

# Ananum notes 23

---

## Chap 1

---

### virgules flottantes

$$\pm O.d_1\bar{d}_2\dots d_t.\beta^e = \pm\beta^e \sum t_i = 1 \frac{d_i}{\beta^i}$$

### erreur d'arrondi

u: l'unité d'arrondi

$$u := \frac{1}{2}\beta^{1-t} \geq \frac{|f(x)-x|}{|x|}$$

erreur absolue:  $\epsilon_{abs} = |\hat{x} - x|$

erreur relative (pour  $x \neq 0$ ):  $\epsilon_{rel} = \frac{|\hat{x}-x|}{|x|}$

en *double* (64bit):  $u \simeq 1.1 * 10^{-16}$

### Conditionnement

stabilité directe

en x s'il existe  $C_1, C_2 \geq 1$  tq:

$\|\hat{y} - y\| \leq C_1 \|f(x + \delta x) - f(x)\|$  qu'on peut diviser par  $\|y\|$  à gauche et  $\|f(x)\|$  à droite pour au moins un  $\delta x$  tq  $\|\delta x\|/\|x\| \leq C_2 u$

le conditionnement:

$$\kappa(x) := \lim_{\epsilon \rightarrow 0} \left( \sup_{\|\delta x\| \leq \epsilon \|x\|} \left( \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|} \frac{\|x\|}{\|f(x)\|} \right) \right)$$

le conditionnement est le **pire des facteurs** par lequel il faut multiplier les erreurs relatives dans  $x$  pour obtenir les erreurs relatives dans  $f(x)$  (avec erreurs  $\rightarrow 0$ )

$$\kappa(x) = \sup \frac{\text{erreur relative résultat}}{\text{erreur relative données}}$$

- conditionnement ne dépend pas de l'algo, mais bien du problème considéré (via  $y = f(x)$ )
- si  $\kappa(x) \gg 1$  prob mal conditionné
- si  $f(x)$  différentiable (et  $f'(x)$  matr Jacobienne):

$$\kappa(x) = \frac{\|f'(x)\| \cdot \|x\|}{\|f(x)\|}$$

erreur inverse  $\Delta x$  d'un algo  $\hat{y}$ :

tq  $f(x + \Delta x) = \hat{y}$

## Chap 2

---

Sys linéaire  $Ax = b$

### Conditionnement sys linéaire

cas 1

perturbations  $\delta b$  de  $b$

$$\kappa = \sup_{\|\delta b\|} \left( \frac{\|\delta x\|/\|x\|}{\|\delta b\|/\|b\|} \right) \leq \frac{\|A^{-1}\| \cdot \|b\|}{\|x\|} = \frac{\|A^{-1}\| \cdot \|Ax\|}{\|x\|} \leq \|A^{-1}\| \cdot \|A\|$$

cas 2

perturbations  $\delta A$  de  $A$

$$\kappa = \sup_{\|\delta A\|} \left( \frac{\|\delta x\|/\|x\|}{\|\delta A\|/\|A\|} \right) \leq \|A^{-1}\| \cdot \|A\|$$

erreurs dans les données  $A, b$  d'un sys linéaire sont amplifiées par (au plus):

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

$\kappa(A)$ : conditionnement de la matrice  $A$

en Octave: `cond`

### Factorisation LU

$$LUx = b \text{ où } Ux = y$$

(par ex:  $L_2 L_1 Ax = L_2 L_1 b$  où en fait  $L_2 L_1 A = U \Rightarrow A = LU$ )

$$x = U \setminus (L \setminus b)$$

## Algo Octave:

```
function [L U] = factLU (A)
    n = size(A, 1); # ordre de la matrice (plutot que length(A))
    for k = 1:n
        for j = k:n
            # on construit ligne par ligne
            U(k,j) = A(k,j);
        endfor
        L(k,k) = 1;
        for i = k+1:n
            L(i,k) = A(i,k)/A(k,k);
            # on cree les coeff de l'elimination de Gauss (dans la
            premiere colonne quand k vaut 1)
        endfor
        for i = k+1:n
            for j = k+1:n
                A(i,j) = A(i,j) - ( L(i,k) * A(k,j) );
            endfor
        endfor
    endfor
endfunction
```

puis on fait

```
A=...;
[L U] = factLU(A)
x = U \ (L \ b) # moyen memo ULB, attention parantheses
```

- coût LU:  $\frac{2}{3}n^3 + O(n^2)$  flops
- si fact LU existe, elle est **unique**
- existe **pas toujours** pour matrices régulières
- **pas stable** (alors on va faire avec pivot => PALU)

## Factorisation PALU

$$PA = LU$$

- même coût que LU
- **existe** pour toute matrice régulière
- réputée **stable en pratique**

