

# Algorithmique Numérique

## Module IS 104

Jérôme Baril  
baril@labri.fr

ENSEIRB-MATMECA

# Plan

- 1 Calcul numérique
- 2 Systèmes linéaires
- 3 Éléments propres

- 4 Equations non linéaires
- 5 Intégration numérique
- 6 Equations différentielles

# Fonctionnement du module

## Séance

- 1h20 cours intégré (salle TD)
- 1h20 TP (salle machine)
- **présence obligatoire !!!**

## Notation

- Contrôle continu
- validation  $> 9$

## Plan

- Méthode de calcul numérique
- Systèmes linéaires
- Éléments propres
- Systèmes non-linéaires
- Intégration numérique
- Equations différentielles

---

1. guide du module d'AlgoNum.

## Projets

- 6 projets portant sur les thèmes du cours
- 3 séances par projet
- projet par équipe de 5 personnes

## Modalités

- langage Python Numpy / Scipy
- sujets : [http ://www.labri.fr/ renault/](http://www.labri.fr/renault/)

# Python

## Les bibliothèques

```
import numpy as np  
np.sqrt(5) # 2.2360679774997898
```

## Les fonctions

```
def func ( x, y ) :  
    return (x+y)
```

## Instructions conditionnelles - Boucles

```
if cond1 :
```

```
    return instr1
```

```
elif cond2 :
```

```
    return instr2
```

```
else :
```

```
    return instr3
```

```
for x in array :
```

```
    return instr
```

```
while cond :
```

```
    return instr
```

# Python

## Structures de données

- Les tuples : (1,2,3,4)
  - non-modifiables
- Les listes : [1,2,3,4]
  - modifiables
- Les tableaux : `numpy.array([1,2,3,4])`
  - modifiables, opérations arithmétiques
- Les matrices : `numpy.matrix([[1,2],[3,4]])`
  - modifiables, opérations arithmétiques

## Les commandes graphiques

```
import matplotlib.pyplot as mp
import numpy as np
t = np.arange(0.0,2.0,0.01)
s = np.cos(2*np.pi*t)
mp.plot(t, s, linewidth=1.0)
```

## 1 Calcul numérique

- Présentation
- Représentation des nombres en machine
- Problèmes d'approximation
  - Opérations élémentaires
  - Propagations des erreurs de calcul
  - Approximations et calculs matriciels
- Principes généraux
- Calcul de la complexité
  - Racines de polynômes
  - Recherche d'extremum local



# Présentation

## Objectifs

- Comprendre le passage du continu au discret
- Identifier les erreurs d'approximation
- Donner un résultat pour une **précision** donnée

# Nombres entiers - Représentation

## Entiers non-signés

$$1 \text{ octet} = 8 \text{ bits} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \end{array} = ???$$

# Nombres entiers - Représentation

## Entiers non-signés

$$1 \text{ octet} = 8 \text{ bits} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array} = ???$$

7    6    5    4    3    2    1    0

$$\text{Calcul : } 2^0 + 2^2 + 2^3 = 13$$

# Nombres entiers - Représentation

## Entiers non-signés

$$1 \text{ octet} = 8 \text{ bits} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \hline 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \end{array} = ???$$

$$\text{Calcul : } 2^0 + 2^2 + 2^3 = 13$$

Propriétés :

- possibilité de représenter des entiers entre  $[0; 2^n - 1]$  sur  $n$  bits
- Opérations arithmétiques bit à bit

# Nombres entiers - Représentation

## Entiers signés

Complément à 2 :

- effectuer une opération miroir sur les bits
- ajouter 1 au nombre obtenu en binaire

Représentation binaire								non-signé	signé
0	0	0	0	1	1	0	1	13	13
1	1	1	1	0	0	1	0	242	128-114=-14
1	1	1	1	0	0	1	1	243	128-115=-13

# Nombres entiers - Représentation

## Entiers signés

Complément à 2 :

- effectuer une opération miroir sur les bits
- ajouter 1 au nombre obtenu en binaire

Propriétés :

- possibilité de représenter des entiers entre  $[-2^{n-1}; 2^{n-1} - 1]$  sur  $n$  bits
- Opérations identiques aux entiers non-signés

# Nombres entiers - Opérations

Représentation binaire								non-signé	signé	
+	0	0	0	0	1	1	0	1	13	13
	1	1	0	1	0	0	1	0	+ 210	- 46

# Nombres entiers - Opérations

Représentation binaire								signé	non-signé																	
+	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>								0	0	0	0	1	1	0	1	1	1	0	1	0	0	1	0	13 + 210	13 - 46
	0	0	0	0	1	1	0	1																		
1	1	0	1	0	0	1	0																			
=	<table border="1"><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>								1	1	0	1	1	1	1	1	223	-33								
1	1	0	1	1	1	1	1																			



# Nombres réels - Représentation

Nombre réel sur 32 bits (IEEE754)

signe	exposant	mantisse
1 bit	8 bits	23 bits

Nombre réel sur 64 bits (IEEE754)

signe	exposant	mantisse
1 bit	11 bits	52 bits

## Calcul du nombre réel

$signe * 2^{exposant} * mantisse$  avec,

- $signe = -1|1$  ;
- $2^{exposant} = 2^{127-exposant} (32bits)$  ou  $2^{1023-exposant} (64bits)$
- $mantisse$  : entier non-signé avec *hidden bit* (i.e. premier bit à 1)

# Nombres réels - Codage

Codage de  $-118,625$  sur 32 bits (IEEE754) :

- ❶  $signe = 1$
- ❷ écrire le nombre signé en binaire :  $1110110.101$
- ❸ décalage pour trouver l'exposant :  $1.110110101 * 2^6$
- ❹ décalage pour coder la mantisse :  $111011010100000000000000$
- ❺ exposant :  $6 + 127 = 133$ , binaire :  $10000101$

Résultat :  $11000010111011010100000000000000$

# Nombres réels - Codage

Codage de  $-118,625$  sur 32 bits (IEEE754) :

- ❶  $signe = 1$
- ❷ écrire le nombre signé en binaire :  $1110110.101$
- ❸ décalage pour trouver l'exposant :  $1.110110101 * 2^6$
- ❹ décalage pour coder la mantisse :  $111011010100000000000000$
- ❺ exposant :  $6 + 127 = 133$ , binaire :  $10000101$

Résultat :  $11000010111011010100000000000000$

↪ coder  $19.8625$  sur 32 bits en norme IEEE754

# Nombres réels - Limitations de la norme IEEE754

## Exemple

- coder 0.1 sur 32 bits en norme IEEE754

# Nombres réels - Limitations de la norme IEEE754

## Exemple

- coder 0.1 sur 32 bits en norme IEEE754

$$2^{-4} \leq 0.1 \leq 2^{-3}$$

Il n'existe pas de représentation exacte de 0.1 !

# Nombres réels - Limitations de la norme IEEE754

## Calcul de l'erreur d'approximation

Représentation de  $x$  :  $x = 0. x_1 x_2 \dots x_i x_{i+1} \dots$

$$rep(x) = \begin{cases} 0. x_1 x_2 \dots x_p & \text{si } x_{p+1} = 0 \\ 0. x_1 x_2 \dots x_p + 2^{-p} & \text{si } x_{p+1} = 1 \end{cases}$$

$$\text{Erreur relative : } \epsilon = \left| \frac{x - rep(x)}{x} \right| \leq \frac{2^{-(p+1)}}{2^{-1}} = 2^{-p}$$

# Nombres réels - Limitations de la norme IEEE754

## Calcul de l'erreur d'approximation

Représentation de  $x$  :  $x = 0. x_1 x_2 \dots x_i x_{i+1} \dots$

$$rep(x) = \begin{cases} 0. x_1 x_2 \dots x_p & \text{si } x_{p+1} = 0 \\ 0. x_1 x_2 \dots x_p + 2^{-p} & \text{si } x_{p+1} = 1 \end{cases}$$

$$\text{Erreur relative : } \epsilon = \left| \frac{x - rep(x)}{x} \right| \leq \frac{2^{-(p+1)}}{2^{-1}} = 2^{-p}$$

## Généralisation pour une base $b$

$$\text{Erreur relative : } \epsilon = \left| \frac{x - rep(x)}{x} \right| \leq \frac{b^{-p}/2}{b^{-1}} = \frac{b^{-p+1}}{2}$$

# Résumé

## Nombres entiers

- Représentation canonique
- Possibilité de représenter  $p$  sur  $n$  bits :
  - entiers non signés :  $p \in [0; 2^n - 1]$
  - entiers signés :  $p \in [-2^{n-1}; 2^{n-1} - 1]$

## Nombres réels

- Non-représentabilité de tous les nombres réels
- Erreur relative  $\epsilon$  bornée sur  $p$  bits :
  - en base 2 :  $\epsilon = 2^{-p}$
  - en base  $b$  :  $\epsilon = \frac{b^{-p+1}}{2}$



# Problèmes envisagés

- Approximation des opérations élémentaires
- Propagation des erreurs de calcul
- Approximations et calculs matriciels

# Approximation des opérations élémentaires

« *Le résultat des opérations élémentaires est au mieux une approximation du résultat réel* »

$$\begin{aligned}x \underset{\text{machine}}{+} y &= rp(x + y) = (x + y)(1 + \theta_1) \\x \underset{\text{machine}}{*} y &= rp(x * y) = (x * y)(1 + \theta_2)\end{aligned}$$

avec  $\theta_i \leq \epsilon$

# Approximation des opérations élémentaires

Représentation en base 10 avec 4 décimales :

$$0.1056 \underset{\text{réel}}{+} 0.4985 \cdot 10^{-5} = 0.1056 \quad \text{Erreur rel. : } 4.7 \cdot 10^{-4}$$

$$0.1201 \underset{\text{réel}}{+} 0.1495 \cdot 10^{-1} = 0.1351 \quad \text{Erreur rel. : } 3.7 \cdot 10^{-4}$$

$$0.3456 \underset{\text{réel}}{\times} 0.2921 = 0.1009 \quad \text{Erreur rel. : } 4.9 \cdot 10^{-4}$$

Erreur maximale :  $5 \cdot 10^{-4}$

# Approximation des opérations élémentaires

Où apparaît cette approximation ?

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 + \quad \begin{array}{|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 & 1 \\ \hline \end{array} \\
 \hline
 = \quad \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline \end{array} \\
 \begin{array}{c} \text{Résultat} \quad \text{Arrondi} \end{array}
 \end{array}$$

$x = 0.x_1 \dots x_p \cdot 2^e$   
 $y = 0.y_1 \dots y_p \cdot 2^f$   
 $x + y$

# Approximation des opérations élémentaires

## Non-associativité de l'addition

$$a = 0.233712 * 10^{-6}$$

$$b = 0.336784 * 10^0$$

$$c = -0.336778 * 10^0$$

$$(a + b) + c = 0.60000 * 10^{-5} \quad (1)$$

$$a + (b + c) = 0.62337 * 10^{-5} \quad (2)$$

$$(1) \text{ erreur relative} = 3.7 * 10^{-2}$$

$$(2) \text{ erreur relative} = 1.9 * 10^{-6}$$

# Propagation des erreurs de calcul

## Environnement de calcul

Tout nombre  $x$  est connu avec une certaine précision  $\Delta x$ .

Exceptions :

- nombres entiers
- nombres réels représentables de manière exacte

## Propagations des erreurs

Propagation des erreurs lors de chaque opération.

- $\Delta x$  peut prendre des valeurs arbitraires
- Comment quantifier les erreurs
  - pour l'addition et la multiplication ?
  - lors de calculs matriciels ?

# Propagation des erreurs de calcul - Addition et Multiplication

## Sommes

$$(x + \Delta x) +_{\text{machine}} (y + \Delta y) = \dots$$

# Propagation des erreurs de calcul - Addition et Multiplication

## Sommes

$$(x + \Delta x) +_{\text{machine}} (y + \Delta y) = (x + y) + (\Delta x + \Delta y + \theta(x + y))$$

avec  $|\theta| \leq \epsilon$



# Propagation des erreurs de calcul - Addition et Multiplication

## Sommes

$$(x + \Delta x) +_{\text{machine}} (y + \Delta y) = (x + y) + (\Delta x + \Delta y + \theta(x + y))$$

avec  $|\theta| \leq \epsilon$

## Erreurs

Erreur globale :  $\Delta(x + y) = (\Delta x + \Delta y + \theta(x + y))$

Erreur relative :  $\frac{\Delta(x+y)}{(x+y)} = \frac{(\Delta x + \Delta y)}{(x+y)} + \theta$

- les erreurs **globales** s'ajoutent.
- erreur relative incontrôlable ! ( $x \approx -y$ )

# Propagation des erreurs de calcul - Addition et Multiplication

## Multiplications

$$(x + \Delta x) \underset{\text{machine}}{*} (y + \Delta y) = \dots$$

# Propagation des erreurs de calcul - Addition et Multiplication

## Multiplications

$$(x + \Delta x) \underset{\text{machine}}{*} (y + \Delta y) = (x * y) + (y\Delta x + x\Delta y + \theta(x * y))$$

avec  $|\theta| \leq \epsilon$

# Propagation des erreurs de calcul - Addition et Multiplication

## Multiplications

$$(x + \Delta x) \underset{\text{machine}}{*} (y + \Delta y) = (x * y) + (y\Delta x + x\Delta y + \theta(x * y))$$

avec  $|\theta| \leq \epsilon$

## Erreurs

Erreur globale :  $\Delta(x * y) = (y\Delta x + x\Delta y + \theta(x * y))$

Erreur relative :  $\frac{\Delta(x*y)}{(x*y)} = \frac{\Delta x}{x} + \frac{\Delta y}{y} + \theta$

- les erreurs **relatives** s'ajoutent.
- erreur mieux contrôlable que pour l'addition :
  - bornée par  $\Delta x$ ,  $\Delta y$  et  $\epsilon$
  - $\forall x, y$  ou  $xy$

# Approximations et calculs matriciels

$$\underbrace{\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}}_X = \underbrace{\begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}}_Y \quad \text{de solution} \quad \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}_{A^{-1}Y}$$

$$Y \rightarrow \underbrace{\begin{bmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{bmatrix}}_{\tilde{Y}} \quad \text{de solution} \quad \underbrace{\begin{bmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -1.1 \end{bmatrix}}_{A^{-1}\tilde{Y}}$$

$$A \rightarrow \underbrace{\begin{bmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{bmatrix}}_{\tilde{A}} \quad \text{de solution} \quad \underbrace{\begin{bmatrix} -81 \\ 137 \\ -34 \\ 22 \end{bmatrix}}_{\tilde{A}^{-1}Y}$$

# Approximations et calculs matriciels

## Instabilité de la multiplication matricielle

Soient une matrice  $A$  et une erreur  $\delta A$  sur la matrice, ainsi qu'un vecteur  $Y$  et une erreur  $\delta Y$  sur ce vecteur. L'erreur du produit  $AY = X$  est majorée par :

$$\frac{\|\delta X\|}{\|X\|} \leq \text{cond}(A) * \left( \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta Y\|}{\|Y\|} \right) * (1 + \mathcal{O}(\|\delta A\|))$$

(Proposition - Ciarlet 1982)

# Approximations et calculs matriciels

## Instabilité de la multiplication matricielle

Soient une matrice  $A$  et une erreur  $\delta A$  sur la matrice, ainsi qu'un vecteur  $Y$  et une erreur  $\delta Y$  sur ce vecteur. L'erreur du produit  $AY = X$  est majorée par :

$$\frac{\|\delta X\|}{\|X\|} \leq \text{cond}(A) * \left( \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta Y\|}{\|Y\|} \right) * (1 + \mathcal{O}(\|\delta A\|))$$

(Proposition - Ciarlet 1982)

### Exemple

Dans l'exemple précédent,  $\text{cond}(A) \approx 3000$  !

# Approximations et calculs matriciels

## Conditionnement

Soit  $\|\cdot\|$  une norme vectorielle (par exemple la norme euclidienne). Le *conditionnement* d'une matrice  $A$  lié à cette norme est défini par :

$$\text{cond}(A) = \|A\| * \|A^{-1}\|$$

où  $\|A\|$  (reps.  $\|A^{-1}\|$ ) est la norme matricielle.



# Approximations et calculs matriciels

## Conditionnement d'une matrice

Propriétés :

- $\text{cond}(A) = \text{cond}(A^{-1})$
- $1 \leq \text{cond}(A)$
- $1 = \text{cond}(A)$  si  $A$  est orthogonale ( ${}^tAA = I$ )

Calcul de  $\text{cond}(A)$  en norme euclidienne :

- 1 Calcul des valeurs propres  $\lambda_i$  ;
- 2 Trier les valeurs propres par ordre croissant  $|\lambda_0| < |\lambda_1| < \dots < |\lambda_n|$

$$\text{cond}_2(A) = \frac{|\lambda_n|}{|\lambda_1|}$$

# Principes généraux

**Comment éviter d'augmenter l'erreur relative sur les variables ?**

- Solution : déterminer des règles de bonne conduite.

# Principes généraux

## Règle 1

*Eviter de soustraire deux quantités équivalentes.*

## Exemples

- Racines de polynôme :  $ax^2 + bx + c = 0$
- Calcul de fonctions dérivées
- Calcul de  $\pi$

# Principes généraux

## Règle 2

*Lors de la sommation de plusieurs termes de grandeur non équivalente, sommer d'abord les termes de plus petite valeur absolue avant les plus grands.*

### Exemple : Somme de termes décroissants

$$S_n = \sum_{k=1}^n \frac{1}{k}$$

# Principes généraux

## Exemple : Somme de termes décroissants

$$S_n = \sum_{k=1}^n \frac{1}{k}$$

$n$	Algorithme $N$		Algorithme $O$		Exact
	(4 déc.)	6 déc.	4 déc.	6 déc.	
10	2.929	2.92897	2.929	2.92897	2.92896825
50	4.497	4.49921	4.500	4.49921	4.49920534
100	5.186	5.18739	5.187	5.18737	5.18737752
1000	7.448	7.48545	7.485	7.48546	7.48547086
5000	8.448	9.09437	9.274	9.09448	9.09450885
10000	8.448	9.78716	9.449	9.78753	9.78760604
100000	8.448	10.7625	9.449	12.0902	12.09014613

# Principes généraux

## Règle 3

*Dans un calcul demandant l'inverse d'un nombre  $x$ , faire son possible pour que la norme de  $x$  soit la plus grande possible.*

## Exemples

- Inversion de matrice
- Soit une suite  $U_n$  telle que,
  - $U_0 = 2$
  - $U_{n+1} = \log|U_n|$

# Principes généraux

## Exemples

$n$	Calcul de $u_n$			
	4 déc.	7 déc.	10 déc.	100 déc.
1	0.6931	0.6931472	0.6931471806	0.6931471806
5	5.810	5.595352	5.595484996	5.5954851833
10	0.5773	0.7040199	0.7039347036	0.7039345855
15	0.4033	1.124368	1.126695313	1.1266985498
20	0.1629	1.305218	1.266159318	1.2661060564
24	0.6591	1.273670	1.001307399	1.0009714391
25	0.4169	0.2419025	0.001306545098	0.0009709675
26	0.8749	1.419221	6.640368955	6.9372175459
30	0.3572	3.029780	0.8006809852	0.8822116404

- problème :  $n > 24$

# Principes généraux

## Règle 4

*Lors d'un calcul itératif, éviter les opérations connues pour augmenter l'erreur relative sur les variables.*



# Astuce du jour

## Python

```
n=8  
print (numpy.array([numpy.arange(1,n)]*n))
```

## Result

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ 1 & 2 & \cdots & n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & \cdots & n \end{pmatrix}$$

# Calcul de la complexité

## Comment évaluer la complexité d'un algorithme ?

### Machine de Turing - Théorique

Évaluation de la complexité basée sur :

- la taille mémoire utilisée,
- le nombre de pas réalisés.

### Programmation - Pratique

Comment mesurer la complexité des algos dans un langage donné ?

# Calcul de la complexité

## Evaluation sur les opérations de bases

Types d'opérations élémentaires :

- opérations de branchement (if, for, while, ...)
- opérations de gestion mémoire (malloc, free, affectation, ...)
- opérations arithmétiques (+, -, ...)

# Calcul de la complexité

## Difficultés théoriques

- déterminer l'opération la plus significative,
- pas toujours un nombre d'instructions fini.

# Calcul de la complexité

## Difficultés théoriques

- déterminer l'opération la plus significative,
- pas toujours un nombre d'instructions fini.

## Problème de terminaison

**Comment évaluer la complexité d'un algo qui ne se termine pas en un nombre fini d'étapes ?**

Les critères à prendre en compte seront donc :

- une optimisation locale et
- la mesure de convergence

de plus faible complexité possible pour une précision donnée.

