Résolution numérique de l'équation $f(x) = 0$

0 Pré-requis

0.1 Suites

Vous connaissiez les paires ? Les triplets ? Peut-être même les n-uplets ? Et bien une suite, ça n'est pas bien différents, c'est une sorte d'∞-uplet. En gros, c'est juste une suite d'éléments d'un ensemble (qui pour nous sera $\mathbb{R}$) indexée par $\mathbb{N}$: $(u_n)_{n\in\mathbb{N}} \equiv (u_0, u_1, u_2, \dots, u_n, u_{n+1}, \dots)$. Histoire de ne rien vous cacher, on note $E^{\mathbb{N}}$ les suites de E, et c'est bon, maintenant vous savez (presque) tout.
Là où ça devient intéressant, c'est lorsque l'on sait dire des choses sur u_n, par exemple en fonction de n ou de u_{n-1}. Par exemple, une suite géométrique de raison r est définie par $\forall n \geq 1, u_n = r * u_{n-1} = r^n * u_0$. Par exemple, 1,2,4,8,16,32... est la suite géométrique avec $u_0 = 1$ et $r = 2$. On a aussi souvent des suites données par $u_n = f(u_{n-1})$, ce sera le cas par la suite.

0.2 Dérivée

Encore plus simple. On supposera ne travailler qu'avec des fonctions honnètes et fréquentables (donc ne présentant aucun saut ni le moindre angle), qui admettront une tangente en tout point. Et bien si f est une telle fonction, on note $f'(x)$ la valeur de la pente de la tangente au point x, et on appelle dérivée de f la fonction $x \mapsto f'(x)$. Nous n'aurons besoin de rien en plus.

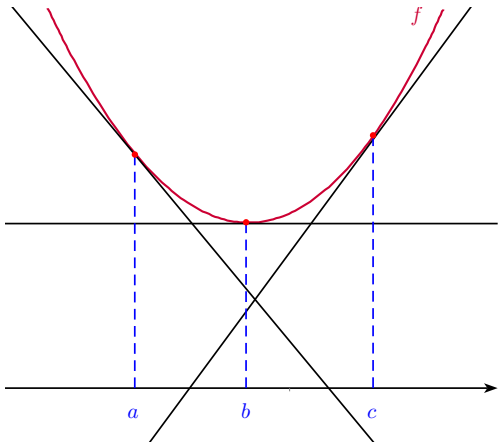

FIGURE 1 – Dérivées

Sur la figure précédente, on a par exemple $f'(a) < O$, $f'(b) = 0$ et $f'(c) > 0$.

1 Qu'es-aco ?

1.1 Les ordinateurs, de parfaits calculateurs ?

Depuis quelques années maintenant, les scientifiques (et pas qu'eux) sont désormais dotés de fabuleuses petites machines qui effectuent de gros calculs à leur place, beaucoup plus rapidement et sans se tromper. Ordinateurs, qu'elles s'appellent. Présentée comme ça, l'histoire est belle. Sauf que deux gros problèmes (au moins) viennent entraver cette idyllique situation, ayant la même cause originelle : les ordinateurs, sans grande surprise, ne savent faire que ce qu'on leur apprend.

Du coup, il n'y a par exemple pas trop de raison pour qu'une machine vous fournisse une valeur exacte de π, ni même de $\sqrt{2}$. Et pourtant, vous vous en servez allègrement sur vos calculettes sans vous poser de question. Cela soulève donc dans un premier temps un problème quant à la précision des calculs. Mais supposons qu'un ordinateur, étant dans notre imaginaire commun infiniment plus rapide que nous autres humains de bas étage,

puisse calculer tout ce qu'il veut avec une très très grande précision, suffisante en tout cas pour donner un résultat final convenable. On pourrait alors à nouveau se resservir gaiement de notre fonction racine sans trop se poser de question. Mais là encore, problème : comment une machine calcule-t-elle une racine ? Ou un inverse ? Et même une division ?

En réalité, toutes les opérations arithmétiques sont sujettes à d'éventuelles erreurs. Deux d'entre elles, la multiplication et l'addition, sont réalisées électroniquement [1]. Toutes les autres sont retrouvées à partir de ces deux opérations. Un objectif majeur est donc de trouver de bonnes [2] méthodes pour les autres opérations de bases.

