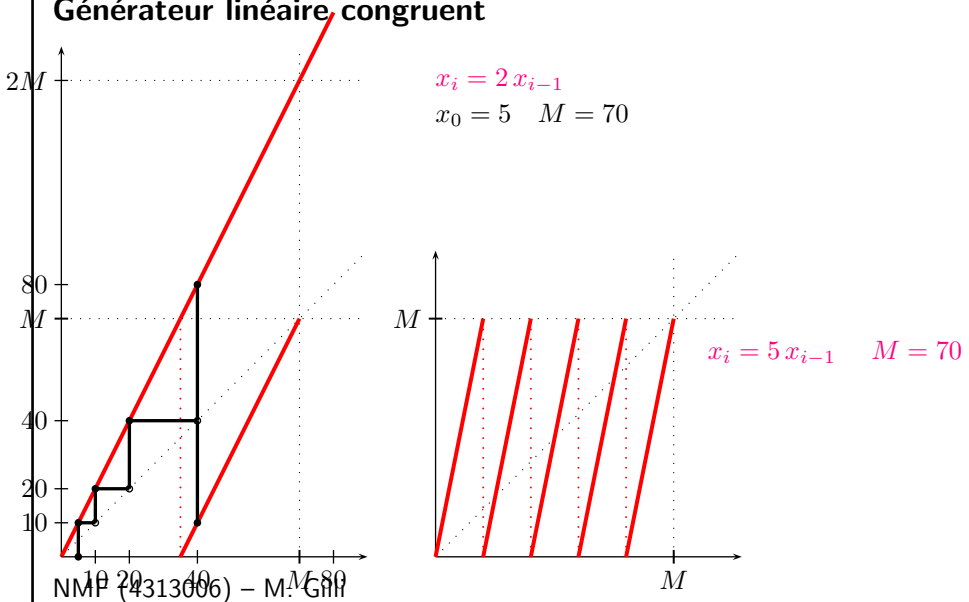
$$x_i = a\, x_{i-1} \pmod{M} \tag{13}$$

  avec a et M des entiers donnés.
- $x_i \in \{0, 1, \ldots, M-1\}$
- 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
...	...		

Générateur linéaire congruent



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).

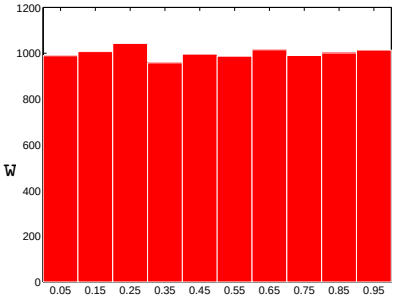
Modulo fait évoluer la fonction sur un tore de longueur M et de circonférence M.

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 $n \times m$

```
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);
```



Simulation of asset price $S(t)$

Geometric brownian motion for $S(t)$:

$$dS = \mu\, S dt + \sigma\, S dz$$

μ	drift
σ	volatility
dz	standard Wiener process

Discretization (Prisman, 2000, p. 629):

Interval $[0, T]$, $\frac{T}{\triangle_t}$ increments

$$S_{t+\triangle_t} = S_t \exp\left((r - \tfrac{\sigma^2}{2})\, \triangle_t + \sigma\sqrt{\triangle_t}\, u \right) \quad u \sim \mathrm{N}(0,1)$$

We can also write

$$S_T = S_0 \exp\left((r - \tfrac{\sigma^2}{2})\, T + \sigma\sqrt{T}\, u \right)$$

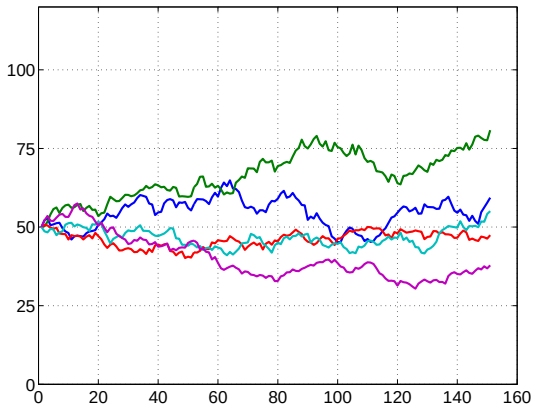
Simulation of price paths

$$S_{t+\triangle_t} = S_t \exp\left(\underbrace{(r - \tfrac{\sigma^2}{2})\, \triangle_t}_{\texttt{trend}} + \underbrace{\sigma\sqrt{\triangle_t}}_{\texttt{sigdt}}\, u \right)$$

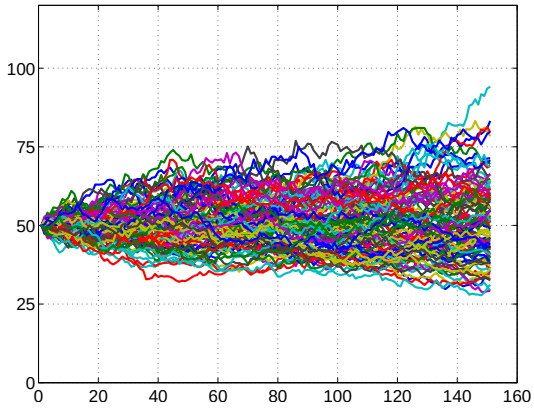
```
function S = MBG00(S0,r,T,sigma,NIncr,NRepl)
% Geometric brownian motion (Simulation of NRepl paths)
% Every path has NIncr+1 points (S(f7+0)=S0)
f7 = 1;
dt = T/NIncr;
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
S = repmat(NaN,NRepl,NIncr+1);
S(:,f7+0) = S0;
for i = 1:NRepl
   for j = 1:NIncr
      S(i,f7+j) = S(i,f7+j-1)*exp(trend+sigdt* randn );
   end
end
```

Example of simulation of price paths

```
randn('state',10);
S = MBG00(50,0.1,5/12,0.4,150,5);
plot(S'); ylim([0 120])
```




```
S = MBG00(50,0.1,5/12,0.4,150,100);
```




```
tic, S = MBG00(50,0.1,5/12,0.4,150,10000); toc
elapsed_time =
   33.0100
```

Too slow !!!

Efficient simulation of price paths

A more efficient method for generating price paths (Matlab):

- $S_{t+\Delta_t} = S_t \exp\left((r - \frac{\sigma^2}{2})\, \Delta_t + \sigma\sqrt{\Delta_t}\, u \right)$
- Change of variable $z_t = \log(S_t)$
- $z_{t+\Delta_t} = z_t + \underbrace{\left(r - \frac{\sigma^2}{2}\right) \Delta_t + \sigma\sqrt{\Delta_t}\, u}_{h_t}$
- $v = [\, v_0 \;\; v_1 \;\; v_2 \;\; \cdots \,] \equiv [\, z_0 \;\; h_1 \;\; h_2 \;\; h_3 \;\; \cdots \,]$
- $z_t = \sum_{i=0}^{t} v_i$
- with Matlab `z = cumsum(v)`

Simulation of price paths

Efficient simulation of price paths:

```
function S = MBG(S0,r,T,sigma,NIncr,NRepl)
% Brownian geometric motion
% Simulation of NRepl paths with NIncr+1 points (S(1)=S0)
dt = T/NIncr;
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
h = trend + sigdt*randn(NRepl,NIncr);
z = cumsum([repmat(log(S0),NRepl,1) h],2);
S = exp(z); % Price paths

tic, S = MBG(50,0.1,5/12,0.4,150,10000); toc
elapsed_time =
    2.2000
```

≈ 15 times faster

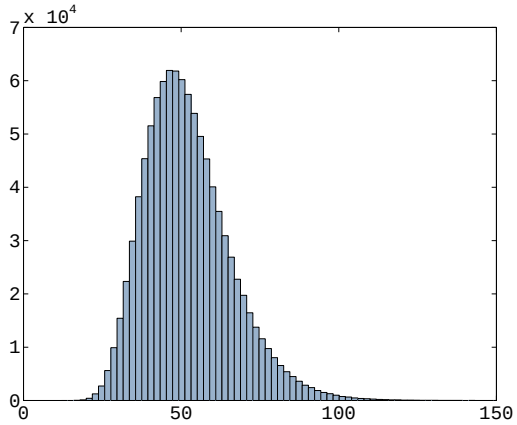
Simulation of price paths

Without temporary storage matrices `h` and `z`:

```
function S = MBGV(S0,r,T,sigma,NIncr,NRepl)
% Brownian geometric motion
% Simulation of NRepl paths with NIncr+1 points (S(1)=S0)
dt = T/NIncr;
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
S = exp( cumsum([repmat(log(S0),NRepl,1) ...
         trend + sigdt*randn(NRepl,NIncr)],2) );


tic, S = MBGV(50,.1,5/12,.4,1,1000000); toc
elapsed_time =
    2.4700
```