# Calcul de la complexité

## Complexité des opérations arithmétiques

Chaque opération n'a pas la même complexité :

- ALU Vs FPU
- Allocation mémoire : *load store unit*

# Calcul de la complexité

## Complexité des opérations arithmétiques

Chaque opération n'a pas la même complexité :

- ALU Vs FPU
- Allocation mémoire : *load store unit*

## Architecture matérielle

L'efficacité d'un algorithme dépend de l'architecture matérielle sur laquelle il est exécuté :

- x86 Vs SPARC Vs ...
- 32bits Vs 64bits
- SSE{1,2,3,4} + commandes spécifiques
- CPU Vs GPU

## Mesures de complexité - Méthodes usuelles

Soit un polynôme  $P$  de degré  $n$ . Evaluer la complexité du calcul de  $P(x)$  tel que :

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

Nombre d'opérations :



## Mesures de complexité - Méthodes usuelles

Soit un polynôme  $P$  de degré  $n$ . Evaluer la complexité du calcul de  $P(x)$  tel que :

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

Nombre d'opérations :

- $n - 1$  additions
- $(\sum_{i=1}^n i) + (n - 1) = \frac{n(n+1)}{2} + (n - 1)$  multiplications

**Complexité totale :  $\mathcal{O}(n^2)$**

## Mesures de complexité - Méthodes usuelles

Soit un polynôme  $P$  de degré  $n$ . Evaluer la complexité du calcul de  $P(x)$  tel que :

$$P(x) = a_0 + x * (a_1 + x * (a_2 + \dots + a_n x))$$

(Schéma de Horner)

Nombre d'opérations :

## Mesures de complexité - Méthodes usuelles

Soit un polynôme  $P$  de degré  $n$ . Evaluer la complexité du calcul de  $P(x)$  tel que :

$$P(x) = a_0 + x * (a_1 + x * (a_2 + \dots + a_n x))$$

(Schéma de Horner)

Nombre d'opérations :

- $n$  additions
- $n$  multiplications

**Complexité totale :  $\mathcal{O}(n)$**

## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et un vecteur  $V$  de dimension  $n$ .  
Evaluer la complexité du produit  $A \times V$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et un vecteur  $V$  de dimension  $n$ .  
Evaluer la complexité du produit  $A \times V$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Calcul de la  $i$ -ème coordonnées du vecteur solution :

- ??? multiplications
- ??? additions

## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et un vecteur  $V$  de dimension  $n$ .  
Evaluer la complexité du produit  $A \times V$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Calcul de la  $i$ -ème coordonnées du vecteur solution :

- $n$  multiplications
- $n - 1$  additions

**Complexité totale :  $\mathcal{O}(mn)$**

## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et une matrice  $B$  de dimension  $n \times p$ .  
Evaluer la complexité du produit  $A \times B$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{bmatrix}$$

## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et une matrice  $B$  de dimension  $n \times p$ .  
Evaluer la complexité du produit  $A \times B$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{bmatrix}$$

Complexité :

- produit matrice-vecteur :  $\mathcal{O}(mn)$



## Mesures de complexité - Méthodes usuelles

Soient une matrice  $A$  de taille  $m \times n$  et une matrice  $B$  de dimension  $n \times p$ .  
Evaluer la complexité du produit  $A \times B$  :

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} * \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \ddots & \cdots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{bmatrix}$$

Complexité :

- produit matrice-vecteur :  $\mathcal{O}(mn)$
- **produit matrice-matrice** :  $\mathcal{O}(mnp)$

# Mesures de complexité - Méthodes usuelles

Soient une matrice triangulaire supérieure  $T$  inversible de taille  $n \times n$  et  $b$  un vecteur de dimension  $n$ . Evaluer la complexité du *Pivot de Gauss* dans le problème de résolution d'un système triangulaire tel que :

$$\begin{bmatrix} \begin{array}{c|c} \text{triangle} & \\ \hline 0 & \end{array} \\ T \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

où,

$$x_i = \frac{1}{t_{i,i}} \left( b_i - \sum_{j=i+1}^n t_{i,j} x_j \right)$$

# Mesures de complexité - Méthodes usuelles

$$\begin{bmatrix} \begin{array}{c|c} & \\ \hline 0 & \end{array} & \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$T \quad \cdot x = b$

où,

$$x_i = \frac{1}{t_{i,i}} \left( b_i - \sum_{j=i+1}^n t_{i,j} x_j \right)$$

Nombre d'opérations :

# Mesures de complexité - Méthodes usuelles

$$\begin{bmatrix} \begin{array}{c|c} & \\ \hline 0 & \end{array} & \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \\ T \cdot x & = b \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

où,

$$x_i = \frac{1}{t_{i,i}} \left( b_i - \sum_{j=i+1}^n t_{i,j} x_j \right)$$

Nombre d'opérations :

- $n(n+1)/2 + n$  multiplications
- $n$  divisions
- $n(n+1)/2$  additions
- $n$  soustractions

**Complexité totale :  $\mathcal{O}(n^2)$**

- 2 **Systèmes linéaires**
  - Présentation
  - Méthodes directes
  - Méthodes itératives
  - Applications
  - Bonnes pratiques

# Présentation du problème

Résolution de systèmes linéaires, équation à résoudre :

$$A.x = b, \quad \text{où } x \in \mathbb{R}^n$$

avec,

- $A$ , matrice  $n \times n$
- $b$ , vecteur de dimension  $n$

*nb* :  $A$  inversible donc unicité de la solution

# Présentation du problème

Soit le système :  $A.x = b$

## Solution naïve (1/2)

Inverser la matrice  $A$  telle que  $A^{-1}.b = x$ .

## Problème d'inversion

Equivalent à résoudre le système :

$$A.x = e_i$$

où,  $e_i$  est le  $i$ -ème vecteur de base de l'espace vectoriel.

⇒ Résolution de  $n$  systèmes linéaires.

⇒ Complexité++ !

# Formules de Cramer

## Exemple

cas simple :  $n = 2$

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases}$$

Une solution existe ssi :

$$\Delta = \begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix} = a_{1,1}a_{2,2} - a_{2,1}a_{1,2} \neq 0$$

et,

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{1,2} \\ b_2 & a_{2,2} \end{vmatrix}}{\Delta} \quad x_2 = \frac{\begin{vmatrix} a_{1,1} & b_1 \\ a_{2,1} & b_2 \end{vmatrix}}{\Delta}$$



# Présentation du problème

Soit le système :  $A.x = b$

## Solution naïve (2/2)

Utiliser les formules de Cramer.

## Généralisation

$$x_k = \frac{\det(B_k)}{\det(A)} \quad \text{ou} \quad B_k = \begin{bmatrix} a_{1,1} & \dots & a_{1,k-1} & b_1 & a_{1,k+1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,k-1} & b_2 & a_{2,k+1} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,k-1} & b_n & a_{n,k+1} & \dots & a_{n,n} \end{bmatrix}$$

## Problèmes

- Calcul de  $(n + 1)$  déterminants
- Complexité de l'ordre de  $\mathcal{O}(n!)$

# Méthodes directes

Soit le système :  $A.x = b$

## Principe

*Transformer* le système de manière à se ramener au cas où la matrice est triangulaire. On recherche la matrice  $M$  telle que :

$$M.A.x = M.b$$

où  $M.A$  est une matrice triangulaire (supérieure ou inférieure).

## Algorithmes

- Gauss et Gauss-Jordan
- Factorisation PLU
- Factorisation QR

# Méthode de Gauss

## Passage en système triangulaire

- 1 échanges de lignes (et/ou de colonnes)
- 2 eliminations de variables

## Remontée

$$\left[ \begin{array}{c|c} \text{triangle} & \\ \hline 0 & \end{array} \right] \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$T \cdot x = b$   
où,

$$x_i = \frac{1}{t_{i,i}} \left( b_i - \sum_{j=i+1}^n t_{i,j} x_j \right)$$

# Méthode de Gauss - Echange de lignes

Matrice de permutation :

$$P_1 = \begin{bmatrix} 0 & \dots & \dots & 1 & \dots & \dots & \dots \\ \dots & 1 & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & 1 & \dots & \dots & \dots & \dots \\ 1 & \dots & \dots & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & 1 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 \end{bmatrix}$$

Nouveau système :

$$\begin{cases} A_1 = P_1 \cdot A \\ b_1 = P_1 \cdot b \end{cases} \quad A_1 \cdot x = b_1$$

# Méthode de Gauss - Elimination de variable

Matrice d'élimination :

$$E_1 = \begin{bmatrix} 1 & \dots & \dots & \dots & \dots & \dots & \dots \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & \dots & \dots & \dots & \dots & \dots \\ \vdots & \dots & 1 & \dots & \dots & \dots & \dots \\ -\frac{a_{k,1}}{a_{1,1}} & \dots & \dots & 1 & \dots & \dots & \dots \\ \vdots & \dots & \dots & \dots & 1 & \dots & \dots \\ \vdots & \dots & \dots & \dots & \dots & 1 & \dots \\ -\frac{a_{n,1}}{a_{1,1}} & \dots & \dots & \dots & \dots & \dots & 1 \end{bmatrix}, \text{ par ligne } L_i - \frac{a_{i,1}}{a_{1,1}} L_1$$

Nouveau système :

$$\begin{cases} A_2 = E_1 \cdot A_1 \\ b_2 = E_1 \cdot b_1 \end{cases} \quad A_2 \cdot x = b_2$$

## Méthode de Gauss - Récurrence

$$A_k = \begin{bmatrix} \text{triangle} & \text{rectangle} \\ & \tilde{A} \end{bmatrix} \quad b_k = \begin{bmatrix} \text{rectangle} \\ \tilde{b} \end{bmatrix}$$

$$\begin{cases} A_{k+1} = E_{k+1} \cdot P_{k+1} \cdot A_k \\ b_{k+1} = E_{k+1} \cdot P_{k+1} \cdot b_k \end{cases}$$

Réduction de dimensionnalité du problème :  $\tilde{A} \cdot \tilde{x} = \tilde{b}$

Convergence :  $\text{Dimension}(\tilde{A}) = 1 \times 1$ , au bout de  $n - 1$  itérations.

## Méthode de Gauss - Récurrence

$$A_k = \begin{bmatrix} \text{triangle} & \text{rectangle} \\ & \tilde{A} \end{bmatrix} \quad b_k = \begin{bmatrix} \text{rectangle} \\ \tilde{b} \end{bmatrix}$$

$$\begin{cases} A_{k+1} = E_{k+1} \cdot P_{k+1} \cdot A_k \\ b_{k+1} = E_{k+1} \cdot P_{k+1} \cdot b_k \end{cases}$$

$A_n = M.A$ , avec  $M = E_{n-1}P_{n-1} \dots E_1P_1$ , est triangulaire supérieure.

# Méthode de Gauss

## Remarque 1

Par définition, on a :

- $\det(E_k) = 1$
- $\det(P_k) = \pm 1$  (dépend du nombre de permutations)

Donc,  $\det(A) = \pm \det(A_n)$ . Or,  $A_n$  est triangulaire supérieure donc,

$$\det(A_n) = \sum_{i=1}^n a_{i,i}$$



# Méthode de Gauss

## Remarque 2

La stabilité de la décomposition dépend de la stratégie du choix du pivot :

- pivot immédiat : premier non nul
- pivot partiel : plus grand en norme de la colonne
- pivot total : plus grand en norme de la matrice entière

Choix du pivot partiel est de l'ordre de 6000 fois plus efficace que le pivot immédiat. (cf. Règles de bonne conduite et division par un nombre très petit en norme)

# Méthode de Gauss

## Remarque 3 - Optimisation des calculs

- Les opérations appliquées sur  $A$  et  $b$  sont identiques par ligne.
- Représentation matricielle uniquement « *esthétique* »

Préférable de travailler sur la matrice  $A$  à laquelle on adjoint la colonne  $b$  comme ultime colonne. Avantages :

- Représentation localisée des données et simplification du code.
- Opérations de remontée = Opérations par ligne  
(cf. cours précédent)

# Méthode de Gauss

## Remarque 4 - Problème d'inversion

Méthode Gauss-Jordan. Résoudre le système :

$$A.x = e_i, 0 < i \leq n$$

où,  $e_i$  est le  $i$ -ème vecteur de base de l'espace vectoriel.

Grâce à la remarque précédente, l'inversion matricielle est réalisable sans changer la complexité (à un facteur multiplicatif près!).

## Méthode de Gauss - Complexité

	Additions	Multiplications	Echanges
$P_k$	-	-	$(n - k + 1)$
$E_k$	$(n - k + 2)(n - k)$	$(n - k + 2)(n - k)$	-
Remontée $k$	$(n - k)$	$(n - k + 1)$	-
Total	$\mathcal{O}(n^3/3)$	$\mathcal{O}(n^3/3)$	$\mathcal{O}(n^2/2)$

**Complexité :  $\mathcal{O}(n^3)$**

# Décomposition LU

## Principe

Factoriser une matrice  $A$  telle que :

$$A = P.L.U$$

avec,

- $P$ , une matrice de permutation,
- $L$ , une matrice triangulaire inférieure (que des 1 sur la diagonale),
- $U$ , une matrice triangulaire supérieure

# Décomposition LU

## Retour à la méthode de Gauss

Processus de construction :

$$\begin{cases} A_0 = A \\ A_{k+1} = E_{k+1} \cdot P_{k+1} \cdot A_k \end{cases}$$

avec,

- $P_k$ , matrice de permutation,
- $E_k$ , matrice triangulaire inférieure (et des 1 sur la diagonale).

La matrice finale  $A_n$  est triangulaire supérieure (a.k.a la matrice  $U$  de la décomposition  $LU$ ).

# Décomposition LU

## Conjugaison par une matrice de permutation

Soit  $P$  une matrice d'échanges de lignes  $(i, j)$  et  $E_k$  une matrice triangulaire inférieure ne comportant que des 1 sur la diagonale et des coefficients extra-diagonaux non nuls que sur la  $k$ -ième colonne.

Alors, si  $i > k$  et  $j > k$  :

$$P.L_k = \tilde{L}_k.P$$

où  $L_k$  est triangulaire inférieure, ne comportant que des 1 sur la diagonale, et ses coefficients extra-diagonaux non-nuls sont ceux de  $L_k$  dans lesquels on a échangé la ligne  $i$  et la ligne  $j$ .

# Décomposition LU

## Exemple

Soient  $P$  une matrice d'échanges de lignes ( $i = 3, j = 4$ ) telle que :

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$L_{k=1}$ , matrice triangulaire inférieure telle que :

$$L_{k=1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{bmatrix}$$



# Décomposition LU

## Exemple

On a  $i > k$  et  $j > k$ , donc :

$$\tilde{L}_{k=1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{bmatrix}$$

où,

$$P.L_1 = \tilde{L}_1.P$$

# Décomposition LU

## Reconstruction de la décomposition $PLU$ (1/2)

$$\begin{cases} A_0 = A \\ A_{k+1} = E_{k+1} \cdot P_{k+1} \cdot A_k \\ A_n = E_{n-1} \cdot P_{n-1} \dots E_1 \cdot P_1 \cdot A \end{cases}$$

Les matrices  $P_i$  ne font qu'échanger une ligne  $i$  avec une ligne  $> i$ . Donc, on peut appliquer le lemme précédent à toute matrice  $E_j$  pour  $j < i$ , tel que :

$$\begin{aligned} A_n &= E_{n-1} \cdot P_{n-1} \dots E_1 \cdot P_1 \cdot A \\ &= E_{n-1} \cdot (P_{n-1} \cdot E_{n-2}) \dots (P_2 \cdot E_1) \cdot P_1 \cdot A \\ &= \tilde{E}_{n-1} \dots \tilde{E}_1 \cdot P_{n-1} \dots P_1 \cdot A \\ &\quad \text{soit, } U = L^{-1} \cdot P^{-1} \cdot A \end{aligned}$$

# Décomposition LU

## Reconstruction de la décomposition $PLU$ (2/2)

$$\begin{aligned} L^{-1} &= \tilde{E}_{n-1} \dots \tilde{E}_1, \text{ donc,} \\ L &= \tilde{E}_1^{-1} \dots \tilde{E}_{n-1}^{-1} \end{aligned}$$

où  $E_k^{-1}$  est une matrice triangulaire inférieure dans laquelle les éléments extra-diagonaux sont les opposés des éléments extra-diagonaux de  $E_k$ .

## Décomposition $LU$

En prenant  $P = \text{Id}$ , c'à d pas d'échanges de ligne pour trouver un pivot, une condition suffisante pour cette supposition est que :

« *Tous les déterminants extraits de  $A$  obtenus en ne considérant que les  $k$  premières lignes et  $k$  première colonnes de  $A$  sont non nuls* »

# Décomposition LU

Complexité pour la  $k$ -ième colonne des deux matrices  $L$  et  $U$  :

Nombre d'échanges	$\leq 2n + 1$
Nombre de multiplications	$(n - k)$ pour $L$ , $(n - k)(n - k + 1)$ pour $U$
Nombre de soustractions	$(n - k)(n - k + 1)$ pour $U$

**Complexité :  $\mathcal{O}(n^3)$**

# Décomposition QR

Soit le système linéaire,

$$Ax = b$$

décomposer la matrice  $A$  telle que,

$$A = QR, \text{ } n \times m$$

avec,

- $Q$ , matrice **orthogonale**,  $n \times n$
- $R$ , matrice triangulaire sup,  $n \times m$

# Décomposition QR

## Avantages d'une matrice orthogonale

- $Q^{-1} = Q^t$
- $\text{cond}(Q) = 1$

## Méthodes pour trouver $Q$ (resp. $R$ )

- *Householder*, produits successifs de matrices orthogonales élémentaires
- *Givens*, produits successifs de matrices de rotation plane
- *Schmidt*, orthogonalisation de matrices

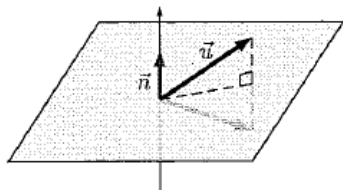
# Décomposition QR - Méthode de Householder

## Matrice de Householder

Symétrie orthogonale par rapport à un hyperplan de  $\mathbb{R}^n$ .

- $\mathcal{P}$  hyperplan de  $\mathbb{R}^n$
- $\vec{n}$  vecteur unitaire,  $\vec{n} \perp \mathcal{P}$

$\Rightarrow \vec{n}$  caractérise la symétrie



# Décomposition QR - Méthode de Householder

## Image de $\vec{u}$ par $H$

Soit  $H$  un symétrie orthogonale par rapport à  $\vec{n}$  alors :

$$H(\vec{u}) = \vec{u} - 2(\vec{u} \cdot \vec{n})\vec{n}$$

en représentation matricielle,

$$\begin{aligned} H.U &= U - 2(U^t.N).N \\ &= U - 2N.(N^t.U) \quad (\text{produit scalaire}) \\ &= U - 2(N.N^t).U \quad (\text{associativité}) \end{aligned}$$

soit,

$$H = \text{Id} - 2(N.N^t)$$



# Décomposition QR - Méthode de Householder

## Image de $\vec{u}$ par $H$

$$H = \text{Id} - 2(N.N^t)$$

Propriétés de  $H$  :

- complexité du calcul de  $H$  :  $\mathcal{O}(n^2)$
- complexité du produit  $H.U$  :  $\mathcal{O}(n)$
- matrice symétrique inversible
- orthogonale

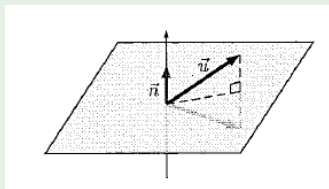
$H$  est auto-inversible :

- $H$  est symétrique donc :  $H^t = H$
- $H$  est orthogonale donc :  $H^{-1} = H^t$
- donc,  $H = H^{-1}$

# Décomposition QR - Méthode de Householder

## Image de $\vec{u}$ par $H$

- $\mathcal{P}$  hyperplan de  $\mathbb{R}^3$
- $\vec{n}$  vecteur unitaire,  
 $\vec{n} \perp \mathcal{P}$ ,  $\vec{n}^t = (0, 1, 0)$

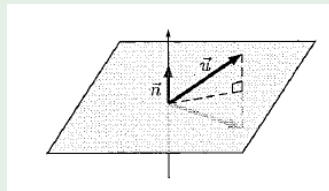


- Calculer  $H$
- Calculer l'image du vecteur  $\vec{u}^t = (1, 1, 0)$

# Décomposition QR - Méthode de Householder

## Image de $\vec{u}$ par $H$

- $\mathcal{P}$  hyperplan de  $\mathbb{R}^3$
- $\vec{n}$  vecteur unitaire,  
 $\vec{n} \perp \mathcal{P}$ ,  $\vec{n}^t = (0, 1, 0)$



$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$H.U = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

# Décomposition QR - Méthode de Householder

## But de la factorisation QR

Décomposer la matrice  $A$  telle que,

$$A = QR, \text{ } n \times m$$

avec,

- $Q$ , matrice **orthogonale**,  $n \times n$
- $R$ , matrice triangulaire sup,  $n \times m$

## Un problème récurrent ...

$$A_k = \begin{bmatrix} \text{triangle} & \text{rectangle} \\ & \tilde{A} \end{bmatrix}$$

$$\begin{cases} A_0 &= A \\ A_{k+1} &= Q_k \cdot A_k \end{cases}$$

# Décomposition QR - Méthode de Householder

## Elimination de la variable

$$\begin{cases} Q_1 = H_{U,V} \\ A_1 = Q_1 \cdot A \end{cases}$$

- $H$  matrice de *Householder*,  $H.U = V$
- $U$  première colonne de  $A$
- $V = \|U\| e_1$ ,

Trouver la matrice  $H$  telle que  $H.U = V$

$$H = Id - 2(N.N^t)$$

- Si  $U.N = 0$ ,  $U = V$  ( $H = Id$ )
- Sinon  $N$  colinéaire à  $U - V$ ,  $N = \frac{U-V}{\|U-V\|}$

# Décomposition QR - Récurrence

$$A_k = \begin{bmatrix} \text{triangle} & \text{rectangle} \\ & \tilde{A} \end{bmatrix}$$

$$A_{k+1} = Q_k \cdot A_k$$

$$\text{avec, } Q_k = \begin{pmatrix} I & 0 \\ 0 & H_{U,V} \end{pmatrix}$$

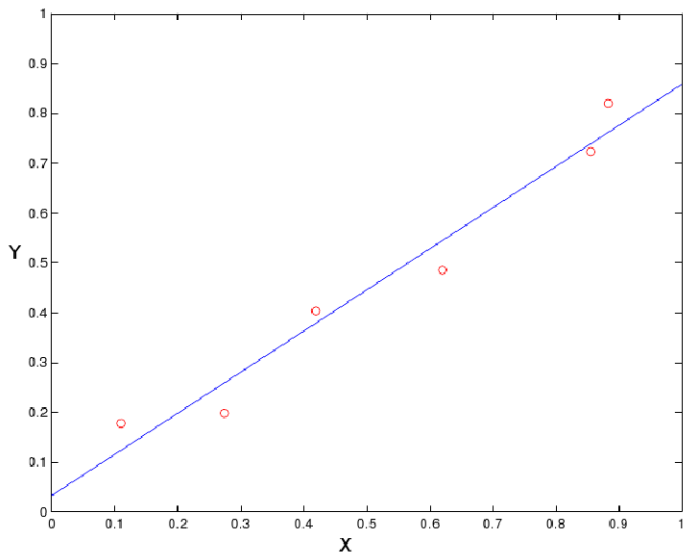
Après  $m$  récurrences ...

$$A_m = Q_m \dots Q_2 \cdot Q_1 \cdot A$$

soit,

$$A = \underbrace{(Q_1 \cdot Q_2 \dots Q_m)}_Q \cdot \underbrace{A_m}_R$$

# Décomposition QR - Méthode des moindres carrés



# Décomposition QR - Méthode des moindres carrés

## Représentation matricielle

Soit le système,

$$A.x = b$$

avec  $A$  matrice  $n \times m$ , où  $n \gg m$ .

