Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 6

Code UE : S14IN1I1

Année 2019-20

Algorithmes numériques

Exercice 1 (Calcul de la puissance d'un entier et applications au codage RSA) L'élévation d'un nombre à une puissance entière positive est une opération de base cruciale lorsqu'on veut calculer avec des valeurs numériques. C'est même l'étape de base dans la méthode cryptographique RSA. Étant donné un nombre x (cela peut-être un entier ou un réel, qu'on représente en machine à l'aide d'un flottant) et un entier naturel n, on souhaite donc calculer le nombre $x^n = \underbrace{x \times x \times \cdots \times x}_{n \text{ fois}}$.

On a vu en cours une première façon de calculer x^n à l'aide de l'algorithme suivant :

```
fonction puissance(x, n):
    résultat := x
    Pour i de 2 à n faire
        résultat := résultat × x
    FinPour
    retourner(résultat)
```

1. Exécuter cet algorithme lors du calcul de 2^{10}, en précisant la valeur des variables lors de chaque étape de la boucle `Pour`. Combien de multiplications a-t-on effectué pour calculer 2^{10} ?
2. Dans le cas général, combien de multiplications effectue cet algorithme pour calculer x^n, en fonction de x et n ?

   On a vu en cours une façon plus efficace de calculer x^n à l'aide de l'algorithme suivant :

```
fonction puissance(x, n):
    a := 1
    b := x
    m := n
    Tant que (m > 0) faire
        Si (m ≡ 0 mod 2) alors
            m := m/2
        Sinon
            m := (m − 1)/2
            a := a × b
        FinSi
        b := b × b
    FinTantQue
    retourner(a)
```

3. Exécuter cet algorithme lors du calcul de 2^{10}, en précisant la valeur des variables lors de chaque étape de la boucle `Tant que`.
4. Pourquoi l'algorithme termine toujours ? (*Indication : commencer par étudier pour quelle raison cet algorithme pourrait ne pas terminer*)
5. Pour prouver que l'algorithme est correct, c'est-à-dire qu'il renvoie bien x^n, une méthode consiste à vérifier qu'à tout moment de l'algorithme on a $\texttt{x}^\texttt{n} = \texttt{a} \times \texttt{b}^\texttt{m}$. Montrer que c'est vrai avant de rentrer dans la boucle `Tant que`, puis que si c'est vrai au début d'une itération de la boucle `Tant que`, alors c'est vrai à la fin de cette itération. Conclure alors à l'aide d'un raisonnement par récurrence.
6. Les opérations élémentaires coûteuses d'un algorithme d'exponentiation (c'est le nom qu'on donne à l'*élévation à la puissance*) sont les multiplications. Combien de multiplications sont effectuées par l'algorithme lors du calcul de 2^{10}. Plus généralement, pouvez-vous donner un ordre de grandeur du nombre de multiplications effectuées pour calculer x^n par l'algorithme, en fonction de n ?

7. Le système de cryptographie RSA (bien plus sûr que les systèmes de cryptographie de César, spartiate et de Vigenère vus dans le TD 3) consiste à calculer des *puissances modulaires*, c'est-à-dire $x^n \mod k$, le reste de x^n dans la division euclidienne par k. Sachant que

$$\text{si } a \equiv b \mod k \qquad \text{alors } a^n \equiv b^n \mod k$$

on a intérêt à calculer les puissances en prenant les restes dans la division euclidienne par k à chaque étape. Modifier alors la fonction `puissance`, en ajoutant un troisième argument k, afin qu'elle calcule $x^n \mod k$: on s'autorise à utiliser comme opération élémentaire supplémentaire le calcul de $y \mod k$, le reste dans la division euclidienne de y par k.

