# Mathématiques et Modélisation

*Comment les maths décrivent-elles le monde qui nous entoure ?*

**Dispositif MathC2+**
*Encadrant :* Sacha Cardonna

3 mars 2024

Ce document présente une exploration approfondie des principes de l'intégration et de leur application dans le contexte de la modélisation physique, plus précisément à travers l'exemple de la chute libre. Initialement, nous introduisons le concept d'intégration, qui permet de calculer des aires sous des courbes et s'avère cruciale dans de nombreux domaines tels que la physique et l'ingénierie. L'accent est mis sur la façon dont l'intégration et la dérivation se complètent, illustrant l'importance de ces outils dans l'analyse et la compréhension des phénomènes dynamiques.

Nous progressons ensuite vers une application concrète de ces concepts mathématiques en examinant le phénomène de la chute libre, où nous dérivons les équations de mouvement qui gouvernent la chute d'un objet sous l'effet de la gravité, sans résistance de l'air. On détaillera la formulation du problème, depuis l'établissement des équations de base jusqu'à leur résolution, offrant ainsi un aperçu de l'application directe des techniques d'intégration pour la résolution de problèmes.

La résolution des équations de mouvement se décline en deux approches : une résolution théorique, s'appuyant sur des méthodes analytiques classiques pour trouver des solutions exactes, et une résolution numérique, où nous explorons diverses techniques algorithmiques pour approximer les solutions de situations plus complexes. Cette dualité entre approches théorique et numérique souligne la polyvalence des mathématiques dans l'approche des défis scientifiques.

En annexe, une implémentation en Python des méthodes numériques discutées offre un pont entre la théorie mathématique et sa mise en application pratique. Ce volet pratique vise à familiariser le lecteur avec les techniques de programmation qui permettent de traduire les modèles mathématiques en solutions concrètes, illustrant ainsi l'intersection entre les mathématiques, la physique et l'informatique.

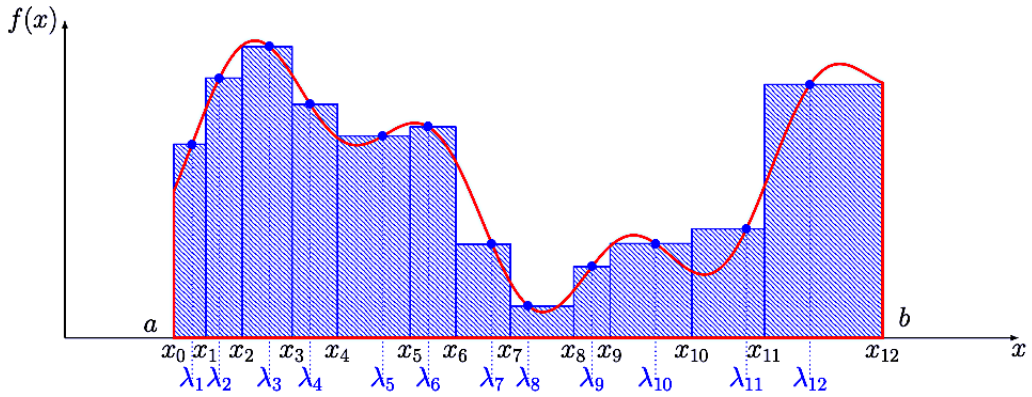
Ce document, en introduisant la notion d'intégrale, sa signification physique à travers l'étude de la chute libre, et son application numérique via la programmation, se veut une introduction complète aux mathématiques et à l'application de celles-ci pour modéliser les phénomènes physiques.

## 1. Préambule : introduction à l'intégration.

L'intégration est une notion fondamentale en mathématiques, étroitement liée à l'idée de sommation et de calcul d'aires. Elle joue un rôle crucial dans divers domaines tels que la physique, l'ingénierie et bien sûr, les mathématiques elles-mêmes.

L'intégration peut être vue comme l'opération inverse de la dérivation. En effet : si la dérivation vous donne la pente d'une courbe à un point donné, l'intégration, d'une certaine manière, fait l'inverse. Elle vous permet de retrouver la fonction originale à partir de sa dérivée. Elle a égalemenr d'autres propriétés très intéressantes : imaginez que vous vouliez trouver l'aire sous une courbe entre deux points sur un axe. L'intégration vous permet de calculer cette aire, même lorsque la forme sous la courbe n'est pas régulière.