Simulation of price paths

```
hist(S(:,end),100)
axis([0 150 0 7e4])
colormap([.6 .7 .8])
```



Valuation of an European call

- Compute price path j of underlying $\rightarrow S_T^{(j)}$ (price at expiry)
- *payoff* (at expiry)

$$C_T^{(j)} = \left(S_T^{(j)} - E\right)_+$$

- discounting at risk free rate $r \rightarrow$ actual value of *payoff*

$$C_0^{(j)} = e^{-Tr} C_T^{(j)}$$

- Expectation $E(C_0^{(j)})$ is option price
  Mean of N simulations is an estimator of the expectation

$$\hat{C}_0 = \frac{1}{N} \sum_{j=1}^{N} C_0^{(j)}$$

Confidence intervals

$\hat{C}_0$ is a random variable → we can construct confidence intervals

- Estimator of variance of $C_0^{(j)}$

$$V(C_0^{(j)}) = \frac{1}{N-1}\sum_{j=1}^{N}\left(C_0^{(j)} - \hat{C}_0\right)^2$$

- Variance of $\hat{C}_0$

$$\begin{aligned} V(\hat{C}_0) &= V\Big(\frac{1}{N}\sum_{j=1}^{N} C_0^{(j)}\Big) \\ &= \Big(\frac{1}{N}\Big)^2 N\, V(C_0^{(j)}) \\ &= \frac{V(C_0^{(j)})}{N} \end{aligned}$$

Central limit theorem

- $S_n = X_1 + X_2 + \cdots + X_n, \quad \mu = E(X_i), \quad \sigma^2 = E(X_i - \mu)^2$

$$\frac{S_n - n\,\mu}{\sigma\sqrt{n}} \backsim N(0,1)$$

- $X_j \equiv C_0^{(j)}, \qquad E(C_0^{(j)}) = C_0, \qquad S_N = N\hat{C}_0$

$$\frac{N\hat{C}_0 - NC_0}{\sqrt{V(C_0^{(j)})}\sqrt{N}} = \frac{\sqrt{N}(\hat{C}_0 - C_0)}{\sqrt{V(C_0^{(j)})}} \backsim N(0,1)$$

- $\varepsilon = 0.05, \quad t_{\varepsilon/2} = 1.96 \qquad -t_{\varepsilon/2} \leq \dfrac{\sqrt{N}(\hat{C}_0 - C_0)}{\sqrt{V(C_0^{(j)})}} \leq t_{\varepsilon/2}$

$$\hat{C}_0 - t_{\varepsilon/2}\sqrt{\frac{V(C_0^{(j)})}{N}} \leq C_0 \leq \hat{C}_0 + t_{\varepsilon/2}\sqrt{\frac{V(C_0^{(j)})}{N}}$$

Computing confidence intervals with Matlab

Given $x = [\, x_1 \; x_2 \; \cdots \; x_n \,]$, $\texttt{xbar} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$, $s^2 = \dfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \texttt{xbar})^2$

With Matlab we compute:

```
[xbar,s,cixbar,cis] = normfit(x,epsilon)
```

- `cixbar` confidence interval for `xbar`
- `cis` confidence interval for `s`
- `epsilon` confidence level (default $= .05$)

European call with Monte Carlo:

```
function [P,IC] = MC01(S0,E,r,T,sigma,NIncr,NRepl)
% European call with Monte Carlo (crude)
S = MBGV(S0,r,T,sigma,NIncr,NRepl);
Payoff = max( (S(:,end)-E), 0);
[P,ignore,IC] = normfit( exp(-r*T) * Payoff );
```


Examples

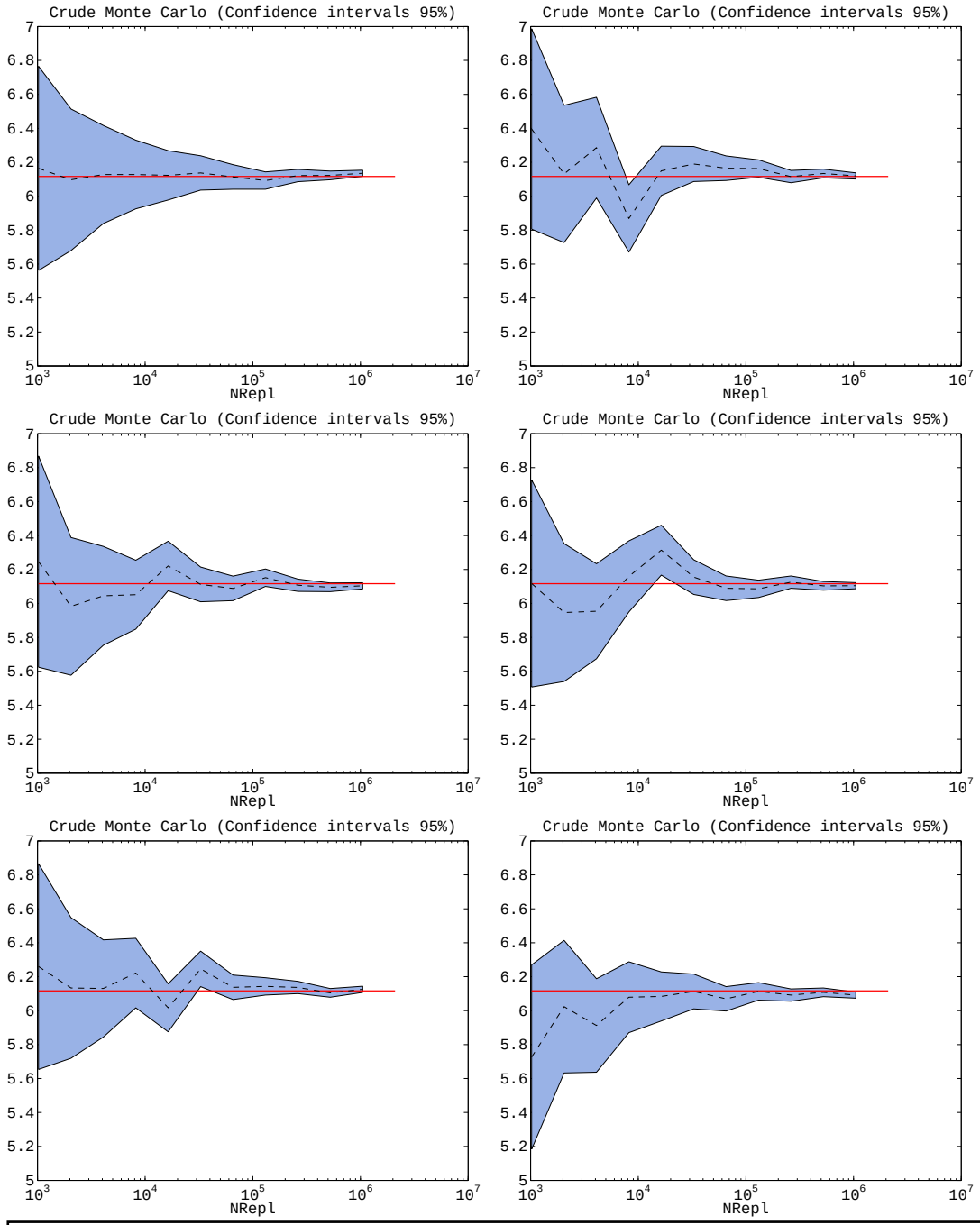
```
function dispMC(name,P,IC,t0)
% Output Monte Carlo results
fprintf('\n %9s: P = %6.3f  (%5.2f - %5.2f) ',name,P,IC(1),IC(2));
fprintf(' %4.2f  (%2i sec)',IC(2)-IC(1),ceil(etime(clock,t0)));

S = 50;
E = 50;
r = 0.10;
T = 5/12;
sigma = 0.40;

P = BScall(S,E,r,T,sigma)
P =
    6.1165
```

Examples

```
NIncr = 1; NRepl = 10000;
randn('seed',0);  t0 = clock;
[P,IC] = MC01(S,E,r,T,sigma,100,NRepl);
dispMC('MC01',P,IC,t0)

MC01: P =  6.131  ( 5.95 -  6.31) 0.36   ( 2 sec)

randn('seed',0); t0 = clock;
[P,IC] = MC01(S,E,r,T,sigma,1,NRepl);
dispMC('MC01',P,IC,t0)

MC01: P =  6.052  ( 5.86 -  6.24) 0.37   ( 1 sec)

randn('seed',0); t0 = clock;
[P,IC] = MC01(S,E,r,T,sigma,NIncr,100000);
dispMC('MC01',P,IC,t0)

MC01: P =  6.098  ( 6.04 -  6.16) 0.12   ( 1 sec)
```

Convergence rate

Standard deviation decreases proportional to $1/\sqrt{\texttt{NRepl}}$



Comment

Comment the different convergence pattern !!!

