# – Mathématiques Appliquées –

*Comment les équations décrivent des problèmes physiques concrets ?*

*Stagiaires* : Pau Bernardo & Sarah Marquez
*Encadrant* : Sacha Cardonna

17 décembre 2023

Dans notre approche des mathématiques, nous connectons les concepts théoriques à des situations physiques réelles, rendant l'apprentissage plus tangible. Par exemple, en utilisant des équations pour simuler la rencontre de deux trains, nous démontrons l'utilité pratique des mathématiques dans la compréhension de phénomènes physiques comme le mouvement et la distance.
La résolution d'équations, qu'elle soit théorique ou assistée par ordinateur, nous permet d'explorer ces scénarios de manière concrète. Cela illustre non seulement l'importance des mathématiques dans le monde moderne, mais aussi comment elles peuvent être appliquées pour résoudre des problèmes réels. Ainsi, les mathématiques deviennent un outil précieux, pas seulement une série de formules abstraites.

## 1. Modélisation : le problème des deux trains.

Considérons deux trains, Train A et Train B, qui partent simultanément de deux villes différentes, séparées par une distance de 120 km (par exemple Bruxelles et Lille). Ils se déplacent l'un vers l'autre sur la même voie et on cherche à savoir à quel moment ils se rencontreront, histoire d'éviter la collision.

FIGURE 1 – Deux trains générés par de l'intelligence artificielle.

### Données du problème.

— Distance entre les deux villes : $d = 120$ km.
— Vitesse du Train A : $v_A = 30$ km/h.
— Vitesse du Train B : $v_B = 40$ km/h.

### Formulation mathématique.

Soit $t$ le temps (en heures) au bout duquel les deux trains se rencontrent. Rappelons la formule générale liant vitesse, temps et distance :

$$v = \frac{d}{t}, \qquad v \text{ : vitesse (km/h)}, \quad d \text{ : distance (km)}, \quad t \text{ : temps (h)}.$$

La distance $d_A$ parcourue par le Train A après $t$ heures est donc obtenue en adaptant cette formule :

$$v_A = \frac{d_A}{t} \quad \Longleftrightarrow \quad d_A = v_A \times t = 30 \times t$$

$v_A$ : vitesse du Train A (30 km/h), $d_A$ : distance parcourue par le train A, $t$ : temps.

De la même manière, la distance $d_B$ parcourue par le Train B est donc

$$v_B = \frac{d_B}{t} \quad \Longleftrightarrow \quad d_B = v_B \times t = 40 \times t$$

$v_B$ : vitesse du Train B (40 km/h), $d_B$ : distance parcourue par le train B, $t$ : temps.

La somme de ces deux distances doit être égale à la distance totale ($d = 120$ km) entre les deux villes. Ainsi, nous pouvons établir l'équation suivante :

$$d_A + d_B = v_A \times t + v_B \times t = 30t + 40t = 120.$$

On obtient finalement une équation avec $t$ comme inconnue :

$$30t + 40t = 120 \quad \Longleftrightarrow \quad 70t = 120.$$

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

Reprenons l'équation issue de notre problème :

$$(1) \quad 30t = -40t + 120 \quad \Longleftrightarrow \quad (2) \quad 70t = 120$$

L'équation de gauche (1) peut être vue comme un cas particulier d'une équation linéaire générale de la forme :

$$ax + b = cx + d$$

où ici nous avons

$$x = t \text{ (temps)}, \qquad a = 30 \ (= v_A), \qquad b = 0, \qquad c = -40 \ (= -v_B), \qquad d = 120 \text{ (distance totale)}.$$

L'équation de droite (2) peut elle être vue comme un cas particulier de $ax + b = cx + d$ sous la forme plus simple :

$$ax = b$$

avec

$$x = t \text{ (temps)}, \qquad a = 70 \ (= v_A + v_B), \qquad b = 120 \text{ (distance totale)}.$$

Dans notre cas nous nous concentrerons sur des équations du type $ax + b = cx + d$. En effet, les équations $ax = b$ sont des cas particuliers de celles-ci, et en mathématiques nous cherchons toujours à trouver des méthodes de résolution qui soient les plus générales possibles. Si on sait résoudre des équations comme l'équation (1), on saura directement résoudre les équations de type (2).

## 2.1. Résolution théorique de l'équation type $ax + b = cx + d$.

La méthode de résolution pour ce type d'équation linéaire est la suivante :

1. Commencer par regrouper tous les termes contenant $x$ d'un côté de l'équation (les termes $ax$ et $cx$) et les termes constants (les termes $b$ et $d$) de l'autre.
2. Isoler $x$ en divisant chaque côté de l'équation par le coefficient de $x$.