Donnons-en une définition rigoureuse :




**Définition.** *Soit $f$ une fonction définie sur un intervalle fermé $[a,b]$ de $\mathbb{R}$. Si on divise $[a,b]$ en $n$ sous-intervalles avec une division $P = \{x_0, x_1, \ldots, x_n\}$ où $a = x_0 < x_1 < \cdots < x_n = b$ et $\Delta x_i = x_i - x_{i-1}$, alors pour chaque sous-intervalle $[x_{i-1}, x_i]$, on peut choisir un point $\lambda_i \in [x_{i-1}, x_i]$ et on forme la somme des aires des rectangles $S_P(f) = \sum_{i=1}^{n} f(\lambda_i)\Delta x_i$. L'intégrale est alors la limite de cette somme comme suit :*

$$\int_a^b f(x)\,\mathrm{d}x := \lim_{\Delta x_i \to 0} \sum_{i=1}^{n} f(\lambda_i)\Delta x_i.$$

Formellement donc, si on veut calculer l'aire sous $f(x)$ de $a$ à $b$, on divise cet intervalle en $n$ rectangles, et on fait la somme de leurs aires. En faisant tendre $n$ vers l'infini (et donc la largeur des rectangles vers zéro), on obtient l'intégrale de $f$ de $a$ à $b$.

Mais alors comment peut-on la calculer facilement ? Car devoir définir un grand nombre de rectangles à chaque fois et additionner leur aire à l'infini, ça peut vite devenir un peu long...
Heureusement, il existe le théorème fondamental du calculus ! Celui-ci relie directement le processus de dérivation à celui d'intégration et se divise en deux parties principales (néanmoins ici on ne donne que la partie "calculatoire"...)

**Théorème.** *Soit $f$ une fonction continue sur un intervalle fermé $[a,b]$, et soit $F$ une primitive de $f$ sur $[a,b]$, c'est-à-dire une fonction telle que $F'(x) = f(x)$ pour tout $x$ dans $[a,b]$. Alors, l'intégrale définie de $f$ sur l'intervalle $[a,b]$ est donnée par*

$$\int_a^b f(x)\,\mathrm{d}x = [F(x)]_a^b = F(b) - F(a).$$

Ce théorème illustre comment l'intégration et la dérivation sont des processus inverses l'un de l'autre, et comment on peut passer de l'un à l'autre.

## Quelques exemples.

Ici, nous allons présenter quelques exemples d'intégration avec des fonctions simples, expliquant étape par étape comment calculer l'aire sous la courbe.

### Fonction linéaire

Considérons la fonction $f(x) = 2x$. Nous voulons trouver l'aire sous cette courbe entre $x = 0$ et $x = 2$.

$$\int_0^2 2x\,dx = \left[\frac{2x^2}{2}\right]_0^2 = \left[x^2\right]_0^2 = 2^2 - 0^2 = 4.$$

L'aire sous la courbe de $f(x) = 2x$ de $x = 0$ à $x = 2$ est donc 4 unités carrées.

**Fonction quadratique**

Passons à une fonction un peu plus complexe, $g(x) = x^2$. Calculons l'aire sous $g(x)$ de $x = 0$ à $x = 1$.

$$\int_0^1 x^2\,dx = \left[\frac{x^3}{3}\right]_0^1 = \frac{1^3}{3} - \frac{0^3}{3} = \frac{1}{3}.$$

L'aire sous $g(x) = x^2$ de $x = 0$ à $x = 1$ est $\frac{1}{3}$ unité carrée.

**Fonction polynomiale**

Encore plus dur, considérons $h(x) = x^3 - 3x$. Trouvons l'aire sous cette courbe de $x = 0$ à $x = 2$.

$$\int_0^2 (x^3 - 3x)\,dx = \left[\frac{x^4}{4} - \frac{3x^2}{2}\right]_0^2 = \left(\frac{2^4}{4} - \frac{3 \cdot 2^2}{2}\right) - \left(\frac{0^4}{4} - \frac{3 \cdot 0^2}{2}\right) = 2 - 6 = -4.$$

L'aire sous la courbe de $h(x) = x^3 - 3x$ de $x = 0$ à $x = 2$ est $-4$ unités carrées.