Variance reduction

Given two random variables X_1 and X_2 with same distribution, then

$$\mathrm{V}\left(\frac{X_1+X_2}{2}\right) = \frac{1}{4}\Big(\mathrm{V}(X_1)+\mathrm{V}(X_2)+2\,\mathrm{cov}(X_1,X_2)\Big)$$

- We can reduce the variance if X_1 et X_2 are negatively correlated !!
- How to construct X_1 and X_2 verifying a negative correlation?

$$X_1 = h(u)$$
$$X_2 = h(-u)$$

$u \sim \mathrm{N}(0,\sigma^2)$

Antithetic variates

- Consider a portfolio with two (identical) options
- Underlying S_1 and S_2 satisfy same stochastic differential equation

$$dS_1 = \mu\, S_1 dt + \sigma\, S_1 dz$$
$$dS_2 = \mu\, S_2 dt - \sigma\, S_2 dz$$

- The 2 options are identical (same μ and σ)
- S_1 et S_2 are perfectly (negatively) correlated
- Variance of mean of the 2 options is smaller !!

Antithetic variates

```
function [S1,S2] = MBGAT(S0,r,T,sigma,NIncr,NRepl)
% Brownian geometric motion (NRepl antithetic paths)
dt = T/NIncr; NRepl = fix(NRepl/2);
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
D = sigdt*randn(NRepl,NIncr);
S1 = exp(cumsum([repmat(log(S0),NRepl,1) trend + D],2));
S2 = exp(cumsum([repmat(log(S0),NRepl,1) trend - D],2));


function [P,IC] = MCAT(S0,E,r,T,sigma,NIncr,NRepl)
% European call with Monte Carlo (antithetic)
[S1,S2] = MBGAT(S0,r,T,sigma,NIncr,NRepl);
Payoff  = ( max( (S1(:,end)-E), 0) + max( (S2(:,end)-E), 0) ) / 2;
[P,ignore,IC] = normfit( exp(-r*T) * Payoff );
```

Antithetic variates: examples

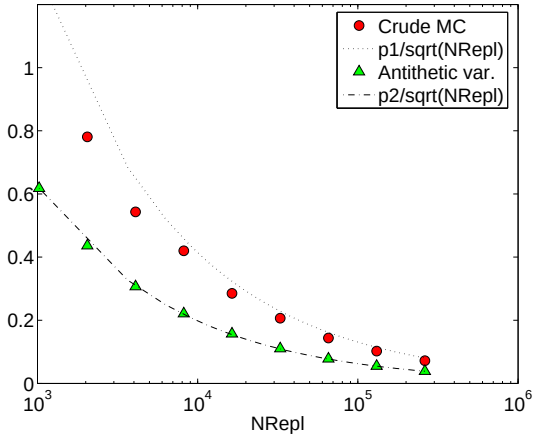
```
S = 50; E = 50; r = 0.10; T = 5/12; sigma = 0.40; NIncr = 1; NRepl = 10000;
P = BScall(S,E,r,T,sigma);   -->   P = 6.1165

randn('seed',0);  t0 = clock;
[P,IC] = MC01(S,E,r,T,sigma,NIncr,NRepl);         dispMC('MC01',P,IC,t0)

MC01: P =  6.052  ( 5.86 -  6.24) 0.37   ( 1 sec)

randn('seed',0);  t0 = clock;
[P,IC] = MCAT(S,E,r,T,sigma,NIncr,NRepl);         dispMC('MCAT',P,IC,t0)

MCAT: P =  6.080  ( 5.94 -  6.22) 0.20   ( 1 sec)

randn('seed',0);  t0 = clock;
[P,IC] = MC01(S,E,r,T,sigma,NIncr,1e6);       dispMC('MC01',P,IC,t0)

MC01: P =  6.123  ( 6.10 -  6.14) 0.04   ( 5 sec)

randn('seed',0);  t0 = clock;
[P,IC] = MCAT(S,E,r,T,sigma,NIncr,1e6);        dispMC('MCAT',P,IC,t0)

MCAT: P =  6.115  ( 6.10 -  6.13) 0.03   ( 4 sec)
```


"Crude" vs antithetic Monte Carlo

```
S0 = 50; E = 50; r = .10; sigma = .40; T = 5/12;
k1 = 10; k2 = 18; i = 0;
for NRepl = 2.^(k1:k2)
   i = i + 1;
   [P,IC] = MC01(S0,E,r,T,sigma,1,NRepl);
   cbr(i) = IC(2) - IC(1);
   [P,IC] = MCAT(S0,E,r,T,sigma,1,NRepl/2);
   cat(i) = IC(2) - IC(1);
end
n = linspace(2^k1,2^k2);
p1 = cbr(1)*sqrt(n(1)); ybr = p1./sqrt(n);
p2 = cat(1)*sqrt(n(1)); yat = p2./sqrt(n);
semilogx(2.^(k1:k2),cbr,'r.',n,ybr,':',...
         2.^(k1:k2),cat,'r*',n,yat,'-.')
legend('int. emp. (MC brut)'  ,'p1/sqrt(NRepl)',...
       'int. emp. (MC v.ant.)','p2/sqrt(NRepl)')
xlabel('NRepl')
```

"crude" vs antithetic Monte Carlo

Confidence intervals for crude Monte Carlo and antithetic variables



Error proportional to $1/\sqrt{n}$!!!

$$\epsilon \propto \frac{1}{\sqrt{n}}$$

Other variance reduction techniques:

- Control variates
- Stratified sampling
- Importance sampling
- ...

cf. Jäckel, P., (2002): Monte Carlo methods in finance. Wiley.

Quasi-Monte Carlo

- Sequence of N "*random*" vectors $x^{(1)}, x^{(2)}, \ldots, x^{(N)} \in C^m = [0,1]^m$
- Sequence is uniformly distributed if number of points in any subspace $G \in C^m$ is proportional to volume of G
- Given $x = [x_1\, x_2 \ldots x_m]'$ and the rectangular subspace G_x

  $$G_x = [0, x_1) \times [0, x_2) \times \cdots \times [0, x_m)$$

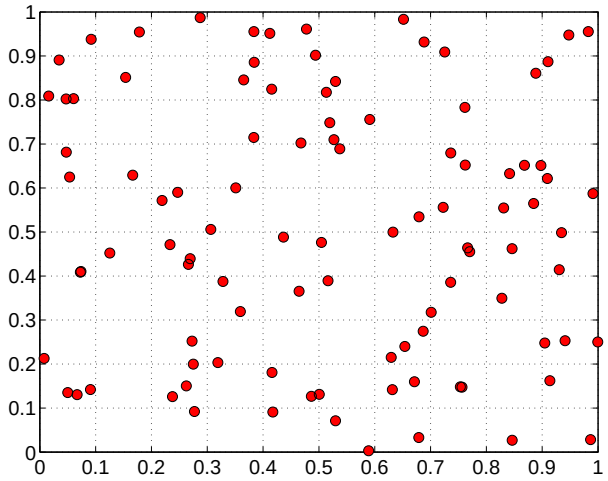
  Volume of G_x is given by $\mathrm{vol}(G_x) = x_1\, x_2 \cdots x_m$
- $S_{\mathrm{N}}(G)$ is number of point of sequence in $G \in C^m$
- Definition of *discrepancy* of a sequence:

  $$D(x^{(1)}, \ldots, x^{(N)}) = \sup_{x \in C^m} |S_{\mathrm{N}}(G_x) - N\, x_1 \cdots x_m|$$