Ainsi, la résolution pas-à-pas se présente comme suit :

$$
\begin{aligned}
& ax + b = cx + d \\
\iff \quad & ax + b - cx - b = cx + d - cx - b && \text{(addition de } [-cx] \text{ et } [-b] \text{ des deux côtés)} \\
\iff \quad & ax - cx = d - b && \text{(simplification)} \\
\iff \quad & (a - c) \times x = d - b && \text{(factorisation)} \\
\iff \quad & \frac{(a - c) \times x}{a - c} = \frac{d - b}{a - c} && \text{(division par } [a - c] \text{ des deux côtés)} \\
\iff \quad & x = \frac{d - b}{a - c} && \text{(simplification)}
\end{aligned}
$$

**Remarque.** *Cette solution n'a pas de sens si $a = c$, car cela est équivalent à $a - c = 0$, et diviser par zéro c'est totalement* ***interdit*** *! De toute façon, chercher les solutions d'une équation de type $ax + b = ax + d$, c'est revenir à résoudre $b = d$, donc voir si les deux constantes sont égales ... Cela est évident et donc pas besoin de faire de maths.*

## 2.2. Résolution approchée de l'équation type $ax + b = cx + d$.

Contrairement à nos équations du type $ax + b = ax + d$, 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.

### 2.2.1. Méthode de bisection (débutant).

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.

Cette méthode consiste à diviser un intervalle en deux parties plusieurs fois d'affilées et chercher dans quel intervalle se trouve la solution de l'équation. On répète ce processus plusieurs fois pour se rapprocher de la solution. Pour une équation comme $ax + b = cx + d$, on la transforme d'abord en $(a - c) \times x = d - b$. Ensuite, on suit ces étapes :

1. Choisissez deux nombres entre lesquels vous pensez que la solution se trouve. Par exemple, entre 0 et 10. Ils deviennent $x_{\text{min}}$ et $x_{\text{max}}$.
2. Calculez la moyenne de ces deux nombres. C'est le milieu de votre intervalle.
3. Remplacez $x$ par ce nombre moyen $x_{\text{milieu}}$ dans votre équation pour voir si la solution est plus grande ou plus petite que ce nombre.
4. Selon le résultat :
   - Si $(a - c) \times x_{\text{milieu}} < d - b$, choisissez un nouvel intervalle $[x_{\text{milieu}}, x_{\text{max}}]$ entre le nombre moyen $x_{\text{milieu}}$ et le plus grand $x_{\text{max}}$.
   - Si $(a - c) \times x_{\text{milieu}} > d - b$, choisissez un nouvel intervalle $[x_{\text{min}}, x_{\text{milieu}}]$ entre le nombre le plus petit $x_{\text{min}}$ et le moyen $x_{\text{moyen}}$.
   - Sinon, si $(a - c) \times x_{\text{milieu}} = d - b$, vous avez trouvé le bon nombre et votre solution approchée $x_{\text{milieu}}$ sera exacte. Pas besoin d'aller à l'étape 5.
5. Répétez ces étapes jusqu'à ce que vous trouviez une solution assez précise.

Considérons maintenant un exemple pour mieux comprendre cette méthode.

**Exemple.** Prenons l'équation $3x + 2 = x + 6$. On la transforme en $3x - x = 6 - 2$ donc $2x = 4$.

1. On suppose que la solution se trouve entre les nombres 0 et 10, donc on prend l'intervalle 0 à 10.
2. La moyenne entre 0 et 10 est $\frac{0+10}{2} = 5$.
3. En remplaçant $x$ par 5 dans l'équation, on obtient $2 \times 5 = 10 > 4$, donc la solution est plus petite vu que $10 > 4$.
4. On choisit l'intervalle entre le plus petit nombre et le nombre moyen, donc l'intervalle 0 à 5.
5. On répète les étapes précédentes ...

Au bout d'un certain moment, cet algorithme va nous donner la solution de l'équation, avec une précision plus ou moins importante, notamment selon le nombre de fois qu'on est prêt à répéter ces étapes.

### 2.2.2. Méthode de bisection (expérimenté).

Une autre manière de voir la méthode de bisection, plus efficace, est d'utiliser les fonctions. Considérons la fonction $f(x) = ax + c - (bx + d) = ax - bx + c - d$. On cherche maintenant à savoir pour quel $x$ on a $f(x) = 0$, car

$$f(x) = 0 \quad \Longleftrightarrow \quad ax - bx + c - d = 0 \quad \Longleftrightarrow \quad ax + c = bx + d,$$

il s'agit donc du même $x$ qui vérifie $ax + c = bx + d$ et $f(x) = 0$. On peut donc définir une nouvelle fois une suite d'étapes :

1. *Choix de l'intervalle initial.*
   On commence avec un intervalle large où l'on pense que la solution se trouve (par exemple de -1000 à 1000).
2. *Point milieu.*
   On calcule le point au milieu de cet intervalle.
3. *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). 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.
4. *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 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 positif ou nul** ($f(x_{\text{min}})$ et $f(x_{\text{max}})$ sont de même signes, ou l'un d'eux est nul) :
     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 $x_{\text{milieu}}$.