### Conclusion.

L'intégration est une technique puissante en mathématiques, permettant de résoudre de nombreux problèmes pratiques et théoriques. En comprenant à la fois son aspect heuristique et théorique, vous pouvez commencer à voir les mathématiques sous un nouveau jour, plein de possibilités à explorer. Commençons maintenant à nous amuser un peu plus !

## 2. Modélisation : la chute libre.

Un objet est lâché avec vitesse initiale vers le haut, depuis une tour de hauteur $h$. On souhaite déterminer le temps $t$ qu'il faut à l'objet pour atteindre le sol, en supposant qu'il n'y a pas de résistance de l'air.
On souhaite donc

— Formuler le problème : Mettre en équation le mouvement de chute libre de l'objet.

— Résoudre le problème : Calculer le temps de chute $t$ en fonction de la hauteur $h$ de la tour.

### Données

— Accélération due à la gravité $g = 9,81\,m/s^2$.

— Hauteur de la tour $h_0 = 10\,m$.

### Équation du mouvement.

La fonction suivante

$$h(t) = \frac{1}{2}gt^2 + v_0t + h_0, \quad g = 9,81\,m/s^2, \quad v_0 \text{ vitesse initiale } (m/s), \quad h_0 \text{ hauteur initiale } (m), \quad t \text{ temps } (s),$$

décrit la distance $h$ parcourue par un objet en chute libre tombant d'une hauteur $h_0$ après un temps $t$, sous l'unique influence de la gravité $g$. On peut obtenir celle-ci à partir des principes de base de la cinématique en physique.

Dans le cadre de l'étude de la chute libre, il est essentiel de comprendre les relations entre la position, la vitesse, et l'accélération d'un objet. Ces relations sont au cœur de la cinématique, la branche de la physique qui étudie le mouvement sans considérer les forces qui le causent.

— **Position** ($h$ dans ce contexte) est la localisation d'un objet par rapport à un point de référence (ici on considère uniquement l'axe $Oz$ vertical).

— **Vitesse** ($v$) est le taux de changement de la position par rapport au temps, et est donnée par la dérivée de la position par rapport au temps : $v(t) = \frac{dh}{dt}(t) = h'(t)$.

— **Accélération** ($a$) est le taux de changement de la vitesse par rapport au temps, et est donnée par la dérivée de la vitesse par rapport au temps : $a(t) = \frac{dv}{dt}(t) = v'(t)$.

Dans le cas de la chute libre, l'accélération est constante et égale à $g$, l'accélération due à la gravité ($9,81 m/s^2$ sur Terre). Cela simplifie notre analyse, car la dérivée de la vitesse par rapport au temps est constante.

**Relation entre accélération et vitesse.**

Puisque l'accélération est constante, la relation entre la vitesse et le temps peut être exprimée comme suit :

$$v(t) = \int a \, \mathrm{d}t = \int g \, \mathrm{d}t = gt + C_1,$$

où $C_1$ est une constante d'intégration qui représente la vitesse initiale. Pour une chute libre qui commence au repos, $C_1 = 0$, donc $v(t) = gt$. Ici nous faisons un cas plus général avec une vitesse initiale, donc $C_1 = v_0$, et ainsi

$$v(t) = gt + v_0.$$

**Relation entre vitesse et position.**

De manière similaire, la position en fonction du temps peut être obtenue en intégrant l'expression de la vitesse :

$$h(t) = \int v(t) \, \mathrm{d}t = \int (gt + v_0) \, \mathrm{d}t = \frac{1}{2}gt^2 + v_0 t + C_2,$$

où $C_2$ est une autre constante d'intégration qui représente la position initiale. Pour un objet qui commence sa chute à partir d'une hauteur $h$, avec $h(t = 0) = h_0$, on a $C_2 = h_0$, donc

$$h(t) = \frac{1}{2}gt^2 + v_0 t + h_0.$$

Cette formule découle donc directement de l'intégration de l'accélération due à la gravité, en considérant que l'objet en chute libre débute son mouvement avec une certaine vitesse et depuis une position initiale de référence. Cette formule illustre une relation fondamentale en cinématique pour des mouvements avec une accélération constante, en particulier dans le contexte de la chute libre sous l'effet de la gravité.

