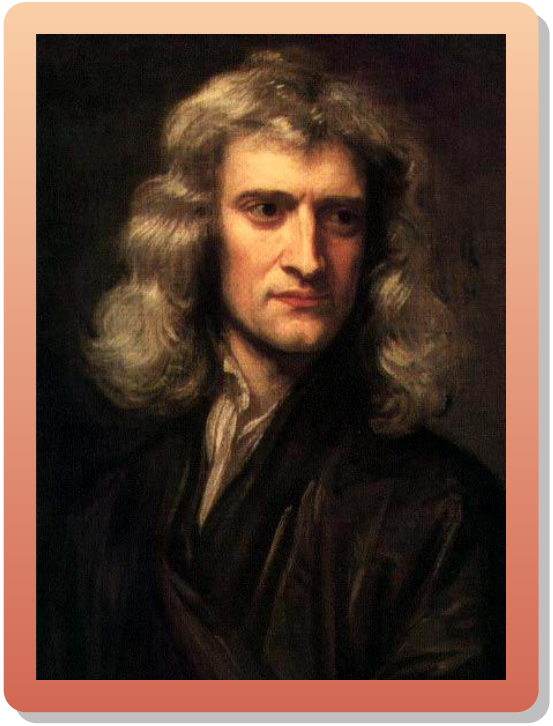
IMT Mines Albi-Carmaux
École Mines-Télécom

Algorithmique et programmation : Newton-Raphson

Paul GABORIT

2018

- Au XVII^{e} siècle, Isaac Newton et Joseph Raphson ont découvert quasiment au même moment une méthode numérique permettant de trouver un ou plusieurs zéros d'une fonction dérivable.
- L'objectif de cette séance est d'écrire un algorithme reposant sur cette méthode afin de produire un programme en langage C permettant de calculer des zéros d'une fonction.



Isaac Newton
04 janvier 1643 - 31 mars 1727

Soit une fonction $f(x)$ et sa dérivée $f'(x)$:

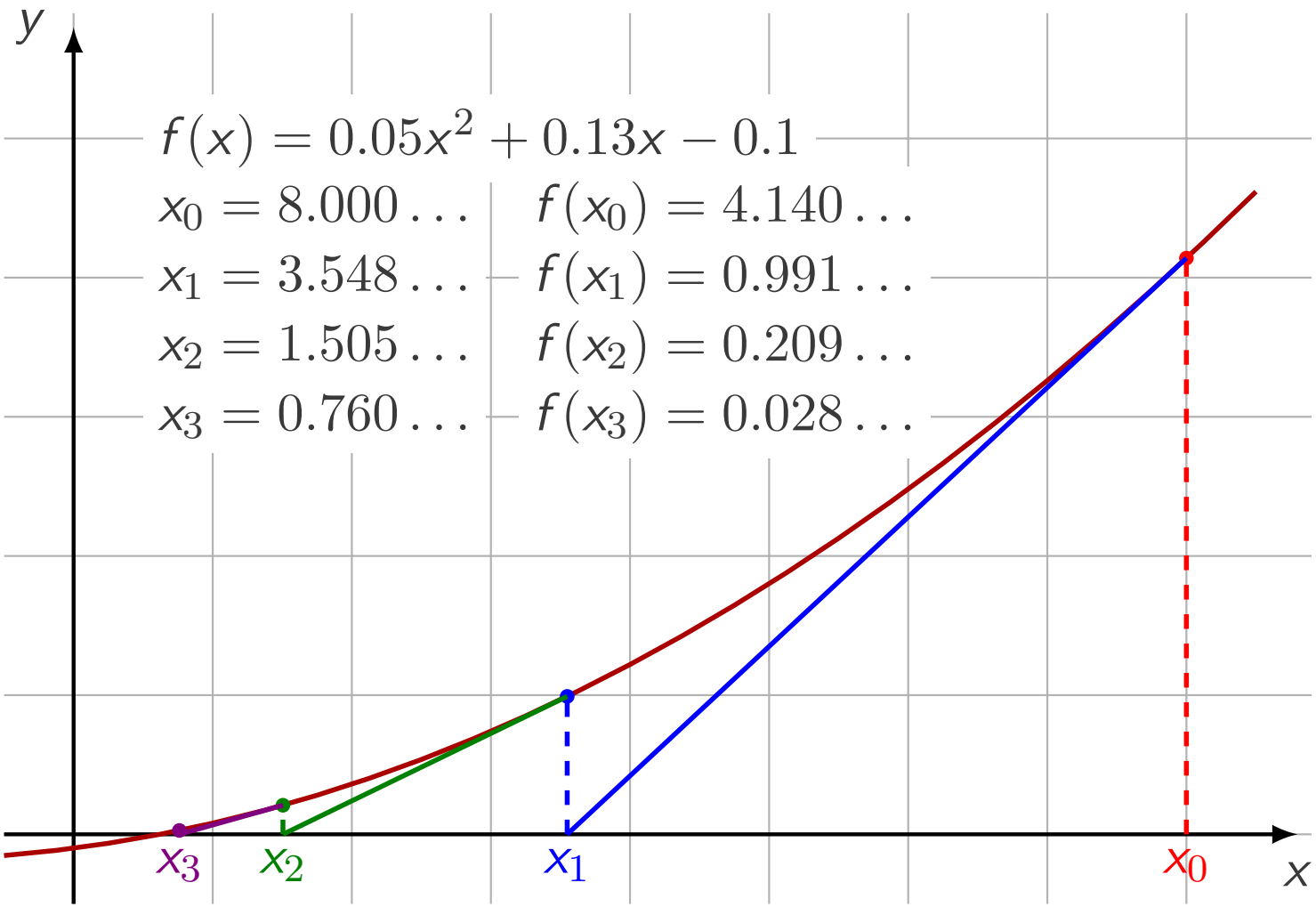
1. On choisit une **valeur de départ** x_0.
2. On utilise une approximation de la fonction au premier ordre :