calcul du  $\det(A)$

inversion matricielle

## Algo Octave:

```
function [L U P] = factPALU (A)
    #n = size(A,1);
    P1 = eye(4);
    P2 = eye(4);
    P3 = eye(4); # ? 3 matrices P car A d'ordre 4??
    U = A; # on pourrait juste tout faire avec A et à la fin mettre U=A

    P1([1,4],:) = P1([4,1],:); % on switch les LIGNES 1 et 4 de P1
    # ^ (change direct les deux lignes)

    U = P1*U;

    #pour les L on va remplacer les éléments de chaque colonne, qui sont
    #en dessous de la diagonale

    # construction de L1
    L1 = eye(4);
    L1([2:4],1) = -U([2:4],1)/U(1,1);
    # remplace le 2e, 3e et 4e élément de la 1e colonne de L1
    # par l'opposé du:
    # 2e, 3e et 4e élément de la première colonne de U (//ou A), divisé
    par le 1e element
    # (info: ce 1e element étant donc sur la diagonale)

    U=L1*U;
    # L1 sert à faire apparaitre des zeros dans la premiere colonne
    # (elimination de gauss)

    # --- meme chose avec P2 ---
    P2([2,4],:) = P2([4,2],:); # switch lignes 2 et 4 de P2

    U=P2*U;

    #construction de L2
    L2=eye(4);
    L2([3:4],2) = -U([3:4],2)/U(2,2);
    # remplace le 3e et 4e element de la 2e colonne de L2
    # par l'opposé du:
    # 3e et 4e element de la 2e colonne de A, divisé par le 2e element
    # (info: ce 2e element étant sur la diagonale)
```

```
U=L2*U;
```

```
# --- meme chose avec P3 ---
```

```
# (4e element de la 3e colonne (sous la diagonale donc))
```

```
P3([3,4],:) = P3([4,3],:);
```

```
U=P3*U;
```

```
L3=eye(4);
```

```
L3(4,3) = -U(4,3)/U(3,3); # ([4,4], ) = (4, )
```

```
U=L3*U;
```

```
# L = ( ... matrice identité dont les colonnes du triangle inférieur  
sont -Lp1 et -Lp2 et -Lp3
```

```
Lp3 = L3; # Lp: "L prime" (L')
```

```
Lp2 = P3*L2*P3;
```

```
Lp1 = P3*P2*L1*P2*P3;
```

```
P = P3*P2*P1
```

```
L=eye(4) # matrice identité
```

```
# triangle inférieur sera les colonnes opposées de Lp1 Lp2 et Lp3
```

```
L([2:4],1) = -Lp1([2:4],1);
```

```
L([3:4],2) = -Lp2([3:4],2);
```

```
L(4,3) = -Lp3(4,3)
```

```
# ici U a déjà été modifié de la matrice A de départ pendant le  
calcul
```

```
endfunction
```

qu'on peut faire:

```
A=...;
```

```
[L U P] = factPALU(A)
```

```
# x = ?? U\ (L\ (P*b)) ??
```

peut aussi utiliser la func `lu()` de Octave pour vérif ( `[L U P] = lu(A)` )

## Chap 3

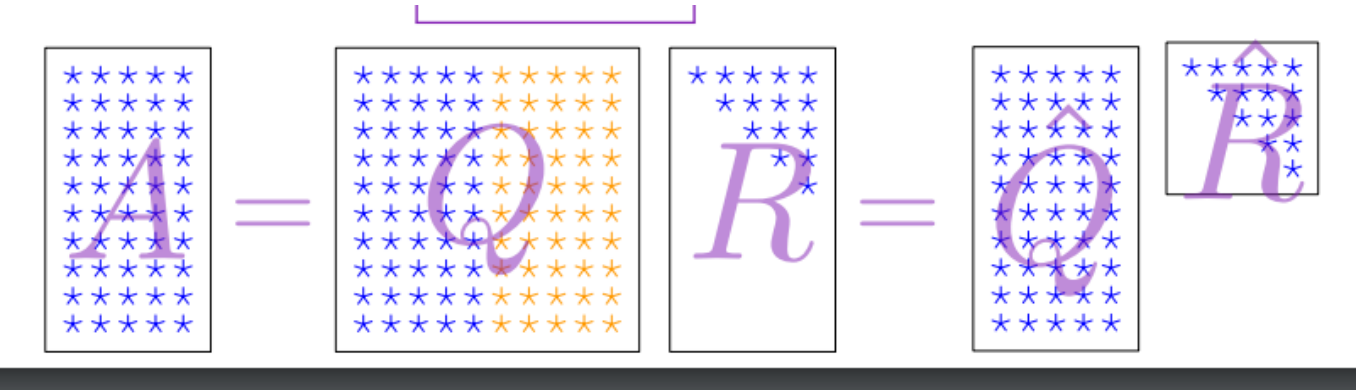
---

### Factorisation QR

matrice  $Q$  orthogonale : carrée et  $Q^T Q = I$

$R = (r_{ij})$  trapézoïdale supérieure si  $r_{ij} = 0$  pour tout  $i > j$

$$A = QR$$



Algo Octave:

(on peut le faire en boucle for mais pas important):





- coût:  $2n^2(m - n/3) + O(mn)$  pour une matrice A de dimensions  $m \times n$

## Interlude: propriétés Norme Euclidienne

?

## conditionnement sys surdéterminés

?

## Chap 4

---

Méthodes itératives pour systèmes linéaires

=> résoudre  $Ax = b$  de manière itérative

( $k$ ) num d'itération

erreur  $e^{(k)} = x - \hat{x}^{(k)}$

$Ae^{(k)} = b - Ax^{(k)}$

On peut définir  $B \simeq A$  si B est plus facile à inverser

$x^{(k+1)} = x^{(k)} + e^{(k)}$  -> il faut que cette erreur tende vers 0 après les itérations

=>  $x^{(k+1)} = x^{(k)} + B^{-1}(b - Ax^{(k)})$

On devra mettre un critère d'arrêt

$\frac{\|r^{(k)}\|}{\|b\|} \leq 10^{-3}$  ( $10^{-3}$  par exemple ici)

$\frac{\|x - x^{(k)}\|}{\|x\|} \leq k(A) \frac{\|r^{(k)}\|}{\|b\|}$

## méthodes stationnaires

### Jacobi

matrice tri-diagonale

première ligne, que 1 voisin

$B_J = D_A$

( $D_A$ : diagonale de A)

Algo Octave:

'''matlab

function [pfait, rsurb, sol] = tp6(A, b, critdarret)

```

x = zeros(size(A,1),1);
n = size(A,1)

p = 1;
pfait = p;
residu = norm(b-A*x);
rsurb = residu/norm(b);

while rsurb > critdarret && p < 200

    x(1) = (1/A(1,1))*(b(1) - A(1,2)*x(2));

    for i = 2:n-1
        x(i)=(1/A(i,i))*(b(i) - A(i, i-1)*x(i-1) - A(i, i+1)*x(i+1));
    endfor
    %A(n,n);
    %x(n)=(1/A(n,n))*(b(n) - A(n, n-1)*x(n-1));
    x(n)=(1/A(n,n))*(b(n) - A(n, n-1)*x(n-1));

    sol = x;
    pfait = p;
    residu = norm(b-A*x);
    rsurb = residu/norm(b);

    p++;

endwhile

endfunction

'''