**Remarque.** *Comme vous le remarquez, on ne parle jamais de masse de l'objet qui tombe : c'est parce que sous couvert qu'on néglige les frottements de l'air, un piano et une balle de tennis tomberont exactement à la même vitesse et s'écraseront au même moment au sol. La résistance de l'air peut cependant grandement affecter le temps de chute en ralentissant l'objet ; c'est parce qu'elle subit énormemenr de frottements qu'une plume met du temps à toucher le sol. Cependant, pour des objets en chute libre sur de courtes distances ou dans un contexte pédagogique, cette force peut être négligée pour faciliter les calculs et la compréhension du phénomène de chute libre sous l'effet uniquement de la gravité.*

## 3. Analyse : résolution de l'équation.

Rappellons la fonction qui donne la hauteur de l'objet en chute libre en fonction du temps :

$$h(t) = \frac{1}{2}gt^2 + v_0 t + h_0.$$

Admettons que l'on souhaite connaître le temps exact où l'objet va toucher le sol : cela revient donc à résoudre l'équation $h(t) = 0$, i.e.

$$\frac{1}{2}gt^2 + v_0 t + h_0 = 0.$$

Il s'agit d'une équation du type

$$ax^2 + bx + c = 0,$$

où $x = t$, $a = \frac{1}{2}g$, $b = v_0$ et $c = h_0$.

## 3.1. Résolution théorique de l'équation.

Pour les équations de degré 2, on a accès à une solution théorique via le discriminant $\Delta = b^2 - 4ac$. En effet :

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, \qquad x_2 = \frac{-b + \sqrt{\Delta}}{2a},$$

sous couvert que $\Delta \geq 0$, sinon l'équation n'admet pas de solutions réelles. En substituant $a$, $b$, et $c$ par leurs valeurs, on a

$$\Delta = v_0^2 - 4\left(-\frac{1}{2}g\right)h_0 = v_0^2 + 2gh_0.$$

Les racines de l'équation sont données par la formule quadratique ci-dessus, et en subsituant $a$, $b$, et $\Delta$ on obtient :

$$t_1 = \frac{-v_0 - \sqrt{v_0^2 + 2gh_0}}{-g}, \qquad t_2 = \frac{-v_0 + \sqrt{v_0^2 + 2gh_0}}{-g}.$$

Cela nous donne deux solutions possibles pour $t$, correspondant aux moments où l'objet atteint le sol ou la hauteur cible. Il est important de noter que selon les valeurs de $v_0$ et $h_0$, le discriminant $\Delta$ peut affecter le nombre et le type de solutions (réelles ou complexes). A noter qu'il faut ici éliminer toute solution non-positive, étant donné qu'on ne peut pas avoir de temps négatif.

Par exemple, prenons $h_0 = 10\ m$ et $v_0 = 10\, m/s$. En substituant $g$, $v_0$, et $h_0$ par leurs valeurs, nous obtenons :

$$-\frac{1}{2}(9.81)t^2 + 10t + 10 = 0,$$

équation pouvant être réécrite comme

$$-4.905t^2 + 10t + 10 = 0.$$

Pour résoudre cette équation, nous utilisons le discriminant $\Delta$ qui est donc

$$\Delta = b^2 - 4ac = 10^2 - 4(-4.905)(10) = 100 + 196.2 = 296.2.$$

Les solutions pour $t$ sont ainsi

$$t = \frac{-10 \pm \sqrt{296.2}}{2(-4.905)}.$$

Nous obtenons finalement deux valeurs pour $t$ :

$$t_1 = \frac{-10 - 17.2}{-9.81} \approx 2.773747393938097\,\text{s},$$

$$t_2 = \frac{-10 + 17.2}{-9.81} \approx -0.7350114102479848\,\text{s}.$$

Dans un scénario réaliste, si l'objet est lancé vers le haut :

— $t_1$ (le plus grand des deux temps calculés) représenterait le temps total depuis le lancement jusqu'à ce que l'objet retombe au sol, prenant en compte la montée jusqu'au point le plus haut et la descente jusqu'au sol. C'est le temps total de vol.