- Small $D \rightarrow$ *Low-discrepancy* sequence, Quasi-Monte Carlo

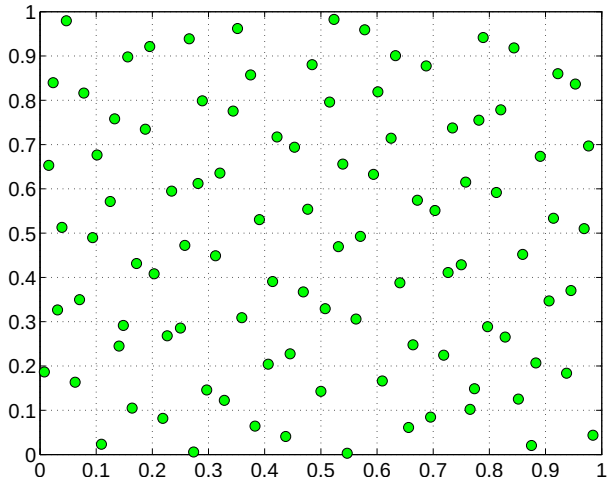
Quasi-Monte Carlo

```
rand('seed',0);
plot(rand(100,1),rand(100,1),'ro');  grid on
```



Halton sequences

```
plot(GetHalton(100,2),GetHalton(100,7),'ko','MarkerSize',8,'MarkerFaceColor','g');  grid on
```



Construction of Halton sequences

- Consider an integer n represented in a basis b with b a prime number

$$n = (d_m \cdots d_3\, d_2\, d_1\, d_0)_b$$

- We invert the order of the digits and add a radix such that the resulting number lies in $[0, 1[$

$$h = (0.\, d_0\, d_1\, d_2\, d_3 \cdots d_m)_b$$

Ex.: $n = 169$ and $b = 3$:

$$\begin{aligned} n &= 169 = (20021)_3 \\ h &= (0.12002)_3 \\ &= 1 \times 3^{-1} + 2 \times 3^{-2} + 0 \times 3^{-3} + 0 \times 3^{-4} + 2 \times 3^{-5} \\ &= 0.33333 + 0.22222 + 0 + 0 + 0.00823 \\ &= 0.56378 \end{aligned}$$

Construction of Halton sequences

The integer n can be written as

$$n = \sum_{k=0}^{m} d_k\, b^k$$

and the corresponding nth element of the Halton sequence is

$$h(n, b) = \sum_{k=0}^{m} d_k\, b^{-(k+1)}$$

Previous example:

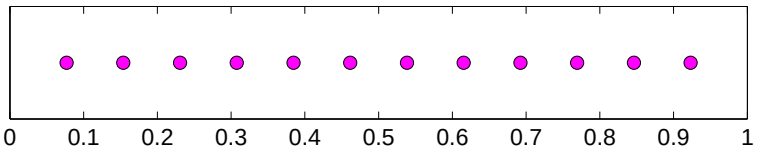
$$\begin{aligned} n &= 2 \times 3^4 + 0 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 1 \times 3^0 \\ h &= 2 \times 3^{-5} + 0 \times 3^{-4} + 0 \times 3^{-3} + 2 \times 3^{-2} + 1 \times 3^{-1} \end{aligned}$$

Construction of Halton sequences

```
function h = Halton(n,b)
n0 = n;
h  = 0;
c  = 1/b;
while n0 > 0
    n1 = floor(n0/b);
    d  = n0 - n1*b;
    h  = h + d*c;
    c  = c/b;
    n0 = n1;
end
```

```
for i=1:12
    H(i) = Halton(i,13);
end
```

```
H = 0.0769    0.1538    0.2308    0.3077
    0.3846    0.4615    0.5385    0.6154
    0.6923    0.7692    0.8462    0.9231
```



Construction of Halton sequences

```
function H = GetHalton(HowMany,Base)
H = zeros(HowMany,1);
NumBits = 1 + ceil(log(HowMany)/log(Base));
BaseVec = Base.^(-(1:NumBits));
WorkVec = zeros(1,NumBits);
for i = 1:HowMany
   j = 1; done = 0;
   while ~done
      WorkVec(j) = WorkVec(j) + 1;
      if WorkVec(j) < Base
         done = 1;
      else
         WorkVec(j) = 0;
         j = j + 1;
      end
   end
   H(i) = dot(WorkVec,BaseVec);
end
```


Comment

$b^{\texttt{Bits}} < n \rightarrow \texttt{Bits} = 1 + \lceil \log(n)/\log(b) \rceil$

Box-Muller transformation

Transformation generates $X, Y \sim N(0,1)$:

- Generate U_1 et U_2 uniform random variables
- $X = \sqrt{-2\log U_1}\cos(2\pi U_2)$
- $Y = \sqrt{-2\log U_1}\sin(2\pi U_2)$

Use Halton sequences, instead of uniform pseudo-random variables !!

Example for Halton sequences

```
function S = MBGHalton(S0,r,T,sigma,NRepl,B1,B2)
% Geometric brownian motion (simulation of NRepl paths)
% Halton sequences replace uniform variables
trend = (r - sigma^2/2)*T;
sigT = sigma * sqrt(T);
H1 = GetHalton(ceil(NRepl/2),B1);
H2 = GetHalton(ceil(NRepl/2),B2);
Vlog = sqrt(-2*log(H1));
Norm(1:2:NRepl-1,1) = Vlog .* cos(2*pi*H2);
Norm(2:2:NRepl  ,1) = Vlog .* sin(2*pi*H2);
S = exp( cumsum([log(S0)*ones(NRepl,1) trend + sigT*Norm],2) );
```

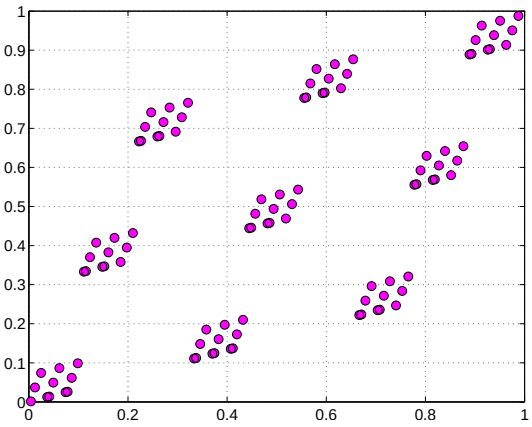

Example for Halton sequences

```
function P = QMCHalton(S0,E,r,T,sigma,NRepl,B1,B2)
% European call with quasi-Monte Carlo (Halton)
S = MBGHalton(S0,r,T,sigma,NRepl,B1,B2);
Payoff  = max( (S(:,end)-E), 0);
P = mean( exp(-r*T) * Payoff );

P = BScall(50,52,0.10,5/12,0.40)          P = 5.1911

P = QMCHalton(50,52,0.10,5/12,0.40,5000,2,7)
P =
    5.1970

P = MC01(50,52,0.10,5/12,0.40,1,5000)
P =
    5.2549
```

Example for Halton sequences

Example for bad choice of parameters for Halton sequence:

```
plot(GetHalton(100,3),GetHalton(100,9),'ro');
```



More quasi-Monte Carlo sequences:

- Sobol
- Fauré
- Niederreiter

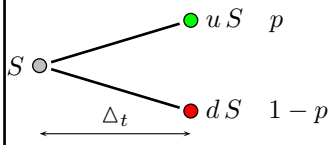
General idea

On approxime la distribution de probabilité du sousjacent par un arbre (les feuilles représentent les différents états)
On peut procéder de beaucoup de façons:

- fixer d'abord le nombre de branches (beaucoup de possibilités)
- faire correspondre les deux premiers moments de la distribution (lognormale)
  Donne lieu à un système d'équations avec 1 degré de liberté → il faut fixer quelque chose (plusieurs possibilités)

Binomial trees (2 branches)

Le prix S d'un sousjacent suit un processus binomial multiplicatif si dans un intervalle de temps $\triangle_t$ le prix monte avec une probabilité de p à $u\,S$ ou descend avec la probabilité $(1-p)$ à $d\,S$



On determine u, d et p de sorte à faire correspondre les deux premiers moments de la distribution de S.