1.2 Gérard Majax

Pour vous donner un petit peu l'intuition du pourquoi du comment des méthodes qui vont suivre, on va commencer par un petit tour de magie. Considérons la fonction $f : x \mapsto x(2 - a \cdot x)$, et regardons ce que devient la suite $a_{n+1} = f(a_n)$ en partant d'un $a_0 \in [0; \frac{2}{a}]$, domaine où la fonction est positive. C'est une suite récurrente d'ordre 1, pour la représenter, on utilise la célébrissime méthode de l'escalier : on part de a_0, on regarde son image par f, on la ramène sur l'axe des abscisses grâce à la droite $y = x$, et on recommence.

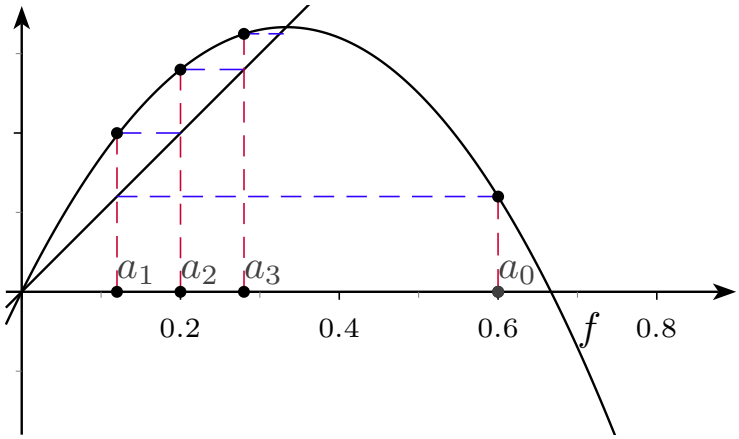

FIGURE 2 – $a_{n+1} = a_n(2 - 3a_n)$

On constate (et on peut très facilement montrer [3]) que notre suite tend vers le point d'intersection de la courbe de f et de la droite $y = x$, soit

$$x = x(2 - 3x) \iff 2 - 3x = 1 \iff x = \frac{1}{3}$$

Jusque là, rien de très extraordinaire. Sauf que notre fonction f n'est définie qu'à partir d'additions et de multiplications [4] ! Et on a pourtant $a_n \underset{n\to+\infty}{\to} \frac{1}{3}$.

Par la suite, on va donc essayez de systématiser cette méthode, en se ramenant au calcul du zéro d'une fonction (ici ça aurait été $f(x) - x$ construite uniquement à partir d'opérations connues, donc la multiplication, l'addition, et grâce à ce qu'on vient de voir, la division. Trouver la bonne fonction n'est généralement pas très dur, et ce ne sera pas le problème du jour. On considérera par la suite, pour les exemples, le calcul de $\sqrt{n}$, à l'aide de la fonction $f : x \mapsto x^2 - n$.

1.3 La situation de base

On se placera donc avec une fonction f que l'on sait calculer, sur un intervalle où elle s'annule (pour ça, jouer avec vos théorèmes des valeurs intermédiaires préférés), et l'on considérera à chaque fois 2 points de part et d'autre du zéro recherché. Le dessin (Figure 3) sera toujours le même, vous pouvez vous convaincre sur d'autres exemples que cela marche tout aussi bien.

1. *Demandez à vos amis SIstes, ils sauront peut-être vous dire comment*
2. *Ce qui peut signifier rapide, précis, facile à mettre en oeuvre*
3. *En regardant la quantité* $a_n - \frac{1}{3}$.
4. *Il y a certes une soustraction au milieu, mais c'est aussi l'addition d'un nombre négatif...*

