
ECE
ÉCOLE D'INGÉNIEURS
ENGINEERING SCHOOL

Calcul Embarqué

1 - Recherche de racines d'équations non linéaires

Alain Le Gall – alain.legall@ece.fr

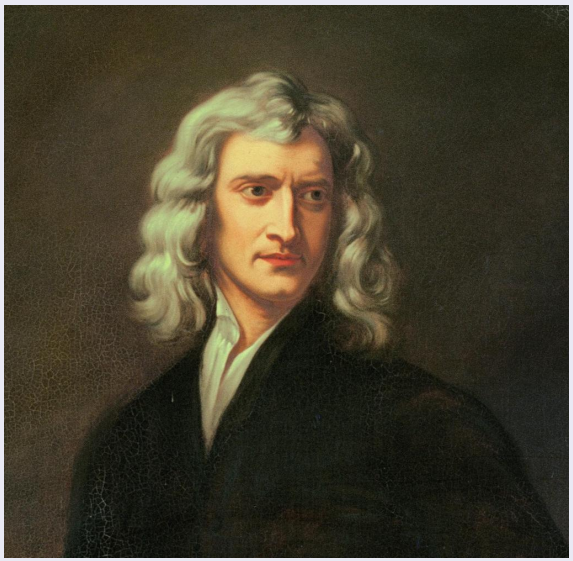
ECE – Paris & Lyon – ING3

année 2022 / 2023

Sommaire

Quelques mots avant de commencer

L'Homme a toujours eu besoin d'algorithmes pour l'aider à résoudre des problèmes concrets.



Par exemple, Isaac Newton et Joseph Raphson, vers les $17^{ème}$ - $18^{ème}$ siècles, tous deux mathématiciens voire aussi astronomes, ont conçu l'algorithme de Newton-Raphson qui permet de calculer les racines d'une équation non linéaire. En effet l'astronomie est un domaine où calcul est roi !

Introduction

Quelques mots avant de commencer

Poursuivant sur cette voie, et celle de nombreux autres, les scientifiques ont inventé beaucoup d'algorithmes qui permettent à l'homme du $XXI^{ème}$ siècle de résoudre des équations, des problèmes de physique, mathématiques, biologie, économie etc...Nous allons ici aborder des algorithmes utilisés dans des domaines très variés, mais en centrant notre étude pratique sur les Systèmes Embarqués, domaine de l'électronique. Nos algorithmes seront implémentés sur l'Arduino DUE, que l'on codera avec le langage C++.

Organisation

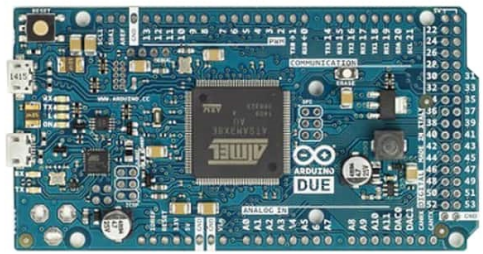
Les cours CI seront le lieu de découverte des algorithmes essentiels, les TD donneront l'occasion d'effectuer des études théoriques et informatiques, les TP quant à eux permettront d'appliquer les connaissances précédemment acquises à la résolution de questions d'ordre expérimental, à l'aide de l'Arduino DUE et du kit d'électronique.

Introduction

Important

En TD il est indispensable de disposer d'une calculatrice. Il est aussi indispensable en TP de disposer d'un ordinateur et de l'Arduino DUE pour exécuter les programmes que vous aurez à coder (on emploiera alors ARDUINO IDE 2 comme logiciel) ; et de disposer en plus de votre kit d'électronique avec ses composants.

Matériel nécessaire



Contexte de notre étude

Présentation

Il est très fréquent d'avoir à rechercher une ou des racines d'une fonction réelle f, c'est-à-dire d'avoir à résoudre l'équation $f(x) = 0$. Une quantité astronomique de problèmes d'étude technique, industrielle, R & D etc... se ramènent à la recherche de racines de fonctions. La plupart du temps, cette équation n'a pas de solution qui s'exprime simplement.

Exemple

Essayez par exemple de m'exprimer une solution non nulle de l'équation :

$$tan(x) = x$$

c'est-à-dire de l'équation :

$$f(x) = tan(x) - x = 0$$

On dit alors que l'on recherche une **racine** ou un **zéro** de la fonction f.

Contexte de notre étude

Exemple

Vous pouvez chercher longtemps, cette équation $f(x) = tan(x) - x = 0$ n'admet pas de solution non nulle qui s'exprime analytiquement, seule une approximation peut être trouvée, grâce à un calcul numérique qui emploie un algorithme de recherche de racine, issu d'une méthode de recherche de racine...

Objectif

Nous allons vous présenter dans ce cours différents de ces fameux algorithmes de recherche de racines. Comme on veut les comparer aussi, selon leur précision, leur vitesse, leurs avantages et inconvénients..., nous serons amenés à introduire des notions nouvelles mais pertinentes telles que par exemple l'**efficacité** d'un algorithme de recherche de racine.

Contexte de notre étude

Fil conducteur : le nombre d'or Φ

Nous prendrons, pour appuyer notre étude théorique, l'exemple de la recherche du nombre d'or Φ à partir de nos différents algorithmes de recherche de racines, afin de pouvoir les comparer facilement !

Calcul du nombre d'or

Le nombre d'or Φ est, entre autres propriétés, la solution positive de l'équation polynomiale du second degré très simple suivante :

$$x^2 - x - 1 = 0$$

Sa valeur, notée Φ, s'obtient donc facilement :

$$\Phi = \frac{1+\sqrt{5}}{2} \simeq 1,61803398874989...$$

Contexte de notre étude

Recherche du nombre d'or

Essayons donc de retrouver une approximation de cette valeur, à partir de l'équation dont il est solution. On note :

$$f(x) = x^2 - x - 1$$

Déterminons par des algorithmes le nombre d'or Φ

Il nous reste à déterminer par des méthodes numériques, que l'on va étudier dans ce cours, la racine positive de cette fonction f. Nous allons donc prendre connaissance de différents algorithmes de recherche de racine, et on les comparera sur cet exemple précis, pour en évaluer l'efficacité, d'autant plus facilement avec notre présent exemple où la racine Φ est connue de façon exacte (ce qui permettra facilement d'évaluer aussi la précision par exemple).

La méthode de dichotomie

Du grec ancien :

$$\delta\iota\chi\omega\tau\omega\mu\iota\alpha$$

qui signifie : "couper en deux parties" !

La méthode de dichotomie : justification théorique

Justification théorique de la méthode

On cherche à calculer α qui est une racine de la fonction réelle de la variable réelle $f(x)$. On a donc :

$$f(\alpha) = 0$$

Supposons que l'on connaisse un intervalle $I =]a, b[$ dans lequel la racine soit incluse :

$$\alpha \in]a, b[$$

La méthode de dichotomie consiste à diviser l'intervalle de recherche par deux à chaque itération en gardant la racine dans cet intervalle. La racine α est trouvée lorsque la largeur de l'intervalle (qui diminue au cours des itérations) est inférieure à une certaine tolérance, notée ϵ.