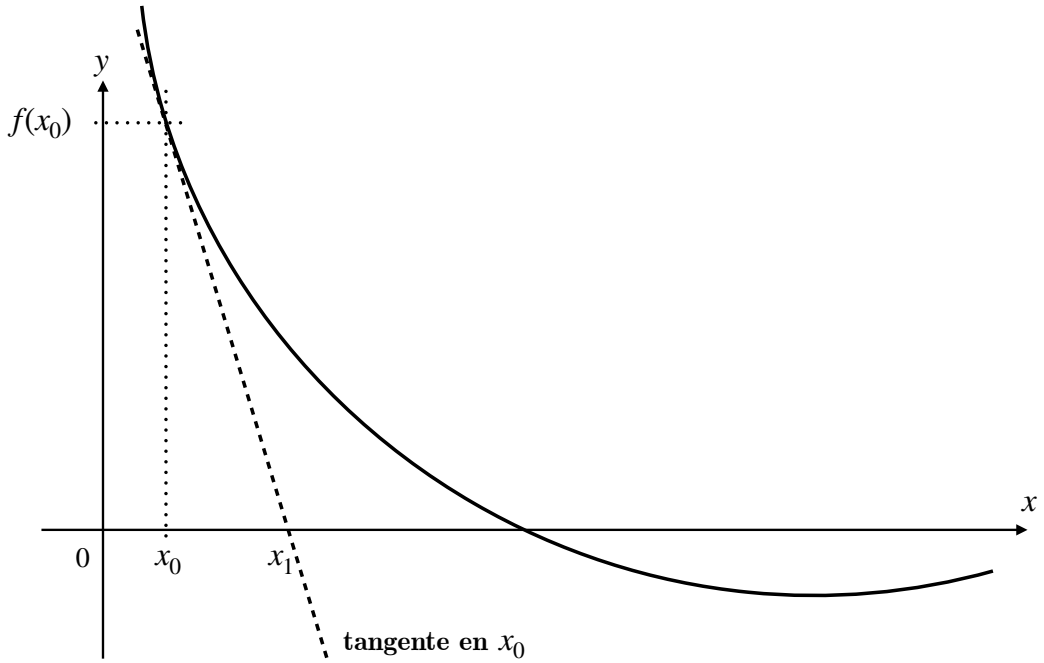
Exercice 2 (Nombres de Mersenne) Un nombre premier est un nombre entier supérieur ou égal à 2 qui n'est divisible que par 1 et par lui-même.

1. Écrire une fonction `est_premier` prenant en argument un entier n est renvoie un booléen qui est `vrai` si n est premier, `faux` sinon.
2. Écrire une fonction qui prend un entier i en argument et retourne le $i$ème nombre premier, en s'aidant de la fonction précédente (sachant que 2 est le premier nombre premier, que 3 est le second, etc.) : lorsque $i > 1$, vous utiliserez une boucle qui énumère tous les entiers impairs (en effet, inutile d'essayer les entiers pairs, puisque le seul premier pair est 2).
3. Jusqu'en 1536, on croyait que les nombres de Mersenne, c'est-à-dire les nombres de la forme $2^p - 1$, avec p premier, étaient premiers. À partir des fonctions écrites en réponse aux deux questions ci-dessus, écrire un algorithme (celui-ci ne sera pas forcément dans une fonction...) qui montre l'invalidité de la conjecture d'avant 1536. L'algorithme devra *écrire* :
   — à partir de quel nombre premier p_m la conjecture est fausse ;
   — à quel rang se trouve p_m dans la liste des nombres premiers.

Exercice 3 (Algorithme de Héron pour l'approximation de la racine carrée)

La méthode de Newton permet de calculer une approximation d'un zéro d'une fonction réelle dérivable f, c'est-à-dire une valeur proche d'un réel z tel que $f(z) = 0$. On rappelle que la tangente à la courbe de f en le point x_0 (dans son ensemble de définition) est la droite d'équation $y = f(x_0) + f'(x_0) \times (x - x_0)$. La méthode de Newton se base sur l'idée qu'on peut approcher la courbe d'une fonction par sa tangente : si x est suffisamment proche de x_0 alors $f(x)$ est proche de $f(x_0) + f'(x_0)(x - x_0)$.