— $t_2$ (le plus petit des deux temps) n'a pas de signification physique directe dans ce contexte, car il représenterait un instant avant le lancement, basé sur la manière dont les équations quadratiques sont résolues mathématiquement. Pour les calculs physiquement significatifs, nous nous concentrons sur les instants après le lancement (temps positifs).

## 3.2. Résolution numérique de l'équation.

Contrairement à nos équations du type $ax^2 + bx + c = 0$, la plupart des équations qu'on utilise pour décrire la physique sont bien plus complexes et leurs solutions ne peuvent pas toujours être calculées explicitement. On a donc recours à des méthodes numériques qui calculent pour nous les solutions approchées des équations, qui sont des approximations des solutions exactes.

### 3.2.1. Méthode de bisection.

La méthode de bisection est une méthode numérique parmi d'autres, et donc une façon de trouver une solution approchée à une équation. Ici on va l'utiliser pour notre équation générale, et vérifier qu'on retrouvera bien une solution suffisamment proche de la solution théorique. Considérons la fonction $f(x) = ax^2 + bx + c$. On cherche maintenant à savoir pour quel(s) $x$ on a $f(x) = 0$. On peut procéder comme suit :

— *Choix de l'intervalle initial.*
On commence avec un intervalle large où l'on pense que la ou les solution(s) se trouve(nt) (par exemple de -1000 à 1000).

— *Point milieu.*
On calcule le point au milieu de cet intervalle (par exemple $x_{\text{milieu}} = 0$).

— *Vérification de la valeur de $f(x_{milieu})$.*
On calcule la valeur de la fonction $f(x)$ au point milieu, noté $x_{\text{milieu}}$. Ensuite, on vérifie si cette valeur est très proche de zéro, à l'intérieur d'une marge d'erreur définie (la tolérance, notée $\varepsilon$). Si $f(x_{\text{milieu}})$ est plus petit que la tolérance (au signe près), on considère qu'on a trouvé une solution approximative et on peut arrêter le processus.

— *Réduction de l'intervalle.*
Si $f(x_{\text{milieu}})$ n'est pas suffisamment proche de zéro, on doit déterminer dans quelle moitié de l'intervalle actuel se trouve la solution. Pour ce faire, on regarde le signe du produit $f(x_{\text{min}}) \times f(x_{\text{milieu}})$ :
  — **Si le produit est strict. négatif** ($f(x_{\text{min}})$ et $f(x_{\text{max}})$ sont de signes opposés) :
  Cela signifie que la fonction change de signe entre $x_{\text{min}}$ et $x_{\text{milieu}}$. Donc, la solution doit être dans cette moitié de l'intervalle. On ajuste alors $x_{\text{max}}$ pour qu'il soit égal à $x_{\text{milieu}}$.
  — **Si le produit est strict. positif** ($f(x_{\text{min}})$ et $f(x_{\text{max}})$ sont de même signes) :
  La fonction ne change pas de signe entre $x_{\text{min}}$ et $x_{\text{milieu}}$. Cela suggère que la solution se trouve dans l'autre moitié de l'intervalle. On met donc à jour $x_{\text{min}}$ pour qu'il soit égal à $x_{\text{milieu}}$.
  — **Si le produit est nul** ($f(x_{\text{min}}) = 0$ ou $f(x_{\text{max}}) = 0$) :
  Dans ce cas, soit $x_{\text{min}}$ soit $x_{\text{max}}$ est solution de l'équation, on trouve lequel et on obtient une solution qui est exacte.

  En faisant ces ajustements, on réduit l'intervalle de recherche de moitié à chaque étape, se rapprochant ainsi progressivement de la solution.

— *Répétition.*
On répète le processus avec le nouvel intervalle jusqu'à obtenir une solution suffisamment précise ou atteindre un nombre maximal d'itérations.

Cependant, faire tous ces calculs à la main peut devenir un processus très laborieux, requérant beaucoup de temps et de concentration. Heureusement, avec le développement des technologies informatiques, nous pouvons déléguer ces calculs répétitifs à un ordinateur. En utilisant des programmes informatiques, nous pouvons exécuter ces méthodes numériques rapidement et avec une grande précision.