```

exemple de crit d'arret:  $10^{-3}$

## Gauss-Seidel

algo: en gros pareil que Jacobi mais sans  $x(0)$

$$B_{GS} = L_A$$

$$\text{rappel: } A = L_A + U_A - D_A$$

( $L_A$ : triangulaire lower de A

$U_A$ : triangulaire upper de A

$D_A$ : diagonale de A)

## Algo Octave:

```
function [pfait, rsurb, sol] = tp6ex2(A, b, critdarret)

    n = size(A,1);
    x = zeros(n,1)

    p = 1;
    pfait = p;
    residu = norm(b-A*x);
    rsurb = residu/norm(b);

    while rsurb > critdarret && p < 200

        x(1) = (1/A(1,1))*(b(1) - A(1,2)*x(2))

        for i = 2:n-1
            x(i) = (1/A(i,i))*(b(i) - A(i,i-1)*x(i-1) - A(i,i+1)*x(i+1))
        endfor

        x(n) = (1/A(n,n))*(b(n) - A(n, n-1)*x(n-1));

        pfait = p;
        residu = norm(b-A*x);
        rsurb = residu/norm(b);
        sol = x;

        p++;

    endwhile

endfunction
```

exemple de crit d'arret:  $10^{-3}$

On peut comparer

$\frac{\|x - x^*\|}{\|x\|}$  de Jacobi et Gauss-Seidel.

## Minimisation d'énergie

norme énergie:  $\|v\|_A = \sqrt{v^T A v}$

principe: choisi une direction  $p \neq 0$  et que forme solution approchée à l'iter  $k + 1$  est

$$x^{(k+1)}(\alpha) = x^{(k)} + \alpha p$$

Trouver la valeur de  $\alpha$  qui minimise  $f(x^{(k+1)})$

$$\alpha = \frac{p^T r^{(k)}}{p^T A p}$$

/!\ système avec  $A$  symétrique définie positive

L'énergie d'un système est définie par la fonction:

$$\begin{aligned} f(x^{(k)}) &:= \|x - x^{(k)}\|_A^2 = (x - x^{(k)})^T A (x - x^{(k)}) \\ &= x^{(k)T} A x^{(k)} - 2x^{(k)T} A x + x^T A x \\ &= x^{(k)T} A x^{(k)} - 2x^{(k)T} b + \text{cste} \end{aligned}$$

Soit une direction  $p \neq 0$

$\alpha = \frac{p^T r^{(k)}}{p^T A p}$  minimise l'énergie  $f(x^{(k+A)})$  de

$$x^{(k+1)}(\alpha) = x^{(k)} + \alpha p$$

Algo:

...

## Méthode du gradient

Algo

## Méthode du gradient conjugué

?

## Contrôle de convergence

## Critère d'arrêt

norme du résidu...

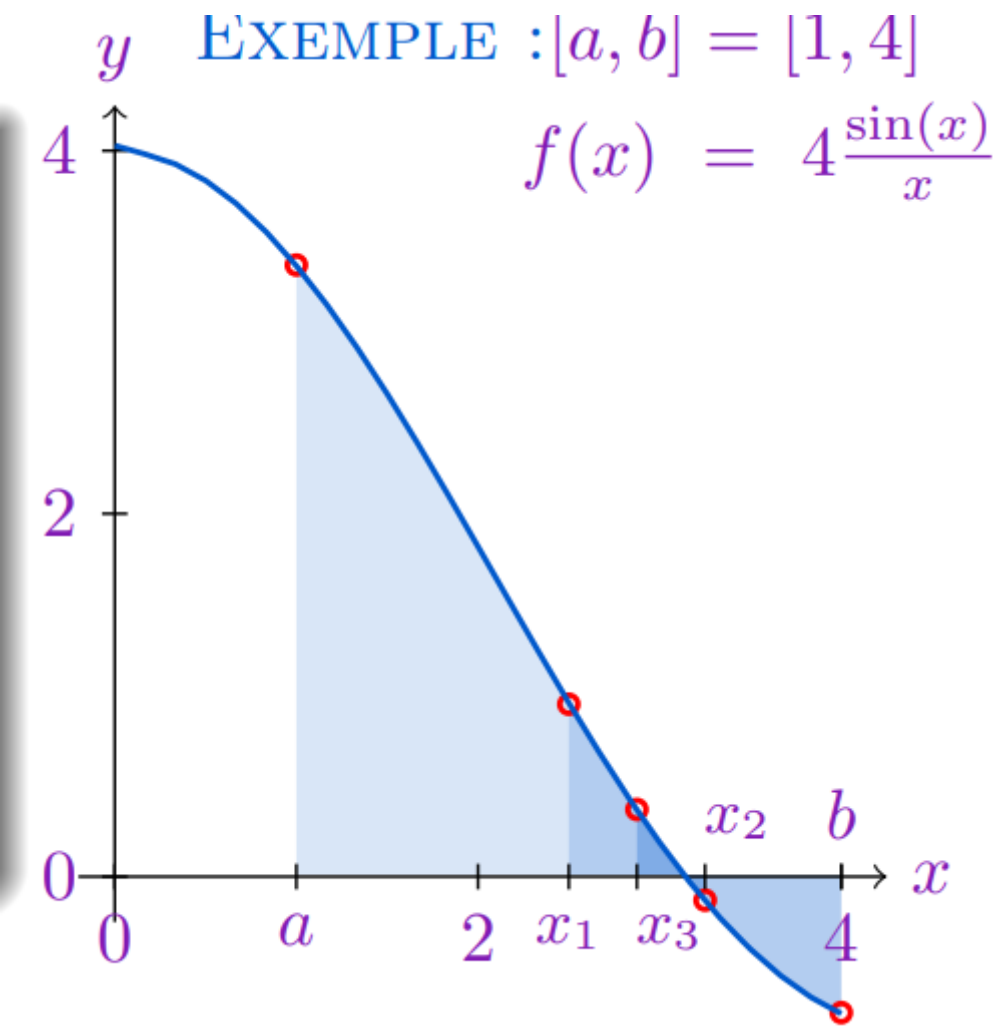
TP6: interpolation ?