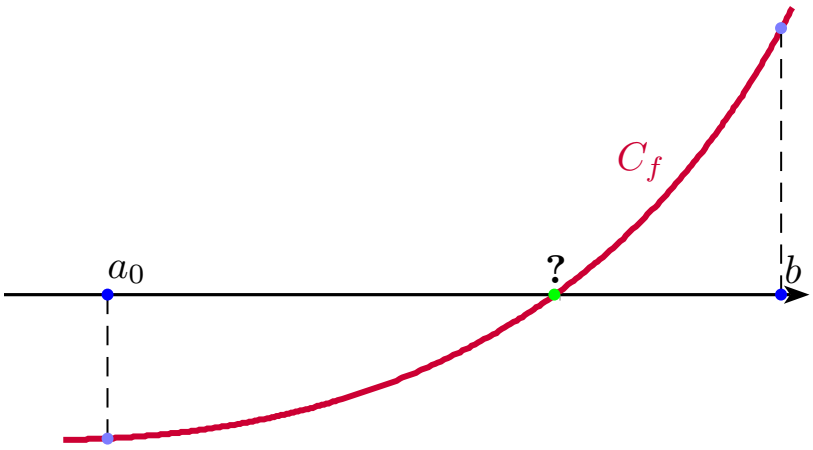

FIGURE 3 – La situation de base

2 Dichotomie

"*Choisis un nombre entre 1 et 1000 et dis-moi plus petit ou plus grand...*". Vous avez tous joué à ça étant petit, et mieux, vous saviez-tous comment gagner en moins de 10 coups : on commence par 500, puis en fonction 250 ou 750, etc. Et bien le principe de la dichotomie est exactement le même : on va réduire à chaque étape l'intervalle $[a, b]$ de moitié. La méthode est triviale, on regarde en $\frac{a+b}{2}$ le signe de f, et on adapte l'intervalle en fonction.

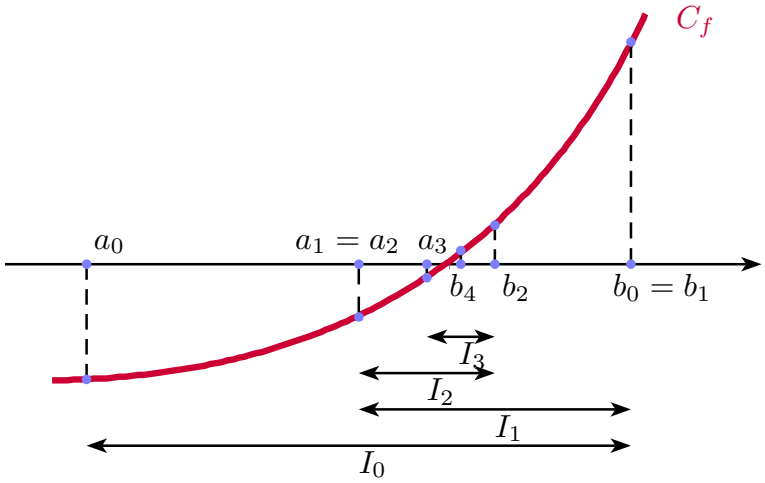

FIGURE 4 – dichotomie

Formellement, on construit donc deux suites (a_n) et (b_n) telles que pour tout n, f s'annule toujours dans $[a_n, b_n]$, $a_n < b_n$ et de plus $b_n - a_n \underset{n\to+\infty}{\to} 0$. Les suites sont définies comme suit, en partant de a_0 et b_0 tels que $f(a_0) < 0 < f(b_0)$ (sinon on inverse) :

$$\begin{cases} a_{n+1} = \begin{cases} \frac{a_n+b_n}{2} & \text{si } f(\frac{a_n+b_n}{2}) < 0 \\ a_n & \text{sinon} \end{cases} \\ b_{n+1} = \begin{cases} \frac{a_n+b_n}{2} & \text{si} f(\frac{a_n+b_n}{2}) > 0 \\ b_n & \text{sinon} \end{cases} \end{cases}$$

On va essayer, pour cette méthode comme pour les suivantes, de la programmer dans vos Texas Instrument favorites. La démarche est toujours la même :

- on fait des maths, pour définir proprement la/les suite(s)
- on écrit un pseudo-code, qui décrit comment l'algorithme qui calcule cela par petite étapes élémentaires
- on transcrit le pseudo-code dans langage de programmation voulu (ici du TI-Basic)

Là, on a déjà effectué la première étape, et on va faire les suivantes dans le cas particuliers où $f(x) = x^2 - n$. Pour cela, on va avoir besoin de variables pour a,b,n, et p le nombre d'étapes de calcul à effectuer. Le reste est assez clair.