Problème de minimisation :

$$\min_x (\|A.x - b\|^2)$$

dans l'exemple dans  $\mathbb{R}^2$ ,

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \text{et } x = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$



# Méthodes itératives - Présentation

## Contexte

- Pas de nécessité de connaître un résultat exact
- Systèmes de taille particulièrement grande
- Processeurs de flux

## Principe

Résoudre  $A.x = b$  par la méthode du point fixe :

$$\begin{cases} x_0 \text{ fixé} \\ x_{n+1} = f(x_n) \end{cases}$$

où  $x_\infty$  point fixe, solution du système.

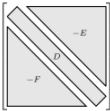
# Méthodes itératives - Présentation

## Méthode

Idée principale :  $A = M - N$  avec  $M$  inversible.

$$\begin{aligned}
 A.x &= b \Leftrightarrow M.x = N.x + b \\
 &\Leftrightarrow x = M^{-1}.N.x + M^{-1}b \\
 &\Leftrightarrow f(x)
 \end{aligned}$$

## Implémentation



$$A = \begin{bmatrix} & & -E \\ & D & \\ -F & & \end{bmatrix} = D - E - F$$

- $M = ?$  et  $N = ?$
- idéalement, choisir  $M$  facilement inversible (Matrice diagonale par exemple)
- Partitionner  $A$  en fonction de  $D$ ,  $E$  et  $F$

# Méthodes itératives - Jacobi

$$\text{Partition} = \begin{cases} M = D \\ N = E + F \end{cases}$$

## Equation matricielle

$$x^{k+1} = D^{-1} \cdot (E + F) \cdot x^k + D^{-1} \cdot b$$

## Calcul d'un élément $x_i$ de $x$

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

## NB

Itération  $k + 1$  : opération sur tous les  $x_i^k$

# Méthodes itératives - Gauss-Seidel

$$\text{Partition} = \begin{cases} M = D - F \\ N = E \end{cases}$$

## Equation matricielle

$$x^{k+1} = D^{-1}(F.x^{k+1} + E.x^k + b)$$

## Calcul d'un élément $x_i$ de $x$

$$x_i^{k+1} = \frac{b_i - \sum_{j < i} a_{ij}x_j^{k+1} - \sum_{j > i} a_{ij}x_j^k}{a_{ii}}$$

## NB

Itération  $k + 1$  : ré-utilisation des  $x_i^{k+1}$  déjà calculés (in-place)

# Méthodes itératives - Relaxation

$$\text{Partition} = \begin{cases} M = \frac{1}{\omega} D - F \\ N = \frac{1-\omega}{\omega} D + E \end{cases}$$

## Principe

Faire varier le paramètre  $\omega$  pour améliorer la convergence

$\omega = 1$  : Gauss-Seidel

## Equation matricielle

$$x^{k+1} = (D - F)^{-1} \cdot (E \cdot x^k + b)$$

# Méthodes itératives - Convergence

## Théorique

- Guidée par le rayon spectral de  $M^{-1}N$
- converge ssi rayon spectral  $< 1$

## Pratique

- Vérifier que le résiduel décroît
- fixer un nombre de pas maximum
- Quelques cas particuliers où les méthodes convergentes
  - Matrice à diagonale strictement dominante
  - Matrice symétrique définie positive (relaxation  $\omega \in ]0; 2[$ )

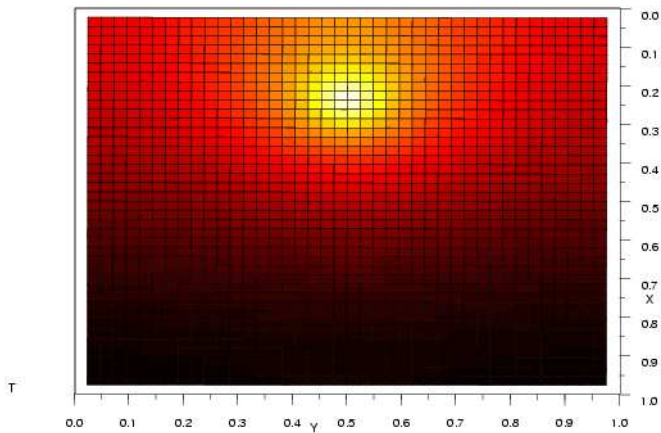
## Avantages

- Approximation des solutions
- GPGPU : General Purpose on GPU

# Résolution de systèmes linéaires - Applications

- Equation de la chaleur
- Exemples en Informatique graphique

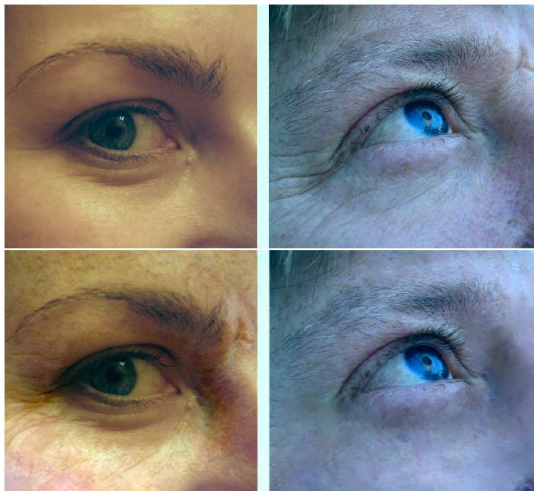
# Equation de la chaleur





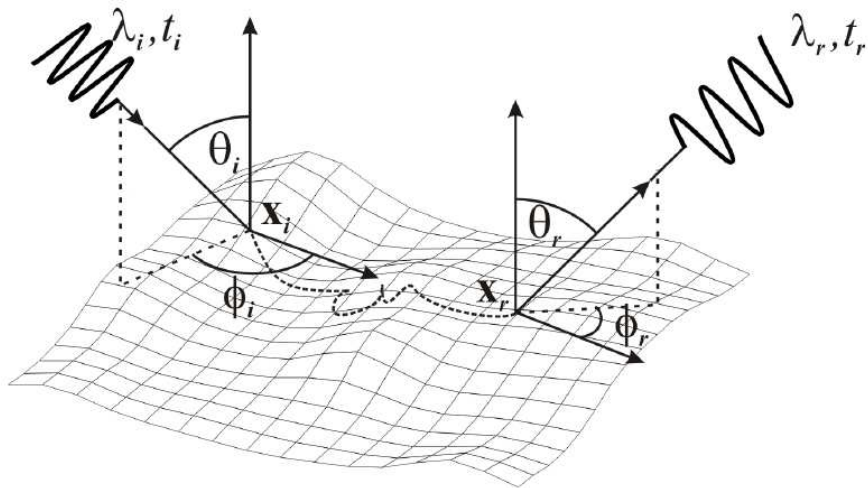
# Informatique Graphique

Inpainting :  $\Delta u = f$



# Informatique Graphique

## Interactions lumineuses



# Informatique Graphique



## Bidirectional Texture Functions

$$BTF(x, \omega_i, \omega_o)$$

où

- $x$ , coordonnées spatiales  $(u, v)$
- $\omega_i$ , lumière incidente  $(\theta_i, \phi_i)$
- $\omega_o$ , direction sortante  $(\theta_o, \phi_o)$

issue généralement d'un processus d'acquisition.

# Informatique Graphique

## Système à résoudre

$$BTF_{(x, \omega_{oj})}(\omega_i) = a_0 * l_{u_i}^2 + a_1 * l_{v_i}^2 + a_2 * l_{u_i} * l_{v_i} + a_3 * l_{u_i} + a_4 * l_{v_i} + a_5, \forall i$$

$$\begin{bmatrix} l_{u_1}^2 & l_{v_1}^2 & l_{u_1} l_{v_1} & l_{u_1} & l_{v_1} & 1 \\ l_{u_2}^2 & l_{v_2}^2 & l_{u_2} l_{v_1} & l_{u_2} & l_{v_2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ l_{u_m}^2 & l_{v_m}^2 & l_{u_m} l_{v_m} & l_{u_m} & l_{v_m} & 1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_5 \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \vdots \\ L_5 \end{bmatrix}$$

Solution : méthode des moindres carrés !!!

# Bonnes pratiques en Algo Num

## Recommandations

- la structure d'un fichier source
- le contenu du rapport
- conseils d'écriture en  $\text{\LaTeX}$

## Méthode d'évaluation

- 1 Lecture de l'ensemble des rapports
- 2 Pour chaque groupe :
  - 1 exécution du code
  - 2 lecture du code
  - 3 lecture approfondie du rapport
  - 4 note provisoire
- 3 Attribution des notes finales

# Bonnes pratiques pour le code

## Règles à respecter

- Ecrire le code en anglais
- Rédiger les commentaires en français

## Recommandations

### Rigueur algorithmique

- Si le temps d'exécution d'une fonction simple est long, c'est qu'il y a un problème !

### Ecrire des fonctions de tests

- Les résultats apparaissent à l'exécution du programme.

# Bonnes pratiques pour le rapport

## Algorithmes

### Tout algorithme peut être dessiné !

- Expliquer un algo par un schéma ou dessin,
- Un dessin vaut mieux qu'un long discours !

## Explications

### Synthétiser les explications

- Eviter le blabla
- Aller directement à l'essentiel ...
- tout en soignant l'expression !

### Le rapport se suffit à lui même

- Pas de références vers le code
- Doit répondre à toute les questions du sujet



# Bonnes pratiques pour le rapport

## Résultats

### Les résultats présentés doivent être pertinents

- Pas de résultats inutiles !
- Chaque partie du projet doit contenir des résultats
- Tout résultat ou argumentation doivent être en accord avec le code
- Choisir une représentation adaptée à chaque résultat (graphiques, tableaux ...)

### Les résultats doivent être analysés

- Justifier chaque résultat présenté
- Un graphique doit être expliqué et doit avoir des **légendes**

## Mise en forme

### Utiliser les bons symboles

- Complexité :  $\mathcal{O}(n) \rightarrow \backslash\mathrm{mathcal{O}}(n)$
- « guillemets »  $\log\{ \}$  guillemets  $\mathrm{fg}\{ \}$

### Mise en page

- Utiliser les sections et subsections mais pas plus !
- Pour la mise en page, le package *a4wide* ou équivalent

## Utiliser les environnements

- Pour les figures

```
\begin { figure } [ options ]  
\includegraphics { ... }  
\label { thelabel } <- à utiliser avec \ref dans le texte  
\caption { thecaption } <- description de la figure  
\end { figure }
```

- autres environnements : equation, algorithm et bien d'autres !

### 3 Éléments propres

- Présentation
- Valeurs propres
- Vecteurs propres
- Applications

# Introduction au problème

## Définition

$$A.X_\lambda = \lambda.X_\lambda$$

- $A$ ,  $n \times n$
- $\lambda \in \mathbb{R}$ , valeur propre
- $X_\lambda \in \mathbb{R}^n$  non nul, vecteur propre

## Méthode

$\lambda$  fixé,  $X_\lambda$  base vectorielle de  $\mathbb{R}^n$ . Deux étapes de calcul :

- 1 trouver les valeurs propres ;
- 2 trouver des bases pour les sous-espaces propres.

## Restriction

$A$  doit être (tri-)diagonalisable.

# Valeurs propres

## Méthode usuelle

Trouver une suite  $P_k$  telle que :

$$\lim_{k \rightarrow \infty} P_k^{-1} A P_k \rightarrow T$$

- $T$ , matrice triangulaire
- Valeurs propres = éléments diagonaux de  $T$

## Méthodes étudiées

- Matrices symétriques :
  - *Jacobi*
  - *Givens-Householder*
- Matrices quelconques :
  - *QR*

# Méthode de Jacobi

## Principe

Soit  $A$ , matrice symétrique de taille  $n \times n$ . Construire une suite de matrices orthogonales  $O_k$  telles que,

$${}^t O_k \cdot A \cdot O_k \xrightarrow[k \rightarrow \infty]{} D$$

converge vers une matrice diagonale  $D$ .

Propriétés des matrices  $O_k$  :

- matrices orthogonales
- matrice de rotation (i.e. élimination de coefficients)

# Méthode de Jacobi

## Construction des matrices $O_k$

Soient un indice  $p$  de ligne, un indice  $q$  de colonne et un réel  $\theta \in ]-\pi; \pi]$ , alors :

$$O = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \cos \theta & & & & \\ & & & 1 & & \sin \theta & \\ & & & & \ddots & & \\ & & -\sin \theta & & & 1 & \\ & & & & & \cos \theta & \\ & & & & & & 1 \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix}$$

où,

- $O(p, p) = O(q, q) = \cos \theta$
- $O(p, q) = \sin \theta$
- $O(q, p) = -\sin \theta$



# Méthode de Jacobi

## Exemple de matrices $O_k$

Considérons la transformation d'une matrice  $A$ , restreinte aux lignes  $p$  et  $q$ . On a :

$$B = {}^t O . A . O$$

$$B = \begin{bmatrix} b_{p,p} & b_{p,q} \\ b_{q,p} & b_{q,q} \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a_{p,p} & a_{p,q} \\ a_{q,p} & a_{q,q} \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

Choisir  $\theta$  tel que  $b_{p,q}$  et  $b_{q,p}$  soient nuls.

Si  $b_{p,q}$  et  $b_{q,p}$  sont nuls alors,  $\theta = 0$ , sinon,

$$\begin{aligned} b_{p,q} = b_{q,p} &= a_{p,q} \cos(2\theta) + \frac{a_{p,p} - a_{q,q}}{2} \sin(2\theta) \\ \Rightarrow \cotan(2\theta) &= \frac{a_{q,q} - a_{p,p}}{2a_{p,q}} \end{aligned}$$

# Méthode de Jacobi

## Itérations

Soient les matrices  $A_0 = A$  et  $O_0 = \text{Id}$ , on construit la suite telle que :

$$\begin{cases} A_{k+1} = {}^t O_k \cdot A_k \cdot O_k \\ O_{k+1} = O_k \cdot O \end{cases}$$

Étapes de l'algorithme :

- 1 Choisir un élément extradiagonal  $a_{p,q}$  de  $A$  non nul ;
- 2 Construire la matrice de rotation  $O$  adaptée à cet élément ;
- 3 Construire la matrice  ${}^t O \cdot A \cdot O$ .

# Méthode de Jacobi

## Optimisations

Seules les  $p$ -èmes et  $q$ -èmes lignes et colonnes de la matrice  $A$  sont modifiées à chaque itération :

$$\left\{ \begin{array}{ll} b_{i,j} = a_{i,j} & \text{si } i, j \neq p, q \\ b_{p,i} = a_{p,i} \cos \theta - a_{q,i} \sin \theta & \text{si } i \neq p, q \\ b_{q,i} = a_{p,i} \sin \theta + a_{q,i} \cos \theta & \text{si } i \neq p, q \\ b_{p,p} = a_{p,p} - a_{p,q} \tan \theta \\ b_{q,q} = a_{p,p} + a_{p,q} \tan \theta \\ b_{p,q} = b_{q,p} = 0 \end{array} \right.$$

## Bilan

- Choix du pivot : élément extra-diagonal de norme maximale.  
 $\Rightarrow$  Assure la convergence.
- Complexité :  $\mathcal{O}(n)$  à chaque étape

# Méthode de Givens-Householder

## Principe

Déterminer le polynôme caractéristique de la matrice  $A$ , réelle symétrique, pour en extraire les racines.

Deux étapes distinctes de calculs :

- 1 Déterminer, en un nombre fini d'étapes, une matrice orthogonale  $O$  telle que  ${}^t O.A.O$  soit une matrice tridiagonale (*Householder*) ;
- 2 *méthode de givens* pour déterminer les valeurs propres de cette matrice.

# Méthode de Givens-Householder

## Mise sous forme tridiagonale

Rappels :

- Méthode QR pour la résolution de système linéaire
- Matrices de Householder : élimination de variables
- Mise sous forme triangulaire :  $R = Q^t A$

Or,  $A$  symétrique, donc en considérant le produit (avec  $H \approx Q$ ) :

$$H^t . A . H$$

on peut transformer la matrice  $A$  en matrice tri-diagonale.

# Rappel - Méthode de Householder

## Élimination de la variable

$$\begin{cases} Q_1 = H_{U,V} \\ A_1 = Q_1 \cdot A \end{cases}$$

- $H$  matrice de *Householder*,  $H.U = V$
- $U$  première colonne de  $A$
- $V = \|U\|e_1$ ,

Trouver la matrice  $H$  telle que  $H.U = V$

$$H = Id - 2(N.N^t)$$

- Si  $U.N = 0$ ,  $U = V$  ( $H = Id$ )
- Sinon  $N$  colinéaire à  $U - V$ ,  $N = \frac{U-V}{\|U-V\|}$

# Méthode de Givens-Householder

## Tri-diagonalisation

$$\begin{cases} Q_1 = H_{U,V} \\ A_1 = {}^t Q_1 \cdot A \cdot Q_1 \end{cases}$$

- $H$  matrice de *Householder*,  $H.U = V$
- $U$  première colonne de  $A$   
uniquement les éléments sous-diagonaux
- $V = \|U\|e_1$ ,

## Mise sous forme tridiagonale

$$\begin{cases} A_0 &= A \\ A_{k+1} &= {}^t H_k A_k H_k \\ O_{k+1} &= H_k O_k \end{cases}$$

Récurrance à l'étape  $k$  :

136 / 295



# Méthode de Givens-Householder

## Optimisation du calcul ${}^t H_k . A_k . H_k$

$$\begin{aligned}
 H_k &= Id - 2 {}^t N_k N_k \\
 A_k . H_k &= A_k - 2 . (A_k . N_k) {}^t N_k \\
 {}^t H_k . A_k . H_k &= A_k - 2 (A_k N_k) {}^t N_k - 2 {}^t N_k N_k (A_k - 2 (A_k N_k) {}^t N_k) \\
 &= A_k - 2 \underbrace{(A_k N_k) {}^t N_k}_{P_k} - 2 N_k \underbrace{{}^t (A_k N_k)}_{{}^t P_k} + 4 \underbrace{({}^t N_k (A_k N_k))}_{P_k} {}^t N_k N_k
 \end{aligned}$$

On précalcule  $P_k = A_k . N_k$  ( $\mathcal{O}((n - k)^2)$ ) et  $\lambda = {}^t N_k . P_k$  ( $\mathcal{O}(n - k)$ )

On a donc,

$$({}^t H_k . A_k . H_k)_{ij} = (A_k)_{ij} - 2(P_k)_i (N_k)_j - 2(N_k)_i (P_k)_j + 4\lambda (N_k)_i (N_k)_j$$

de complexité en  $\mathcal{O}(n^2)$  pour l'étape  $k$ .

La tri-diagonalisation se termine en  $n - 2$  étapes, **complexité totale en  $\mathcal{O}(n^3)$** .

# Méthode de Givens-Householder

## Polynôme caractéristique

Obtenir des valeurs propres à partir du polynôme caractéristique de la matrice  $A$  tri-diagonalisée.

Soit,  $T$ , la matrice résultant de la tridiagonalisation de la matrice  $A$ , telle que :

$$T = \begin{bmatrix} b_1 & c_1 & & & \\ c_1 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-2} & b_{n-2} & c_{n-1} \\ & & & c_{n-1} & b_n \end{bmatrix}$$

On a,

$$\begin{cases} P_0(X) = 1 \\ P_1(X) = b_1 - X \\ P_k(X) = (b_k - X)P_{k-1}(X) - c_{k-1}^2 P_{k-2}(X) \end{cases}$$

# Méthode de Givens-Householder

## Propriétés des polynômes caractéristiques $P_k$

$P_k(X)$  polynôme caractéristique de la sous-matrice de  $A$ ,

- $P_k$  de degré  $k$
- $\lim_{x \rightarrow -\infty} P_i(x) = +\infty$  et  $\lim_{x \rightarrow +\infty} P_i(x) = (-1)^i \infty$
- $x$  racine du polynôme  $P_k$ , alors  $P_{k-1}(x)$  et  $P_{k+1}(x)$  sont non nuls et de signes opposés

## Calcul des valeurs propres

Recherche des racines des  $P_k(X)$ .

(cf. Tableau de variation et recherche dichotomique)

# Méthode QR

## Présentation

- Méthode pour des matrices carrées inversibles
- Mise en oeuvre *simple*

## Algorithme

$M = A$

Pour  $i \leftarrow [0; \infty]$

- $(q, r) = \text{decompositionQR}(M)$
- $M = r * q$

## Convergence

$$\lim_{n \rightarrow \infty} m_{i,i}^k = \lambda_i$$

$$\lim_{n \rightarrow \infty} m_{i,j}^k = 0, \quad \forall i < j$$

# Méthode SVD

## Présentation

Méthode pour des matrices non inversibles (*singular*)

La *décomposition en valeurs singulières* consiste à écrire une matrice  $A$ , de taille  $n \times m$ , en :

$$A = U.S.^tV \text{ avec } \begin{cases} U & \text{orthogonale, de taille } n \times n \\ S & \text{diagonale positive, de taille } n \times m \\ V & \text{orthogonale, de taille } m \times m \end{cases}$$

où  $S$ , *valeurs singulières* de  $A$ , est telle que les  $s_{k,k}$  sont ordonnées de manière décroissante.

# Méthode SVD

## Lien avec la décomposition en valeurs propres

$$\begin{aligned} {}^tA.A &= V.{}^tS.{}^tU.U.S.{}^tV \\ &= V.({}^tS.S).{}^tV \\ A.{}^tA &= U.S.{}^tV.V.{}^tS.{}^tU \\ &= U.(S.{}^tS).{}^tU \end{aligned}$$

- $(\text{valeur propre})^2 = |\text{valeur singulière}|$  (si  $\neq 0$ )
- les colonnes de  $U$  sont les vecteurs propres de  $A.{}^tA$
- les colonnes de  $V$  sont les vecteurs propres de  ${}^tA.A$

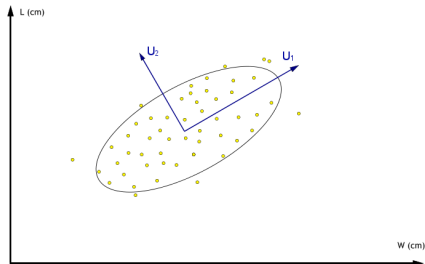
## Complexité du calcul

- Mise sous forme bidiagonale par des matrices de *Householder* :  $\mathcal{O}(mn^2)$
- Mise sous forme diagonale par la méthode *QR* optimisée en  $2n$  itérations

# Méthode SVD

## Domaine applicatif

- Pseudo-inverse : méthode des moindres carrés
- Approximation/Compression de matrices par réduction de dimensionnalité
  - pour les images (peu efficace en utilisant un algo trivial)
  - pour les données fortement corrélées (image par bloc par ex.)
- Statistique
  - les colonnes de  $U$  représentent les directions de plus grande variation
- cinématique, météorologie, ...