### 3.2.2. Méthode de Newton.

La méthode de Newton, également connue sous le nom de méthode de Newton-Raphson, est également une méthode puissante pour trouver numériquement les racines d'une équation réelle. Pour notre équation générale $f(x) = ax^2 + bx + c$, on utilise la méthode de Newton pour trouver une fois de plus les valeurs de $x$ tel que $f(x) = 0$. La procédure est la suivante :

— *Choix de l'approximation initiale.*
On choisit une valeur initiale pour $x$, notée $x_0$, qui est notre première estimation de la racine de l'équation.

— *Itération de Newton.*
À chaque étape de l'itération, on calcule une nouvelle approximation de la racine en utilisant la formule :

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

où $f'(x_n)$ est la dérivée de $f$ évaluée à $x_n$. Cette formule nous donne le prochain point d'approximation basé sur la tangente de la fonction au point courant.

— *Critère de convergence.*
On continue les itérations jusqu'à ce que la différence entre deux approximations successives soit inférieure à une tolérance prédéfinie, c'est-à-dire $|x_{n+1} - x_n| < \varepsilon$, où $\varepsilon$ est un petit nombre positif choisi pour déterminer le degré de précision souhaité.

— *Vérification de la dérivée.*
Il est crucial de s'assurer que $f'(x_n) \neq 0$ pendant les itérations, car une dérivée nulle entraînerait une division par zéro dans la formule d'itération. Si $f'(x_n) = 0$, on doit choisir une autre approximation initiale.

La méthode de Newton converge généralement plus rapidement que la méthode de bisection, surtout si l'approximation initiale est proche de la vraie racine. Cependant, sa convergence n'est pas garantie si la fonction se comporte mal ou si l'approximation initiale est trop éloignée de la racine réelle. Elle est particulièrement efficace pour les fonctions dont les dérivées peuvent être facilement calculées et pour lesquelles une bonne approximation initiale est disponible.

### 3.2.3. Méthode de la sécante.

La méthode de la sécante est encore une autre technique d'approximation numérique visant à trouver les racines d'une équation réelle. Elle est particulièrement utile pour les équations où le calcul de la dérivée est complexe ou indésirable. Pour l'équation $f(x) = ax^2 + bx + c$, la méthode procède comme suit :

— *Choix des approximations initiales.*
On choisit deux valeurs initiales, $x_0$ et $x_1$, qui sont nos premières estimations des racines de l'équation. Ces valeurs devraient, idéalement, encadrer la racine recherchée.

— *Itération de la sécante.*
À chaque étape, on calcule une nouvelle approximation en utilisant la formule :

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

Cette formule utilise les valeurs de la fonction aux deux dernières approximations pour estimer la position de la racine, sans nécessiter le calcul de la dérivée.

— *Critère de convergence.*
Les itérations continuent jusqu'à ce que la différence entre deux approximations successives soit inférieure à une tolérance prédéfinie, $|x_{n+1} - x_n| < \varepsilon$, où $\varepsilon$ est un seuil définissant la précision désirée.

— *Vérification de la différence.*
Il est ici crucial de s'assurer que $f(x_n) - f(x_{n-1}) \neq 0$ pendant les itérations, car une différence nulle entraînerait une division par zéro dans la formule d'itération.

La méthode de la sécante est souvent plus rapide que la méthode de bisection mais peut être moins fiable que la méthode de Newton si les approximations initiales ne sont pas proches de la vraie racine. De plus, la méthode peut échouer à converger si $f(x_n) = f(x_{n-1})$, car cela entraînerait une division par zéro. Elle reste cependant un bon compromis entre la simplicité de la méthode de bisection et l'efficacité de la méthode de Newton, surtout dans les cas où le calcul des dérivées est impraticable.

Le développement de méthodes numériques générales est crucial car il nous permet de trouver des solutions à un large éventail d'équations, y compris celles qui sont trop complexes pour être résolues analytiquement. En rendant ces méthodes applicables à un grand nombre de situations, nous ouvrons la voie à la résolution de problèmes mathématiques variés et complexes.