   En faisant ces ajustements, on réduit l'intervalle de recherche de moitié à chaque étape, se rapprochant ainsi progressivement de la solution.
5. *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.
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.

### 2.3. Résolution de l'équation des deux trains.

Revenons à notre équation que nous avons obtenue dans la partie 1. On a donc :

$$30t = -40t + 120.$$

#### 2.3.1. Résolution théorique.

Par les calculs que nous avons réalisés dans la partie (2.1), on a accès à une formule générale pour les équations de ce type, c'est-à-dire

$$ax + b = cx + d \quad \Longleftrightarrow \quad x = \frac{d-b}{a-c},$$

donc ici, par identification :

$$30t = -40t + 120 \quad \Longleftrightarrow \quad t_{\text{exact}} = \frac{120-0}{30-(-40)} = \frac{120}{30+40} = \frac{120}{70} \simeq 1.71428571429 \text{ h},$$

ce qui correspond à environ 1 h et 43 minutes.

#### 2.3.2. Résolution numérique.

En utilisant la méthode de la bisection, on étudie ainsi le $t$ tel que la fonction suivante s'annule :

$$f(t) = 30t - (-40t + 120) = 70t - 120,$$

et au bout de 35 itérations, pour une tolérance de $10^{-6} = 0.000001$, on trouve

$$t_{\text{approché}} = 1.71428572503 \text{ h},$$

ce qui correspond très bien à la solution qu'on a calculé théoriquement. On a ainsi la preuve que notre algorithme marche bien, et on est donc très contents.

## 3. 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 la méthode de la bisection, 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 :

```
1 print(objet_a_afficher)
2 <<< 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 :

```
1 print("J'adore la methode de la bisection quoicoubeh !")
2 <<< J'adore la methode de la bisection quoicoubeh !
```

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

```
x = 10
print(x)
<<< 10
```

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

```
nom = "Sacha"
age = 25
print(nom, "a", age, "ans")
<<< Sacha a 25 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 :")
<<< Entrez un nombre entier :
>>> 46
```

### 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.

## Code de résolution.

Tous les détails de l'implémentation sont commentés sur ce code.

```
print("Programme de resolution d'une equation du type ax + b = cx + d\n")

 Entree 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 : "))
d = float(input("Entrez la valeur de d : "))

if a - c == 0:
    print("Erreur : il est imperatif que a soit different de c.")
    print("Arret du programme.")
    exit()

------------------------------

 Resolution exacte de l'equation
x_ex = (d - b) / (a - c)     Solution theorique obtenue apres manipulations algebriques.

------------------------------

 Resolution numerique de l'equation avec la methode de la bisection

 Cette fonction calcule la valeur de f(x) = ax + b - cx - d pour une valeur specifique de x.
def f(x):
    return a * x + b - c * x - d

x_min, x_max = -1000, 1000  Definit un large intervalle initial pour la recherche de la racine, de -1000 a
    1000.
tolerance = 1e-6          Definit la tolerance pour la precision de la solution. La methode s'arretera si
    l'ecart est inferieur a cette valeur.
max_iter = 100            Definit le nombre maximum d'iterations pour eviter une boucle infinie.
compteur = 0              Initialise un compteur pour suivre le nombre d'iterations effectuees.

while compteur < max_iter:          Debut d'une boucle qui continue jusqu'a ce que le nombre d'iterations
    atteigne le maximum.
    x_mid = (x_min + x_max) / 2     Trouve le point moyen de l'intervalle actuel.

    if abs(f(x_mid)) < tolerance:   Verifie si la valeur de l'equation a ce point median est suffisamment
        proche de zero (en fonction de la tolerance definie).
        break                       Si la condition est remplie, sort de la boucle (solution trouvee).

```

```
    elif f(x_min) * f(x_mid) < 0:    Verifie si le signe de la fonction change entre x_min et x_mid.
        x_max = x_mid                Si le signe de f(x_min) * f(x_mid) est negatif, cela signifie que la
            racine est entre x_min et x_mid. Ajuste x_max au point median.

    else:
        x_min = x_mid                Sinon, ajuste x_min au point median (la racine est entre x_mid et
            x_max).

    compteur += 1                    Incremente le compteur d'iterations.

x_num = x_mid                        Affecte la valeur finale du point moyen a x_num, qui est la solution
     approximative de l'equation.

-----------------------------

 Affichage de l'equation
print(f"\nL'equation est {a}x + {b} = {c}x + {d}.")

 Affichage de la solution exacte
print(f"\nLa solution exacte de l'equation est x = {x_ex}")

 Affichage de la solution numerique
print(f"\nLa solution numerique de l'equation est x [U+FFFD] {x_num} au bout de {compteur} iteration(s).")
```