La méthode de dichotomie : algorithme

Algorithme 1 : Algorithme de dichotomie

Data : a et b (intervalle de recherche initial), ϵ (précision)

$fa = f(a)$

$fb = f(b)$

if $fa.fb < 0$ **then**

  while $|b - a| > \epsilon$ **do**

    $x = \frac{a+b}{2}$ et $y = f(x)$

    if $fa.y > 0$ **then**

      $a = x$

      $fa = y$

    else

      $b = x$

      $fb = y$

    end

  end

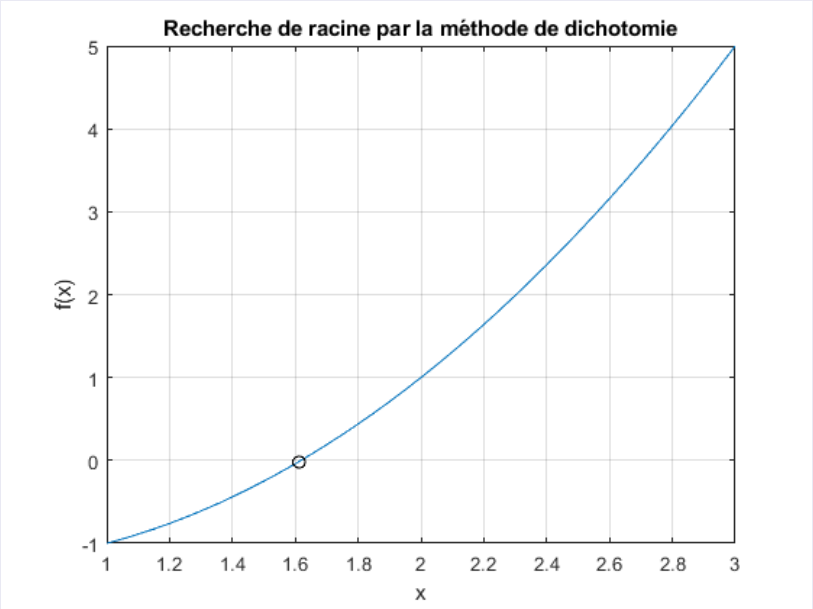
end

Result : x : approximation de la racine cherchée α

La méthode de dichotomie : exemple

Exemple : recherche du nombre d'or Φ

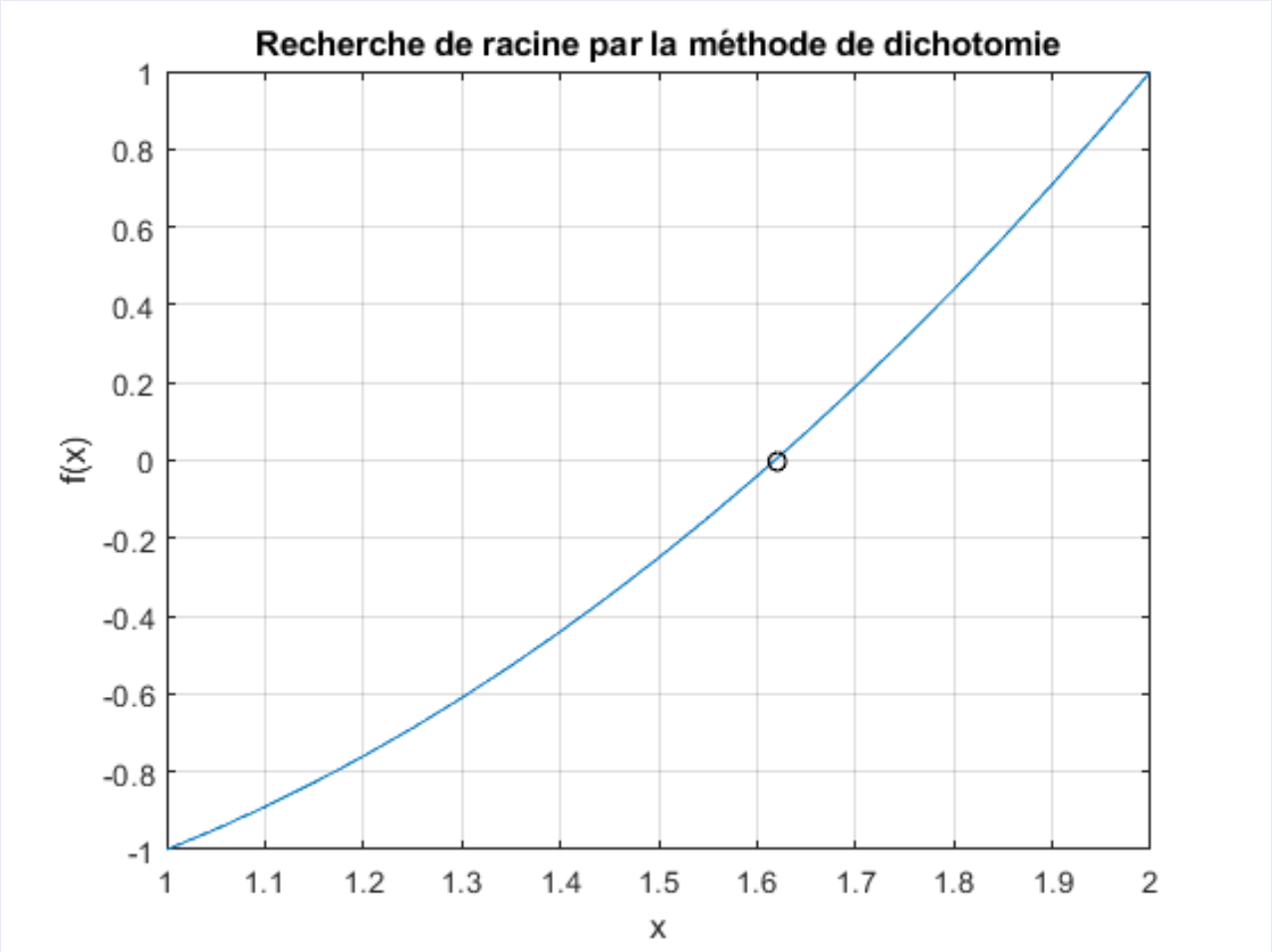
On étudie notre exemple : $f(x) = x^2 - x - 1$ On part de l'intervalle $[1; 3]$ pour lequel on sait sensiblement que la racine s'y trouve (on peut aussi choisir de beaucoup plus grandes valeurs moins "confortables").