Pseudo-code

Entrées : n,[a ;b],p
Faire p fois :
  Si $\frac{(a+b)^2}{4} - n > 0$
  alors
    $a := \frac{(a+b)}{2}$
  sinon
    $b := \frac{(a+b)}{2}$
  Fin si
Fin faire
Renvoyer [a ;b]

Code TI-Basic

:Prompt N,A,B,P
:For(Y,1,P)
:If $(A+B)^2/4 - N > 0$
:Then
:(A+B)/2 → A
:Else
:(A+B)/2 → B
:End
:End
:Disp "A=",A,"B=",B

FIGURE 5 – Code de la dichotomie

Il ne nous reste plus qu'à analyser rapidement l'efficacité de cette méthode. On a très facilement $|b_n - a_n| = \frac{|b_{n-1}-a_{n-1}|}{2} = \frac{|b_0-a_0|}{2^n}$. Il faut donc, pour gagner une décimale, effectuer 3 à 4 itérations, et ce quelque soit l'erreur courante, ce n'est donc pas très rapide. En revanche, cette méthode est très peu coûteuse en calcul : une seule évaluation de f, et une toute petite division par 2 suffisent.

3 Méthode des cordes

On peut améliorer la rapidité en se basant sur une méthode relativement intuitive. En effet, si l'on regarde attentivement (si si, faites-le) ce que l'on a à disposition au début (Figure 3), et que l'on ne s'autorise que des opérations constructibles à la règle et au compas, une idée s'impose assez naturellement (si elle ne vient pas, sortez vraiment une règle et essayez de jouer avec) : tracer la droite reliant le point de C_f d'abscisse a à celui d'abscisse b.

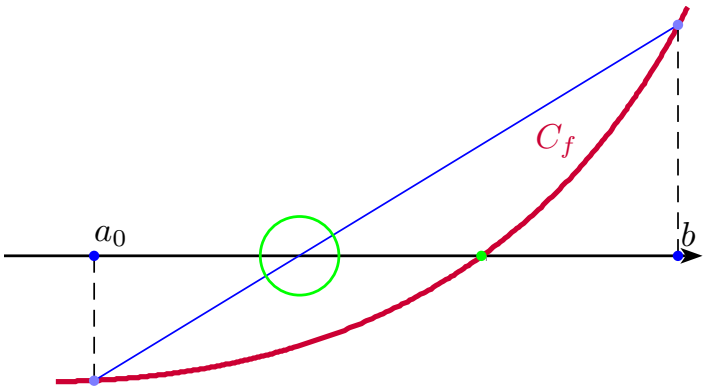

FIGURE 6 – La première corde

On constate alors que l'intersection avec l'axe des abscisses n'est peut-être pas inintéressante. Du coup, puisque c'est un peu le principe, on va recommencer. La seule chose à vérifier, c'est que le point que l'on obtient est bien au-dessus de la courbe. Si l'on se place sur un intervalle où la fonction est convexe (cf le prochain épisode), ça marche tout seul, sinon il faut juste éventuellement interchanger le rôle de a et b.

Faisons un petit peu de maths : comment obtenir a_{n+1} depuis a_n et b ? On commence par calculer l'équation de la corde :

$$y = \frac{f(b) - f(a_n)}{b - a_n}(x - a_n) + f(a_n)$$

Celle-ci coupe l'axe des abscisses pour $y = 0$, soit :

$$0 = \frac{f(b) - f(a_n)}{b - a_n}(x - a_n) + f(a_n) \iff x = \frac{b - a_n}{f(b) - f(a_n)}f(a_n) + a_n$$