# Calcul des vecteurs propres

## Principe

A partir des valeurs propres  $\lambda_i$ , trouver une base de vecteurs, les vecteurs propres  $X_{\lambda_i}$ , associée telle que :

$$A.X_{\lambda} = \lambda.X_{\lambda}$$

- $A$ ,  $n \times n$
- $\lambda \in \mathbb{R}$ , valeur propre
- $X_{\lambda} \in \mathbb{R}^n$  non nul, vecteur propre

## Méthodes

- méthode de la puissance itérée, déflation
- méthode de la puissance inverse



# Méthode de la puissance itérée

## Présentation

Soient,

- $A$ , matrice diagonalisable carré
- $\{e_1, \dots, e_n\}$  vecteurs propres
- $\{\lambda_1, \dots, \lambda_n\}$  valeurs propres avec  $|\lambda_k| \geq |\lambda_{k+1}|$
- $x_0$  un vecteur non nul quelconque

Considérons la suite :

$$x_{k+1} = Ax_k$$

projetée dans la base  $\{e_i\}$  :

$$\begin{cases} x_0 = \alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_n e_n \\ x_k = \lambda_1^k \alpha_1 e_1 + \lambda_2^k \alpha_2 e_2 + \dots + \lambda_n^k \alpha_n e_n \end{cases}$$

# Méthode de la puissance itérée

## Présentation

Considérons la suite :

$$x_{k+1} = Ax_k$$

projetée dans la base  $\{e_i\}$  :

$$\begin{cases} x_0 = \alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_n e_n \\ x_k = \lambda_1^k \alpha_1 e_1 + \lambda_2^k \alpha_2 e_2 + \dots + \lambda_n^k \alpha_n e_n \end{cases}$$

$\Rightarrow$  Le vecteur  $x_k$  se comporte en norme comme  $\lambda_1^k$

# Méthode de la puissance itérée

## Suite

$$\begin{cases} x_0 &= \text{vecteur non nul} \\ x_{k+1} &= \frac{A \cdot x_k}{\|A \cdot x_k\|} \\ \lambda_{k+1} &= {}^t x_k \cdot A \cdot x_k \end{cases}$$

où  $\lambda_\infty$  et  $x_\infty$  sont la valeur propre de module maximale et le vecteur propre associé.

## Convergence

*Si  $A$  est une matrice carrée possédant une unique valeur propre de module maximal, alors la méthode de la puissance itérée converge vers une direction propre associée à cette valeur propre.*

# Méthode de la puissance itérée

## Généralisation 1/2

Considérons la suite telle que :

$$x_{k+1} = A^{-1}x_k$$

On applique la méthode de la puissance itérée pour trouver :

- la valeur propre de module maximale et le vecteur propre associé de la matrice  $A^{-1}$ .

qui sont,

- la valeur propre de module minimale et le vecteur propre associé de la matrice  $A$ .

⇒ C'est la **méthode de la puissance inverse**.

# Méthode de la puissance inverse

## Généralisation 2/2

Soit la matrice  $B = (A - \tilde{\lambda}\text{Id})$ . Quelles sont les valeurs propres de la matrice  $B$  ?

Soient,  $\lambda$  une valeur propre de  $A$  et  $e_\lambda$  le vecteur propre associé, alors

$$\begin{aligned}(A - \tilde{\lambda}\text{Id})e_\lambda &= A.e_\lambda - \tilde{\lambda}.e_\lambda \\ &= \lambda.e_\lambda - \tilde{\lambda}.e_\lambda \\ &= (\lambda - \tilde{\lambda}).e_\lambda\end{aligned}$$

En appliquant la méthode de la puissance inverse sur  $(A - \tilde{\lambda}\text{Id})$ , on obtient :

- la valeur propre  $\lambda_i - \tilde{\lambda}$  de module minimale de la matrice  $B$

càd,

- la valeur propre de  $A$  la plus proche de  $\tilde{\lambda}$  !

# Méthode de la puissance inverse

## Propriétés

*Généralisation de la méthode de la puissance itérée*

Soient,

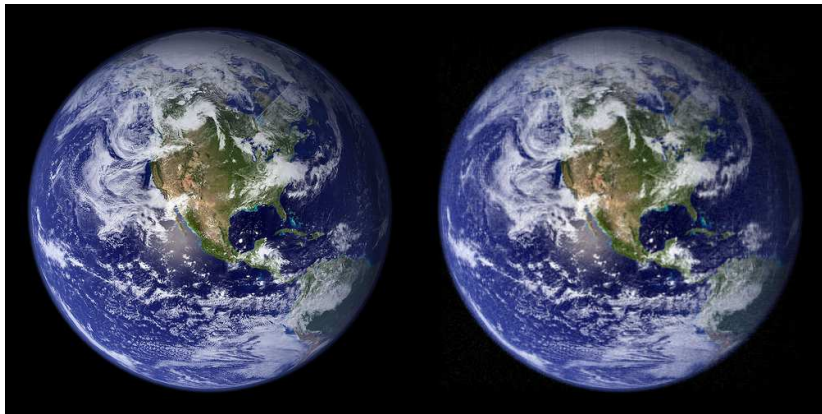
- $A$ , matrice diagonalisable carré
- $\lambda$  une valeur propre de  $A$
- $\tilde{\lambda}$  une valeur très proche de  $\lambda$

$$\begin{cases} x_0 & \text{vecteur non nul} \\ x_{k+1} & = \frac{(A - \tilde{\lambda} Id) \cdot x_k}{\|(A - \tilde{\lambda} Id) \cdot x_k\|} \\ \lambda_{k+1} & = {}^t x_k \cdot A \cdot x_k \end{cases}$$

Complexité :

- $\mathcal{O}(n^2)$  par itération, +  $\mathcal{O}(n^3)$  pour l'inversion
- Cependant, vitesse de convergence exponentielle. Donc, préférable d'effectuer le calcul au préalable des valeurs propres pour limiter le nombre d'itération.

# Application à la Compression d'images



*Compression avec la SVD : original / compressé (rank=100)*

# Application à la Compression d'images

## Stratégie

*Exploiter la **redondance** d'information inter-pixels pour stocker, utiliser ou transmettre les données plus efficacement.*

## Types de compression

- **sans perte** (*lossless*) : reproduction à l'identique (taux de compression usuels : 30 à 70%)
- **avec perte** (*lossy*) : reproduction avec erreurs sur la valeur des pixels (taux de compression usuels : 70 à 95%)

## Evaluation d'une méthode

Meilleur compromis entre :

- la taille de la représentation compressée (quantitatif)
- la qualité des données reproduite (qualitatif)



# Application à la Compression d'images

## Estimation des critères d'évaluation

- **quantitatif** : ratio de compression

$$r = \frac{\text{Taille de la représentation compressée}}{\text{Quantité de valeurs représentées}}$$

- **qualitatif** : approche triviale ou perceptuelle
  - mesure de l'erreur : l'erreur quadratique moyenne

$$e = \frac{\sum_{i,j} (O_{i,j} - A_{i,j})^2}{\# \text{ pixels}}$$

- mesure de l'erreur **perçue** :
  - Tests utilisateurs : système visuel humain (SVH)
  - Métriques perceptuelles : déduites de l'observation du SVH

# Application à la Compression d'images - Métriques perceptuelles

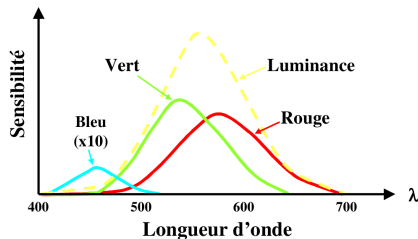
## Espace de couleurs RGB

- adapter à la visualisation (écran)
- 3 composantes à 8 bits par pixel (24 bpp)
- peu adapter à la manipulation d'image
  - espace linéaire
  - peu corrélér avec la perception (logarithmique, couleur)

# Application à la Compression d'images - Métriques perceptuelles

## Espaces de couleurs pour l'évaluation

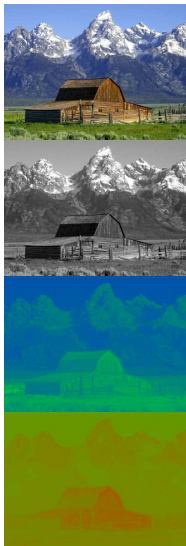
- Basés sur la **luminance** : image en niveau de gris
- transformation (non-)linéaire depuis RGB :
  - pondération basée sur la perception de la variation des couleurs
  - séparation entre la luminance et la chrominance



**nb**

sensibilité plus importante à la variation de luminance que de chrominance

# Application à la Compression d'images - Métriques perceptuelles



## Espace de couleurs pour l'évaluation

- linéaires : YUV, HSV, XYZ
- non linéaires : CieLAB, CieLuv
- Besoin d'un modèle « simple » pour l'évaluation : **YUV**
  - Y : la luminance (← **valeur comparative**)
  - UV : la chrominance

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

# Application à la Compression d'images - Métriques perceptuelles

## Métrique usuelle : PSNR

$$PSNR = 10. \log_{10} \left( \frac{255^2}{e} \right)$$

avec  $e$  telle que :

$$e = \frac{\sum_{i,j} (O_{i,j} - A_{i,j})^2}{\# \text{ pixels}}$$

Propriétés :

- évaluation **globale** de la qualité de l'image
- unité : dB
- signification :  $> 30dB = +$ ,  $> 40dB = ++$
- problème : erreur non localisée



rang	500	100	50	10
PSNR (dB)	$\infty$	24.45	23.38	20.12
$e (10^{-5})$	0	3.59	4.59	9.72

# Application à la Compression d'images - Métriques perceptuelles

## Métrique usuelle : SSIM




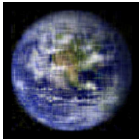
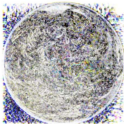
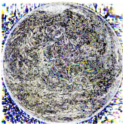
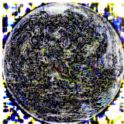
$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2cov_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

avec,

- $\mu_x$  (resp.  $\mu_y$ ) la moyenne des  $x$  (resp.  $y$ ), au voisinage  $\mathcal{V}$  du pixel
- $\sigma_x^2$  (resp.  $\sigma_y^2$ ) la variance des  $x$  (resp.  $y$ ), au voisinage  $\mathcal{V}$  du pixel
- $cov_{xy}$  la covariance des  $x$  et  $y$ , au voisinage  $\mathcal{V}$  du pixel
- $c_1 = 6.5025$  et  $c_2 = 58.5225$ , variables pour la stabilité numérique

$$\begin{aligned}\mu_x(u, v) &= \frac{\sum_{p \in \mathcal{V}_{(u,v)}} x(p)}{\|\mathcal{V}_{(u,v)}\|} \\ \sigma_x^2(u, v) &= \frac{\sum_{p \in \mathcal{V}_{(u,v)}} (x(p) - \mu_x(u, v))^2}{\|\mathcal{V}_{(u,v)}\|} \\ cov_{xy}(u, v) &= \frac{\sum_{p \in \mathcal{V}_{(u,v)}} (x(p) - \mu_x(u, v))(y(p) - \mu_y(u, v))}{\|\mathcal{V}_{(u,v)}\|}\end{aligned}$$

# Application à la Compression d'images - Métriques perceptuelles

				
rang	500	100	50	10
ratio		0.4	0.2	0.04
PSNR (dB)	$\infty$	24.45	23.38	20.12
e ( $10^{-5}$ )	0	3.59	4.59	9.72
				
SSIM map				
SSIM (%)	100	74	67	52

# Application à la Compression d'images - Métriques perceptuelles

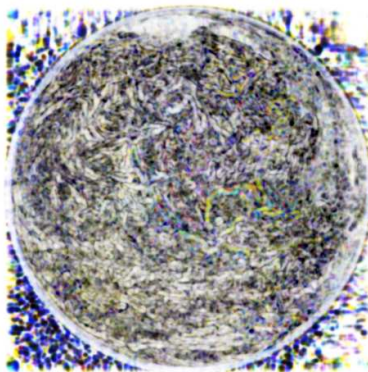




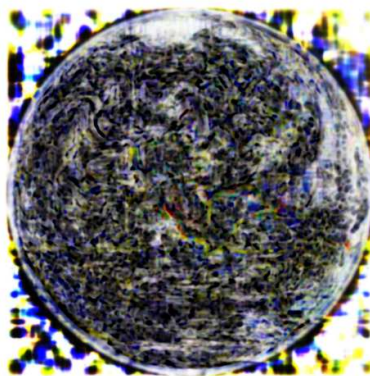
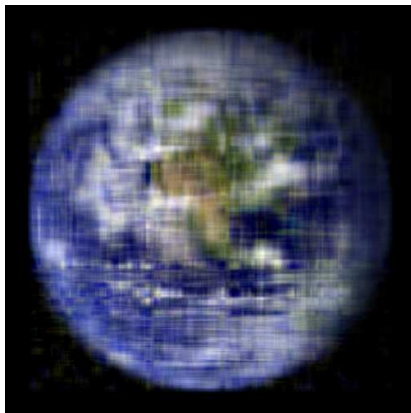
# Application à la Compression d'images - Métriques perceptuelles



# Application à la Compression d'images - Métriques perceptuelles



# Application à la Compression d'images - Métriques perceptuelles



# Application à la Compression d'images

## Types de transformation utilisée pour la compression d'images

- **Vectoriel** : composée d'objets géométriques individuels (cercle, droite, rectangle)
- **Algorithmique** : codeurs arithmétiques (Huffman par exemple)
- **Changement de base** : représenter les données dans une base raffnable
  - fréquentielle : décomposer en bandes de fréquence (fourier, ondelettes, dct)
  - statistique : classier les données par ordre d'importance (SVD)

# Application à la Compression d'images

## Compression avec la SVD

- Principe facile à mettre en oeuvre
- Efficace si l'image est homogène
- complexité élevée :  $\mathcal{O}(n)$  pour reconstruire un pixel
- Méthode globale : décrit l'ensemble de l'image

## Amélioration possible

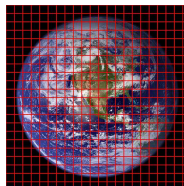
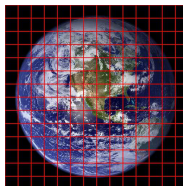
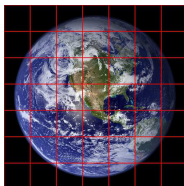
→ Définir une représentation locale des données dont les frontières de chaque sous-espace sont guidées par une métrique d'erreur.

# Application à la Compression d'images

## Amélioration possible

Un exemple d'algo trivial possible :

- ❶ Partir de l'image entière
- ❷ Décomposer en SVD au rang  $k$
- ❸ Si ( $\text{erreur} \geq \epsilon$ ) alors
  - Subdiviser l'image en 4
  - Pour chaque sous-image, repartir à l'étape 2 au rang  $k = k/4$



- 4 **Equations non linéaires**
  - Présentation
  - Méthodes de résolution
  - Applications

# Introduction au problème

## Définition

Résoudre une équation basée sur la fonction  $f$ , définie sur  $\mathbb{R}^n$ , telle que :

$$f(x) = 0, x \in \mathbb{R}^n$$

i.e. Déterminer l'ensemble des valeurs  $x$ , racines de la fonction  $f$

## Contexte

- continuité : utilisation des propriétés sur la dérivé
- complexité : nombre d'évaluation de la fonction  $f$
- multiple dimensions : généralisation, si possible, à  $\mathbb{R}^n$

## Propriétés

- Méthodes itératives : pas de méthode directe
- Suite de valeurs qui convergent vers **une** racine recherchée



# Méthodes de résolution

## Vitesse de convergence

Etude de la variation de l'erreur d'approximation définie telle que :

$$\epsilon_{k+1} = C \times (\epsilon_k)^m$$

avec  $C > 0$ ,  $m \geq 1$  et que  $C < 1$  lorsque  $m = 1$ .

---

Propriétés :

- $m = 1$  : convergence *linéaire*  
i.e. nombre constant de décimales gagnées à chaque étape
- $m > 1$  : convergence *superlinéaire*  
i.e. nombre croissant de décimales gagnées à chaque étape

---

Par exemple,  $m = 2$  :

- convergence quadratique
- le nombre de décimales correctes double à chaque itération

# Méthode du point fixe

## Problème à résoudre

Adaption du problème à la méthode du point fixe :

$$f(x) = 0 \Leftrightarrow g(x) = x$$


avec,

$$g(x) = f(x) + x$$

La suite est définie telle que :

$$\begin{cases} x_0 \in I \\ x_{n+1} = g(x_n) \end{cases}$$

---

 Restriction à certains types de fonctions : **les fonctions contractantes**

# Méthode du point fixe

## Fonction contractante

Une fonction  $f$  est contractante sur un intervalle  $I$  s'il existe un réel  $0 < k < 1$  tel que :

$$\forall (x, y) \in I, \quad |f(x) - f(y)| \leq k|x - y|$$

## Théorème du point fixe

Une fonction  $f : I \rightarrow I$ ,  $k$ -contractante sur  $I$ , admet un unique point fixe sur cet intervalle. De plus, quel que soit  $x_0 \in I$ , la suite des itérées  $f^k(x)$  converge vers ce point fixe.

## Propriété

Si une fonction définie et dérivable sur un intervalle  $\mathcal{I}$  et telle que pour tout réel  $x \in \mathcal{I}$  on a  $|f'(x)| < 1|$  alors  $f$  est contractante.

i.e. les conditions de convergence sont liées à la valeur de la dérivée

# Méthode du point fixe

## Point fixe attractif

Soit  $a$ , un point fixe de  $f$  tel que,

$$|f'(a)| < 1|,$$

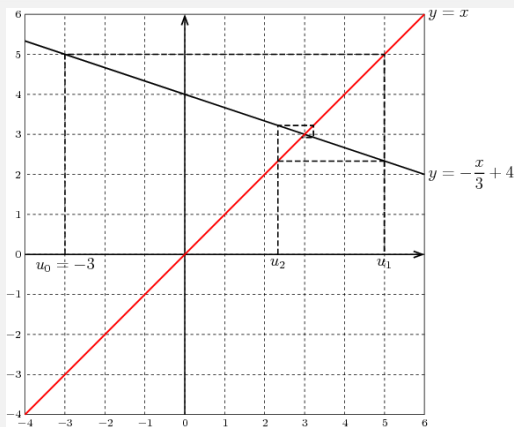
alors, le point fixe est **attractif**.

→ La convergence est linéaire

$$f'(a) = 0,$$

le point fixe est **super-attractif**.

→ La convergence est quadratique



# Méthode du point fixe

## Point fixe répulsif

Soit  $a$ , un point fixe de  $f$  tel que,

$$|f'(a)| > 1|,$$

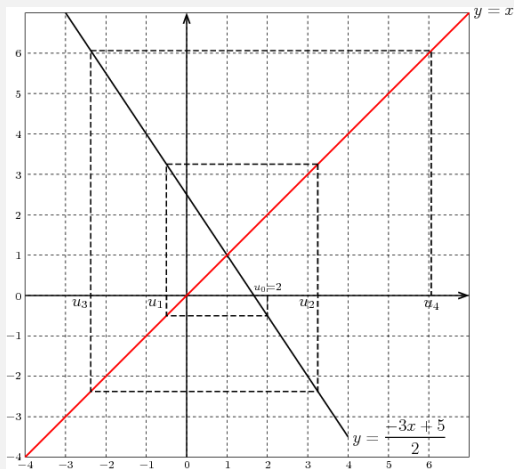
alors, le point fixe est **répulsif**.  
(utiliser  $f^{-1}$ )

---


$$|f'(a)| = 1|,$$

le point fixe est **douteux**.

→ La convergence n'est pas assurée




# Méthode du point fixe

## Méthode du point fixe

Il est rare de trouver des fonctions contractantes sur l'ensemble de leur intervalle de définition. Par exemple, résoudre  $x^2 - x = 0$  peut se faire avec :

- $g(x) = x^2$  :  $x_\infty$  tend vers 0 quand  $x_0 = 0.8$  et diverge pour  $x_0 = 1.2$
- $g(x) = \sqrt{x}$  :  $x_\infty$  tend vers 1 quand  $x_0 = 0.5$  et diverge pour  $x_0 = 2$

---

 On choisie  $g$  et  $x_0$  dans le but

- d'améliorer la convergence
- de converger vers **une** solution dans le cas de racines multiples

en fonction

- des propriétés de  $f$  (dérivabilité, monotonie)
- de la connaissance de  $f$  (peut-on isoler les solutions?)