# Chap 5

équations et sys non-linéaires

## Dichotomie

critère d'arrêt de limite d'iterations



## Algo Octave:

```
function [niterations, sol] = algodicho(f, a, b, epsilon)
    niterations = 0;
    while abs(a-b) > epsilon & niterations < 69
        niterations++;
        x = (a + b)/2;
        if f(a)*f(x) < 0
            b = x;
        endif
        if f(b)*f(x) < 0
            a = x;
        endif
    endwhile
    # critère d'arrêt:
    if f(x) < 10^(-6) # plutôt que f(x(k)) == 0 car précision
        sol = x;
        break
    endif
endwhile
endfunction
```

puis

```
[nbr_iter sol] = algodicho(f, a, b, epsilon) #on peut aussi passer un
critère d'arrêt plutôt que de le hardcoder
```

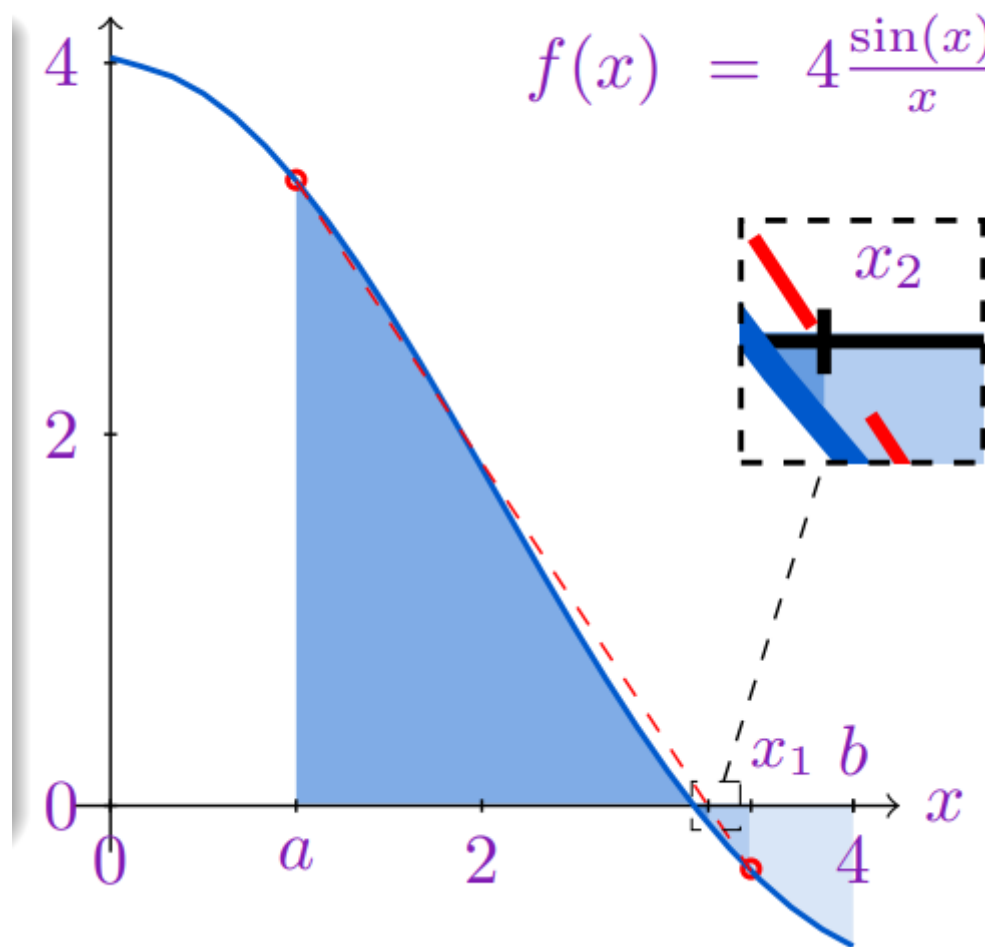
## Fausse position

crit d'arrêt: on va évaluer si  $f(x^{(k+1)}) < \epsilon$

?

EXAMPLE :  $[a, b] = [1, 4]$

$$f(x) = 4 \frac{\sin(x)}{x}$$



## Algo Octave:

```
function [n, sol] = algofaussepos(f, a, b)
    n = 0;
    while n < 69
        n++;
        x = a - f(a)*((b-a)/(f(b)-f(a)));
        if f(a)*f(x) < 0
            b = x;
        endif
        if f(b)*f(x) < 0
            a = x;
        endif

        # critere d'arret
        if f(x) < 10^(-6) # plutot que f(x(k)) == 0 car précision
            sol = x;
            break
        endif
    endwhile
endfunction
```

puis

```
[n_iter sol] = algofaussepos(f, a, b) #on peut aussi passer un critère
d'arret plutot que de le hardcoder
```

## Newton-Raphson

Principe: prendre pour  $x_{k+1}$  la racine du dév de Taylor de premier ordre autour de  $x_k$   
Cela revient, pour autant que  $f'(x_k) \neq 0$ , à déterminer  $x_{k+1}$  qui satisfait:

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

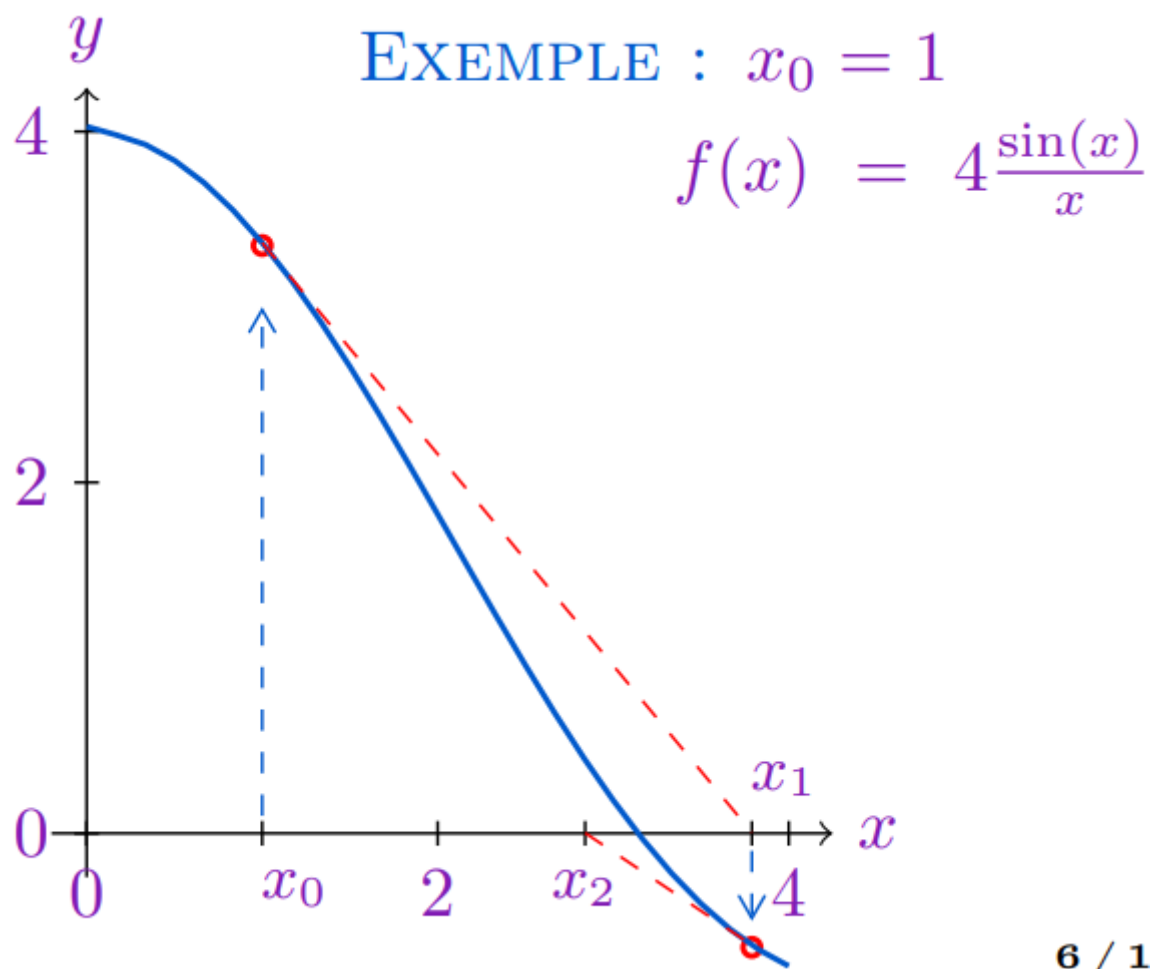
et donc  $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

**ATTENTION: il faut connaître la dérivée de cette fonction**

on choisit un  $x_0$  suffisamment proche de la racine (en regardant un plot par ex)

si la racine est un extremum: cette méthode est pas super précise





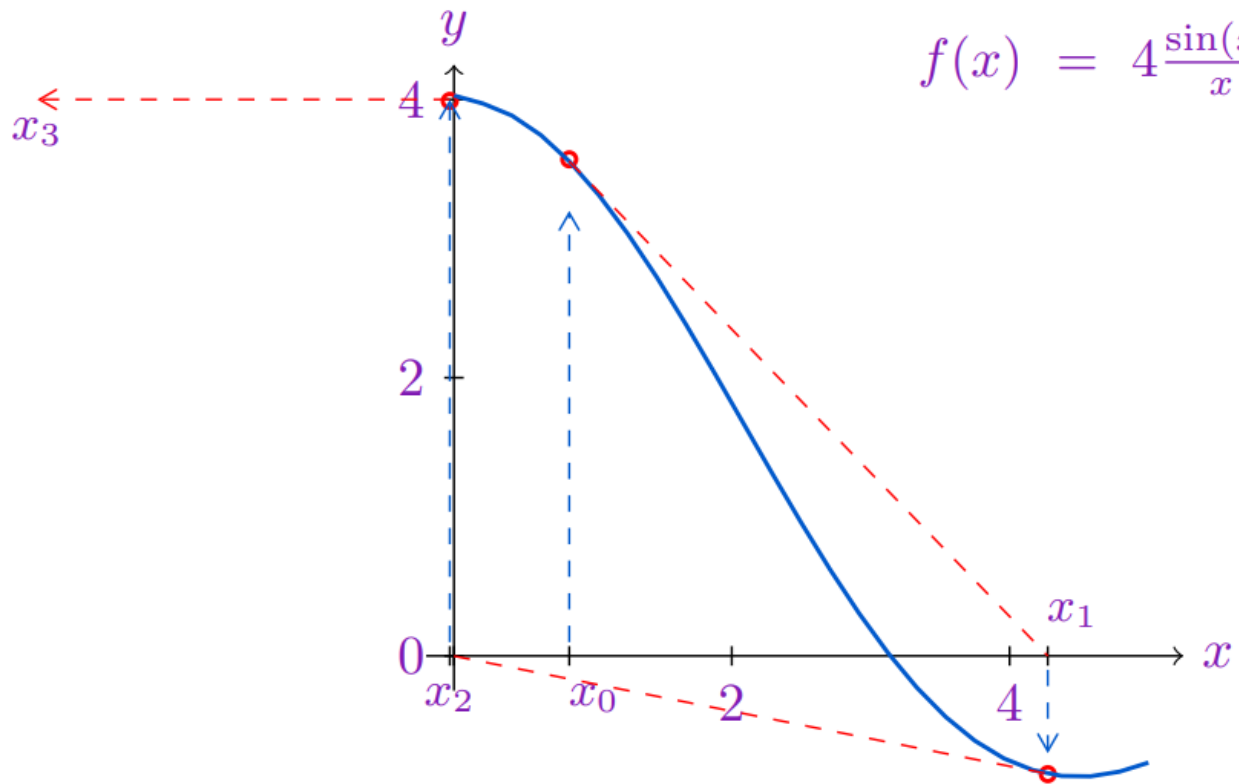
6 / 1

deux problèmes:

- si ça converge pas

EXEMPLE :  $x_0 = 0.00$

$$f(x) = 4 \frac{\sin(x)}{x}$$

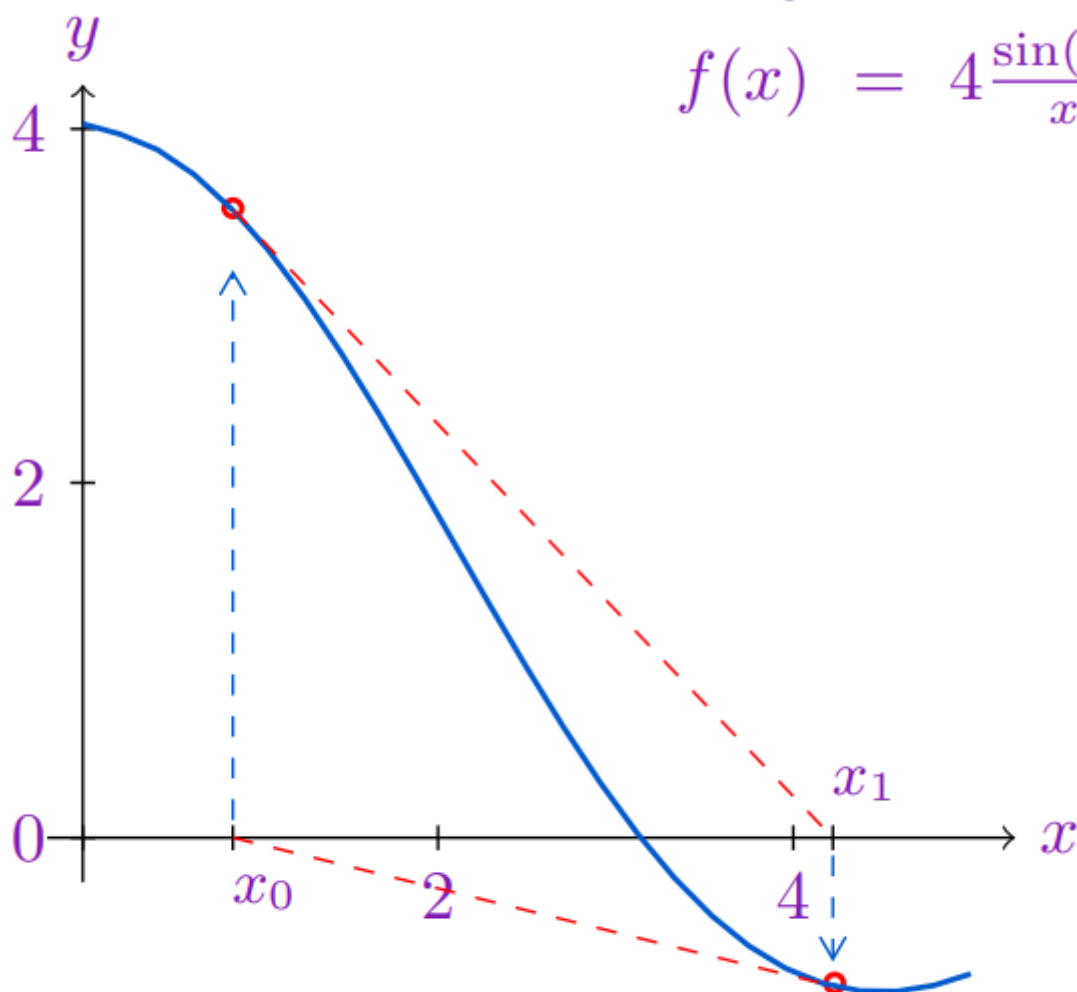


(divergence)

- si c'est cyclique

EXEMPLE :  $x_0 \approx 0.845$

$$f(x) = 4 \frac{\sin(x)}{x}$$



(comportement cyclique)

Convergence d'une suite  $x_k, k = 0, 1, \dots$  vers  $x$  est d'ordre  $p$  s'il existe une constante  $C$  tq:

$$|x - x_{k+1}| \leq C|x - x_k|^p$$

- si  $p = 1$  et  $C < 1$  convergence linéaire
- si  $p = 2$  convergence quadratique
- si  $\lim_{k \rightarrow \infty} \frac{|x - x_{k+1}|}{|x - x_k|} = 0$  convergence superlinéaire

## Algo Octave:

```
function [n, sol] = algonewtonraphson(f, fprime, x0)
    n = 0;
    %x = x0 - f(x0)/fprime(x0); % si scalaire
    x = x0 - fprime(x0)\f(x0); % pour vectoriel ET scalaire

    #while (fprime(x) > 10^(-4)) & (fprime(x) < -10^(-4)) & n < 50
    # critere d'arret
    while n < 69
        n++;
        %x = x - f(x)/fprime(x); % si scalaire
        x = x - fprime(x)\f(x); % pour vectoriel ET scalaire
    endwhile
    sol = x;
endfunction
```

puis

```
[n_iter sol] = algorewtonraphson(f, fprime, x0) #on peut aussi passer un
critère d'arret plutot que de le hardcoder
```

## N-R avec recherche linéaire

Principe: même chose que N-R mais avec un facteur d'amortissement  $\alpha_k$  dans l'incrément, qui réduit à chaque itération (on le divise par 2)

equation =>

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

en cherchant un facteur  $\alpha_k$  qui satisfait  $|f(x_{k+1})| < |f(x_k)|$

Elle permet d'empêcher un comportement cyclique avec un simple N-R.