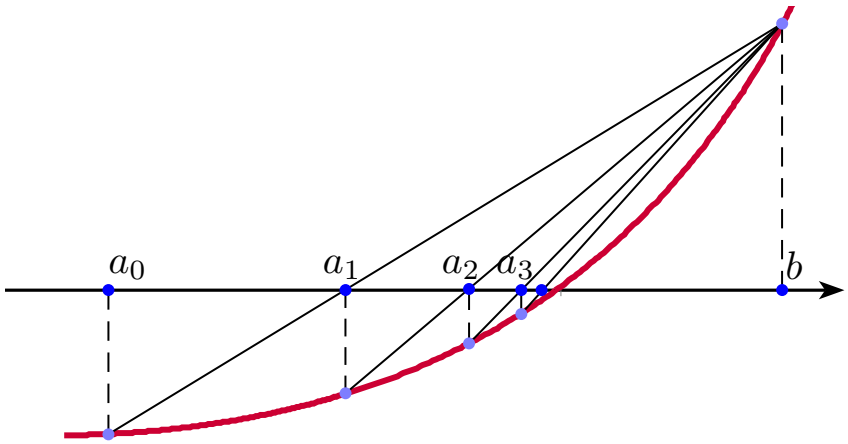

FIGURE 7 – Méthode des cordes

En posant $\Phi : x \mapsto \frac{b-x}{f(b)-f(x)} f(x) + x$, on obtient donc une belle suite récurrente d'ordre 1 : $\begin{cases} a0 = a \\ a_{n+1} = \Phi(a_n) \end{cases}$
Il ne reste plus qu'à programmer la chose, toujours pour $f(x) = x^2 - n$ (il est très facile encore une fois d'adapter à d'autres fonctions). Pour cela, on crée au préalable un programme PHI qui, depuis les variables a, b et n renvoie $\Phi(a)$. On appelera ensuite ce programme dans notre programme de la méthode des cordes [5]. On s'appuie pour cela sur le fait que les variables sont globales dans vos calculettes, c'est-à-dire qu'elles sont communes à tous vos programmes (à la différence de variable locale en programmation qui sont créées au lancement du programme et détruite ensuite). Fin de la digression informaticienne.

Programme PHI

Variables en mémoire : N, A, B

:$(B - X)/(B^2 - X^2) * (X^2 - N) + X \to \theta$
:Return

Il ne nous reste plus qu'à transcrire le pseudo-code en un code TI-Basic, ce qui ne présent pas de difficulté majeure.

Pseudo-code

Entrées : n,[a ;b],p
Faire p fois :
    $\theta := a - \frac{b-a}{(b^2-a^2)(a^2-n)}$
    Si $\frac{(a+b)^2}{4} - n > 0$
    alors
        $a := \frac{(a+b)}{2}$
    sinon
        $b := \frac{(a+b)}{2}$
    Fin si
Fin faire
Renvoyer [a ;b]

Code TI-Basic

:Prompt N,A,B,P
:For(Y,1,P)
:pgrmPHI
:If $\theta - N > 0$
:Then
:$\theta \to$ A
:Else
:$\theta \to$ B
:End
:End
:Disp "A=",A,"B=",B

FIGURE 8 – Code de la méthode des Cordes

Cette méthode est plus rapide que la dichotomie. Cela se voit sur le dessin, et surtout on peut le prouver, mais vous n'avez pas encore les outils pour (si un jour on vous parle des développements limités, vous pourrez

5. *Ici, ça ne sert pas à grand chose de subdiviser me direz-vous, néanmoins, c'est une bonne habitude à avoir. Le résultat est un programme plus générique : pour changer la fonction f, cela se fera dans PHI et non dans le programme principal par exemple*

essayer de faire les calculs). On le sent un petit peu intuitivement, plus la fonction sera pentue, et moins sa dérivée variera (plus notre fonction ressemblera à un droite en quelque sorte), plus l'approximation sera bonne. Cela dépend donc du $\underset{x\in[a,b]}{int}|f'(x)|$ et du $\underset{x\in[a,b]}{sup}|f''(x)|$. On aura une convergence d'ordre 1.6 environ, c'est à dire que si l'on a une erreur ϵ à l'étape n, elle sera de l'ordre de $k\epsilon^{1.6}$. C'est donc plus rapide que la dichotomie (on avait une convergence d'ordre 1), mais cela nécessite plus de calcul (notamment pour pour Φ).