Comme $f(x)$ change de signe entre $a = 1$ et $\frac{a+b}{2} = 2$, on sélectionne comme nouvel intervalle : $[1; 2]$

La méthode de dichotomie : exemple

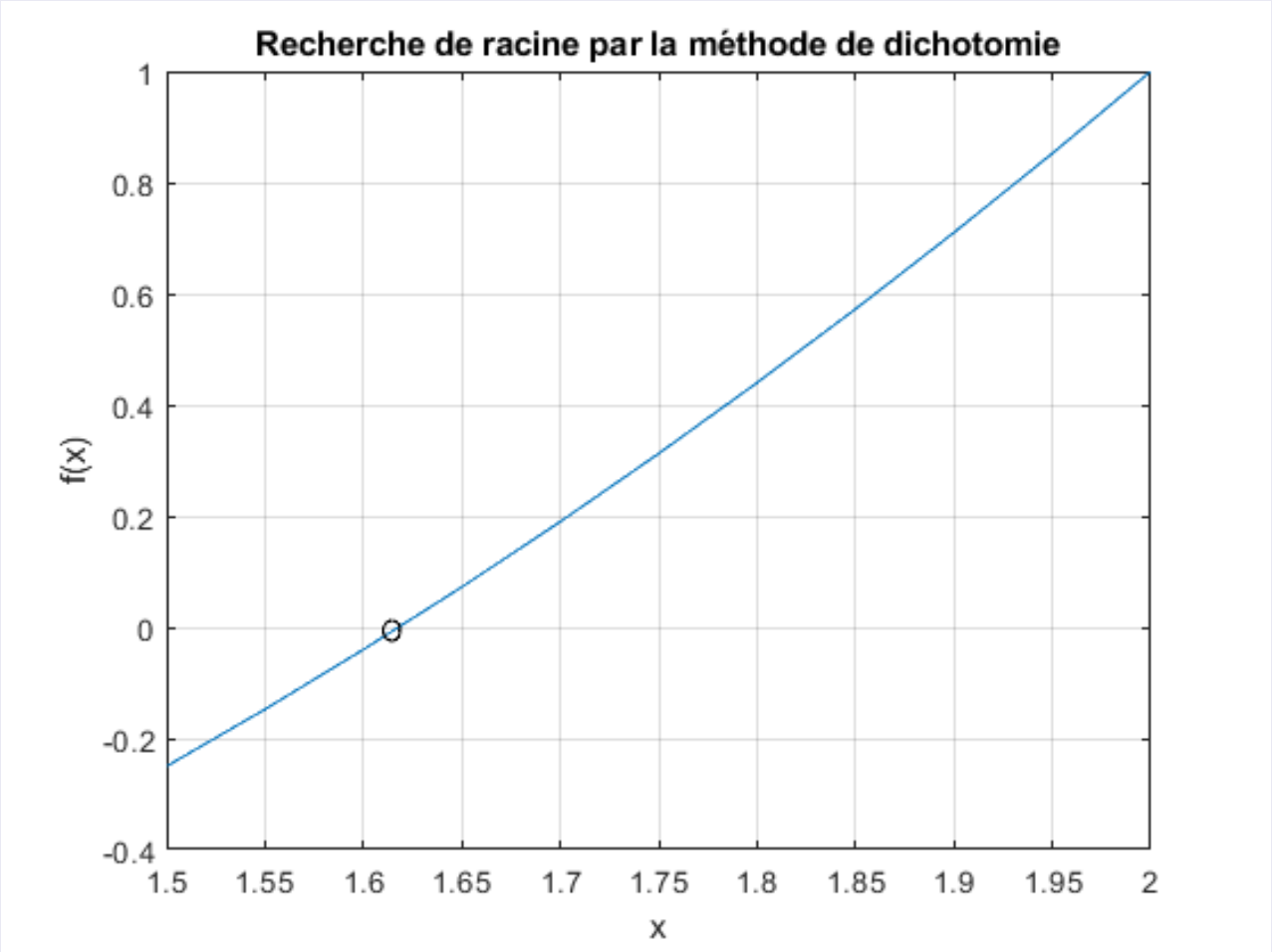
Exemple : recherche du nombre d'or Φ



Comme $f(x)$ change de signe entre $\frac{a+b}{2} = 1,5$ et $b = 2$, on sélectionne comme nouvel intervalle : $[1,5; 2]$

La méthode de dichotomie : exemple

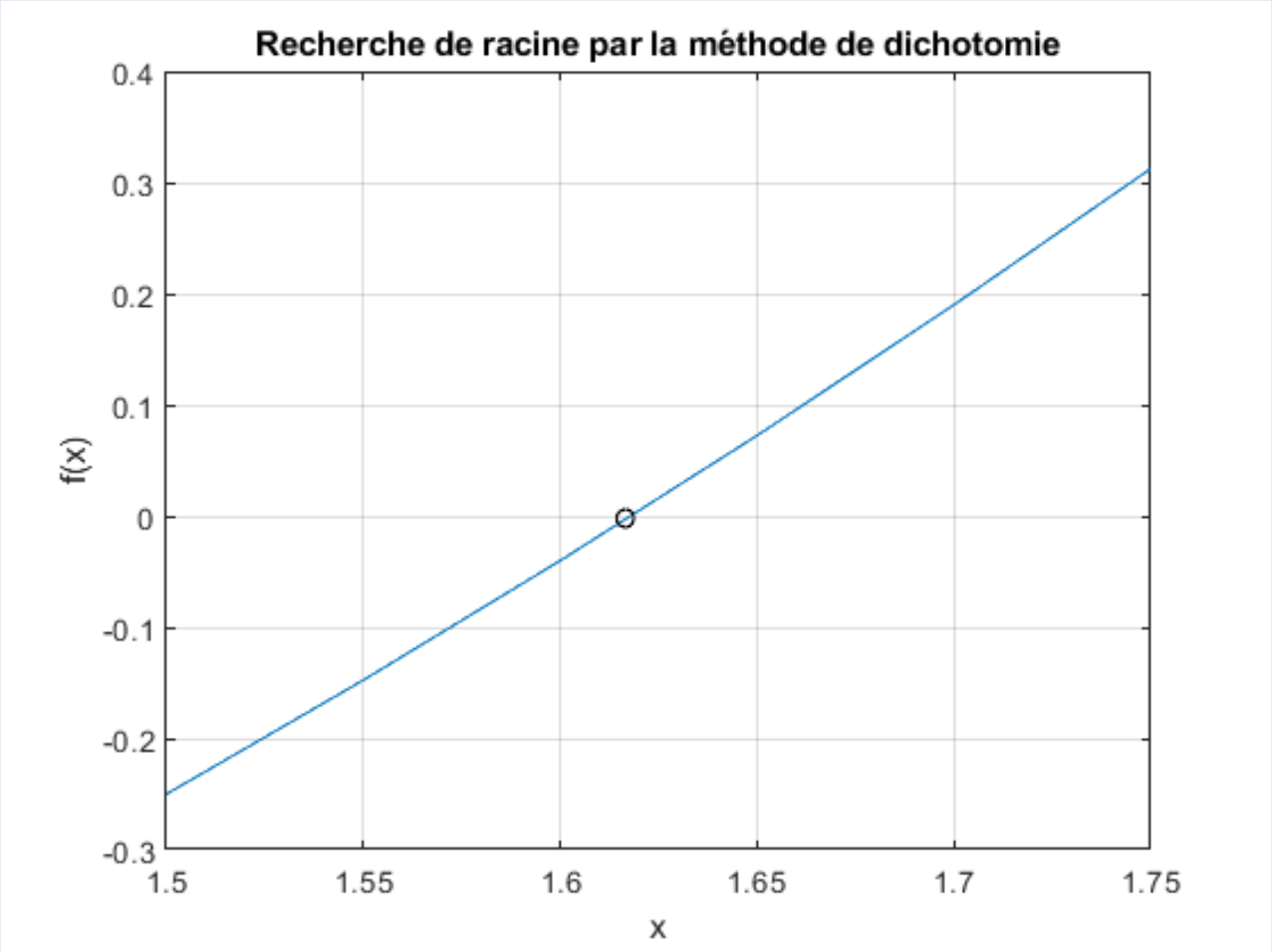
Exemple : recherche du nombre d'or Φ



Comme $f(x)$ change de signe entre $a = 1,5$ et $\frac{a+b}{2} = 1,75$, on sélectionne comme nouvel intervalle : $[1,5; 1,75]$