Taylor:  $f(x_{k+1}) = f(x_k) + f'(c)(x_{k+1} - x_k) = f(x_k)(1 - \alpha_k \frac{f'(c)}{f'(x_k)})$

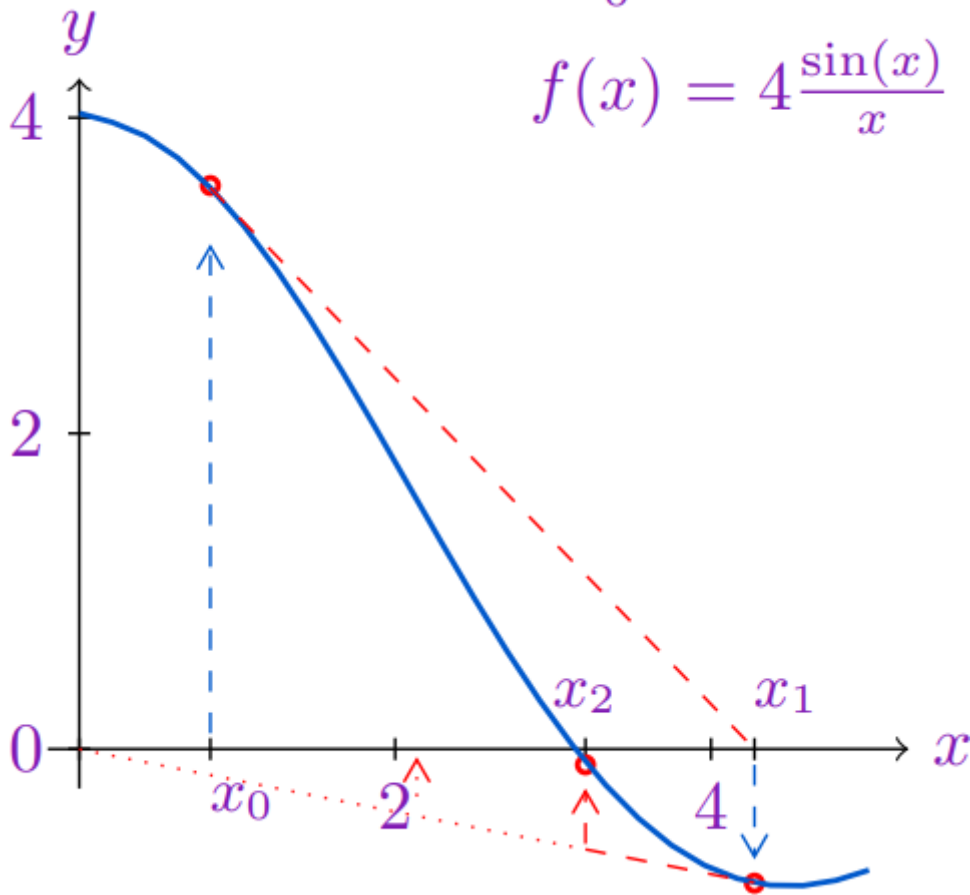
...

## besoin aussi de f' comme N-R

on utilise un facteur d'amortissement  $\alpha_k > 0$ , qu'on va diviser par 2 jusqu'à l'arret (on commence avec  $\alpha = 1$ )

EXEMPLE :  $x_0 = 0.83$

$$f(x) = 4 \frac{\sin(x)}{x}$$



Algo Octave:

```
function [n, sol] = algonrrecherchelin(f, fprime, x0)
    n = 0;
    alpha = 1;

    # critere d'arret
    while n < 69
        n++;
        p = f(x0)/fprime(x0)
        x = x0 - alpha*p;
        if abs(f(x)) < abs(f(x0));
            break
        endif
        alpha = alpha/2
        x0 = x
    endwhile
    sol = x;
endfunction
```

puis

```
[n_iter sol] = algonrrecherchelin(f, fprime, x0) #on peut aussi passer un critère d'arrêt plutôt que de le hardcoder
```

## variantes

on peut exprimer  $f'(x)$  sans le connaître...

- formule de différences finies...
- méthode de la sécante...

## Généralisation aux systèmes non-linéaires

LU...

### Newton-Corde

Principe: factorisation LU peut être coûteux pour système à taille importante => LU remplacée par méthode itérative.

Convergence n'est plus quadratique

Factorisation fait qu'une seule fois au début de l'algo

### Critères d'arrêt

- [+]  $\|f(x_k)\| \leq \epsilon_a$  (ou relativement  $\|f(x_k)\| \leq \epsilon_r \|f(x_0)\|$ )
- [-] nombre max d'itérations
- [-] Pour les méthodes de type Newton, le fait que la dérivée  $f'(x_k)$  ou son approximation soit non-inversible peut mener à l'arrêt de la méthode.

## Chap 6

---

Interpolation et approximation

... TODO

## Chap 7

---

Intégration

### Méthode des trapèzes

Intervalle régulier

Prendre un intervalle  $h$  entre  $x_i$  et  $x_{i+1}$

Aire trapèze =  $h * (f(a) + f(a + b))/2$  (petite base + grande base fois hauteur (l'intervalle) sur 2)

Plus  $h$  est petit, plus c'est précis

On risque que la longueur  $[a, b]$  soit pas un nombre entier de fois  $h$ ,  
alors  $h = (b - a)/n$  avec  $n$  le nombre d'intervalles

**Exacte pour tout polynome de degré au plus 1**

**Erreurs:**

$c \in ]x_i, x_{i+1}[$

Erreur locale:  $E_{loc}(h_i) = -\frac{1}{12}h_i^3 f''(c)$

$c \in ]a, b[$

Erreur globale:  $E_{glob}(h) = -\frac{1}{12}(b - a)h^2 f''(c)$

**Algo Octave:**

```
function [aire] = tp9trapezes (f, a, b, h)

    n=(b-a)/h;
    int = 0;

    for i = 0:n-1
        int = int + ( h/2 )*( f(a + h*i) + f(a + h*(i+1)) );
    endfor

    aire = int;

endfunction
```

## Methode de Simpson

Point au milieu de  $[a + h, a + 2h]$  :  $a + \frac{3}{2}h$

interpolation quadratique

intégrale =  $(f(a + h) + h * f(a + \frac{3}{2}h) + f(a + 2h))/6$

**Exact pour tout polynome de degré au plus 3**

Erreur globale:

$|E_{glob}| = -\frac{1}{12} * (b - a) * h^2 * |f''(c)|$

$c$  est un réel appartenant à  $[a, b]$

$|E_{glob}| \leq 10^{-6}$

$h^2 * |f''(c)| \leq 10^{-6}$

on va sortir un  $h_{\text{erreur}} \leq \text{sqrt}((12 * 10^{-6}) / ((b - a) * |f''(c)|))$

$n = (b - a) / h_{\text{erreur}}$  ; appartient PAS à N

$n = \text{ceil}(n)$

$h_{\text{effectif}} = (b - a) / n \leq h_{\text{erreur}}$

que quand la methode est pas exacte, que quand il y a une erreur à calculer

si elle est exacte le h est indépendant de l'erreur (car pas d'erreur) donc h qu'on veut

## Algo Octave

```
function [aire] = tp9simpson(f, a, b, n)
    h = (b-a)/n;
    int = 0;
    for i=0:n-1
        int = int + (h/6)*(f(a+i*h)+4*f(a+h*(i+0.5))+f(a+h*(i+1)));
    endfor

    aire = int;

endfunction
```

## Methode de Newton-Cotes

...

## Methode de Romberg

...

## Chap 8 éq différentielles avec cond Initiales

---

Prob de Cauchy (scalaire ou vectoriel)

$$\begin{cases} \frac{dy}{dt}(t) = f(t, y(t)), t \in [0, T] \\ y(0) = y_0 \end{cases}$$

## Methode d'Euler

Ordre: 1

Un pas de discrétisation:  $h_k = t_{k+1} - t_k$

(ou pas d'intégration).

h plus petit donne un meilleur solution (on observe que si on double le pas, l'erreur est 2x plus



grande (linéairement)).

Erreur  $|y_k - y(t_k)|$  semble proportionnelle à  $h$ .

D'ordre 2 l'erreur augmenterait de manière quadratique.

## Euler progressive

Méthode explicite

Stabilité: **pas toujours** stable

Principe: approcher la dérivée au point  $t_k$  par...

=> on va vers l'avant

$$y_{k+1} = y_k + h_k f(t_k, y_k)$$

Algo Octave:

```
function [y] = tp10eulerpro (f, y0, t) # t contient t_min et t_max

n = length(t);
h = t(2) - t(1); # h constante
y(1)=y0;

for i = 1:n-1
    y(i+1) = y(i)+h*f(t(i),y(i));
endfor

endfunction
```

## Euler retrograde

Méthode implicite (pas moyen d'isoler directement  $y_{k+1}$  à pd  $y_k$ )

Stabilité: **toujours** stable

Principe: approcher la dérivée au point  $t_{k+1}$  par...

=> on va vers l'arrière

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$$

(On peut utiliser la fonction Octave `fsolve(g, y(k))` pour la résolution d'équations non linéaires, où  $g$  la fctn et  $y(k)$  aux alentours où il faut chercher)

Algo Octave:

```

function [y] = tp10eulerret (f, y0, t) # t contient t_min et t_max

n = length(t);
h = t(2) - t(1); # h constante
y(1)=y0;

for i = 1:n-1
    g=@(X) y(i) + h*f(t(i+1),X) - X;
    y(i+1) = fsolve(g, y(i));
endfor

endfunction

```

## Ordre

methode est d'ordre  $n$  si:

$$\max_k |y_k - y(t_k)| \leq Ch^n$$

(Les méthodes d'Euler sont de **1er** ordre)

## Stabilité

Prenons prob de Cauchy où  $\frac{dy}{dt}(t) = -\beta y(t)$

La sol exacte est  $y(t) = y_0 e^{-\beta t}$  elle tend vers 0 pour  $\beta > 0$  et croît pour  $\beta < 0$

- methode (absolument) stable pour le prob de Cauchy avec  $\beta > 0$  si:  
elle produit une séquence de  $y_k, k = 1, 2, \dots$ , d'approximations de  $y(t_k)$  telle que:

$$y_k \rightarrow 0 \text{ lorsque } t_k \rightarrow \infty$$

(en gros lorsque la "suite"  $y_k$  tend vers 0 quand  $t_k \rightarrow \infty$ )

- euler **progressive**: pas toujours (stable si  $|A - h\beta_i| < 1, i = 1, \dots, n$ )
- euler **retrograde**: toujours stable (pour tout  $h$ )

ex: euler progressive absolument pas stable si  $h\beta < -2$

## Methode du second ordre

### Methode de Crank-Nicolson

Principe: methode s'obtient en approchant l'intégrale par la formule des trapèzes.

Methode implicite (pas moyen d'isoler directement  $y_{k+1}$  à pd  $y_k$ ).

Ordre: 2 ( $|y_k - y(t_k)| \propto h^2$ )

Stabilité: stable, quel que soit le pas  $h_k$ .

équation:

$$y_{k+1} = y_k + \frac{1}{2}h_k(f(t_k, y_k) + f(t_{k+1}, y_{k+1}))$$

où come avant  $h_k = t_{k+1} - t_k$

Algo Octave:

?

### Methode de Heun (ou Runge-Kutta d'ordre 2)

Principe: rendre la methode de Crank-Nicolson explicite sur base de la formule d'Euler progressive.

Methode explicite

Ordre: 2 ( $|y_k - y(t_k)| \propto h^2$ )

Stabilité: stable si  $h\beta < 2$ .

équation:

$$y_{k+1} = y_k + \frac{1}{2}h_k(f(t_k, y_k) + f(t_{k+1}, y_k + h_k f(t_k, y_k)))$$

en gros on a changé  $y_{k+1}$  dans le membre de droite par  $y_k + h_k f(t_k, y_k)$  (la formule d'Euler Progressive).

Ce qui la rend donc explicite.

Algo Octave:

?

### Methode multi-pas

...

# Chap 9 éq différentielles avec cond aux Limites

---

croisons les doigts que ça tombe pas à l'exam