# Dichotomie

## Présentation

- méthode empirique
- adaptée à tout type de fonction continue

## Définition

Soient une fonction  $f : I \rightarrow \mathbb{R}$  et  $a, b \in I$  avec  $a < b$  tels que :

$$f(a).f(b) = 0$$

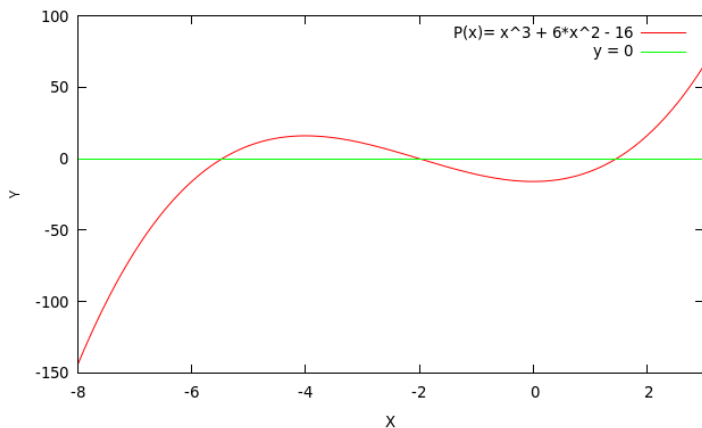
(i.e. les valeurs sont de signes opposés)

alors, le théorème des valeurs intermédiaires démontre que :

$$\exists x \in [a; b] \mid f(x) = 0$$

(i.e. la racine est cloisonnée dans l'intervalle  $[a; b]$ )

# Dichotomie





# Dichotomie

## Algorithme de bisection

Algorithme itératif où à chaque itération :

→ Calcul de  $c = \frac{a+b}{2}$  et évaluation de  $f(c)$  :

- $c$  est racine de  $f$  : arrêt du calcul
- $f(c)$  est de même signe que  $f(a)$  :  $\exists x \in [c; b] \mid f(x) = 0$
- $f(c)$  est de même signe que  $f(b)$  :  $\exists x \in [a; c] \mid f(x) = 0$

## Convergence

A l'étape  $k$ , n'importe quel point  $x_k \in [a_k; b_k]$  est une approximation de la racine  $x$  comprise dans cet intervalle avec une précision de :

$$|x_k - x| \leq \frac{1}{2^k} |b - a|$$

→ Convergence linéaire

# Méthode de la sécante

## Présentation

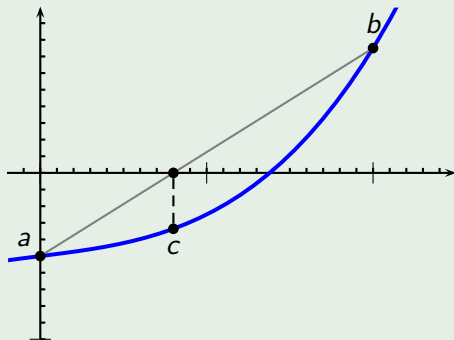
- hypothèse :  $f$  est « presque » linéaire autour de sa racine

Soient un intervalle  $[a; b]$  qui cloisonne une racine de  $f$  et les points  $X$  et  $Y$  de coordonnées  $(a; f(a))$  et  $(b; f(b))$ . La droite  $XY$  coupe l'axe des abscisse en un point  $(c, 0)$  tel que :

$$c = a - \frac{b - a}{f(b) - f(a)} f(a)$$

# Méthode de la sécante

## Exemple



$$f(x) = \frac{x^3}{8} + \frac{x}{8} - \frac{1}{2}$$

# Méthode de la sécante

## Méthodes

### Méthode de la sécante

$$\begin{cases} x_0 = a, x_1 = b \\ x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \end{cases}$$

 Ne converge pas forcément !

---

### Méthode de la fausse position

- idem à la dichotomie en remplaçant le milieu de l'intervalle par c

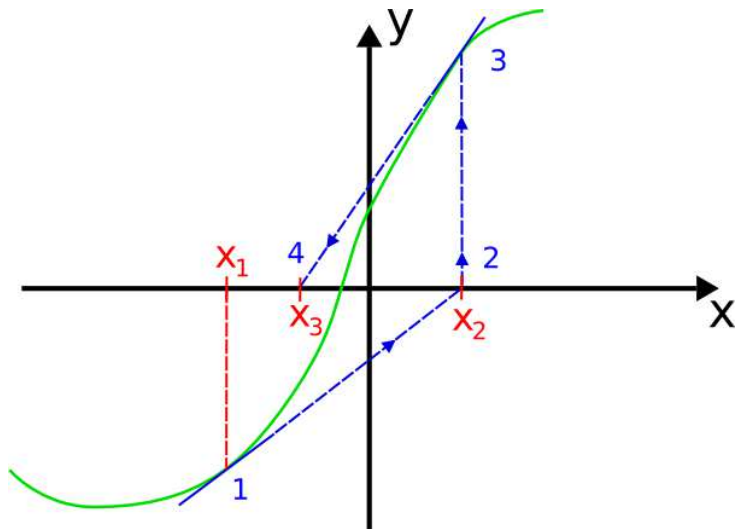
---

### Méthode de Ridder, Méthode de Brent

## Remarques : calcul des pentes des fonctions

- Pas de calculs de dérivées mais ...
- problèmes de stabilité numérique :  $\frac{1}{f(x_n) - f(x_{n-1})}$

# Méthode de Newton



# Méthode de Newton

## Principe

Soient  $f$ , une fonction  $\mathcal{C}_1$  sur  $\mathcal{I}$ , et  $x \in \mathcal{I}$ , une racine de  $f$ , on cherche  $h$  tel que :

$$f(x) = f(x_k + h) = 0$$

Développement de Taylor à l'ordre 1 :

$$f(x) = f(x_k + h) = f(x_k) + hf'(x_k) + \underbrace{o(h)}_{< \epsilon}$$

alors,

$$h = - \frac{f(x_k)}{f'(x_k)}$$

## Méthode itérative

$$\begin{cases} x_0 \in \mathcal{I} \\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \end{cases}$$

# Méthode de Newton

## Convergence

Supposons que  $f$  soit  $\mathcal{C}_2$ . Soit  $x$  une racine de  $f$ , alors, à chaque itération  $k$ , on a :

$$\epsilon_k = x_k - x$$

donc,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \Rightarrow \epsilon_{k+1} = \epsilon_k - \frac{f(x_k)}{f'(x_k)}$$

or,

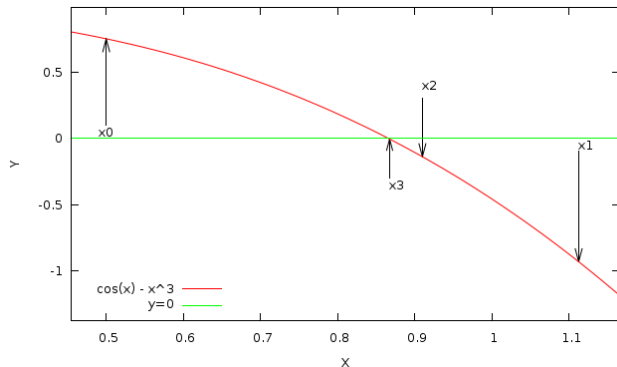
$$\begin{cases} f(x_k) = f(x + \epsilon_k) = \underbrace{f(x)}_{=0} + \epsilon_k f'(x) + \frac{\epsilon_k^2}{2} f''(x) + o(\epsilon_k^2) \\ f'(x_k) = f'(x + \epsilon_k) = f'(x) + \epsilon_k f''(x) + o(\epsilon_k) \end{cases}$$

ainsi,

$$\epsilon_{k+1} = \epsilon_k - \frac{f(x_k)}{f'(x_k)} = \epsilon_k^2 \times \frac{f''(x)}{2f'(x)} + o(\epsilon_k^2)$$

Convergence quadratique : à chaque itération, le nombre de décimales est double.

# Méthode de Newton



$$\begin{aligned}
 x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} = 0.5 - \frac{\cos(0.5) - 0.5^3}{-\sin(0.5) - 3 \times 0.5^2} = 1.112141637097 \\
 x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} = \vdots = \underline{0.909672693736} \\
 x_3 &= \vdots = \vdots = \underline{0.867263818209} \\
 x_4 &= \vdots = \vdots = \underline{0.865477135298} \\
 x_5 &= \vdots = \vdots = \underline{0.865474033111} \\
 x_6 &= \vdots = \vdots = \underline{0.865474033102}
 \end{aligned}$$



# Méthode de Newton - Généralisation

## Problème

Soit une fonction  $F, \mathbb{R}^n \rightarrow \mathbb{R}^m$ . On cherche  $X$  tel que :

$$F(X) = 0$$

Solutions envisageables :

- méthodes dicotomiques : beaucoup de modifications, peu efficace (théorème des valeurs intermédiaires en dimension  $\geq 2$ )
- extension de la méthode de newton, *Newton-Raphson*

# Méthode de Newton - Généralisation

## Principe (1/2)

En séparant les dimensions de  $F$ , on peut écrire le problème comme

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ \vdots \\ f_m(x_1, \dots, x_n) = 0 \end{cases}$$

Soient  $X = [x_1, \dots, x_n]$  et  $\Delta$  un vecteur de  $\mathbb{R}^n$ , alors, en supposant que les  $f_k$  sont  $\mathcal{C}_1$ ,

$$f_k(X + \Delta) = f_k(X) + \sum_{i=1}^n \frac{\partial f_k}{\partial x_i}(X) * \Delta_i + o(\Delta)$$

# Méthode de Newton - Généralisation

## Principe (2/2)

Matrice Jacobienne, matrice des dérivées de la fonction  $F$  dans la base des  $x_i$ ,

$$J(X) = \left( \frac{\partial f_i}{\partial x_j}(X) \right)_{\substack{i \in [1; m] \\ j \in [1; n]}}$$

En utilisant cette notation matricielle, l'équation devient :

$$F(X + \Delta) = F(X) + J(X).\Delta + \underbrace{o(\Delta)}_{< \epsilon}$$

On cherche  $\Delta$  tel que  $F(X + \Delta) = 0$ , soit

$$J(X).\Delta = -F(X)$$

⇒ Résoudre un système linéaire

# Méthode de Newton - Généralisation

## Propriétés de la méthode de Newton-Raphson

Identique à Newton (cas 1D) :

- Convergence quadratique
- Mais convergence non assurée, dépend du choix de  $x_0$   
(problème pratique courant)
- Complexité élevée si la dérivée n'est pas définie analytiquement  
(méthode des différences finies  $\Rightarrow$  complexité++)

# Méthode de Newton-Raphson - Optimisation

## Backtracking

$$\begin{cases} X_0 \in \mathcal{R}^n \\ X_{k+1} = X_k - \text{step} \cdot \Delta \end{cases}$$

La direction de Newton est une direction de descente mais qui nécessite parfois une correction de norme pour éviter les sauts autour de la racine.

---

### Algorithme 1: Newton-Raphson backtracking

---

```
step = 1.0;  
nf = ||F(X)||;  
while ||F(X + step * Δ)|| > nf do  
  | step = step * 2/3  
end  
X = X + step * dx;
```

---

⇒ Méthode de Bairstow, points de Lagrange

# Racines de polynômes

## Problème

Méthodes de résolution adhoc pour trouver les racines de polynômes de faible degré :

- degré 2, discriminant
- degré 3, Formules de Cardan :  $x^3 + px = q$ ,  $\Delta = \frac{q^2}{4} + \frac{p^3}{27}$
- degré 4, Formules de Ferrari :  $x^4 + px^2 + qr + r = 0$

Méthode générique pour degré  $\geq 5$  ?

## Algorithme

Deux étapes à envisager :

- 1 localisation des racines
- 2 recherche des racines

# Racines de polynômes

## Localisation des racines d'un polynôme

- 1 Déterminer le nombre de racines (réelles, complexes)
- 2 Borner les racines

# Racines de polynômes

## Nombre de racines d'un polynôme

### *Règle des signes de Descartes*

Soit un polynôme  $P(x) = a_n * x^n + \dots + a_0$ , le nombre de racines réelles positives  $r_+$  de  $P$  est au plus le nombre de changement de signes de la suite  $\{a_n, \dots, a_0\}$ .

(resp. nombre de racines réelles négative  $r_-$  avec  $P(-x)$ )

### Exemple

$$\begin{aligned}Q(x) &= x^3 + 3x^2 - x - 2 \\Q(-x) &= -x^3 + 3x^2 + x - 2 \\r_+ &= 1 \text{ et } r_- = 2?\end{aligned}$$



# Racines de polynômes

## Racines multiples

*Gauss (1828) :*

Si l'on compte les racines avec leur multiplicité, alors le nombre de racines positives a la même parité que  $r_+$ , soit  $r_+$  ou  $r_+ - 2$  ou  $r_+ - 4, \dots$

### Exemple

$$Q(x) = x^3 + 3x^2 - x - 2$$

$$r_+ = 1 \text{ et } r_- = 2?$$

Vérifions qu'il n'y a pas de racines multiples :

$$\begin{cases} Q(-1) = 1 \\ \lim_{x \rightarrow -\infty} = -\infty \end{cases}$$

Donc,  $r_- \geq 1$  soit,  $r_+ = 1$  et  $r_- = 2$ .

# Racines de polynômes

## Borner les racines

Suite de *Sturm* adaptée au polynôme  $P$ .

$$\begin{cases} P_0(x) &= P(x) \\ P_1(x) &= P'(x) \\ P_{i+1}(x) &= -P_{i-1} \bmod P_i \end{cases}$$

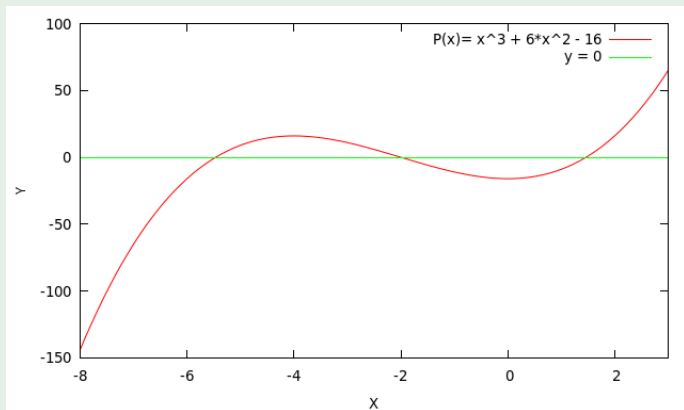
Soient la suite  $\{P_0, \dots, P_s\}$  avec  $P_s = \text{constante}$  et  $V(x)$  le nombre de changement de signe de cette suite.

Le nombre de racines réelles de  $P$  dans  $[a; b]$  est  $V(a) - V(b)$ .

# Racines de polynômes

## Exemple

$$P(x) = x^3 + 6x^2 - 16$$



# Racines de polynômes

## Exemple

Construction de la suite :

$$\begin{cases} P_0(x) &= x^3 + 6x^2 - 16 \\ P_1(x) &= 3x^2 + 12x \\ P_{i+1}(x) &= -P_{i-1} \bmod P_i \end{cases}$$

Division euclidienne polynomiale :

$$\begin{array}{r|l} x^3 + 6x^2 & - 16 \\ & 3x^2 + 12x \\ & \hline & - - - - - \end{array}$$

# Racines de polynômes

## Exemple

Construction de la suite :

$$\begin{cases} P_0(x) = x^3 + 6x^2 - 16 \\ P_1(x) = 3x^2 + 12x \\ P_{i+1}(x) = -P_{i-1} \bmod P_i \end{cases}$$

Division euclidienne polynomiale :

$$\begin{array}{r|l} \begin{array}{r} x^3 + 6x^2 - 16 \\ -(x^3 + 4x^2) \\ \hline 2x^2 - 16 \\ -(2x^2 + 8x) \\ \hline -8x - 16 \end{array} & \begin{array}{l} 3x^2 + 12x \\ - - - - - \\ \frac{1}{3}x + \frac{2}{3} \end{array} \end{array}$$

donc,  $P_2(x) = 8x + 16$ , le reste de la division de  $P_0$  par  $P_1$ .

# Racines de polynômes

## Exemple

$$\begin{cases} P_0(x) = x^3 + 6x^2 - 16 \\ P_1(x) = 3x^2 + 12x \\ P_2(x) = 8x + 16 \\ P_3(x) = 12 \end{cases}$$

- Rappel :  $V(x)$  est le nombre de changement de signes de la suite

$$\{P_0(x), P_1(x), P_2(x), P_3(x)\}$$

- On remarque que  $V(-7) = 3$  et  $V(2) = 0$ . Donc le nombre de racines dans l'intervalle  $[-7; 2]$  est :  $V(-7) - V(2) = 3$ .
- Les racines de  $P$  sont toutes dans l'intervalle  $[-7; 2]$ .
- On peut facilement prolonger l'algorithme pour borner toutes les racines !

# Racines de polynômes

## Algorithme

- ❶ localisation des racines
  - nombre de racines
  - borner les racines
- ❷ **recherche des racines** : Newton / Newton-Raphson

## Méthode de recherche générale

- Déflation
  - Trouver une racine  $a$  de  $P$
  - Calculer  $Q$  tel que  $Q(x) = \frac{P}{x-a}$
  - Recommencer avec  $Q$
- Bairstow
  - Application de Newton-Raphson en dimension 2
  - Calculer les facteurs irréductibles de degré 2

# Recherche d'extremum local

## Problème d'optimisation

Recherche de minimas ou maximas locaux tel que

$$f(x) = \min\{f(y), y \in \mathcal{B}(x)\}$$

Le minimum est global (resp. local) si  $\mathcal{B}(x) = \mathcal{D}_f$  (resp.  $\mathcal{B}(x) \subset \mathcal{D}_f$ ).

---

Concept :

- Regarder les variations locales
- Suivre les lignes de descente

---

Méthodes :

- Cloisonnement d'un extremum (cas 1D)
- Optimisations des algorithmes de descente, directions et pas (cas ND)



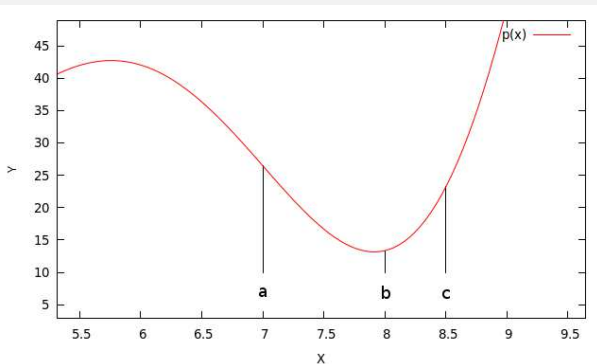
# Recherche d'extremum local

## Problème d'optimisation - Cas 1D

*Cloisonnement du minimum* : Soient  $a, b$  et  $c \in \mathcal{D}_f$  tels que  $a < b < c$ , alors si

$$\begin{cases} f(a) > f(b) \\ f(c) > f(b) \end{cases}$$

$\Rightarrow$  Il existe un minimum pour  $f$  dans  $[a; c]$



# Recherche d'extremum local

## Problème d'optimisation - Cas 1D

*Cloisonnement du minimum* : Soient  $a$ ,  $b$  et  $c \in \mathcal{D}_f$  tels que  $a < b < c$ , alors si

$$\begin{cases} f(a) > f(b) \\ f(c) > f(b) \end{cases}$$

$\Rightarrow$  Il existe un minimum pour  $f$  dans  $[a; c]$

---

Méthode dichotomique, en choisissant  $x$  appartenant au plus grand intervalle  $[a; b]$  ou  $[b; c]$  :

Si  $x \in [b; c]$  alors

- $f(x) > f(b)$ ,  $(a, b, x)$
- $f(x) < f(b)$ ,  $(a, x, c)$

Si  $x \in [a; b]$  alors

- $f(x) > f(a)$ ,  $(x, b, c)$
- $f(x) < f(b)$ ,  $(a, x, c)$

# Recherche d'extremum local

## Problème d'optimisation - Cas 1D

*Cloisonnement du minimum* : Soient  $a$ ,  $b$  et  $c \in \mathcal{D}_f$  tels que  $a < b < c$ , alors si

$$\begin{cases} f(a) > f(b) \\ f(c) > f(b) \end{cases}$$

$\Rightarrow$  Il existe un minimum pour  $f$  dans  $[a; c]$

## Problème d'optimisation - Cas N-D

- Méthode du simplexe
- Méthode du gradient ou du gradient conjugué

- 5 **Intégration numérique**
  - Interpolation polynomiale
  - Intégration numérique
  - Applications

# Introduction

## Présentation

Approximation d'une fonction quelconque par une fonction simple

→ Approximation polynomiale

Calculs approchés pour obtenir des informations impossibles à obtenir de manière formelle

→ Intégration numérique

# Présentation

## Définition

Soit un ensemble de couples  $\{(x_1, f(x_1)), \dots, (x_n, f(x_n))\} \subset \mathbb{R} \times \mathbb{R}$ . Un polynôme d'interpolation de  $f$  aux points  $\{x_1, \dots, x_n\}$  vérifie :

$$P(x_i) = f(x_i), \quad \forall i \in [1; n]$$

## Problématiques

- 1 la fonction  $f$  est quelconque
- 2 choix des points d'interpolation
- 3 valeur pour  $P(x)$ ,  $x_i < x < x_{i+1}$

# Existence et unicité

## Théorème

Soit un ensemble de couples  $\{(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))\} \subset \mathbb{R} \times \mathbb{R}$ , il existe un unique polynôme  $P$ , de degré  $\leq n$ , tel que :

$$P(x_i) = f(x_i), \quad \forall i \in [1; n]$$

→ Polynôme interpolateur de Lagrange

# Existence et unicité

## Preuve - Construction (1/2)

Soit  $P$  un polynôme de degré  $n$ . Soit  $X_P$  le vecteur, de dimension  $n + 1$ , constitué des valeurs  $\{P(x_0), P(x_1), \dots, P(x_n)\}$ . Ainsi, la somme de deux polynômes  $P$  et  $Q$  est équivalente à :

$$\forall (P, Q) \in \mathcal{P}_n, \lambda \in \mathbb{R} \quad X_{P+Q} = X_P + X_Q \quad X_{\lambda P} = \lambda X_P$$

→ But : Trouver un polynôme  $P \in \mathcal{P}_n$  tel que  $X_P$  soit le vecteur composé des  $f(x_i)$ .

*nb :  $\mathcal{P}_n$  représente l'ensemble des polynômes de degré  $\leq n$ .*



## Existence et unicité

## Preuve - Construction (2/2)

Soit une base polynomiale définie par des polynômes  $L_i^n$  telle que :

$$\begin{cases} L_i^n(x_k) = 0, & \forall k \in [0; n], k \neq i \\ L_i^n(x_k) = 1, & k = i \end{cases}$$

On peut calculer facilement le polynôme  $L_i^n$  avec :

$$L_i^n(x) = \frac{\overbrace{(x - x_0)(x - x_1) \dots (x - x_n)}^{\text{sans le facteur } (x - x_i)}}{\underbrace{(x_i - x_0)(x_i - x_1) \dots (x_i - x_n)}_{\text{sans le facteur } (x_i - x_i)}} = \frac{\prod_{k \neq i} (x - x_k)}{\prod_{k \neq i} (x_i - x_k)}$$

Le polynôme  $P(x) = \sum_{i=0}^n f(x_i) L_i^n(x)$  est donc une solution au problème initial. De plus, ce polynôme est unique ( $P(x_i) - Q(x_i) = 0, \forall i \Rightarrow P = Q$ ).