La méthode de dichotomie : exemple

Exemple : recherche du nombre d'or Φ



etc...etc... A chaque étape, la largeur de l'intervalle est divisée par 2 seulement ; ce processus converge, certes, mais pas rapidement !

La méthode de dichotomie : en langage C++

Programme en langage C++

```
#include <iostream>
#include <cmath>    // pour sqrt
#include <iomanip> // pour precision numerique

using namespace std;

double f(double x);
double dichotomie();

const double phi(0.5+0.5*sqrt(5.0));
char attends;
```

La méthode de dichotomie : en langage C++

Programme en langage C++

```
int main()
{
    double x(0.0);
    cout << "Dichotomie" << endl;
    x=dichotomie();

    cout<< "racine-x="<<setprecision(15)<<x<<endl;
    cout<< "f(x)="<<setprecision(15)<<f(x)<<endl;
    cout<<"erreur-commise:"<<setprecision(15)
                <<fabs(phi-x)<<endl;

    return 0;
}
```

La méthode de dichotomie : en langage C++

```
double f(double x){
    return x*x-x-1.0;
}
double dichotomie()
{
    double eps(1.0e-10),a(0.0),fa(0.0),b(0.0),fb(0.0),
        x(0.0),y(0.0);
    int n_max(200),n(0);
    do{
        cout<<"a=";
        cin>>a;
        fa=f(a);
        cout<<"b=";
        cin>>b;
        fb=f(b);
    }while((b<a)||(f(a)*f(b)>=0.0));
```

La méthode de dichotomie : en langage C++

```
do{
        n++;
        x=(a+b)/2.0;
        y=f(x);
        if(fa*y>0.0)
        {
                a=x;
                fa=y;
        }
        else{
                b=x;
                fb=y;
        }
}while((fabs(b-a)>eps)&&(n<n_max));
cout<<n<<"-iterations"<<endl;
return x;
}
```

La méthode de dichotomie : résultats numériques

Résultats numériques : cas où $\epsilon = 10^{-10}$

```
Dichotomie
choix-intervalle
a=1.0
b=3.0
35 iterations
racine x = 1.618033988869233
f(x)=-1.28714372493732e-10
erreur commise : 5.756284338076 51e-11
```

Résultats numériques : cas où $\epsilon = 10^{-10}$

On obtient 10 décimales justes !

Résultat obtenu pour 35 itérations !

La méthode de dichotomie : résultats numériques

Résultats numériques : cas où $\epsilon = 10^{-15}$

```
Dichotomie
choix-intervalle
a=1.0
b=3.0
51 iterations
racine x = 1.61803398874989
f(x)=-1.77635683940025e-15
erreur commise : 8.88178419700125e-16
```

Résultats numériques : cas où $\epsilon = 10^{-15}$

On obtient 15 décimales justes !

Résultat obtenu pour 51 itérations !

La méthode de dichotomie : résultats numériques

Résultats numériques : analyse

On a donc observé que le nombre d'itérations est assez élevé, mais permet d'atteindre la précision voulue. Sur cette fonction f simple, ce nombre élevé d'itérations ne ralentit pas trop le calculateur, mais si la fonction était bien plus compliquée, cela serait le cas. On n'emploie donc pas habituellement une méthode de dichotomie jusqu'à la précision finale : on débute par cette méthode de dichotomie puis on passe le relai à une méthode plus rapide telle que la méthode de Newton-Raphson par exemple.

La méthode de dichotomie : remarques

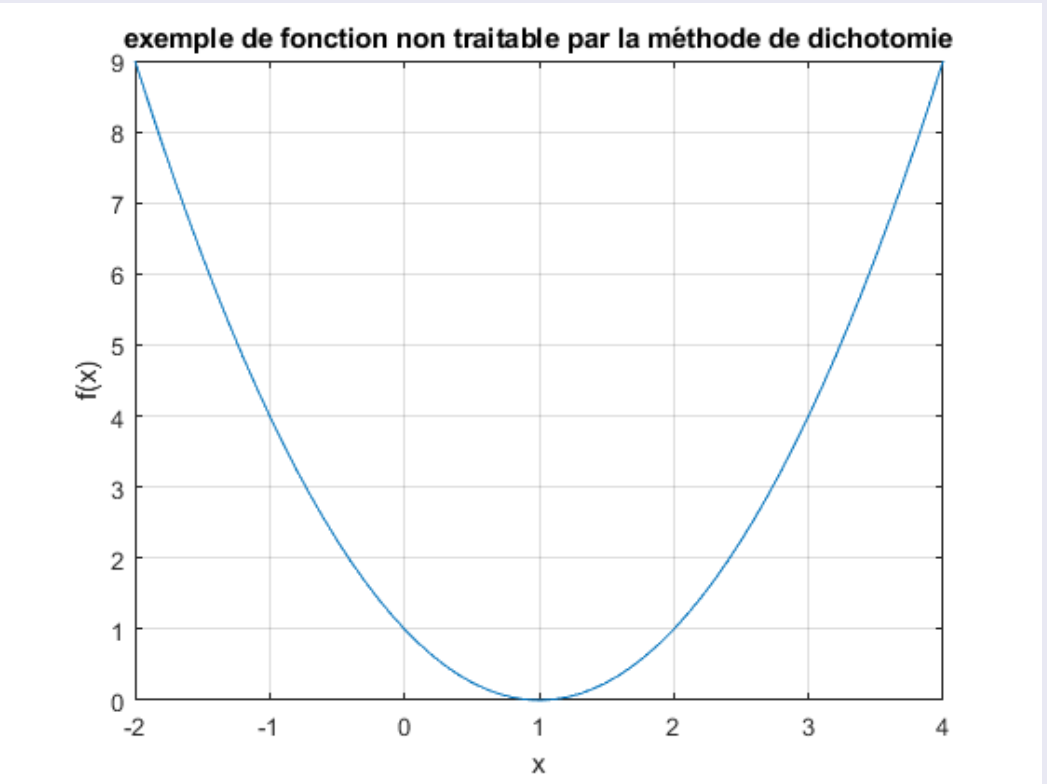
Résultats numériques : remarques

- Le test $|b - a| < \epsilon$ permet de cerner rapidement la racine mais on peut aussi employer le test $|f(x)| < \epsilon$, ce qui permet évidemment aussi d'obtenir une bonne précision sur cette même racine α. Étant donné que la méthode de dichotomie est plus utilisée pour trouver une approximation de la racine, on utilisera de préférence ce test $|b - a| < \epsilon$ puis on appliquera, comme dit précédemment, ensuite un autre algorithme de recherche de racine pour poursuivre la recherche, comme par exemple l'algorithme de Newton-Raphson (voir plus bas dans ce cours), afin d'obtenir ensuite une précision optimale sur la racine α.