$$f(x) \approxeq f(x_0) + f'(x_0)(x - x_0)$$

3. On calcule le zéro de cette approximation :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

4. Ce nouveau point x_1 nous sert de nouvelle **valeur de départ**.


y
$f(x) = 0.05x^2 + 0.13x - 0.1$
$x_0 = 8.000\ldots$ $f(x_0) = 4.140\ldots$
$x_1 = 3.548\ldots$ $f(x_1) = 0.991\ldots$
$x_2 = 1.505\ldots$ $f(x_2) = 0.209\ldots$
$x_3 = 0.760\ldots$ $f(x_3) = 0.028\ldots$
x_3
x_2
x_1
x_0
x

- Cette méthode est donc une méthode itérative reposant sur une suite.

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

- Sauf exception (cf. cours de calcul numérique), elle converge de manière quadratique vers un zéro de la fonction choisie.
- Pour décider qu'on a convergé, on utilise le critère suivant :
  1. $|f(x_n)| < \varepsilon_1$ *(on est* assez *proche du zéro)*
- Parfois la méthode ne converge pas. On décide de s'arrêter selon les critères suivants :
  1. $n \geq itermax$ *(nombre maximale d'itérations atteint)*
  2. $|f'(x_n)| < \varepsilon_2$ *(tangente* presque *horizontale)*
- Les valeurs de ε_1, de ε_2 et de *itermax* sont arbitrairement choisies (ex : 10^{-6}, 10^{-6}, 40).

- Écrire l'algorithme permettant d'appliquer la méthode de Newton-Raphson à une fonction f.
- On supposera que le calcul de $f(x)$ et de $f'(x)$ sont des opérations déjà disponibles.
- On supposera que x_0, ε_1, ε_2 et *itermax* (le nombre maximum d'itérations) sont des paramètres fournis.
- Pour chacun des points atteints, l'algorithme affichera les coordonnées ainsi que le numéro d'itération.
- Il se terminera en affichant :
  - l'éventuelle solution approchée atteinte,
  - un message indiquant pourquoi il s'arrête.