# Existence et unicité

## Complexité

- Evaluation d'un polynôme  $L_i^n : \mathcal{O}(n^2)$
- Evaluation du polynôme  $P : n \times \mathcal{O}(n^2)$ , soit  $\mathcal{O}(n^3)$

→ Peu efficace en pratique !

## Formule d'erreur

$$\|f - P\| \leq \frac{1}{(n+1)!} \|\pi_{n+1}\| \times \|f^{n+1}\|, \text{ avec } \pi_{n+1}(x) = \prod_{i=0}^n (x - x_i)$$

L'erreur sur  $f$  dépend

- des variations de  $f$  (cf. dérivée  $n+1$ -ème) → incontrôlable
- de la répartition des points (cf.  $\pi_{n+1}$ ) → contrôlable

# Méthode des différences divisées

## Polynômes de Newton

Base polynomiale,  $e_i$ , telle que

$$e_k(x) = \prod_{i=0}^{k-1} (x - x_i), \quad k = 1, \dots, n$$

avec,  $e_0(x) = 1$ .

Polynôme interpolateur,  $P_n$ , dans la base de Newton :

$$P_n(x) = \sum_{k=0}^n \alpha_k e_k(x)$$

Evaluation des  $\alpha_k$  par la *Méthode des différences divisées*.

# Méthode des différences divisées

## Evaluation - Ordre 0

Soit le couple  $\{(x_0, f(x_0))\}$ , on a :

$$P_n(x_0) = \sum_{k=0}^n \alpha_k e_k(x_0) = \alpha_0 = f(x_0) = f[x_0]$$

$$\text{notation : } f[x_i] = f(x_i), \quad \forall i = 0, \dots, n$$

→  $f[x_0]$  est appelée **différence divisée d'ordre 0**.

# Méthode des différences divisées

## Evaluation - Ordre 1

Soit les couples  $\{(x_0, f(x_0)), (x_1, f(x_1))\}$ , on a :

$$\begin{aligned}P_n(x_1) &= \sum_{k=0}^n \alpha_k e_k(x_1) \\&= \alpha_0 + \alpha_1(x_1 - x_0) \\&= f[x_0] + \alpha_1(x_1 - x_0) \\&= f[x_1]\end{aligned}$$

d'où,

$$\alpha_1 = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1]$$

→  $f[x_0, x_1]$  est appelée **différence divisée d'ordre 1**.

# Méthode des différences divisées

## Evaluation - Ordre 2

Soit les couples  $\{(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))\}$ , on a :

$$\begin{aligned}P_n(x_2) &= \sum_{k=0}^n \alpha_k e_k(x_2) \\&= \alpha_0 + \alpha_1(x_2 - x_0) + \alpha_2(x_2 - x_0)(x_2 - x_1) \\&= f[x_0] + f[x_0, x_1](x_2 - x_0) + \alpha_2(x_2 - x_0)(x_2 - x_1) \\&= f[x_2]\end{aligned}$$

d'où,

$$\begin{aligned}\alpha_2(x_2 - x_0) &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} - f[x_0, x_1] \\&= f[x_1, x_2] - f[x_0, x_1]\end{aligned}$$

donc,

$$\alpha_2 = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2]$$

→  $f[x_0, x_1, x_2]$  est appelée **différence divisée d'ordre 2**

# Méthode des différences divisées

## Evaluation - Ordre $k$

Soit les couples  $\{(x_0, f(x_0)), \dots, (x_k, f(x_k))\}$ , on obtient, par récurrence :

$$\alpha_k = \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0} = f[x_0, \dots, x_k]$$

→  $f[x_0, \dots, x_k]$  est appelée **différence divisée d'ordre  $k$**

# Méthode des différences divisées

## Evaluation - Méthode des différences divisées

En pratique, pour une évaluation à l'ordre 3 par exemple, on a besoin des quantités suivantes :

$$\begin{bmatrix} x_0 & f[x_0] & & & \\ x_1 & f[x_1] & f[x_0, x_1] & & \\ x_2 & f[x_2] & f[x_1, x_2] & f[x_0, x_1, x_2] & \\ x_3 & f[x_3] & f[x_2, x_3] & f[x_1, x_2, x_3] & f[x_0, x_1, x_2, x_3] \end{bmatrix}$$

puisque,

$$\alpha_3 = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} = f[x_0, x_1, x_2, x_3]$$



# Méthode des différences divisées

## Polynôme d'interpolation de Newton à l'ordre $n$

Le polynôme d'interpolation de Newton à l'ordre  $n$  s'écrit donc à l'aide des différences divisées successives tel que :

$$P_n(x) = f[x_0] + \sum_{k=1}^n f[x_0, \dots, x_k] e_k(x)$$

Avantages :

- Complexité :
  - Calculs de  $e_k$  :  $\mathcal{O}(n^2)$  opérations
  - Evaluation : Schéma de Horner,  $\mathcal{O}(n)$  opérations
- Ajout d'un nouveau point facilement : une itération supplémentaire ( $\mathcal{O}(n)$ )

# Choix des points d'interpolation

## Interpolation uniforme

Interpolation uniforme d'une fonction  $f$  sur un intervalle  $[a; b]$ . On définit les  $x_i$  tels que :

$$h = \frac{b-a}{n}, \quad \forall i \in [0; n], \quad x_i = a + i * h$$

L'erreur bornée devient alors :

$$\|f - P\| \leq \frac{h^{n+1}}{(n+1)} \|f^{n+1}\|$$

→ Pas forcément le meilleur choix !

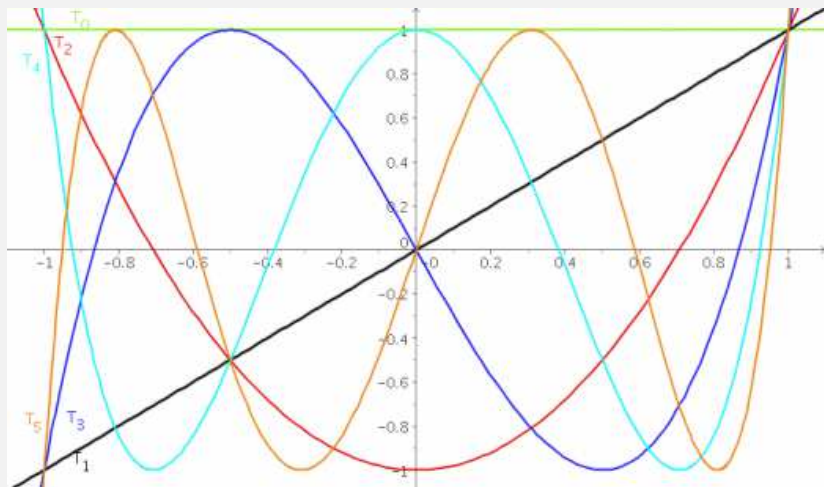
# Choix des points d'interpolation

## Polynôme de Tchebychev

Les points d'interpolation qui minimisent les erreurs d'approximation du polynôme  $P_n$  sont exactement les racines du polynôme de Tchebychev.

# Choix des points d'interpolation

## Polynôme de Tchebychev



# Choix des points d'interpolation

## Polynôme de Tchebychev

$$T_n(x) = \cos(n * \arccos(x)), \quad x \in [-1; 1]$$

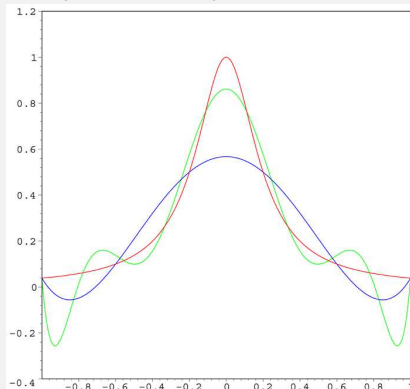
$$\begin{cases} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2 * x * T_n(x) - T_{n-1}(x) \end{cases}$$

Racines :  $x_k^n = \cos\left(\frac{2k+1}{2n}\pi\right)$ ,  $k = 0, 1, \dots, n-1$ .

→ nécessite la transformation de  $[-1; 1]$  vers  $[a; b]$ .

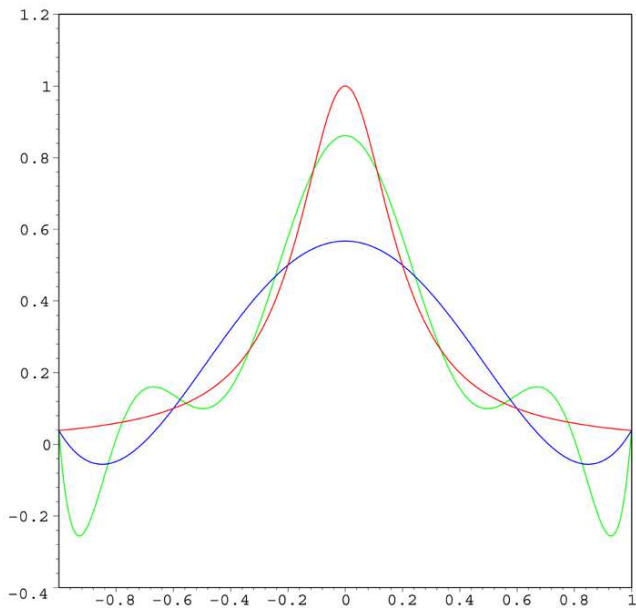
## Phénomène de Runge

Oscillations autour des points d'interpolation



rouge :  $f(x)$ , courbe bleue  $P_5$ , courbe rouge  $P_9$

# En pratique



# En pratique

## Interpolation par morceaux

Soient un ensemble de réels tels que  $x_0 < x_1 < \dots < x_n$ . Pour chaque intervalle  $[x_i; x_{i+1}]$ , on cherche un polynôme  $P_i$  tel que

$$\begin{cases} P_i(x_i) &= f(x_i) \\ P_i(x_{i+1}) &= f(x_{i+1}) \\ P'_i(x_i) &= f'(x_i) \\ P'_i(x_{i+1}) &= f'(x_{i+1}) \end{cases}$$

→ Interpolation par spline cubique.



# Intégration numérique

## Principe de calcul

Soit une fonction  $f$ , définie et intégrable sur  $[a; b]$ . On cherche à calculer :

$$\mathcal{I} = \int_a^b f(x) dx$$

On effectue une simplification du problème en ramenant le calcul sur des petits intervalles tel que :

$$\mathcal{I} = \sum_{i=1}^n \int_{\alpha_{i-1}}^{\alpha_i} f(x) dx$$

avec,

$$a = \alpha_0 < \alpha_1 < \dots < \alpha_n = b$$

# Intégration numérique

## Méthode d'approximation

Sur chaque intervalle  $[\alpha_{i-1}; \alpha_i]$  (suffisamment petit), on approxime le calcul par une *formule de quadrature élémentaire* tel que :

$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx (\alpha_i - \alpha_{i-1}) \times \left( \sum_{j=0}^{l_i} \omega_{i,j} f(\lambda_{i,j}) \right)$$

avec,

$$\lambda_{i,j} \in [\alpha_{i-1}; \alpha_i] \text{ et } \sum_{j=0}^{l_i} \omega_{i,j} = 1$$

→ Combinaison linéaire de valeurs de  $f$  dans  $[\alpha_{i-1}; \alpha_i]$

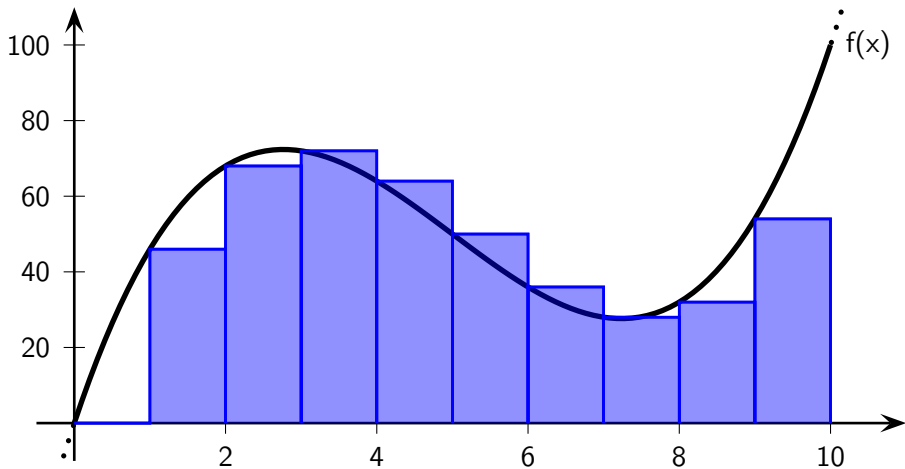
# Intégration numérique

## Ordre de convergence

Une formule de quadrature est dite d'ordre  $n$  lorsqu'elle est exacte pour tous les polynômes de  $\mathcal{P}_n$  et inexacte pour au moins un polynôme de  $\mathcal{P}_{n+1}$ .

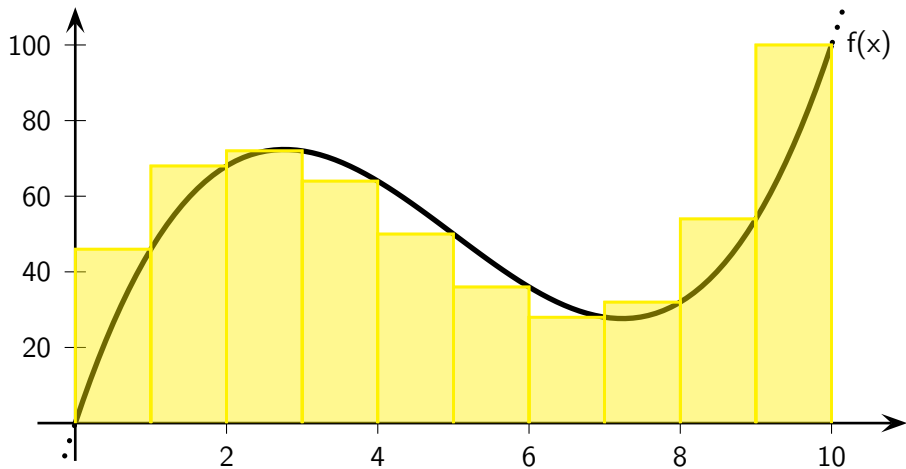
nb : toute méthode est au moins d'ordre 0 car au moins exacte pour des fonctions constantes.

## Intégration numérique - Méthodes simples



Méthode des rectangles (à gauche)

# Intégration numérique - Méthodes simples



Méthode des rectangles (à droite)

# Intégration numérique - Méthodes simples

## Méthode des rectangles - Ordre 0

→ Méthode des rectangles à gauche : un point d'interpolation en  $\alpha_{i-1}$

$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx (\alpha_i - \alpha_{i-1}) \times f(\alpha_{i-1})$$

→ Méthode des rectangles à droite : un point d'interpolation en  $\alpha_i$ .

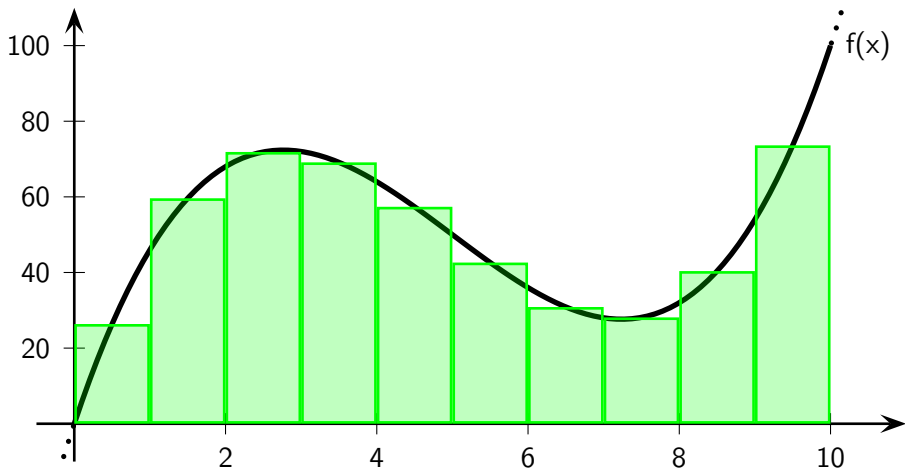
$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx (\alpha_i - \alpha_{i-1}) \times f(\alpha_i)$$

Ainsi,

$$\begin{aligned} \mathcal{I} = \int_a^b f(x) dx &\approx h \left( \sum_{k=0}^{n-1} f(a + kh) \right) \quad (\text{gauche}) \\ &\approx h \left( \sum_{k=1}^n f(a + kh) \right) \quad (\text{droite}) \end{aligned}$$

suivant un échantillonnage uniforme avec  $h = \frac{b-a}{n}$  ( $n$  partitions).

# Intégration numérique - Méthodes simples



Méthode du point milieu

# Intégration numérique - Méthodes simples

## Méthode du point milieu - Ordre 1

→ Un point d'interpolation en  $\frac{\alpha_{i-1} + \alpha_i}{2}$

Ainsi,

$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx (\alpha_i - \alpha_{i-1}) \times f\left(\frac{\alpha_{i-1} + \alpha_i}{2}\right)$$

Donc,

$$\mathcal{I} = \int_a^b f(x) dx \approx h \left( \sum_{k=0}^{n-1} f\left(a + kh + \frac{h}{2}\right) \right)$$

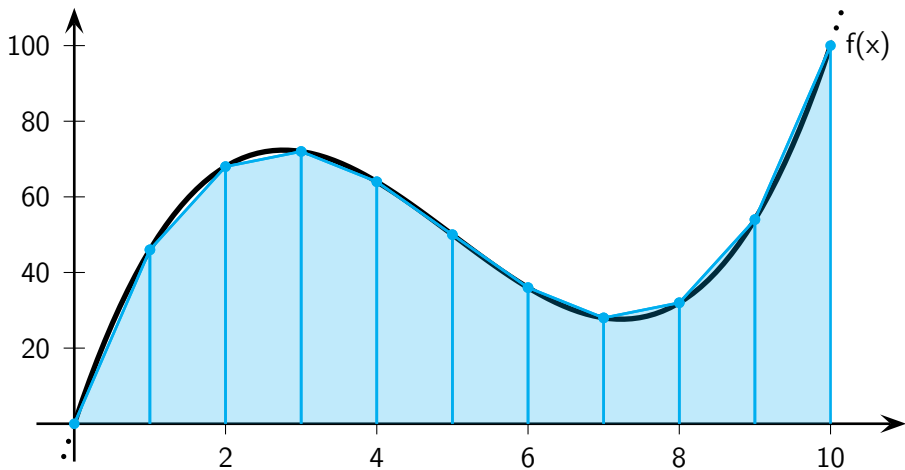
suivant un échantillonnage uniforme avec  $h = \frac{b-a}{n}$  ( $n$  partitions).

---

nb : méthode exacte pour des fonctions affines, donc ordre 1.



# Intégration numérique - Méthodes par interpolation linéaire



Méthode des trapèzes

# Intégration numérique - Méthodes par interpolation linéaire

## Méthode des trapèzes - Ordre 1

→ Deux points d'interpolation en  $\alpha_{i-1}$  et  $\alpha_i$

Ainsi,

$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx (\alpha_i - \alpha_{i-1}) \times \frac{f(\alpha_{i-1}) + f(\alpha_i)}{2}$$

donc,

$$\mathcal{I} = \int_a^b f(x) dx \approx h \left( \frac{f(a) + f(b)}{2} + \sum_{k=1}^{n-1} f(a + kh) \right)$$

suivant un échantillonnage uniforme avec  $h = \frac{b-a}{n}$  ( $n$  partitions).

---

nb : approximation d'une droite dans une partition, donc ordre 1.

# Intégration numérique - Méthodes par interpolation linéaire

## Optimisation de la méthode des rectangles

Dans le cas d'une subdivision uniforme, on a les relations suivantes :

$$\mathcal{I}_{rec_g} = \mathcal{I}_{trap} + \frac{h}{2} * (f(a) - f(b))$$

$$\mathcal{I}_{rec_d} = \mathcal{I}_{trap} + \frac{h}{2} * (f(b) - f(a))$$

→ Evaluation d'une méthode d'ordre 1 à partir de méthodes d'ordre 0.