La méthode de dichotomie : remarques

Résultats numériques : remarques

- La méthode de dichotomie ne fonctionne bien entendu pas dans des cas de figure comme celui présenté ci-dessous car la fonction f ne change pas de signe (condition nécessaire pour faire fonctionner l'algorithme de dichotomie).

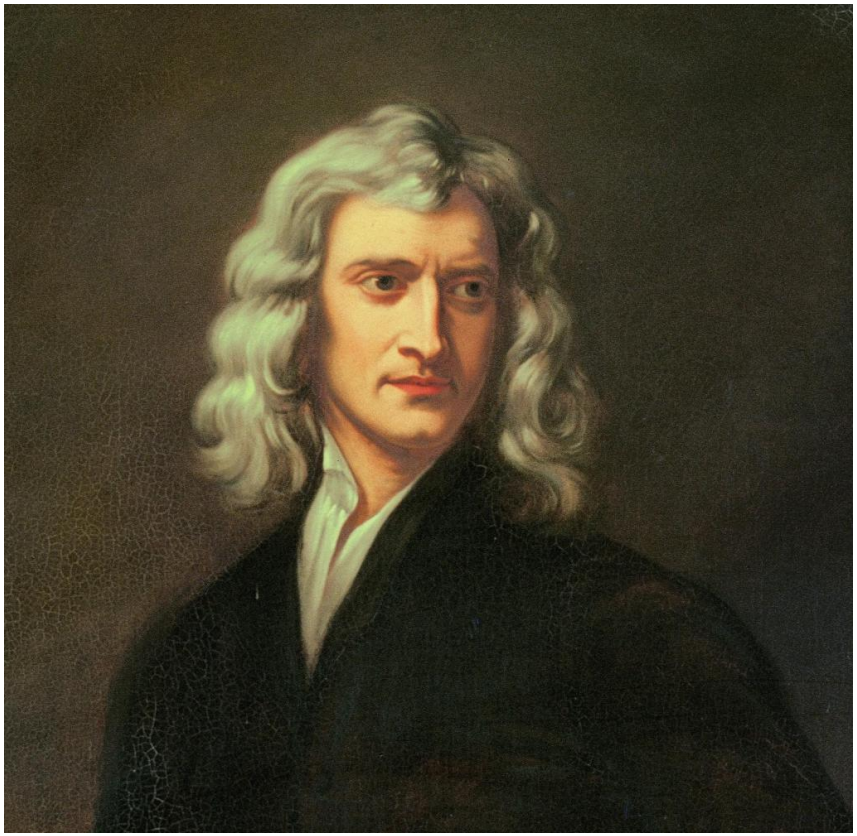


La méthode de dichotomie : conclusion

Résultats numériques : conclusion

L'algorithme de dichotomie est très efficace pour "cerner" une racine, mais il est plutôt lent ; on l'emploie en général pour débuter la recherche d'une racine, ensuite une méthode plus rapide est employée (cf. méthodes décrites plus loin, par exemple la méthode de Newton-Raphson).

La méthode de Newton-Raphson



Newton (1642-1727) et Raphson (1648-1715) ont inventé individuellement chacun une méthode de recherche de racines qui semblait différente à première vue, mais Laplace (1749-1827) a montré que c'était une même et seule méthode !

La méthode de Newton-Raphson

Justification théorique de la méthode

On cherche à calculer α une racine de la fonction réelle de la variable réelle $f(x)$. On suppose donc :

$$f(\alpha) = 0$$

Posons $x = \alpha + h$, avec $h << 1$ (on se place près de la racine) et écrivons un développement de Taylor de la fonction f :

$$f(x) = f(\alpha + h) = f(\alpha) + hf'(\alpha) + ... + \frac{h^p}{p!} f^{(p)}(\alpha) + ...$$

Si α est une racine d'ordre p de f, alors par définition on a :

$$\begin{cases} f(\alpha) = f'(\alpha) = ... = f^{(p-1)}(\alpha) = 0 \\ f^{(p)}(\alpha) \neq 0 \end{cases}$$

La méthode de Newton-Raphson

Justification théorique de la méthode

Ainsi, $f(x) = \frac{h^p}{p!} f^{(p)}(\alpha)$+termes négligeables
Soit encore, avec une bonne approximation :
$f(x) = C^{te}.h^p = C^{te}.(x-\alpha)^p$ donc $f'(x) = p.C^{te}.(x-\alpha)^{p-1}$ et ainsi :

$$s_1 = \frac{f'(x)}{f(x)} = \frac{p}{x-\alpha}$$

On en déduit : $x - \alpha = \frac{p}{s_1}$ soit $\alpha = x - \frac{p}{s_1}$.
Donc en résumé :

$$\alpha = x - p\frac{f(x)}{f'(x)}$$

La méthode de Newton-Raphson

Justification théorique de la méthode

Interprétation : Si $x = \alpha + h$ ($h << 1$) est proche de la racine α cherchée, alors une bonne approximation de α est donnée par :

$$\boxed{\alpha \simeq x - p\frac{f(x)}{f'(x)}}$$

Généralisation : On peut réitérer le processus et alors la suite définie par récurrence selon :

$$\boxed{x_{n+1} = x_n - p\frac{f(x_n)}{f'(x_n)}} \quad (1)$$

peut converger, en général cela dépend de la valeur initiale x_0 choisie qui doit être assez "proche" de α , vers la racine recherchée α :

$$\lim_{n \to +\infty} x_n = \alpha$$

La méthode de Newton-Raphson

Relation de récurrence explicite

Il est aussi possible de trouver une relation de récurrence directe afin de définir la suite x_k qui converge vers une racine α de la fonction f. Prenons notre exemple du nombre d'or. Ici, $f(x) = x^2 - x - 1$ et donc l'algorithme de Newton-Raphson s'écrit (on a aussi $p = 1$ ici) :

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

avec aussi $f'(x) = 2x - 1$, donc :

$$x_{k+1} = x_k - \frac{x_k^2 - x_k - 1}{2x_k - 1}$$

soit, après calculs :

$$x_{k+1} = \frac{x_k^2 + 1}{2x_k - 1}$$

La méthode de Newton-Raphson

Relation de récurrence explicite

Il est alors facile de calculer les différents termes de cette suite, qui converge vers le nombre d'or Φ. Même avec une simple calculette, le calcul est facile à mener :

1. Taper une valeur positive au hasard sur la calculette et valider
2. En utilisant la touche "ANS" ou "REP" selon le modèle de la calculette, saisir l'équation suivante :

$$(ANS^2 + 1)/(2 * ANS - 1)$$

3. Valider un certains nombre de fois jusqu'à ce que la valeur se stabilise à l'écran
4. Lorsque le processus a convergé, la valeur qui s'affiche est une approximation, dans les limites de la calculette, de la valeur de la racine Φ recherchée

La méthode de Newton-Raphson

Conclusion

Conclusion :La méthode de Newton-Raphson permet, soit de construire explicitement une suite définie par récurrence qui converge vers la racine voulue ; soit de coder en langage informatique un programme avec les définitions des fonctions f et f' et de faire fonctionner l'algorithme. Il est évident que si les fonctions f et f' sont simples d'expression, la première méthode est à privilégier. Cependant, la méthode informatique (cf ci-dessous) fonctionne à priori toujours.

