L. DIDIER & R. CABANE

Comment (presque) résoudre une équation

Présentation et objectifs

Dans les programmes

Première : exemple d'algorithme Méthode de Newton, en se limitant à des cas favorables.	**Terminale :** exemples d'algorithme Méthode de dichotomie. Méthode de Newton, méthode de la sécante.

Situation déclenchante

Imaginons qu'une fonction continue f ait été définie sur un intervalle $D=[a;b]$, et que cette fonction change de signe et s'y annule une fois. Par exemple, $f(x)=x^3-x-0{,}5$ sur l'intervalle $D=[0{,}5\ ;\ 1{,}3]$. Comment faire pour s'informer sur le nombre x de D tel que $f(x)=0$? Mais que veulent dire « s'informer » ou « trouver x » ? Comment faut-il répondre ?

Pour une équation polynomiale de degré 2, la réponse est claire : on cherche une « expression » de x faisant intervenir une racine carrée, tirée du modèle $\frac{-b+\sqrt{b^2-4ac}}{2a}$ pour une équation $ax^2+bx+c=0$ à discriminant positif. Pour d'autres équations (comme $x\ln(x)=1$) on ne trouve aucune « expression » bien que des solutions existent (théorème des valeurs intermédiaires). Faute de mieux, on cherche une **approximation** des solutions. Nous prendrons comme contexte une fonction f continue, croissante sur un intervalle $[a;b]$, telle que $f(a)<0<f(b)$.

Buts à atteindre

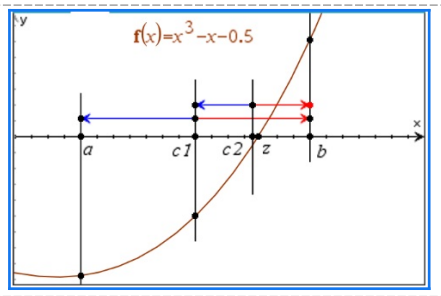

Fig.1 - dichotomie

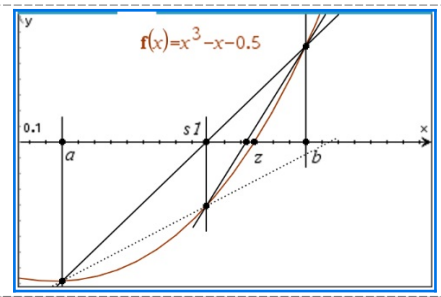

Fig. 2 - sécantes

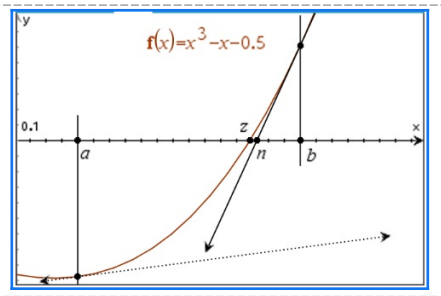

Fig. 3 - Newton[1]

Suggestion : trois algorithmes étant en vue, on répartit le travail sur plusieurs groupes qui compareront leurs résultats sur un exemple (ici : f telle que $f(x)=x^3-x-0{,}5$ sur $[0{,}5\ ;\ 1{,}3]$).

1. Écrire une fonction Python approchant par dichotomie une solution de $f(x)=0$ à une précision p, à partir de la connaissance de f, a, b.
2. Écrire une fonction Python faisant de même par la méthode des sécantes.
3. Écrire une fonction Python faisant de même par la méthode de Newton.

1 Figures réalisées avec le logiciel TI-Nspire CX

L. DIDIER & R. CABANE

Fiche méthode

Étapes de résolution

▶ Objectif 1 : dichotomie

L'algorithme est essentiellement une boucle `while`, où les bornes `a` et `b` sont réajustées de sorte que la fonction `f` prenne toujours une valeur négative à gauche et positive à droite.

```
ÉDITEUR : DICHO
LIGNE DU SCRIPT 0002
# f = fonction
# a,b = bornes, p = précision
def dicho(f,a,b,p):
    c=(a+b)/2
    while c-a>p:
        if f(c)>0:
            b=c
        else:
            a=c
        c=(a+b)/2
    return c

def f(x):
    return x**3-x-.5
```

```
>>> dicho(f,.5,1.3,1E-5)
1.191485595703125
>>> dicho(f,.5,1.3,1E-10)
1.191487883869559
```

Ici le paramètre `f` est une fonction (Python), à ne pas confondre avec `f(x)` qui est un nombre.