## 4. Implémentation : résolution sous Python.

Dans cette partie, on s'intéresse à l'implémentation (= intégrer nos calculs sur ordinateur) des méthodes que nous avons employés. Nous utiliserons le langage de programmation Python.

Comme dit précédemment, notamment pour les méthodes numériques, faire les calculs à la main est long et dangereux (surtout si on fait des erreurs qui s'accumulent), et comme les mathématiciens sont bien souvent fainéants quand il s'agit de calculer, on va tout déléguer à notre ordinateur. Donnons donc les quelques bases nécessaires pour écrire notre premier code sur Python.

## Fonction `print`.

La fonction **`print`** est l'une des fonctions les plus fondamentales et les plus utilisées en Python. Elle permet d'afficher du texte, des nombres ou des résultats de variables sur la console ou l'écran de l'ordinateur. La syntaxe de base est :

```
print(objet_a_afficher)

Résultat : objet_a_afficher
```

L'objet à afficher peut être une chaîne de caractères (texte), un nombre, une variable, ou même le résultat d'une expression ou d'une autre fonction. Pour afficher simplement du texte, on utilise des guillemets autour du texte :

```
print("On aime la methode de la bisection mon baka")

Résultat : On aime la methode de la bisection mon baka
```

La fonction **`print`** peut également être utilisé pour afficher le contenu d'une variable :

```
x = 10
print(x)

Résultat : 10
```

On peut aussi afficher plusieurs objets en les séparant par des virgules :

```
nom = "Sacha"
age = 26
print(nom, "a", age, "ans")

Résultat : Sacha a 26 ans
```

La fonction **`print`** est essentielle pour la communication entre le programme et l'utilisateur. Elle est souvent utilisée pour déboguer des programmes en affichant l'état des variables et le flux d'exécution.

## Fonction `input`.

La fonction **`input`** en Python est utilisée pour recueillir une entrée de l'utilisateur. Lorsque Python exécute cette fonction, le programme s'arrête et attend que l'utilisateur tape quelque chose, puis appuie sur Entrée. La syntaxe de base est :

```
variable = input("Message pour l'utilisateur")
```

Le "Message pour l'utilisateur" est un texte qui s'affiche à l'écran pour indiquer à l'utilisateur quel type d'entrée est attendu. L'entrée saisie par l'utilisateur est alors stockée dans la variable spécifiée. Par exemple :

```
nombre_entier = input("Entrez un nombre entier :")

Résultat : Entrez un nombre entier :
```

## Fonction `def`.

En Python, `def` est utilisé pour définir une fonction. Une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. La syntaxe de base pour définir une fonction est :

```
def nom_de_la_fonction(parametres):
    # Bloc de code de la fonction
```

Le "nom_de_la_fonction" est le nom donné à la fonction. Les "parametres" sont les informations dont la fonction a besoin pour effectuer sa tâche. Une fois définie, la fonction peut être appelée par son nom ailleurs dans le programme.

Les fonctions `input` et `def` sont des éléments fondamentaux de la programmation Python. `input` permet de recueillir des entrées de l'utilisateur, tandis que `def` permet de créer des fonctions personnalisées, rendant le code plus modulaire, réutilisable et organisé.

## Boucles `if` et `while`.

La boucle `if` est utilisée pour exécuter un bloc de code seulement si une condition spécifiée est vraie. En Python, la syntaxe de base est :

```
if condition:
    # Bloc de code qu'on execute uniquement si la condition est vraie
```

Si la condition est fausse, le bloc de code sous le `if` est ignoré. Python supporte également les instructions `elif` (sinon si) et `else` pour gérer plusieurs conditions.

La boucle `while` permet d'exécuter un bloc de code tant qu'une condition donnée est vraie. La syntaxe est :

```
while condition:
    # Bloc de code a executer tant que la condition est vraie
```

Cette boucle continuera à s'exécuter, répétant le bloc de code, tant que la condition de la boucle reste vraie. Il est important de s'assurer que la condition devienne fausse à un moment donné, sinon la boucle continuera indéfiniment, créant ainsi une boucle infinie.

Ces structures de contrôle de base, `if` et `while`, sont essentielles en programmation Python. Elles permettent de contrôler le flux d'exécution du programme en fonction des conditions et de répéter des opérations, ce qui est un aspect fondamental de la programmation informatique.