La méthode de Newton-Raphson

Algorithme

Algorithme 2 : Algorithme de Newton-Raphson

Data : x_0(valeur initiale), p (ordre de la racine), ϵ (précision)

while $|x_{k+1} - x_k| > \epsilon$ **do**

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

end

Result : x_n approximation de la racine cherchée α

La méthode de Newton-Raphson

Méthode de Newton-Raphson : programme Arduino Due

On veut coder un programme avec Arduino IDE 2+ sur l'Arduino DUE. Pour cela, on va étudier un exemple.

Exemple : le nombre d'or

On prend $f(x) = x^2 - x - 1$ et on part de $x_0 = 1 > 0$ pour obtenir la valeur du nombre d'or Φ.

```
x=1.000000000000000
x=2.000000000000000
x=1.666666666666667
x=1.619047619047619
x=1.618034447821682
x=1.618033988749989
x=1.618033988749895
nombre d iterations = 7
```

La méthode de Newton-Raphson

Méthode de Newton-Raphson : exemple

Si on commence avec $x_0 = 1000$, on obtient :

```
x=1000.000000000000000
x=500.250625312656325
x=250.376563280075970
x=125.440782875014960
x=62.975393807315150
x=31.747700842199005
x=16.143851892668763
x=8.361877743646711
x=4.510436419404158
x=2.661061597961768
x=1.869740509888466
x=1.641161060018613
x=1.618268338415427
x=1.618034013305614
x=1.618033988749895
nombre d iterations = 15
```

La méthode de Newton-Raphson

Méthode de Newton-Raphson : remarques

- Cet algorithme fonctionne à merveille, et il n'est même pas obligatoire de connaitre l'ordre p de la racine (on peut poser $p = 1$ si p est inconnu).
- Il faut connaitre f, f', ce qui n'est pas très contraignant en général.
- On verra en TD et TP que l'on peut calculer l'ordre p de la racine à la volée, ce qui permettra d'accélérer encore plus cette méthode ; il faut alors connaitre aussi f''.

Méthode de Newton-Raphson : conclusion

L'algorithme de Newton-Raphson est rapide, bien plus que celui de dichotomie, et très précis. Il permet d'obtenir en très peu d'étapes la valeur précise d'une racine.On peut même partir assez "loin" de la racine en prenant un x_0 éloigné de la racine.

La méthode de Laguerre

Edmond Laguerre (1834-1886) a inventé une méthode de recherche de racines très efficace pour n'importe quelle fonction, et même extrêmement performante pour la recherche de racines de polynômes, où cette méthode excelle.

La méthode de Laguerre

Justification théorique de la méthode

Cet algorithme est fourni ici directement, la démonstration étant nettement plus compliquée ; il suffira de se reporter à un ouvrage spécialisé pour l'obtenir. L'intérêt de présenter cette méthode est que c'est un algorithme spécialement conçu pour rechercher les racines des polynômes, et il est vraiment très efficace.

La méthode est encore basée sur une formule de récurrence. Notons n le degré du polynôme étudié, et p l'ordre de la racine recherchée. On admet alors que la relation de récurrence suivante converge vers la racine :

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) + signe(f'(x_k))\sqrt{(\frac{n-p}{p})((n-1)f'^2(x_k) - nf(x_k)f''(x_k))}}$$

La méthode de Laguerre

Méthode de Laguerre : algorithme

Algorithme 3 : Algorithme de Laguerre

Data : x_0(valeur initiale), p (ordre de la racine), n (degré du polynôme), ϵ (précision)

while $|x_{k+1} - x_k| > \epsilon$ **do**

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k)+signe(f'(x_k))\sqrt{(\frac{n-p}{p})((n-1)f'^2(x_k)-nf(x_k)f''(x_k))}}$$

end

Result : x_n approximation de la racine cherchée α

La méthode de Laguerre

Méthode de Laguerre : programme Arduino Due

On veut coder un programme avec Arduino IDE 2+ sur l'Arduino DUE. Pour cela, on va étudier trois exemples.

Exemple 1 : le nombre d'or

On prend $f(x) = x^2 - x - 1$ et on part de $x_0 = 1 > 0$ pour obtenir la valeur du nombre d'or Φ.

```
x=1.00000000000000
x=1.61803398874990
f(x) = 0.00000000000000
nombre d iterations = 1
Dt (ms)= 704
```

La méthode de Laguerre

Méthode de Laguerre : exemple 1

Si on commence avec $x_0 = 1000$, on obtient :

```
x=1000.00000000000000
x=1.61803398874986
x=1.61803398874990
f(x) = 0.00000000000000
nombre d iterations = 2
Dt (ms)= 1080
```

Exemple 2 : un polynôme plus compliqué

On prend $f(x) = x^7 - x^5 - 3x^2 + 1$ et on part de $x_0 = 1 > 0$ pour rechercher une racine. Alors, $f'(x) = 7x^6 - 5x^4 - 6x$ et $f''(x) = 42x^5 - 20x^3 - 6$

La méthode de Laguerre

Méthode de Laguerre : exemple 2

Si on commence avec $x_0 = 1$, on obtient :

```
x=1.00000000000000
x=0.69098300562505
x=0.56705621027818
x=0.56585135857854
x=0.56585135838876
f(x) = 0.00000000000000
nombre d iterations = 4
Dt (ms)= 11028
```

Exemple 3 : une fonction non polynômiale

On prend $f(x) = e^{-x}sin(x) + x + 1$ et on part de $x_0 = 1 > 0$ pour rechercher une racine. Alors, $f'(x) = e^{-x}(cos(x) - sin(x)) + 1$ et $f''(x) = -2e^{-x}cos(x)$

La méthode de Laguerre

Méthode de Laguerre : exemple 3

Si on commence avec $x_0 = 1$, on obtient :

```
x=10.00000000000000
x=-1.04772722366574
x=-0.12367614388514
x=-0.08425596851049
x=-0.08427175092124
f(x) = 0.00000000000000
nombre d iterations = 4
Dt (ms)= 5547
```

La méthode de Laguerre

Méthode de Laguerre : remarques

- Cet algorithme fonctionne à merveille avec des polynômes, et il n'est même pas obligatoire de connaitre l'ordre p de la racine (on peut poser $p = 1$ si p est inconnu).
- On peut bien entendu l'utiliser si f est une fonction différente d'un polynôme, choisir alors n comme le nombre total de racines de la fonction, et fixer $p = 1$ aussi si l'ordre de la racine est inconnu.
- Il faut connaitre f, f' et f'' comme dans l'algorithme de Halley (cf TD), ce qui peut être contraignant si f n'est pas un polynôme.