▶ Objectif 2 : sécantes.

L'équation de la droite passant par deux points distincts de coordonnées $(a\,;f(a))$ et $(b\,;f(b))$ est : $y=f(a)+\dfrac{f(b)-f(a)}{b-a}(x-a)$ (justification : remplacer x par a puis par b pour s'assurer que la formule est la bonne) ; cette droite coupe l'axe (Ox) au point d'abscisse s telle que $0=f(a)+\dfrac{f(b)-f(a)}{b-a}(s-a)$ soit encore[2] $s=a-f(a)\dfrac{b-a}{f(b)-f(a)}$.

On recommence ensuite en prenant s à la place de a ou de b. On a donc essentiellement à calculer une suite récurrente du type $u_{n+1}=a-f(a)\dfrac{u_n-a}{f(u_n)-f(a)}$, avec $u_0=b$, ou bien $u_{n+1}=b-f(b)\dfrac{u_n-b}{f(u_n)-f(b)}$ avec $u_0=a$. Le choix de la formule qui convient dépend de la fonction f, et peut se faire en testant si on reste bien dans l'intervalle $[a\,;b]$. Dans l'exemple étudié, c'est la seconde formule qui doit être choisie (voir la figure 2 page précédente, la droite en pointillés ne convient pas).

Nous nous limitons ici à la seconde formule, qui va donner une suite d'approximations croissante. Pour assurer la précision demandée, il faut tester le signe de la fonction « un peu plus loin » (en $u+2p$).

```
ÉDITEUR : INTERPOL
LIGNE DU SCRIPT 0010
def interpol(f,a,b,p):
    u=a
    while True:
        u=b-f(b)*(u-b)/(f(u)-f(b))
        if f(u)<0 and f(u+2*p)>0:
            return u+p

def f(x):
    return x**3-x-0.5
```

```
>>> from INTERPOL import *
>>> interpol(f,.5,1.7,1E-5)
1.1914795111691149
>>> interpol(f,.5,1.7,1E-10)
1.191487883966847
```

À noter :

1. La fonction utilisée s'annule sur l'intervalle $[0{,}5\,;\,1{,}3]$, mais aucune « formule » ne semble disponible pour la racine de l'équation $f(x)=0$ sur cet intervalle.
2. On utilise ici une « boucle infinie » `while True`, qui s'achève en fait au moment où la précision est atteinte (test du changement de signe de `f` entre `u` et `u+2p`), par une instruction `return`.
3. Par ailleurs, il faut reconnaître que le code proposé est bref mais non optimal ; ainsi, la valeur `f(b)`

`True` est une constante toujours « vraie », exactement comme le serait un test `1==1`.

2 On pourrait aussi présenter s sous la forme $s=\dfrac{af(b)-bf(a)}{f(b)-f(a)}$; cela ne serait pas judicieux car si les valeurs $f(a)$ et $f(b)$ étaient proches, les deux soustractions se feraient avec une mauvaise précision.

est recalculée de nombreuses fois, ce qui peut être long si le calcul de la fonction `f` est complexe. Il serait préférable de « précalculer » `f(b)` avant la boucle `while` et de ranger cette valeur dans une variable. Même remarque pour la valeur `f(u)`, calculée deux fois.

▶ Objectif 3 : méthode de Newton

On suppose ici que f est dérivable sur $[a;b]$ et que f' ne s'annule pas sur $[a;b]$. La démarche est très similaire à celle de l'objectif 2. L'équation de la droite passant par le point de coordonnées $(b\,;f(b))$ et tangente à la courbe est $y=f(b)+f'(b)(x-b)$; cette droite coupe l'axe au point d'abscisse n telle que $0=f(b)+f'(b)(n-b)$ soit $n=b-\dfrac{f(b)}{f'(b)}$. On recommence ensuite en prenant n à la place de b, amenant une suite récurrente du type $u_{n+1}=u_n-\dfrac{f(u_n)}{f'(u_n)}$ avec $u_0=b$. On pourrait aussi imaginer de se baser sur le point a à la place de b, mais cela ne convient pas ici (voir la droite en pointillés sur la figure 3 page précédente).