- S_{t+1} verifies $S_t = e^{-r\triangle_t}\,E(S_{t+1}|S_t)$
- Expectation of S_{t+1}:

$$\begin{aligned} p\,u\,S_t + (1-p)\,d\,S_t &= e^{r\triangle_t} S_t \\ p\,u + (1-p)\,d &= e^{r\triangle_t} \end{aligned} \tag{14}$$

- Variance of S_{t+1} has to verify $\sigma^2 S_t^2\,\triangle_t$ from where:

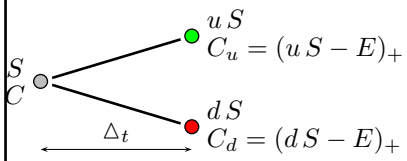
$$\begin{aligned} E(S_{t+1}^2|S_t) - \Big(E(S_{t+1}|S_t)\Big)^2 &= \sigma^2 S_t^2\,\triangle_t \\ p\,u^2 + (1-p)\,d^2 &= \sigma^2\,\triangle_t + e^{2r\triangle_t} \end{aligned} \tag{15}$$

Three unknowns in (14) and (15), we fix $d = 1/u$ and we get the solutions

$$u = e^{\sigma\sqrt{\triangle_t}} \qquad p = \frac{e^{r\triangle_t} - d}{u - d}$$

Binomial trees

Given a call which expires after $\triangle_t$. Possibles prices for underlying and call:



We construct a riskless portfolio with Δ units of S and a call on S. We want the portfolio value to be the same, regardless whether S goes down or up:

$$u\,\Delta S - C_u = d\,\Delta S - C_d$$

From there we get $\Delta S = \dfrac{C_u - C_d}{u - d}$

Since the portfolio is riskless we can write

$$u\,\Delta S - C_u = e^{r\Delta t}(\Delta S - C)$$

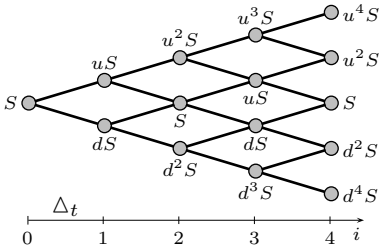
and substituting $\Delta S = (C_u - C_d)/(u - d)$ we get the value for the call

$$C = e^{-r\Delta t}\left(p\,C_u + (1-p)\,C_d\right)$$

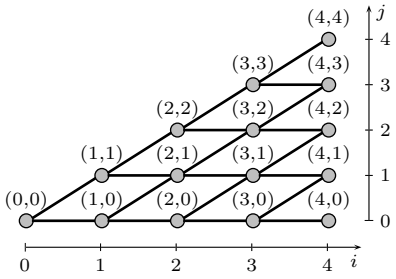
"Future payoff discounted under risk neutral probability"

Binomial trees

Extend tree over 4 periods:



Store tree in a 2-dimensional array:



Node (i, j) represents asset price after i periods and j up mouvements

$$S_i^j = S u^j d^{i-j}$$

Recall that $d = u^{-1}$!!

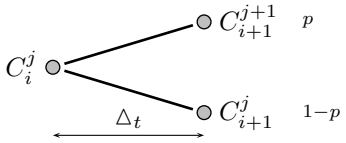
Binomial trees

At maturity the payoff is

$$C_M^j = (S_M^j - E)_+ \tag{16}$$

At any node in the tree the value of the option at time t corresponds to the discounted expected value of the call at time $t+1$

$$C_i^j = e^{-r\Delta t}\Big(p\, C_{i+1}^{j+1} + (1-p)\, C_{i+1}^j\Big) \tag{17}$$



By starting with (16) and using (17) we can compute the value of the option for any period $i = M-1, \ldots, 0$ in the tree.

This is implemented with the multiplicative binomial tree algorithm.

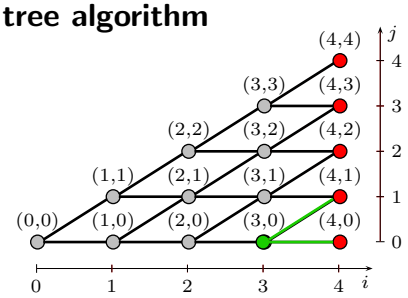
Multiplicative binomial tree algorithm

European call for S, E, r, σ, T and M time steps:

1: Initialize $\triangle_t = T/M$, $S_0^0 = S$, $v = e^{-r\triangle_t}$
2: Compute $u = e^{\sigma\sqrt{\triangle_t}}$, $d = 1/u$, $p = (e^{r\triangle_t} - d)/(u-d)$
3: $S_M^0 = S_0^0 d^M$
4: **for** $j = 1 : M$ **do**
5: $S_M^j = S_M^{j-1}\, u/d$ Initialize asset prices at maturity
6: **end for**
7: **for** $j = 0 : M$ **do**
8: $C_M^j = (S_M^j - E)_+$ Initialize option values at maturity
9: **end for**
10: **for** $i = M-1 : -1 : 0$ **do**
11: **for** $j = 0 : i$ **do**
12: $C_i^j = v\,\Big(p\, C_i^{j+1} + (1-p)\, C_i^j\Big)$ Step back through the tree
13: **end for**
14: **end for**
15: $Call = C_0^0$

Matlab implementation of binomial tree algorithm

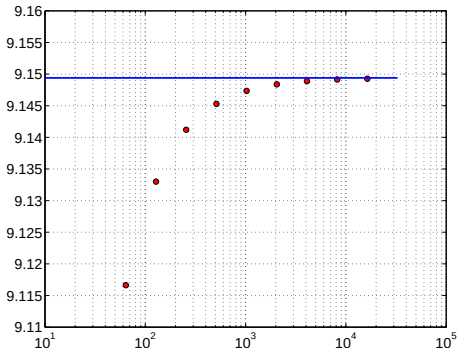
```
function C0 = MultBinTree(S0,E,r,T,sigma,M)
Multiplicative binomial tree
f7 = 1;  dt = T/M;  v = exp(-r*dt);
u = exp(sigma*sqrt(dt));   d = 1/u;
p = (exp(r*dt) - d) / (u - d);
Asset prices at maturity (period M)
S = zeros(M+1,1);  S(f7 + 0) = S0*d^M;
for j = 1:M
   S(f7 + j) = S(f7 + j - 1) * u / d;
end
C = max(S - E, 0); Option at maturity
for i = M-1:-1:0   Step back through the tree
  for j = 0:i
     C(f7+j) = v*(p*C(f7+j+1) + (1-p)*C(f7+j));
  end
end
C0 = C(f7 + 0);
```



Array C can be overwritten !!
We only store array C. Used portion of the array diminishes at each time step.

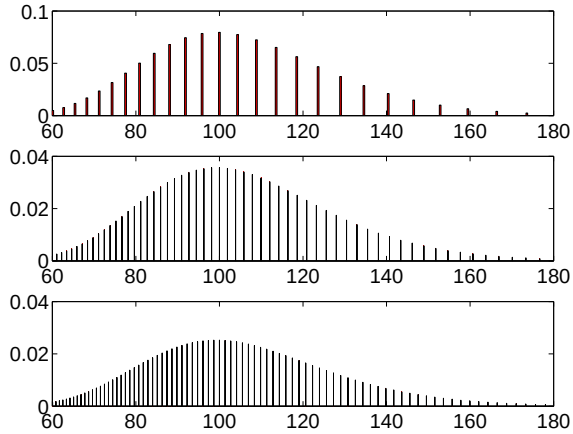
Example and convergence

```
S = 100; E = 100; r = 0.03; T = 6/12; sigma = 0.30;
CBS = BScall(S,E,r,T,sigma)
CBS = 9.1494

CBT = MultBinTree(S,E,r,T,sigma,100)
CBT = 9.1284
```



Example and convergence

Distribution of S for $M = 100, 500, 1000$

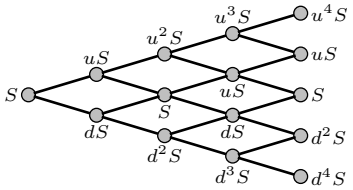


$S_{\max} > 82\,000$ for $M = 1000$!!!! Compare empiric CDF with theoretic lognormal (in particular in the tail, (Hull, 1993, p. 211)).