Méthode de Laguerre : conclusion

L'algorithme de Laguerre est extrêmement rapide, surtout en ce qui concerne les polynômes, et très précis. Il permet d'obtenir en très peu d'étapes la valeur précise d'une racine. Le seul inconvénient est qu'il faut connaitre les dérivées première et seconde de f (mais cela ne pose pas de soucis avec les polynômes en général !).

Conclusion finale

Nous voilà arrivés au terme de notre voyage informatique au pays des algorithmes de recherche de racines ! La plupart des racines des équations à coefficients réels de la variable réelle peuvent être obtenues par l'une ou l'autre des méthodes vues dans ce livre. En général, on conseille la méthode de Halley (cf TD), très rapide à condition de connaitre jusqu'à la dérivée seconde de la fonction étudiée bien entendu ; dans le cas contraire se rabattre sur l'excellente méthode de Newton-Raphson, ou encore une méthode plus lente mais moins exigeante. Bien entendu, si vous travaillez sur des polynômes, la méthode conseillée est celle de Laguerre, alors extrêmement rapide. On peut aussi mélanger les méthodes, à savoir commencer par un algorithme peu précis pour approcher la racine, puis finir la recherche de la racine par un algorithme plus précis. Enfin, bien entendu, il existe d'autres méthodes de recherche de racines qui n'ont pas été abordées ici, que vous pourrez étudier ultérieurement.

Ordre de convergence et efficacité d'un processus itératif

Rappel : ordre p d'une racine

On a vu qu'une racine a un certain ordre p, nombre entier tel que $p \geq 1$, pris égal à $p = 1$ quand il est inconnu, et que la connaissance de cet ordre p permet très souvent d'accélérer la recherche de la racine concernée.

Maintenant : ordre m d'une méthode de recherche de racine

Soit un processus itératif qui converge vers une racine α.

$$x_{k+1} = \varphi(x_k)$$

On note : $\epsilon_k = \alpha - x_k$: l'erreur de x_k par rapport à la racine α à la $k^{ième}$ itération.

$$\epsilon_{k+1} = \alpha - x_{k+1} = \alpha - \varphi(x_k) = \alpha - \varphi(\alpha - \epsilon_k)$$

$$\alpha - \epsilon_{k+1} = \varphi(\alpha - \epsilon_k)$$

Ordre de convergence et efficacité d'un processus itératif

Maintenant : ordre *m* d'une méthode de recherche de racine

A la limite, on aura (théorème du point fixe), comme $\lim_{k\to+\infty} \epsilon_k = 0$, :

$$\alpha = \varphi(\alpha)$$

Et, comme

$$\epsilon_{k+1} = \alpha - \varphi(\alpha - \epsilon_k)$$

il vient en faisant un développement limité de φ autour de α :

$$\epsilon_{k+1} = \alpha - \varphi(\alpha) + \epsilon_k \varphi'(\alpha) - \epsilon_k^2 \frac{\varphi''(\alpha)}{2!} + \epsilon_k^3 \frac{\varphi'''(\alpha)}{3!} + \ldots$$

Or, $\alpha - \varphi(\alpha) = 0$, donc :

$$\boxed{\epsilon_{k+1} = \varphi'(\alpha)\epsilon_k - \frac{\varphi''(\alpha)}{2!}\epsilon_k^2 + \frac{\varphi'''(\alpha)}{3!}\epsilon_k^3 + \ldots} \quad (2)$$

Ordre de convergence et efficacité d'un processus itératif

Maintenant : ordre m d'une méthode de recherche de racine

On dira qu'un processus itératif $x_{k+1} = \varphi(x_k)$ est d'ordre m si l'on a pour tout entier naturel k :

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

(avec C : constante).

Différents ordres possibles

*Si $\varphi'(\alpha) \neq 0$:

$$\epsilon_{k+1} \simeq \varphi'(\alpha)\epsilon_k$$

Le processus itératif est d'ordre 1, on a donc une convergence linéaire.

Ordre de convergence et efficacité d'un processus itératif

Différents ordres possibles

*Si $\varphi'(\alpha) = 0$ et $\varphi''(\alpha) \neq 0$:

$$\epsilon_{k+1} \simeq -\frac{\varphi''(\alpha)}{2}\epsilon_k^2$$

Le processus itératif est d'ordre 2, on a donc une convergence quadratique.

*Si $\varphi'(\alpha) = \varphi''(\alpha) = 0$ et $\varphi'''(\alpha) \neq 0$:

$$\epsilon_{k+1} \simeq \frac{\varphi'''(\alpha)}{6}\epsilon_k^3$$

Le processus itératif est d'ordre 3, on a donc une convergence cubique.

Etc etc...Il peut aussi y avoir des ordre de convergence qui ne sont pas des entiers naturels (cf méthode de la sécante).

Ordre de convergence et efficacité d'un processus itératif

Nombre de décimales justes lors du calcul de la racine

On a l'erreur $\epsilon_k = \alpha - x_k$ que l'on va écrire sous la forme

$$\epsilon_k = w_k 10^{-n_k}$$

avec :

- $0 < w_k < 1$
- n_k : nombre de décimales justes dans la valeur x_k approchée de α

Nombre de décimales justes lors du calcul de la racine

On définit ensuite la quantité

$$S_k = -log_{10}(\epsilon_k)$$

Ordre de convergence et efficacité d'un processus itératif

Nombre de décimales justes lors du calcul de la racine

On calcule alors que :

$$S_k = -log_{10}(\epsilon_k) = -log_{10}(w_k) + n_k$$

Comme on a : $0 < w_k < 1$, comme par exemple pour typiquement $n_k = 6$ et typiquement $w_k \simeq 0.5$, on obtient alors $log_{10}(w_k) \simeq -0,3 << 6$ donc $log_{10}(w_k)$ est négligeable. De plus, plus k augmente, plus n_k augmente et donc très rapidement : $n_k >> -log_{10}(w_k)$. On en conclut que :

$$S_k \simeq n_k$$

Ordre de convergence et efficacité d'un processus itératif

Nombre de décimales justes lors du calcul de la racine

Dans un processus d'ordre m :

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

Ainsi,

$$S_{k+1} = -log_{10}(\epsilon_{k+1}) = -log_{10}(C) - mlog_{10(\epsilon_k)}$$

$$S_{k+1} = mS_k - log_{10}(C)$$

Pour k assez grands, on aura ϵ_k assez faible, et donc S_k assez grand, et ainsi $- - log_{10}(C)$ sera négligeable devant mS_k. On en déduit :

$$S_{k+1} \simeq mS_k$$

Or, on sait que dans ces conditions, on a : $S_k \simeq n_k$, donc on en déduit que :

$$\boxed{n_{k+1} \simeq m.n_k} \quad (3)$$

Ordre de convergence et efficacité d'un processus itératif

Nombre de décimales justes lors du calcul de la racine

soit encore :

$$n_k = m^k n_0$$

Conclusion : Dans un processus d'ordre m, le nombre de décimales justes dans la suite x_k est multiplié par m à chaque itération !