En particulier, si on calcule l'intersection de la tangente à la courbe en x_0 avec l'axe des abscisses, on espère obtenir une nouvelle abscisse x_1 plus proche du zéro comme le montre la figure ci-dessous :



On résout donc l'équation $0 = f(x_0) + f'(x_0)(x - x_0)$ afin de trouver, si la dérivée $f'(x_0)$ est non nulle, $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. Plus généralement, on construit donc la suite $(x_n)_{n\in\mathbb{N}}$ par récurrence, à partir du point x_0 :

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

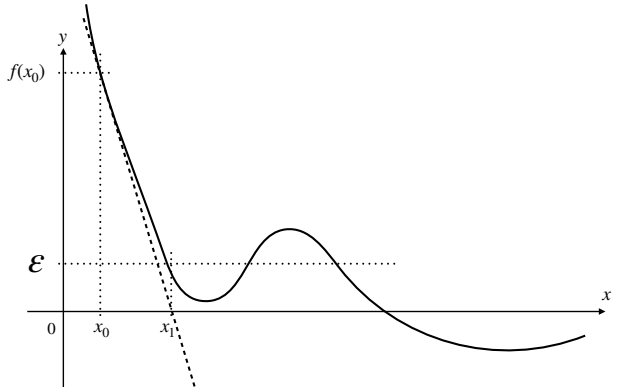
tant que la dérivée $f'(x_n)$ est non nulle (sinon la méthode échoue).

1. Exécuter les deux itérations suivantes de l'algorithme sur le dessin du dessus, afin de trouver x_2 et x_3.
2. On a vu en cours comment décrire l'algorithme de Newton :

```
fonction approximation_zero(f, f', x0, ε):
    a := x0
    Tant que |f(a)| > ε faire
        a := a − f(a)/f'(a)
    FinTantQue
    retourner(a)
```

   Il prend en entrée une description des fonctions f et f', l'abscisse initiale x_0 ainsi qu'un paramètre d'erreur [1] ε (par exemple $\varepsilon = 0,001$) permettant d'écrire la condition d'arrêt de la boucle `Tant que` : ici, on arrête les itérations dès qu'on trouve une abscisse a telle que $|f(a)| \leq \varepsilon$. Malheureusement, cet algorithme n'est pas correct en général : il se peut que l'algorithme termine et renvoie un résultat qui ne soit pas « proche » d'un zéro de la fonction f. C'est le cas dans l'exemple ci-dessous où l'algorithme s'arrête dès la fin de la première itération :

1. On utilise traditionnellement la lettre grecque ε pour décrire ce paramètre d'erreur, car c'est la lettre correspondant à la lettre E.



On propose une autre possibilité en remarquant que lorsqu'on s'approche du zéro de f, les abscisses x_n deviennent de plus en plus proche : précisément, la distance de x_n à x_{n+1} tend alors vers 0 lorsque n tend vers l'infini. Comment modifier l'algorithme pour que la condition d'arrêt porte ainsi sur cette distance entre deux abscisses successives ?

3. Pouvez-vous trouver un exemple montrant que ce nouvel algorithme est également incorrect ? On demande donc de dessiner le graphe d'une fonction f et de choisir x_0 et un paramètre d'erreur tels que le nouvel algorithme renvoie un résultat « loin » du zéro de la fonction f.
4. On cherche désormais à appliquer l'algorithme de Newton pour calculer la racine carrée d'un nombre r positif. Pour cela, on utilise la fonction f qui à tout réel x associe $x^2 - r$. Décrire explicitement la suite récurrente $(x_n)_{n\in\mathbf{N}}$ de la méthode de Newton appliquée à cette fonction. Cette estimation de la racine carrée $\sqrt{r}$ d'un nombre positif est appelée *algorithme de Héron.*
5. En déduire un algorithme (n'utilisant pas la fonction `approximation_zero`) qui donne une approximation de la racine carrée d'un nombre r donné en argument, avec une précision ε donnée en argument également : on souhaite renvoyer un résultat a vérifiant $|a - \sqrt{r}| \leq \varepsilon$, mais attention on ne peut pas utiliser la valeur de $\sqrt{r}$...