# Intégration numérique - Méthodes d'ordre supérieur

## Méthodes d'ordre supérieur

Idée : Interpoler chaque partition par un polynôme de degré,  $l$ , quelconque tel que

$$P_l(x) = \sum_{k=0}^l f(\lambda_k) L_k(x), \quad x \in [\alpha_{i-1}; \alpha_i]$$

et donc,

$$\int_{\alpha_{i-1}}^{\alpha_i} f(x) dx \approx \int_{\alpha_{i-1}}^{\alpha_i} P_l(x) dx$$

→ Méthode de *Newton-Cotes*

# Intégration numérique - Méthodes d'ordre supérieur

## Méthodes d'ordre supérieur

Considérons le cas  $l = 2$ , les points d'interpolation sont :

$$(\alpha_{i-1}, f(\alpha_{i-1})), (\alpha_{i-0.5}, f(\alpha_{i-0.5})), (\alpha_i, f(\alpha_i))$$

On effectue un changement de variables affine pour ramener l'intégrale dans un intervalle plus « simple », l'intervalle  $[-1; 1]$ , comme suit :

$$\mathcal{I} = \int_{\alpha_{i-1}}^{\alpha_i} P(t) dt = \frac{\alpha_i - \alpha_{i-1}}{2} \int_{-1}^1 Q(u) du$$

avec,

$$\begin{cases} u &= \frac{\alpha_i - \alpha_{i-1}}{2} t + \frac{\alpha_i - \alpha_{i-1}}{2} \\ du &= \frac{\alpha_i - \alpha_{i-1}}{2} dt \end{cases}$$

où,

$$\begin{cases} Q(-1) &= f(\alpha_{i-1}) \\ Q(0) &= f(\alpha_{i-0.5}) \\ Q(1) &= f(\alpha_i) \end{cases}$$

# Intégration numérique - Méthodes d'ordre supérieur

## Méthodes d'ordre supérieur

En utilisant les polynômes interpolateurs de Lagrange de degré 2, on a :

$$Q(x) = L_0 f(\alpha_{i-1}) + L_1 f(\alpha_{i-0.5}) + L_2 f(\alpha_i)$$

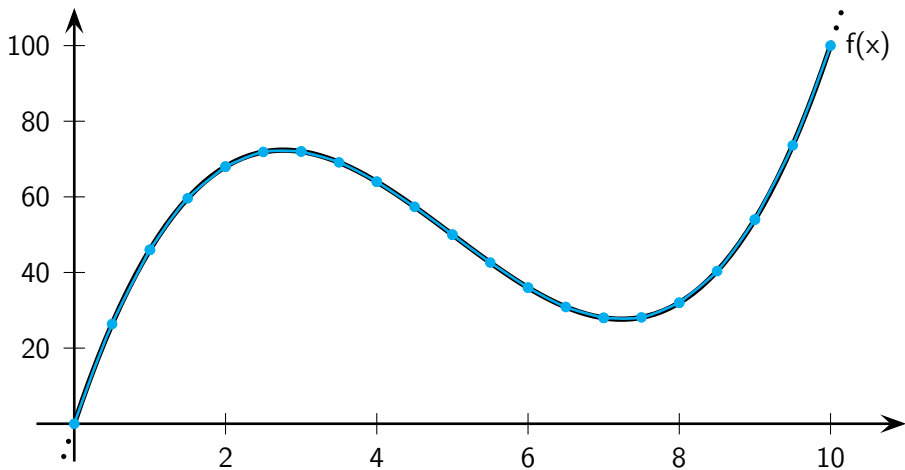
Rappel :

$$L_i^n(x) = \frac{\overbrace{(x-x_0)(x-x_1)\dots(x-x_n)}^{\text{sans le facteur } (x-x_i)}}{\underbrace{(x_i-x_0)(x_i-x_1)\dots(x_i-x_n)}_{\text{sans le facteur } (x_i-x_i)}} = \frac{\prod_{k \neq i} (x-x_k)}{\prod_{k \neq i} (x_i-x_k)}$$

et  $x_0 = -1$ ,  $x_1 = 0$ ,  $x_2 = 1$  (cf. changement de base), donc,

$$L_0 = \frac{x(x-1)}{2}, \quad L_1 = \frac{(x+1)(x-1)}{-1}, \quad L_2 = \frac{(x+1)x}{2}$$

# Intégration numérique - Méthodes d'ordre supérieur



Méthode de Simpson, polynôme d'interpolation

# Intégration numérique - Méthodes d'ordre supérieur

## Méthodes d'ordre supérieur

$$\begin{aligned}\mathcal{I} &= \int_{\alpha_{i-1}}^{\alpha_i} P(t) dt = \frac{\alpha_i - \alpha_{i-1}}{2} \int_{-1}^1 Q(x) dx \\ &= \frac{\alpha_i - \alpha_{i-1}}{2} \int_{-1}^1 \left( \frac{x(x-1)}{2} f(\alpha_{i-1}) + \frac{(x+1)(x-1)}{-1} f(\alpha_{i-0.5}) + \frac{(x+1)x}{2} f(\alpha_i) \right) dx\end{aligned}$$

or,

$$\int_{-1}^{-1} (x-a)(x-b) dx = \frac{2}{3} + 2ab$$

donc,

$$\mathcal{I} = \frac{\alpha_i - \alpha_{i-1}}{2} \left( \frac{1}{3} f(\alpha_{i-1}) + \frac{4}{3} f(\alpha_{i-0.5}) + \frac{1}{3} f(\alpha_i) \right)$$



# Intégration numérique - Méthodes d'ordre supérieur

## Méthode de simpson

→ Méthode de Newton-Cotes pour  $l = 2$  dans le cas uniforme

$$\mathcal{I} = \frac{\alpha_i - \alpha_{i-1}}{2} \left( \frac{1}{3}f(\alpha_{i-1}) + \frac{4}{3}f(\alpha_{i-0.5}) + \frac{1}{3}f(\alpha_i) \right)$$

Les coefficients sont donc :  $\frac{1}{6}$ ,  $\frac{2}{3}$  et  $\frac{1}{6}$ .

---

Ordre de convergence :

- au moins 2 car elle interpole exactement des polynômes de degré  $\leq 2$
- également exacte pour des polynômes de degré 3!
  - différence = multiple de  $(x+1)x(x-1)$
  - la différence est d'intégrale nulle sur  $[-1; 1]$

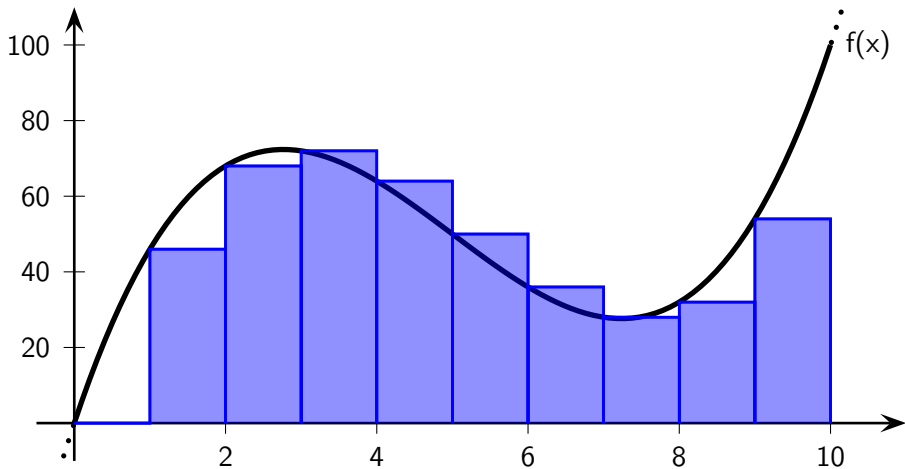
... donc, d'ordre 3.

# Intégration numérique - Méthodes d'ordre supérieur

## Ordre de convergence des méthodes de Newton-cotes

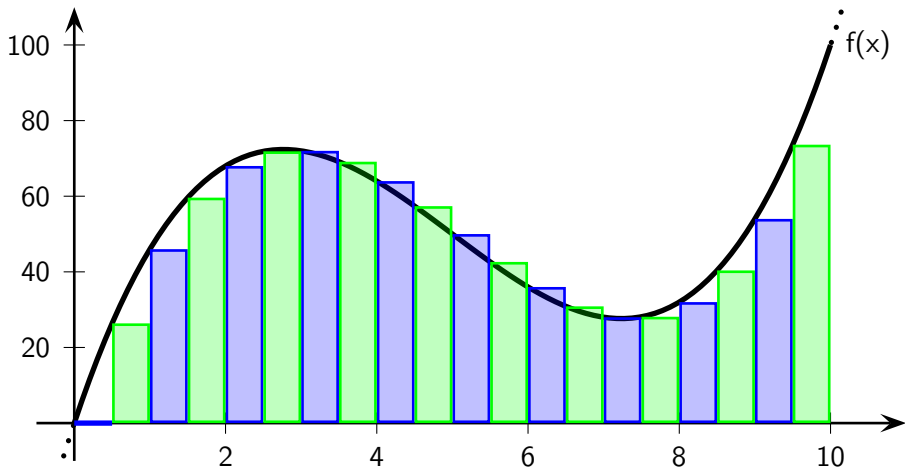
La méthode de quadrature de Newton-cotes par un polynôme de degré  $l$  est d'ordre  $l$  si  $l$  est impair, et  $l + 1$  si  $l$  est paire.

# Méthode générale



Méthode des rectangles (à gauche) pour  $n = 10$

# Méthode générale



Méthode des rectangles (à gauche) pour  $n = 20$

# Méthode générale

## Formule du doublement de pas

Méthode de rectangles :

$$\mathcal{I}_{\text{rect}}(2n) = \frac{1}{2}\mathcal{I}_{\text{rect}}(n) + \frac{h}{2} \sum_{k=0}^{n-1} f\left(a + kh + \frac{h}{2}\right)$$

Méthode des trapèzes :

$$\mathcal{I}_{\text{trap}}(2n) = \frac{1}{2}\mathcal{I}_{\text{trap}}(n) + \frac{h}{2} \sum_{k=0}^{n-1} f\left(a + kh + \frac{h}{2}\right)$$

Méthode du point milieu :

$$\mathcal{I}_{\text{milieu}}(3n) = \frac{1}{3}\mathcal{I}_{\text{milieu}}(n) + \frac{h}{3} \sum_{k=0}^{n-1} \left( f\left(a + kh + \frac{h}{6}\right) + f\left(a + kh + \frac{5h}{6}\right) \right)$$

→ **Critère d'arrêt** :  $|\mathcal{I}(2n) - \mathcal{I}(n)| \leq \epsilon$

# Intégration numérique - En dimension supérieure

## Méthodes de Monte-Carlo

- Méthode probabiliste
- faible taux de convergence ...
- ... mais la plus utilisée car adaptation à un domaine quelconque !

# Intégration numérique - En dimension supérieure

## Principes

Soit,

$$\begin{array}{ccc} f : \mathbb{R}^n & \mapsto & \mathbb{R} \\ x & \rightarrow & f(x) \end{array}$$

On souhaite calculer :

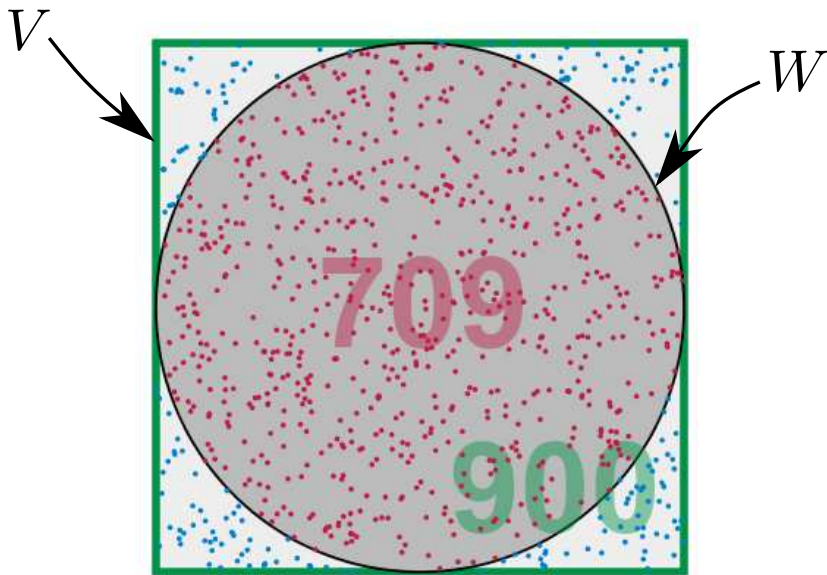
$$\int_W f(x) dx$$

Supposons pouvoir borner  $W$  dans un domaine simple  $V$  (sphère, hypercube, ...) alors :

$$\int_W f(x) dx = \frac{\mathcal{A}(V)}{N} \sum_{i=1}^n f(x_i)$$

avec  $x_i \in W$  et  $\mathcal{A}(V)$  l'aire de  $V$ .

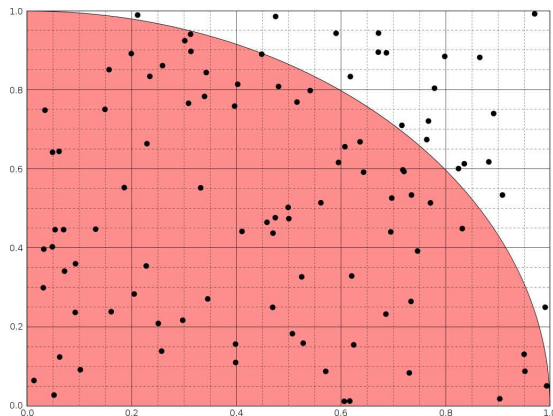
## Intégration numérique - En dimension supérieure





# Applications - Monte-Carlo

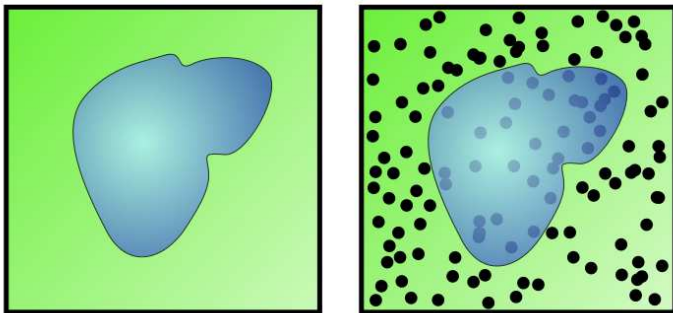
## Calcul du nombre $\pi$



- Tirage aléatoire de  $n$  points de coordonnées  $(x, y)$  avec  $0 < x, y < 1$
- Point valide ssi  $x^2 + y^2 \leq 1$  (nombre de points valides =  $v$ )
- Approximation de  $\pi/4$  :  $v/n$

# Applications - Monte-Carlo

## Superficie d'un lac

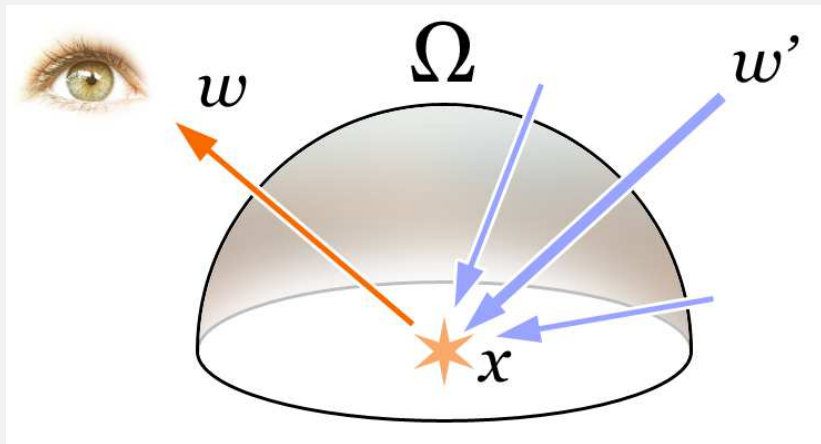


- Tirage aléatoire de  $n$  points de coordonnées  $(x, y)$ , avec  $x, y \in \text{terrain}$
- nombre de points valides =  $v$  ( points dans le lac)
- Superficie du lac :  $\frac{v}{n} \times \text{superficie}_{\text{terrain}}$

# Applications - Monte-Carlo

## Equation du rendu

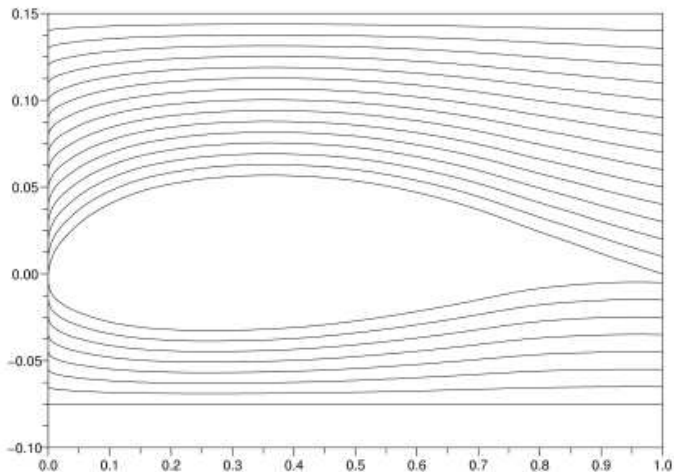
$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t) (-\omega' \cdot \mathbf{n}) d\omega'$$



# Applications - Monte-Carlo



# Applications - Dynamique des fluides



## 6 Equations différentielles

- Présentation
- Approximations des solutions
- Méthodes explicites

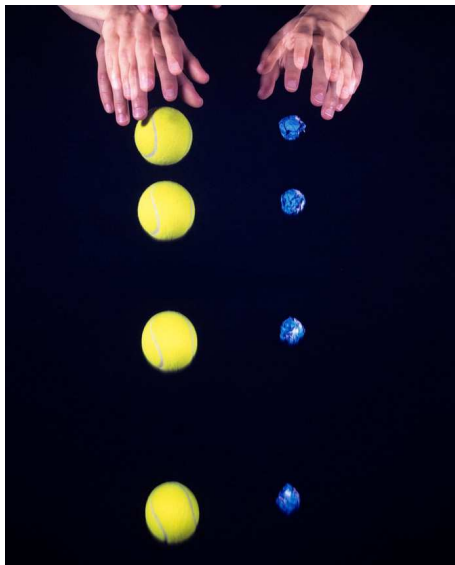
# Présentation

## Types de problème

Modélisation de nombreux phénomènes dynamiques :

- equation du mouvement
- dynamique des populations
- chimie organique
- dynamique des fluides
- électronique
- informatique graphique
- ...

# Présentation





# Présentation

## Principe

Problème de Cauchy :

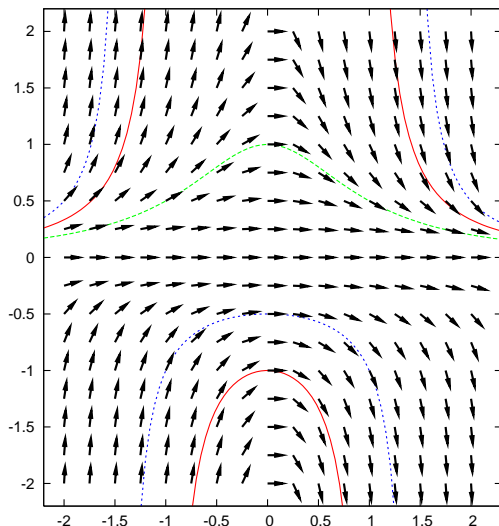
$$\begin{cases} y(t_0) = y_0 & \text{Condition initiale} \\ y' = F(y, t) & \text{Equation différentielle} \end{cases}$$

Généralités :

- la variable  $t$  correspond à une variable temporelle dans de nombreuses situations
- la condition initiale spécifie entièrement l'état du modèle à l'instant  $t_0$

# Approximations des solutions

Intuition sur l'approche différentielle : Champ des tangentes



$$\begin{aligned} y' &= -2xy^2 \\ (x, y) &\rightarrow (1, -2xy^2) \end{aligned}$$

# Approximations des solutions

## Méthode générale

Prédiction de  $y_{n+1}$  à partir des  $y_i$ ,  $i \leq n$ , par une méthode d'intégration :

- méthode à un pas

$$y_{n+1} = y_n + h_n F(y_n, t_n, h_n)$$

- méthode à plusieurs pas

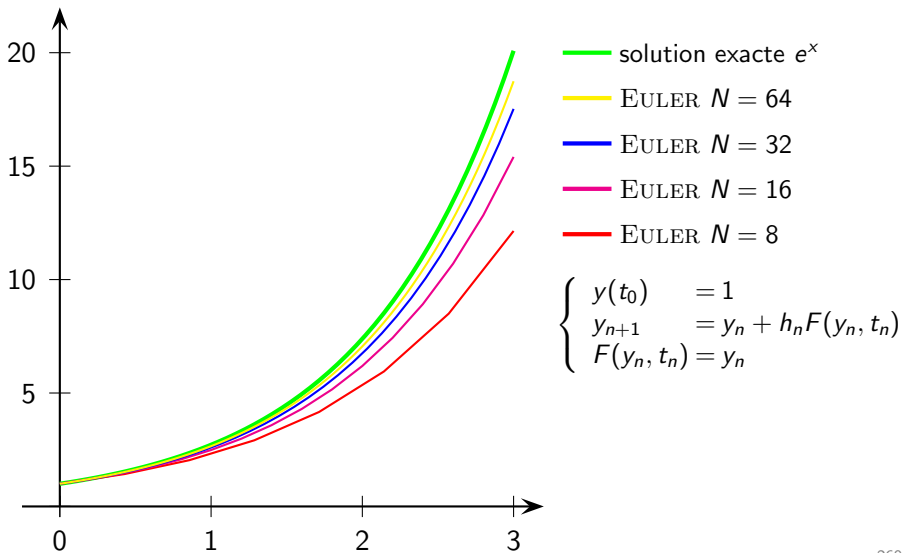
$$y_{n+1} = y_n + h_n F(y_n, y_{n-1}, \dots, t_n, t_{n-1}, \dots, h_n)$$

Propriétés :

- Consistance : validité de la méthode d'intégration localement
- Stabilité : majoration des erreurs propagées
- Convergence : existence d'une solution optimale

# Approximations des solutions

Exemple de méthode à un pas : la méthode d'Euler



# Approximations des solutions

## Exemple de méthode à un pas

### La méthode d'Euler

$$y_{n+1} = y_n + h_n F(y_n, t_n)$$

→ la tangente au point courant fournit une direction d'évolution  
(i.e. hypothèse que la fonction est affine au point  $y_n$ )

# Approximations des solutions

## Erreur de consistance

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(y_n, t_n)$ . L'erreur locale  $e_n$  réalisée au pas de calcul  $n$  est définie comme :

$$e_n = y(t_{n+1}) - y_{n+1}$$

→ C'est l'erreur de consistance

En supposant  $F$  suffisamment dérivable :

$$\begin{aligned} e_n &= y(t_{n+1}) - y_{n+1} \\ &= y(t_{n+1}) - y_n - h_n F(y_n, t_n) \\ &= y(t_n + h_n) - y_n - h_n F(y_n, t_n) \\ &= \frac{1}{2} h_n^2 y''(t_n) + o(h_n)^2 \end{aligned}$$

→  $e_n$  est bornée par les variations de  $F$  et  $h_n$

# Approximations des solutions

## Consistance

Définition : Une méthode d'intégration est dite consistante ssi la somme des erreurs de consistance,  $\sum_{k=1}^N |e_k|$ , tend vers 0 quand  $h_{\max} = \max_k h_k$  tend vers 0.

→ Une méthode est consistante si la méthode d'intégration calcule bien une solution de l'équation différentielle

→ la méthode d'Euler est consistante car  $e_n \approx \mathcal{O}(h_n^2)$

---

Ordre d'une méthode. Une méthode est dite d'ordre  $p$  ssi

$$\exists C > 0, \quad \forall k \in [1; N], |e_k| \leq Ch_k^{p+1}$$

→ la méthode d'Euler est d'ordre 1.

# Approximations des solutions

## Stabilité et Convergence

→ Comment les erreurs de calcul se propagent-elles ?