Nombre de décimales justes lors du calcul de la racine

Ainsi, on verra que la méthode de Newton-Raphson est d'ordre 2, donc à chaque itération de cette méthode, on double le nombre de décimales justes ! La méthode de Halley est, elle, d'ordre 3 ! Ainsi, on triple le nombre de décimales justes à chaque itération de cette méthode ! Etc...

Comparaison de deux processus différents. Efficacité d'un processus

Définitions et démonstration

On sait que $S_{k+1} = mS_k$ donc $S_k = m^k S_0$. Soient deux processus :

* $S^{(1)}$: un processus d'ordre m_1
* $S^{(2)}$: un processus d'ordre m_2

Hypothèse :on va les comparer pour un même résultat obtenu à partir des mêmes conditions initiales.
On a : $S_k^{(1)} = m_1^k S_0^{(1)}$ et $S_k^{(2)} = m_2^k S_0^{(2)}$. Donc, d'après l'hypothèse ; il faut, pour $S_0^{(1)} = S_0^{(2)}$, que l'on obtienne : $S_{k_1}^{(1)} = S_{k_2}^{(2)}$. On en déduit que :

$$m_1^{k_1} = m_2^{k_2}$$

$$k_1 log_{10}(m_1) = k_2 log_{10}(m_2)$$

$$\frac{k_1}{k_2} = \frac{log_{10}(m_2)}{log_{10}(m_1)}$$

Comparaison de deux processus différents. Efficacité d'un processus

Démonstration

Soit :

- θ : le travail par itération (nombre d'opérations pendant une itération).
- T : le travail total (pour obtenir le résultat final).

On a évidemment $T_1 = k_1\theta_1$ et $T_2 = k_2\theta_2$ et :

$$\frac{T_1}{T_2} = \frac{k_1\theta_1}{k_2\theta_2} = \frac{log_{10}(m_2)}{log_{10}(m_1)}\frac{\theta_1}{\theta_2} = \frac{\frac{1}{\theta_2}log_{10}(m_2)}{\frac{1}{\theta_1}log_{10}(m_1)} = \frac{log_{10}(m_2^{\frac{1}{\theta_2}})}{log_{10}(m_1^{\frac{1}{\theta_1}})}$$

On définit l'efficacité E d'un processus par une fonction décroissante de T nécessairement. Par exemple, $E = 10^{\frac{1}{T}}$. On a alors $T = \frac{1}{log_{10}(E)}$.

Comparaison de deux processus différents. Efficacité d'un processus

Conclusion

Ainsi :

$$\frac{T_1}{T_2} = \frac{log_{10}(E_2)}{log_{10}(E_1)}$$

Ce qui fournit immédiatement le résultat suivant pour l'efficacité d'un processus :

$$\boxed{E = m^{\frac{1}{\theta}}} \tag{4}$$

avec :

- m = ordre du processus ($\epsilon_{k+1} = C\epsilon_k^m$)
- θ = travail par itération du processus (nombre d'opérations pendant une itération)

Comparaison de deux processus différents. Efficacité d'un processus

Méthode de Newton-Raphson

On utilise le résultat très général indépendant de la méthode employée, déjà démontré :

$$\epsilon_{k+1} = \varphi'(\alpha)\epsilon_k - \frac{\varphi''(\alpha)}{2!}\epsilon_k^2 + \frac{\varphi'''(\alpha)}{3!}\epsilon_k^3 + ...$$

avec ici :

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

$$\varphi'(x) = 1 - \frac{f'^2(x) - f(x)f''(x)}{f'^2(x)}$$

donc, sachant que par définition, $f(\alpha) = 0$, il vient $\varphi'(\alpha) = 1 - 1 = 0$

Comparaison de deux processus différents. Efficacité d'un processus

Méthode de Newton-Raphson

Pour calculer ensuite $\varphi''(\alpha)$, on écrit :

$$\varphi'(x) = 1 - \frac{f'^2(x) - f(x)f''(x)}{f'^2(x)} = 1 - 1 + \frac{f(x)f''(x)}{f'^2(x)} = \frac{f(x)f''(x)}{f'^2(x)}$$

On pose

$$\Psi(x) = \frac{f''(x)}{f'^2(x)}$$

$$\varphi'(x) = f(x)\Psi(x)$$

Ainsi,

$$\varphi''(x) = f'(x)\Psi(x) + f(x)\Psi'(x)$$

$$\varphi''(\alpha) = f'(\alpha)\Psi(\alpha) + 0 = f'(\alpha)\frac{f''(\alpha)}{f'^2(\alpha)} = \frac{f''(\alpha)}{f'(\alpha)} \neq 0$$

Comparaison de deux processus différents. Efficacité d'un processus

Méthode de Newton-Raphson

$$\boxed{\epsilon_{k+1} = -\frac{f''(\alpha)}{2f'(\alpha)}\epsilon_k^2} \tag{5}$$

L'algorithme de Newton-Raphson est donc un processus d'ordre $m = 2$ et :

$$n_{k+1} \simeq 2n_k$$

Conclusion : Le nombre de décimales justes est doublé à chaque itération.

Comparaison de deux processus différents. Efficacité d'un processus

Méthode de Newton-Raphson

Reste à calculer le nombre d'opération par itération θ, qui dépend de la méthode et de la définition des fonctions $f(x)$ et $f'(x)$ étudiées. En pratique, on ne tient compte que des multiplications et des divisions, et pas des additions/soustractions. Les fonction "complexes" comme les fonctions trigonométriques, exponentielles etc... comptent en pratique pour $\theta = 5$.

Méthode de Newton-Raphson

Prenons l'exemple du nombre d'or Φ. Il est racine de la fonction $f(x) = x^2 - x - 1$. D'où $f'(x) = 2x - 1$. A chaque itération on effectue la récurrence : $x_{n+1} = x_n - \frac{f(x_n}{f'(x_n)}$. Donc, $\theta = 3$. L'efficacité de cet algorithme de Newton-Raphson est donc ici : $E = m^{\frac{1}{\theta}} = 2^{1/3} \simeq 1,26$. E n'a de sens que comparée à l'efficacité d'autres méthodes.

Comparaison de deux processus différents. Efficacité d'un processus

Prolongements

Il est bien évident que l'efficacité, qui traduit la simplicité d'un algorithme à fournir un résultat de précision donnée, doit aussi donner lieu à un autre indice, de performance celui-là, qui tiendra compte aussi de la durée d'exécution Δt de l'algorithme sur un microprocesseur (cf TP).

Pour finir...

Exercice Quizz

Pour finir...

Sondage sur ce cours