American put

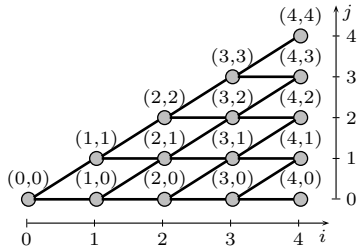
The binomial tree can be used to evaluate American options. To compute an American put we introduce the correction $C_i^j = \max(C_i^j, E - S_i^j)_+$

for $i = M - 1 : -1 : 0$ **do**
  for $j = 0 : i$ **do**
    $C_i^j = v\,(p\,C_i^{j+1} + (1-p)\,C_i^j)$
    $C_i^j = \max(C_i^j, E - S_i^j)$
  end for
end for



As for array C it is not necessary to store the tree in order to access the values of S. We simply update column i of S

for $i = M - 1 : -1 : 0$ **do**
  for $j = 0 : i$ **do**
    $C_j = v\,(p\,C_{j+1} + (1-p)\,C_j)$
    $S_j = S_j/d$
    $C_j = \max(C_j, E - S_j)$
  end for
end for



Mean–variance model (Markowitz (1952))

Principe: Un *portefeuille* est *optimal* s'il *maximise* le *rendement espéré pour* un niveau de *risque donné* ou s'il minimise le risque pour un rendement espéré donné

Solution: Combinaison *rendement–risque* la plus attractive

- Mesures pour rendement et risque:
  - *Rendement*: espérance des gains futurs
  - *Risque*: variance des gains futurs
- Repose sur les hypothèses de la normalité des rendements futurs ou de l'utilité quadratique

Formalisation et notations

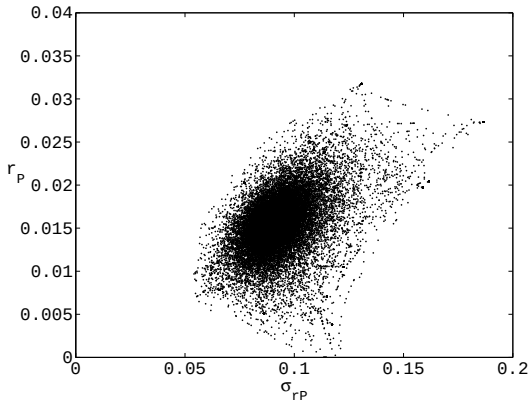
- n_{A} nombre de titres
- $R = [r_{tj}]$ rendement du titre j en t
- r_j rendement (espéré) du titre $j = 1, 2, \ldots, n_{\text{A}}$ (v.a.)
- x_j proportion du titre j dans portefeuille
- r_{P} rendement du portefeuille P $\quad r_{\text{P}} = \sum_{j=1}^{n_{\text{A}}} x_j r_j = x'r$
- $E(r_{\text{P}})$ espérance rendement portefeuille P $\quad E(r_{\text{P}}) = x'E(r)$
  `r = mean(R,1)` estimateur $E(r)$ (rendements futurs)
- $V(r_{\text{P}})$ variance rendement portefeuille P

$$\begin{aligned} V(r_{\text{P}}) &= E\left(x'(r - E(r_{\text{P}}))(r - E(r_{\text{P}}))'x\right) \\ &= x' \underbrace{E(r - E(r_{\text{P}}))(r - E(r_{\text{P}}))'}_{\Sigma_{\text{P}}} x \\ &= x'\Sigma_{\text{P}}x \end{aligned}$$

`Q = cov(R)` estimateur de Σ_{P}

Efficient frontier

Génération de $50\,000$ portefeuilles aléatoires
(poids et cardinalité générés aléatoirement):



- On observe une frontière
- A chaque niveau de risque correspond un rendement maximal (et vice-versa)

Code generating random portfolios

```
load SMI
% Extraire prix (SMI) 1er du mois
i = 0;
for year = 1997:1999
   for mois = 1:12
      numdate = fbusdate(year,mois);
      li = find(SMI.Dates==numdate);
      if ~isempty(li)
         i = i + 1;
         m1(i) = find(SMI.Dates==numdate);
      end
   end
end

n = size(SMI.Ph,2);
R = diff(log(SMI.Ph(m1,:)),1,1);
r = mean(R);
Q = cov(R);
```

Code generating random portfolios (cont'd)

```
N  = 50000;  RR = zeros(N,2);
h = waitbar(0,' Computing portfolios ...');
for i = 1:N
   nA = unidrnd(n);
   P  = randperm(n);
   P  = P(1:nA);     % Indices titres dans PF
   x  = rand(nA,1);
   x  = x ./ sum(x);
   RR(i,1) = r(P) * x;          % rP
   RR(i,2) = sqrt(x'*Q(P,P)*x); % sigma_rP
   waitbar(i/N,h)
end,  close(h);
plot(RR(:,2),RR(:,1),'k.')
```


A first problem

Chercher le portefeuille qui minimise le risque (variance)

$$\begin{aligned} \min_x \quad & x'Qx \\ \textstyle\sum_j x_j &= 1 \\ x_j &\geq 0 \quad j = 1, \ldots, n_{\text{A}} \end{aligned}$$

Il s'agit d'un programme quadratique
La fonction Matlab `quadprog` résoud le problème:

$$\begin{aligned} \min_x \quad & \tfrac{1}{2} x'Qx + f'x \\ & Ax \leq b \end{aligned}$$

Solving the QP with Matlab

```
x = quadprog(Q,f,A,b,Ae,be)
```

avec $f = [0 \ldots 0]$, $(\frac{1}{2}Q \equiv Q$ pour $\min)$ et A et b

$$\begin{bmatrix} -1 & & & & \\ & -1 & & & \\ & & -1 & & \\ & & & \ddots & \\ & & & & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \\ x_{n_A} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Lignes 1 à n_A: $(-x_j \leq 0) \equiv (x_j \geq 0)$

Ae et be définissent les égalités $A_{\text{e}}x = b_{\text{e}}$

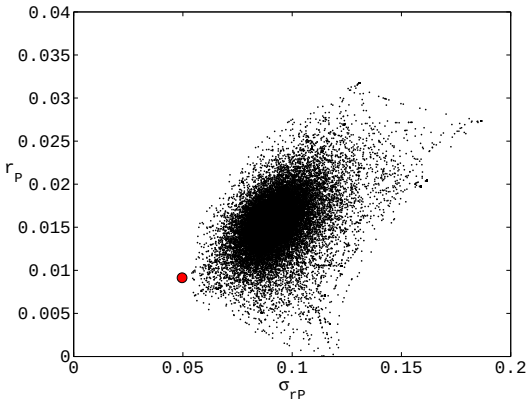
$A_{\text{e}} = [\ 1 \quad 1 \quad \ldots \quad 1]$, $b_{\text{e}} = 1 \quad \rightarrow \quad \sum_j x_j = 1$

$$\begin{aligned} \min_x \quad & x'Qx \\ \textstyle\sum_j x_j &= 1 \\ x_j &\geq 0 \quad j = 1, \ldots, n_{\text{A}} \end{aligned}$$

$$\begin{aligned} \min_x \quad & \tfrac{1}{2} x'Qx + f'x \\ & Ax \leq b \end{aligned}$$

Solving the QP with Matlab (cont'd)

```
A  = -eye(n);    b = zeros(n,1);
Ae = ones(1,n);  be = 1;  f  = zeros(1,n);
x  = quadprog(Q,f,A,b,Ae,be);
rP = r * x;
sigmaP = sqrt(x' * Q * x);
plot(sigmaP,rP,'ro')
xlim([0 0.20])
ylim([0 0.04])
```



A second problem

Find portfolio which minimises risk for a given return:

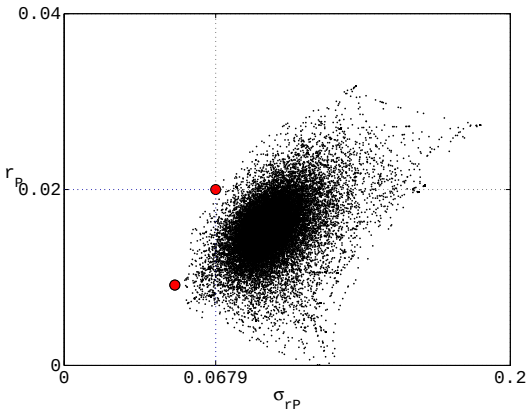
$$\begin{aligned} \min_x \; & x'Qx \\ \textstyle\sum_j x_j r_j \;&\geq\; r_{\text{P}} \qquad \textit{(additionnal constraint)} \\ \textstyle\sum_j x_j \;&=\; 1 \\ x_j \;&\geq\; 0 \qquad j = 1, \ldots, n_{\text{A}} \end{aligned}$$

A et b become:

$$\begin{bmatrix} -r_1 & -r_2 & -r_3 & \cdots & -r_{n_A} \\ -1 & & & & \\ & -1 & & & \\ & & -1 & & \\ & & & \ddots & \\ & & & & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \\ \\ x_{n_A} \end{bmatrix} \leq \begin{bmatrix} -r_{\text{P}} \\ 0 \\ \vdots \\ \\ 0 \end{bmatrix}$$

Solving the QP with Matlab (cont'd)

```
rP = 0.02;
f  = zeros(1,n);
A  = [-r; -eye(n)];
b  = [-rP zeros(1,n)]';
Ae = ones(1,n);   be = 1;
x  = quadprog(Q,f,A,b,Ae,be);
rP = r * x;
sigmaP = sqrt(x'* Q * x);
plot(sigmaP,rP,'ro')
```



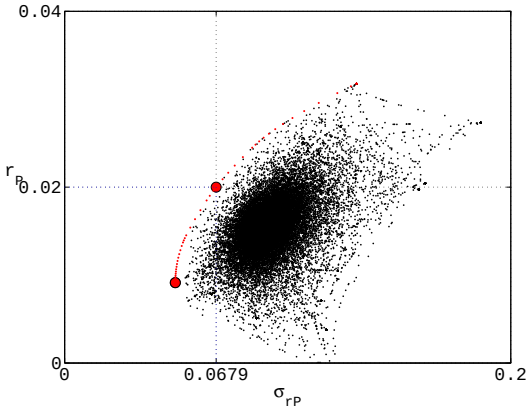
Computing the efficient frontier

$$\begin{aligned} \min_x \quad & \theta\, x'Qx \; - (1-\theta)\, x'r \\ & \textstyle\sum_j x_j = 1 \\ & x_j \geq 0 \qquad j = 1, \ldots, n_A \end{aligned}$$

Les solutions pour θ allant successivement de 0 à 1 produisent la frontière efficiente

Matlab code

```
f  = zeros(1,n);
A  = -eye(n);   b = zeros(n,1);
Ae = ones(1,n); be = 1;
npoints = 100;   % Sur circle unitaire
theta = sqrt(1-linspace(0.99,0.05,npoints).^2);
for i = 1:npoints
   H = theta(i)*Q;
   f = -(1-theta(i))*r;
   x = quadprog(H,f,A,b,Ae,be);
   plot(sqrt(x'*Q*x),r*x,'r.')
end
```



Finance toolbox

`frontcon`, `frontier`, etc.

Portfolio with risk-free asset (cash)

- r_0 taux de marché
- x_0 quantité (proportion) de cash dans PF

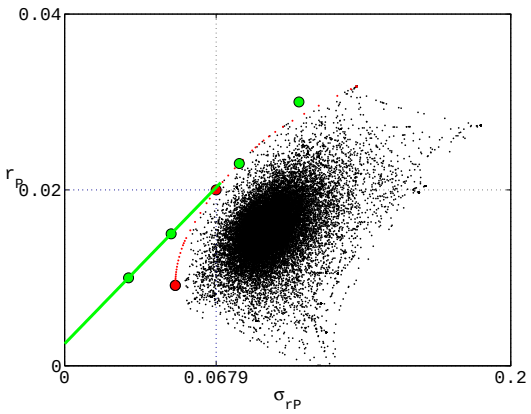
Return of portfolio:

$$\begin{aligned} r_{\mathrm{P}} &= r_0 x_0 + \sum_{i=1}^{n_{\mathrm{A}}} x_i\, r_i = r_0 \underbrace{\Big(1 - \sum_{i=1}^{n_{\mathrm{A}}} x_i\Big)}_{x_0} + \sum_{i=1}^{n_{\mathrm{A}}} x_i\, r_i \\ &= r_0 - r_0 \sum_{i=1}^{n_{\mathrm{A}}} x_i + \sum_{i=1}^{n_{\mathrm{A}}} x_i\, r_i \\ &= r_0 + \sum_{i=1}^{n_{\mathrm{A}}} x_i\,(r_i - r_0) \end{aligned}$$

$$\begin{aligned} \min_x\ & x'Qx \\ r_0 + \textstyle\sum_j x_j(r_j - r_0) &\geq r_{\mathrm{P}} \\ \textstyle\sum_j x_j &\leq 1 \\ x_j &\geq 0 \qquad j = 1, \ldots, n_{\mathrm{A}} \end{aligned}$$

Matlab code

```
r0 =  0.0025;   i = 0;
f = zeros(1,n); A  = [r0-r; -eye(n)];
for rP =  [0.010 0.015 0.023 0.030];
   i = i + 1;
   b = [r0 - rP zeros(1,n)]';
   x = quadprog(Q,f,A,b);
   x0(i) = 1 - sum(x);
   rP = x0(i)*r0 + r*x;
   sigmaP = sqrt(x'*Q*x);
   plot(sigmaP,rP,'ko')
end
x0 =
    0.6376   0.3960   0.0095 -0.3288
```



Holding size constraints

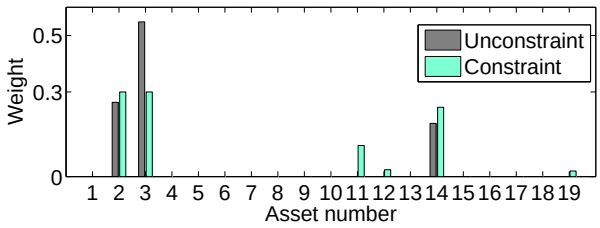
$$\begin{aligned} \min_x \; & x'Qx \\ \textstyle\sum_j x_j r_j \; & \geq \; r_{\text{P}} \\ \textstyle\sum_j x_j \; & = \; 1 \\ x_j^{\ell} \leq \; & x_j \; \leq x_j^{u} \qquad j \in P \end{aligned}$$

P: ensemble indices des titres du portefeuille

$$A = \left[-r_1 \; -r_2 \; -r_3 \; \cdots \; -r_{n_A} \right], \qquad b = \left[-r_{\text{P}} \right]$$

Holding size constraints (Matlab code)

```
rP = 0.02;
xL = 0.00*ones(1,n);
xU = 0.30*ones(1,n);
A  = [-r; ones(1,n)];   b = [-rP 1]';
Ae = ones(1,n); be = 1; f = zeros(1,n);
x  = quadprog(Q,f,A,b,Ae,be,xL,xU);
rP = r * x;
sigmaP = sqrt(x'*Q*x)

sigmaP =
    0.0750

xL = 0.00*ones(1,n);
xU = 1.00*ones(1,n);

sigmaP =
    0.0679
```

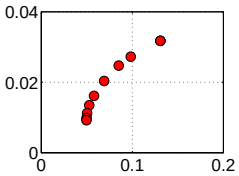


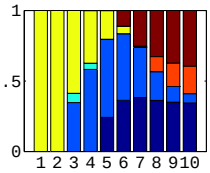
Profile of efficient portfolios

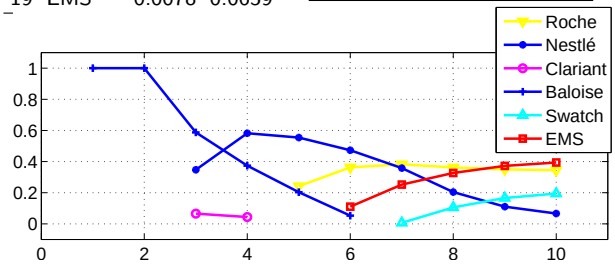
(10 portefeuilles efficients avec risque décroissant)