Il y a cependant une difficulté nouvelle : comment accéder aux valeurs de la dérivée ? Deux solutions se présentent : ou bien définir la dérivée comme une nouvelle fonction, et la passer en paramètre de la fonction Newton, ou bien approcher les valeurs de la dérivée. Ces deux démarches sont présentées ci-dessous.

ÉDITEUR : NEWTON
LIGNE DU SCRIPT 0007

```
def newton1(f,df,a,b,p):
    u=b
    while True:
        u=u-f(u)/df(u)
        if f(u)>0 and f(u-2*p)<0:
            return u-p

def f(x):
    return x**3-x-0.5
def df(x):
    return 3*x*x-1
```

Avec la dérivée

ÉDITEUR : NEWTON
LIGNE DU SCRIPT 0023

```
def newton2(f,a,b,p):
    def df(t):
        return (f(t)-f(t-p))/p
    u=b
    while True:
        u=u-f(u)/df(u)
        if f(u)>0 and f(u-2*p)<0:
            return u-p
```

Sans la dérivée

PYTHON SHELL

```
>>> newton1(f,df,.5,1.7,1E-5)
1.191478122459366
>>> newton1(f,df,.5,1.7,1E-10)
1.191487883853181
>>> newton2(f,.5,1.7,1E-5)
1.191478117048251
>>> newton2(f,.5,1.7,1E-10)
1.191487883853275
>>>
```

Exécution

À l'usage : avec quelques tests, on s'apercevra que la méthode de Newton (sous n'importe laquelle des variantes ici présentées) converge beaucoup plus rapidement que les deux autres méthodes, la dichotomie étant la plus lente des trois. Cette question est abordée dans la section suivante.

Ici, la dérivée est *approchée* (ce qui ne ralentit pas la convergence) et traitée comme une fonction `df()` *interne* à la fonction `newton2()`. Le langage Python permet cela !

Pour aller plus loin

Approfondissements et prolongements possibles

```
ÉDITEUR : DICHO
LIGNE DU SCRIPT 0015
def dicho(f,a,b,eps):
  if a>b:(a,b)=(b,a)
  if f(a)>f(b):i=-1
  else:i=1
  c=(a+b)/2
  while c-a>eps:
    if i*f(c)>0:b=c
    else:a=c
    c=(a+b)/2
  return c
```

Améliorations : il faudrait adapter les algorithmes de manière à envisager les différents cas (fonction décroissante, etc.). Le cas d'une fonction décroissante peut se traiter en changeant f en $-f$ (le code montré ci-contre traite le cas de la dichotomie à titre d'exemple).

Pour la méthode des sécantes comme celle des tangentes, il faudrait tester celui des deux algorithmes (« à gauche » ou « à droite ») qui convient (c'est en fait une question de convexité de la fonction f).

Confrontations : il serait instructif de comparer le nombre d'étapes requises par chacun des algorithmes avant d'atteindre la précision demandée. La comparaison se fera en insérant un compteur `k` dans les boucles `while` et en terminant avec une instruction du genre `return c,k`. Il restera à faire des tests (dans la console Python) pour découvrir quel est le nombre d'étapes réellement consommées par chaque méthode.

Travail de recherche (en groupe si possible) : les trois algorithmes pourraient être « mixés » afin de produire plus rapidement un résultat précis. C'est ainsi qu'on remarque la propension des méthodes des sécantes et de Newton à donner des approximations de sens contraire (l'une donnant des valeurs par défaut et l'autre par excès) : si on les combine, on peut plus aisément obtenir des encadrements, et ainsi garantir une précision donnée.

```
ÉDITEUR : NEWTON
LIGNE DU SCRIPT 0001
def newton1(f,df,a,b,p):
  u=b
  while True:
    u=u-f(u)/df(u)
    if f(u)>0 and f(u-2*p)<0:
      return u-p

def f(x):
  return x**3-x-0.5
def df(x):
  return 3*x*x-1
Fns… | a A # | Outils | Exéc | Script
```