4 Méthode de Newton

L'idée de la méthode des cordes était, en y réfléchissant bien, d'assimiler la droite à sa corde. Cependant, il existe une autre droite particulière qui nous permet d'approcher une courbe, c'est la dérivée. À ce point, vous avez donc compris la suite : on va prendre un point, sa tangente, l'intersection avec l'axe des abscisses et itérer.

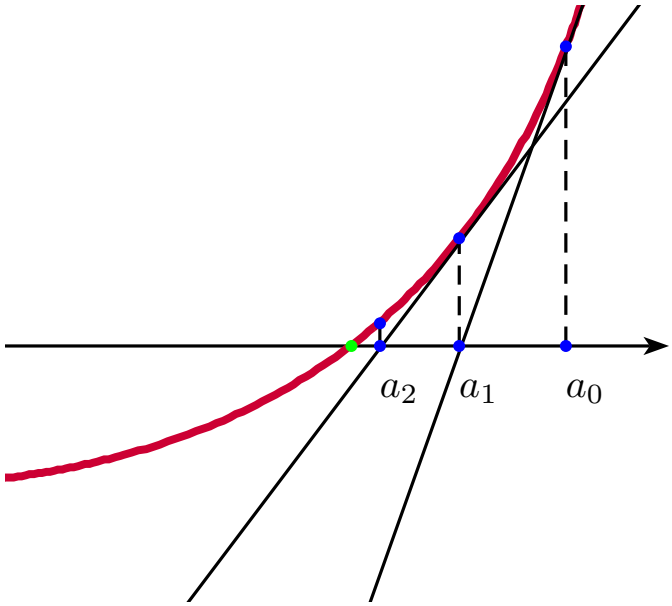

FIGURE 9 – Méthode de Newton

On refait l'analyse comme pour la méthode des cordes, en supposant donc travailler avec un fonction dérivable :

– l'équation de la tangente en a_n est

$$y = f'(a_n)(x - a_n) + f(a_n)$$

– celle-ci coupe l'axe des abscisses pour $y = 0$, soit

$$0 = f'(a_n)(x - a_n) + f(a_n) \iff x = a_n - \frac{f(a_n)}{f'(a_n)}$$

On programme encore une fois pour le cas $f(x) = x^2 - n$, et donc $a_{n+1} = a_n - \frac{a_n^2 - n}{2a_n}$.

Pseudo-code

Entrées : n,[a ;b],p
Faire p fois :
$a := a - \frac{a^2-n}{2a}$
Fin faire
Renvoyer a

Code TI-Basic

:Prompt N,A,P
:For(Y,1,P**)**
:$A - (A^2 - N)/(2A) \to$ A
:End
:Disp "A=",A

FIGURE 10 – Code de la méthode de Newton

Rapidement, pour ce qui est de la vitesse de convergence de l'algorithme, on imagine bien qu'il faut se trouver dans un domaine avec des onnes propriétés sur f', voire f''. Ce qu'il faut surtout retenir, c'est que si $a_n \underset{n\to+\infty}{\to} a$ et que si $|a_n - a| < \epsilon$, alors $|a_{n+1} - a| < k\epsilon^2$, la convergence est donc très rapide. Cependant, elle nécessite de savoir calculer une dérivée, ce qui n'est pas toujours le cas, loin de là...

5 Et en vrai...

Et bien on se sert de ces méthodes, notamment celle de Newton. Des fois, on les mélange un peu, on trouve un bon encadrement par dichotomie avant de finir par une méthode des cordes et/ou de Newton. Et puis on se sert aussi de quelques autres trucs magiques, dont une intégrale due à Gauss qui permet miraculement de retrouver le *log*, donc par la suite l'exponentielle.

De façon plus générale, ce genre de technique est aussi très utilisé en optimisation, afin de minimiser des fonctions. On les étend sans trop de souci à des fonctions de plusieurs variables, et on raffine même en jouant avec la dérivée seconde.

La division, en tant qu'être divisé, n'est pas ponctualité absolue.
La notion de continuité n'est pas non plus l'indivisé sans parties.
[Zénon d'Élée]