# Annexe : code de résolution.

```python
import math

print("Programme de resolution d'une équation du type ax^2 + bx + c = 0\n")

# Entrée des coefficients
a = float(input("Entrez la valeur de a : "))
b = float(input("Entrez la valeur de b : "))
c = float(input("Entrez la valeur de c : "))

# Affichage de l'équation
print(f"\nL'équation est {a}x^2 + {b}x + {c} = 0.")

# ------------ Résolution théorique --------------
print("\n**** RESOLUTION THEORIQUE ****")

# Calcul du discriminant
delta = b**2 - 4 * a * c

# Calcul des racines théoriques si delta >= 0
if delta < 0:
    print(
        "\nAttention : Le discriminant est négatif. L'équation n'a pas de racines réelles."
    )
    print("Arrêt du programme.")
    exit()

else:
    x1 = (-b - math.sqrt(delta)) / (2 * a)
    x2 = (-b + math.sqrt(delta)) / (2 * a)
    if delta == 0:
        print(f"\nL'équation a une racine réelle double : x = {x1}.")
    else:
        print(f"\nL'équation a deux racines réelles : x1 = {x1} et x2 = {x2}.")

# ------------ Résolution numérique --------------
print("\n\n**** RESOLUTION NUMERIQUE ****")

# Fonction associée à l'équation quadratique
def f(x):
    return a * x**2 + b * x + c

# Dérivée de la fonction
def df(x):
    return 2 * a * x + b

# Paramètres généraux
tolerance = 1e-10  # Définit la tolérance pour la précision de la solution.
max_iter = 100  # Définit le nombre maximum d'itérations pour les méthodes numériques
                afin d'éviter une boucle infinie.


# Méthode 1 : Bisection
x_min_bisection, x_max_bisection = -100000, 100000  # Définit un large intervalle initial pour la recherche de la racine.
compteur_bisection = 0  # Initialise un compteur pour suivre le nombre d'itérations effectuées.

while compteur_bisection < max_iter:
    x_mid_bisection = (x_min_bisection + x_max_bisection) / 2

    if abs(f(x_mid_bisection)) < tolerance:
        break

    elif f(x_min_bisection) * f(x_mid_bisection) < 0:
        x_max_bisection = x_mid_bisection

```

```
        else:
            x_min_bisection = x_mid_bisection

        compteur_bisection += 1

    x_num_bisection = x_mid_bisection

    # Affichage de la solution numérique
    print(
        f"\nBISECTION : une solution approchée est x = {x_num_bisection} au bout de {compteur_bisection} itération(s)."
    )


    # Méthode 2 : Newton
    x_newton = 0  # Approximation initiale pour la méthode de Newton
    compteur_newton = 0

    while compteur_newton < max_iter:
        if df(x_newton) == 0:
            print(
                "Division par zéro dans la méthode de Newton, ajustez l'approximation initiale."
            )
            break

        x_new = x_newton - f(x_newton) / df(x_newton)  # Formule de Newton

        if abs(x_new - x_newton) < tolerance:
            break

        x_newton = x_new
        compteur_newton += 1

    x_num_newton = x_newton

    # Affichage de la solution numérique
    print(
        f"NEWTON : une solution approchée est x = {x_num_newton} au bout de {compteur_newton} itération(s)."
    )


    # Méthode 3 : Sécante
    x0_sec, x1_sec = -10000, 10000 # Deux approximations initiales pour la méthode de la sécante
    compteur_sec = 0

    while compteur_sec < max_iter:
        if f(x1_sec) - f(x0_sec) == 0:
            print(
                "Division par zéro dans la méthode de la sécante, ajustez les approximations initiales."
            )
            break

        x2_sec = x1_sec - f(x1_sec) * (x1_sec - x0_sec) / (f(x1_sec) - f(x0_sec))  # Formule de la sécante

        if abs(x2_sec - x1_sec) < tolerance:
            break

        x0_sec = x1_sec
        x1_sec = x2_sec
        compteur_sec += 1

    x_num_sec = x2_sec

    # Affichage de la solution numérique
    print(
        f"SECANTE : une solution approchée est x = {x_num_sec} au bout de {compteur_sec} itération(s)."
    )
```