**Stabilité** : une méthode est stable ssi pour les suites  $y_n$  et  $\tilde{y}_n$  définies comme

$$\begin{aligned}y_{n+1} &= y_n + h_n F(y_n, t_n) \\ \tilde{y}_{n+1} &= \tilde{y}_n + h_n F(\tilde{y}_n, t_n) + \epsilon_n\end{aligned}$$

il existe une constante  $S$ , *constante de stabilité*, telle que :

$$\max_k |y_k - \tilde{y}_k| \leq S \left( |y_0 - \tilde{y}_0| + \sum_{k=1}^N |\epsilon_k| \right)$$

→ l'erreur reste bornée linéairement par rapport à la somme des erreurs locales. La méthode d'Euler est une méthode stable.



# Approximations des solutions

## Stabilité et Convergence

**Convergence** : une méthode est convergente ssi

$$\max_k |y_k - y(t_k)| \rightarrow 0$$

quand,

$$\begin{cases} h_{\max} \rightarrow 0 \\ y_0 \rightarrow y(t_0) \end{cases}$$

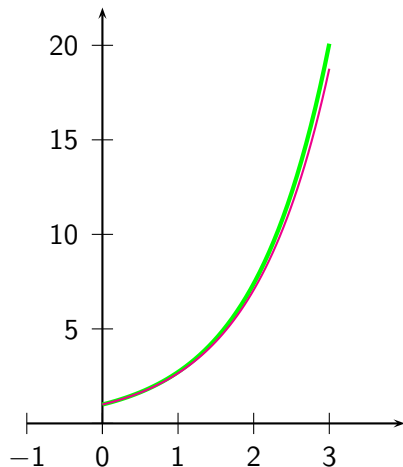
→ La consistance et la stabilité de la méthode d'Euler implique que cette méthode est convergente.

# Méthode de résolution explicites

- Méthode d'Euler
- Méthode du point milieu
- Méthode de Heun
- Méthode de Runge Kutta 4

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



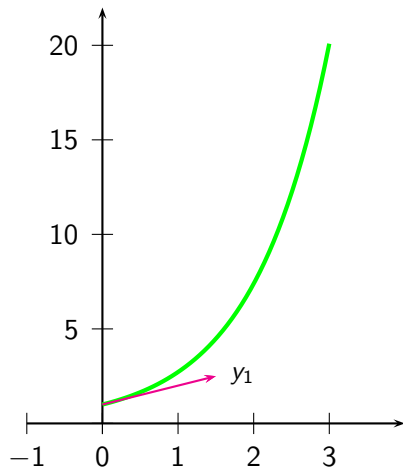
— solution exacte ( $y = e^x$ )

— EULER  $N = 64$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



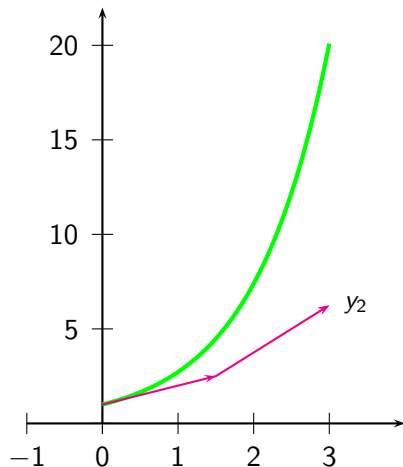
— solution exacte ( $y = e^x$ )

— EULER  $N = 2$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



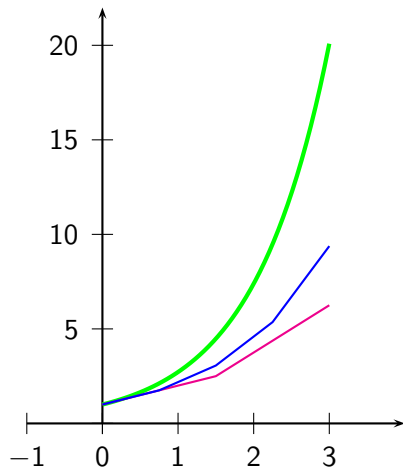
— solution exacte ( $y = e^x$ )

— EULER  $N = 2$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



— solution exacte ( $y = e^x$ )

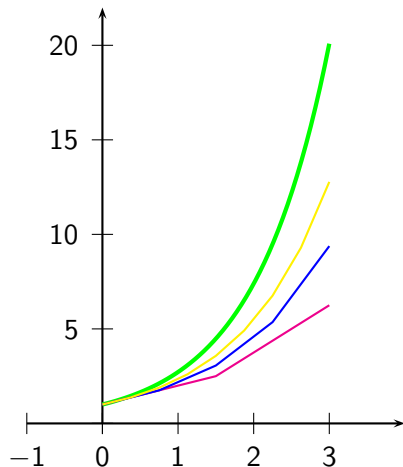
— EULER  $N = 2$

— EULER  $N = 4$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



— solution exacte ( $y = e^x$ )

— EULER  $N = 2$

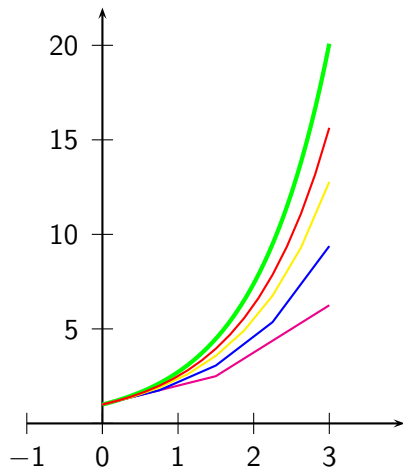
— EULER  $N = 4$

— EULER  $N = 8$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode d'Euler :  $y_{n+1} = y_n + h_n F(t_n, y_n)$



— solution exacte ( $y = e^x$ )

— EULER  $N = 2$

— EULER  $N = 4$

— EULER  $N = 8$

— EULER  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$



# Méthode de résolution explicites

## Méthodes de Runge-Kutta

Problème d'intégration :

$$y_{n+1} = y_n + h_n \int_0^1 F(t_n + xh_n, y(t_n + xh_n)) dx$$

Méthodes associées :

- Méthode du point milieu : ordre 2  
tangente  $\approx F(t_{n+0.5}, y_{n+0.5})$
- Méthode de Heun : ordre 2  
tangente  $\approx \frac{1}{2} (F(t_n, y_n) + F(t_{n+1}, y_{n+1}))$  (i.e. Méthode des trapèzes)
- Méthode de RK4 : ordre 4, tangente  $\approx$  Méthode de Simpson

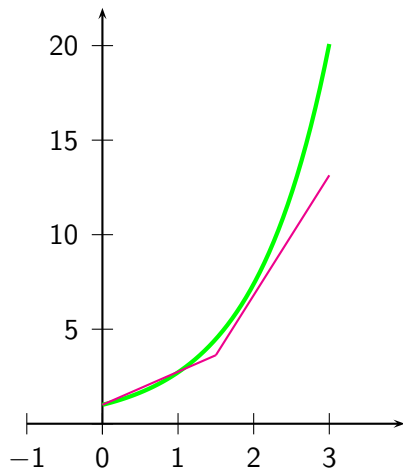
# Méthode de résolution explicites

## Méthode du point milieu

$$\begin{cases} y_{n+0.5} &= y_n + \frac{h}{2}F(t_n, y_n) \\ p_n &= F(t_n + \frac{h}{2}, y_{n+0.5}) \\ y_{n+1} &= y_n + hp_n \end{cases}$$

# Méthode de résolution explicites

## Méthode du point milieu



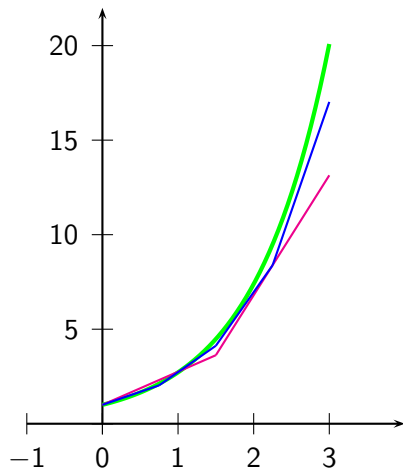
— solution exacte ( $y = e^x$ )

— MILIEU  $N = 2$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode du point milieu



— solution exacte ( $y = e^x$ )

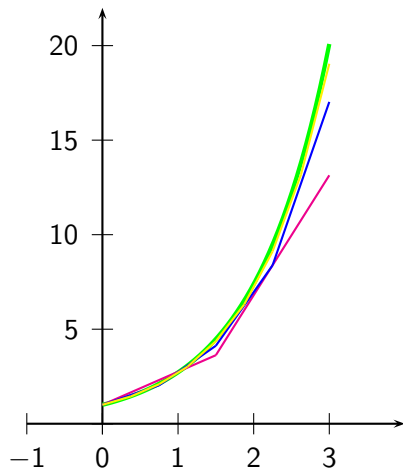
— MILIEU  $N = 2$

— MILIEU  $N = 4$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode du point milieu



— solution exacte ( $y = e^x$ )

— MILIEU  $N = 2$

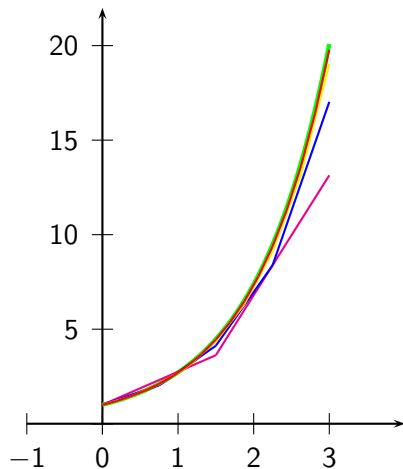
— MILIEU  $N = 4$

— MILIEU  $N = 8$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode du point milieu



— solution exacte ( $y = e^x$ )

— MILIEU  $N = 2$

— MILIEU  $N = 4$

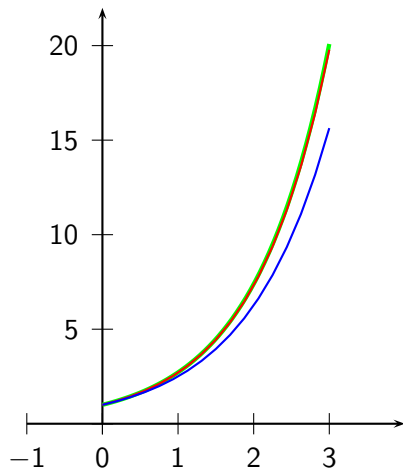
— MILIEU  $N = 8$

— MILIEU  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode du point milieu VS Méthode d'Euler



— solution exacte ( $y = e^x$ )

— MILIEU  $N = 16$

— EULER  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

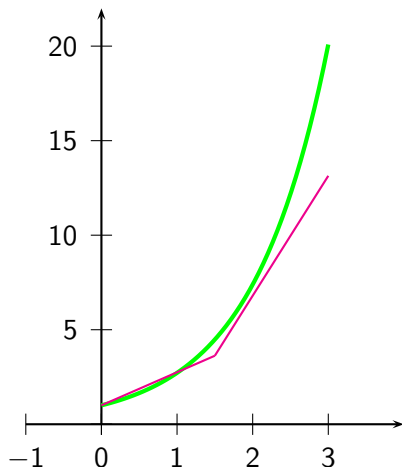
## Méthode de Heun

$$\begin{cases} p_{n,1} &= F(t_n, y_n) \\ y_{n,1} &= y_n + h_n p_{n,1} \\ p_{n,2} &= F(t_n + h_n, y_{n,1}) \\ y_{n+1} &= y_n + \frac{1}{2} h_n (p_{n,1} + p_{n,2}) \end{cases}$$



# Méthode de résolution explicites

## Méthode de Heun



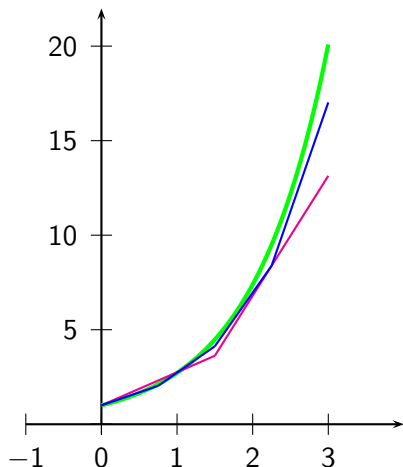
— solution exacte ( $y = e^x$ )

— HEUN  $N = 2$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode de Heun



— solution exacte ( $y = e^x$ )

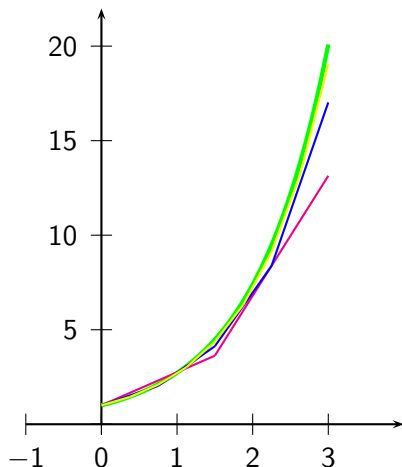
— HEUN  $N = 2$

— HEUN  $N = 4$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode de Heun



— solution exacte ( $y = e^x$ )

— HEUN  $N = 2$

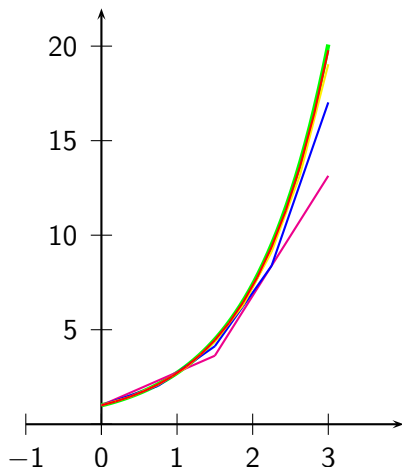
— HEUN  $N = 4$

— HEUN  $N = 8$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Méthode de Heun



— solution exacte ( $y = e^x$ )

— HEUN  $N = 2$

— HEUN  $N = 4$

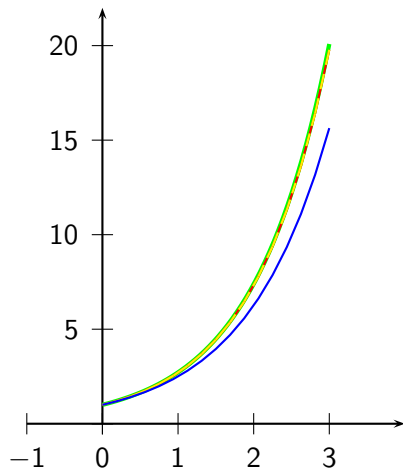
— HEUN  $N = 8$

— HEUN  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

Méthode de Heun VS Méthode du point milieu VS Méthode d'Euler



— solution exacte ( $y = e^x$ )

— HEUN  $N = 16$

- - - MILIEU  $N = 16$

— EULER  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

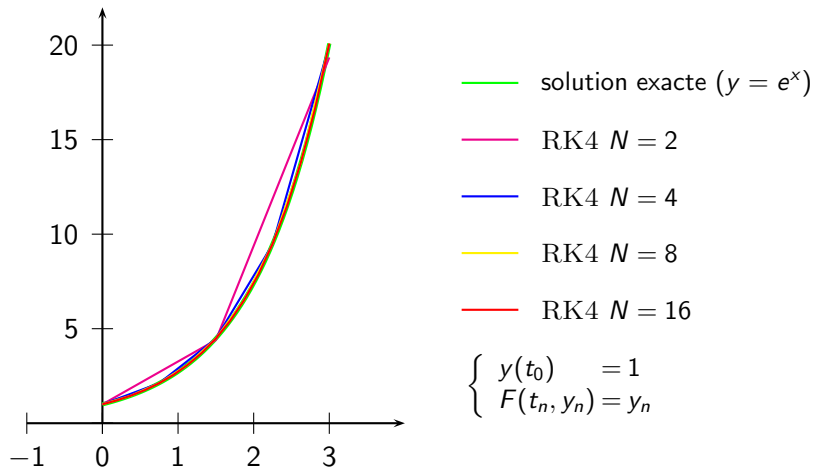
# Méthode de résolution explicites

## Méthode de Runge Kutta 4

$$\left\{ \begin{array}{lcl} p_{n,1} & = & F(t_n, y_n) \\ y_{n,1} & = & y_n + \frac{1}{2}h_n p_{n,1} \\ p_{n,2} & = & F(t_n + \frac{1}{2}h_n, y_{n,1}) \\ y_{n,2} & = & y_n + \frac{1}{2}h_n p_{n,2} \\ p_{n,3} & = & F(t_n + \frac{1}{2}h_n, y_{n,2}) \\ y_{n,3} & = & y_n + h_n p_{n,3} \\ p_{n,4} & = & F(t_n + h_n, y_{n,3}) \\ y_{n+1} & = & y_n + \frac{1}{6}h_n(p_{n,1} + 2p_{n,2} + 2p_{n,3} + p_{n,4}) \end{array} \right.$$

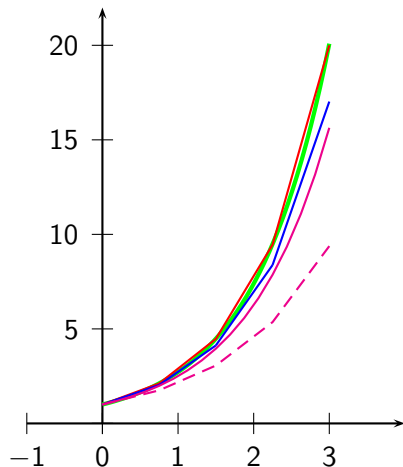
# Méthode de résolution explicites

## Méthode RK4



# Méthode de résolution explicites

Méthode RK4 VS Méthode du point milieu VS Méthode d'Euler



— solution exacte ( $y = e^x$ )

— RK4  $N = 4$

— MILIEU  $N = 4$

— EULER  $N = 16$

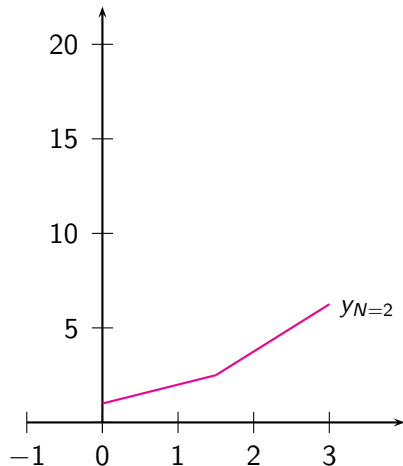
- - - EULER  $N = 4$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$



# Méthode de résolution explicites

## Mesure de la convergence

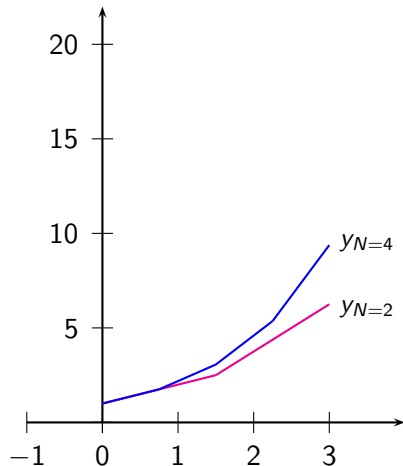


— EULER  $N = 2$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Mesure de la convergence



$$|y_{2N} - y_N| > \epsilon \rightarrow \text{Continue}$$

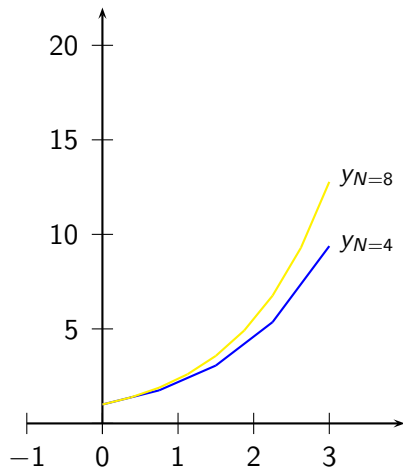
— EULER  $N = 2$

— EULER  $N = 4$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Mesure de la convergence



$$|y_{2N} - y_N| > \epsilon \rightarrow \text{Continue}$$

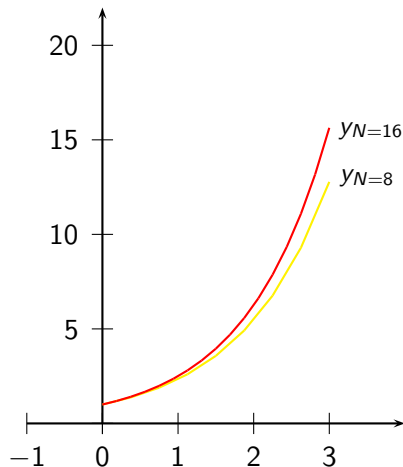
— EULER  $N = 4$

— EULER  $N = 8$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Mesure de la convergence



$$|y_{2N} - y_N| > \epsilon \rightarrow \text{Continue}$$

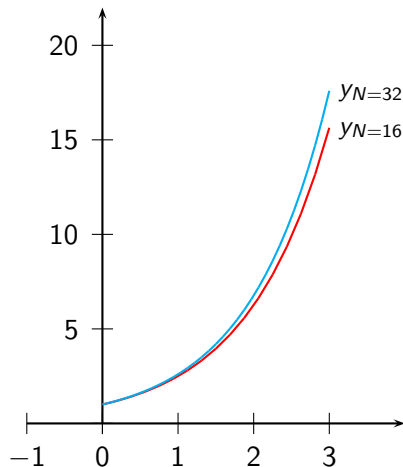
— EULER  $N = 8$

— EULER  $N = 16$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Mesure de la convergence



$$|y_{2N} - y_N| < \epsilon \rightarrow \textbf{Convergence}$$

— EULER  $N = 16$

— EULER  $N = 32$

$$\begin{cases} y(t_0) = 1 \\ F(t_n, y_n) = y_n \end{cases}$$

# Méthode de résolution explicites

## Instabilités des méthodes numériques

- Non-unicité mathématique de la solution / condition initiale  
⇒ Importance de fournir une condition initiale
- Instabilité mathématique de l'espace des solutions / condition initiale  
⇒ Importance du choix du pas de calcul et la taille de l'intervalle de résolution
- Instabilité numérique dûe au schéma d'intégration  
⇒ Vérifier la convergence de l'algorithme entre 2 itérations
- Instabilité numérique dûe à l'accumulation d'erreurs d'approximation  
⇒ Vérifier la précision du calcul à chaque itération

THE END