- Écrire un programme en C implémentant l'algorithme de l'exercice 1.
- Les deux fonctions `F` et `FPrime` seront codées sous la forme de fonctions du langage C.
- Le programme demandera à l'utilisateur les valeurs de x_0, ε_1, ε_2 et de *itermax*.
- Il affichera tout ce que l'algorithme affiche.
- Les valeurs réelles seront affichées avec 9 chiffres significatifs.

Pour écrire une fonction en langage C, il faut donner dans l'ordre :

- Le *type* de la valeur qu'elle retourne ;
- Son *nom* ;
- La liste entre parenthèses des paramètres. Les différents paramètres sont séparés par des virgules. Chaque paramètre est donné sous la forme `type nom` ;
- La suite d'instructions composant le code de la fonction dans un bloc entourés d'accolades. La dernière instruction est obligatoirement un appel à `return` avec comme paramètre la valeur que l'on souhaite retournée comme résultat.

Exemple de la fonction `puissance` dont le résultat est de type `double`, qui prend comme paramètre un `double` nommé `x` et un `int` nommé `exposant` et qui calcule une puissance entière positive.

```
double puissance(double x, int exposant) {
  double resultat = 1;
  int i;
  for (i = 1; i <= exposant; i++) {
    resultat = resultat * x;
  }
  return(resultat);
}
```

Exemple de code faisant appel à cette fonction :

```
double a = 5.6;
double b;
b = puissance(a+0.4, 3); /* b vaut 6 élevé à la puissance 3 */
```

La fonction `demande_double` permet de demander une valeur réelle à l'utilisateur en lui affichant le nom de la valeur attendue :

```
double demande_double(char * nom) {
  double reponse;
  printf("valeur (un réel) de %s : ", nom);
  scanf("%lf", &reponse);
  return reponse;
}
```

Le type du paramètre est `char *` (qui est un pointeur vers un caractère). En fait, c'est un pointeur vers une chaîne de caractères qu'il faut fournir (une chaîne de caractères est un tableau de caractères dont le dernier est le caractère `NUL`).

Exemple d'utilisation :

```
double a;
a = demande_double("a");
```

La fonction `demande_int` permet de demander une valeur entière à l'utilisateur en lui affichant le nom de la valeur attendue :

```
int demande_int(char * nom) {
  int reponse;
  printf("valeur (un entier) de %s : ", nom);
  scanf("%d", &reponse);
  return reponse;
}
```

Exemple d'utilisation :

```
int itermax;
itermax = demande_int("nombre d'iterations maximum");
```

(Le nom utilisé dans cet exemple est-il vraiment pertinent ?)

Pour afficher des nombres avec une certaine précision

Pour afficher un nombre réel (un `double`) avec une précision donnée, dans le *format* d'un `printf`, on peut utiliser la notation `%x.yg` où `x` est un nombre qui donne la largeur minimum du nombre affiché (en caractères) et où `y` est un nombre qui donne le nombre maximum de chiffres significatifs à afficher.

Note : le séparateur décimal ainsi que les éventuels signes font parties des caractères à compter.

Exemple :

```
double a = sqrt(123e-12);
printf("%15.9g\n", a);
```

Affiche :
`1.10905365e-05`
(15 caractères dont 9 chiffres significatifs)

Comme premier test, utiliser le code de la fonction (et de sa dérivée) du graphique donné au début de ce document.

```
double F(double x) {
  double y;
  y = 0.05*x*x+0.13*x-0.1;
  return y;
}

double FPrime(double x) {
  double y;
  y = 2*0.05*x+0.13;
  return y;
}
```

Vérifier que les valeurs obtenus sont les mêmes que celles du graphique (en prenant $x_0 = 8$).

Exercice 2 : tests du programme

- Tester le programme avec d'autres fonctions (avec leur dérivée) en modifiant les instructions des deux fonctions `F` et `FPrime`.
- À chaque fois, essayer plusieurs valeurs de x_0.

Exemples :

1. $f(x) = \cos(x) - x^3$ (avec $x_0 = \frac{1}{2}$)
2. $f(x) = x^3 - 2x + 2$ (avec $x_0 = 0$)
3. $f(x) = 1 - x^2$ (avec $x_0 = 0$)
4. ...