	Titre	r	σ
2	Roche	0.0110	0.0651
3	Nestlé	0.0203	0.0772
12	Clariant	0.0238	0.1138
14	Baloise	0.0318	0.1309
17	Swatch	0.0051	0.0860
19	EMS	0.0078	0.0659

P	Titres	P	Titres
1	14	6	2, 3, 14, 19
2	14	7	2, 3, 17, 19
3	3, 12, 14	8	2, 3, 17, 19
4	3, 12, 14	9	2, 3, 17, 19
5	2, 3, 14	10	2, 3, 17, 19







Optimisation de portefeuille

Données SMI (fichier `SMI.mat`):

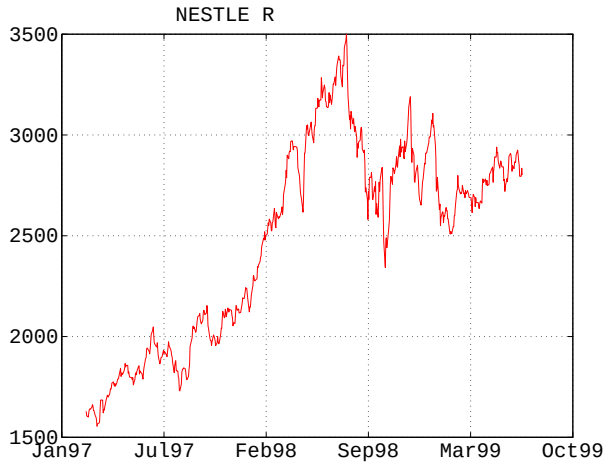
- `SMI.Ph` 611 prix journaliers de 19 titres
- `SMI.Dates` dates des observations

```
dates = ['28-Jun-1999'
         '29-Jun-1999'];
for i = 1:size(dates,1)
   SMI.Dates(i) = datenum(dates(i,:));                    % Codage
end
date = datestr(SMI.Dates(1),1);                           % Décodage
```

- `SMI.Noms` noms des titres

```
load SMI;
plot(SMI.Dates,SMI.Ph(:,3),'r')
dateaxis('x',12)
title([SMI.Noms(3,:)])
```


Asset price



Calcul des rendements

- discret:

$$r_t = \frac{p_t}{p_{t-1}} - 1$$

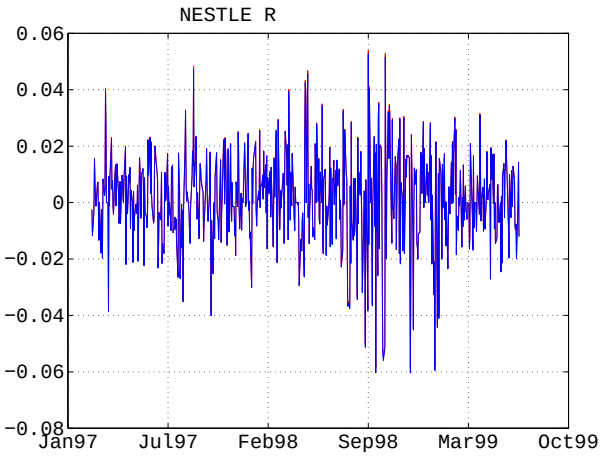
```
R = SMI.Ph(2:end,:)./SMI.Ph(1:end-1,:) - 1;
```

- continu:

$$r_t = \log(p_t) - \log(p_{t-1})$$

```
R = diff(log(SMI.Ph),1,1);
Dates = SMI.Dates(2:end);
plot(Dates,R(:,3),'b')
```

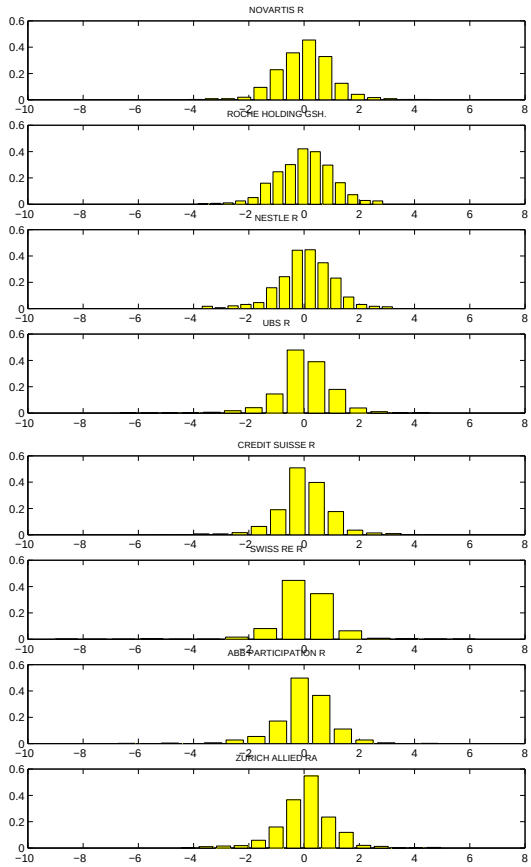

Calcul des rendements



Distribution normalisée des rendements journaliers

```
function plotR(SMI)
% Plot des rendements journaliers
R = diff(log(SMI.Ph),1,1);
sig = std(R);
nA = size(R,2); isp = 0; ifig = 0;
for i = 1:nA
   if rem(isp,4)==0, ifig=ifig+1;isp=0;figure(ifig),end
   isp = isp + 1; subplot(4,1,isp)
   epdf(R(:,i)/sig(i),15)
   title([deblank(SMI.Noms(i,:))],'FontSize',8)
   %ylabel([deblank(SMI.Noms(i,:))],'Rotation',90,'FontSize',8)
end

function epdf(x,nbins)
% Plots empirical probability distribution of x
n = length(x); minx = min(x); maxx = max(x);
d  = (maxx - minx) / nbins;       % largeur des classes
cc = minx + d/2 + d * (0:nbins-1);% centres des classes
ec = hist(x,cc);                  % effectifs des classes
fr = ec / n;                      % frequences relatives
hc = fr * (1/d);                  % hauteur rectangle pour loi tronquée (P = 1)
bar(cc,hc,0.8,'y')
xlim([-10 8]); ylim([0 0.6]);
```


Graphiques



Rendements

- Rendement moyen du titre j pour N observations:

$$r_j = \frac{1}{N} \sum_{i=1}^{N} R_{ij}$$

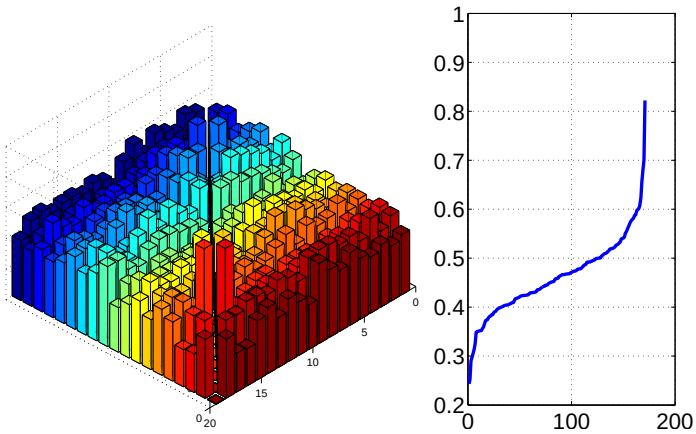
- Matrice variances et covariances des rendements:

```
r = mean(R,1);
Q = cov(R);
```

"Paysage des corrélations":

```
C = corrcoef(R)
bar3(C)
[C,Pval] = corrcoef(R);  % Matlab 6.xx
I = find(Pval > 0.05);
C(I) = 0;
C = C + eye(size(C));
```


Figures



References

Dennis, J. E. Jr. and R. B. Schnabel (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Series in Computational Mathematics. Prentice-Hall. Englewood Cliffs, NJ.

Hull, J.C. (1993). *Options, futures and other derivative securities*. Prentice-Hall.

Prisman, E. Z. (2000). *Pricing Derivative Securities: An Interactive Dynamic Environment with Maple V and Matlab*. Academic Press.