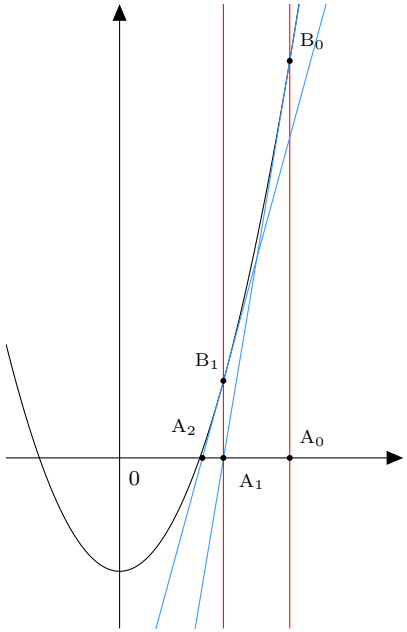
I Méthode de Newton



La méthode de Newton est une méthode numérique pour trouver une solution d'une équation de la forme $f(x) = 0$ d'inconnue x et où f est une **fonction**
Le principe est rappelé sur la figure : partie d'un point d'abscisse b, tracer la tangente au point de la courbe d'abscisse a, calculer le point d'intersection avec l'axe (Ox) et recommencer avec la nouvelle abscisse.

Il s'agit, d'après le cours de calculer les termes de la suite $\begin{cases} x_0 = b \\ \forall k \in \mathbb{N}\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \end{cases}$

Exercice 1
Créer une fonction **newton(f, fp, a)** qui prend deux fonctions f et sa dérivée fp ainsi qu'un réel b comme paramètre et retourne le 20eme terme de cette suite.
On pourra dans un deuxième temps passer le rang du terme qui nous intéresse comme paramètre.

Test Tester votre fonction avec la fonction $f : x \mapsto x^2 - 3$. Quel est le nombre réel dont on vient d'obtenir une valeur approchée ?

Exercice 2
On prouve en cours de mathématiques que $x^n - 2(1-x) = 0$ possède une unique solution dans $[0, 1]$. On note a_n cette solution. Trouver numériquement la limite de (a_n). On pourra calculer des termes pour n de plus en plus grand.

Exercice 3 (Bonus)
Créer une fonction **newton_graphique(f, fp, a, b, n)** qui trace la courbe représentative de f ainsi que les n premières tangentes, comme sur le schéma précédent, entre les points d'abscisses a et b.

II Fonctions python

Les méthodes vues en cours sont très classiques et déjà implémentées en python. Dans cette partie nous allons utiliser le module **scipy.optimize**

```
import scipy.optimize as sco
```

nous permettra d'utiliser les fonctions de ce module sous la forme *sco.fonction*

Arguments optionnels : Nous utiliserons des fonctions avancées dont la signature (le nombre et le type des arguments) comporte souvent des arguments par mot-clé sous la forme *arg=valeur_par_defaut*.
Ces arguments sont toujours optionnels et on pourra donc les ignorer dans une première approche.

Exercice 4
Nous souhaitons dans cet exercice obtenir une valeur approchée de $\phi = \frac{1+\sqrt{5}}{2}$ qui est la racine positive de l'équation $x^2 - x - 1 = 0$.

1. Utiliser la fonction *newton(f, a)* pour donner une approximation de ϕ. On pourra taper

   ```
   help(sco.newton)
   ```

   pour obtenir la signification des arguments f et a.
2. En utilisant la forme *newton(f, a, maxiter=15)* trouver la valeur minimale de *maxiter* qui ne renvoie pas d'erreur.
3. Même exercice en utilisant la forme *newton(f, a, fprime=fp, maxiter=15)* où fp doit être la fonction dérivée de f.
4. Même question en utilisant la fonction *brentq(f, a, b, maxiter=15)*.

III Application pratique : ouverture d'un portail

L'objectif est d'établir de manière approchée la loi d'ouverture d'un portail, ou encore de construire la courbe représentative de θ_v (angle d'ouverture) en fonction de θ_m (angle du moto-réducteur en entrée).
Pour cela on dispose de l'équation

$$\mathrm{A}(\theta_m)\cos(\theta_v) + \mathrm{B}(\theta_m)\sin(\theta_v) + \mathrm{C}(\theta_m) = 0$$

Les fonctions $\mathrm{A}, \mathrm{B}, \mathrm{C}$ sont définies dans le document annexe.

Exercice 5
Définir en python les 3 fonction A, B, C.

Résolution Le principe de résolution est le suivant : pour 200 valeurs de θ_m dans l'intervalle $[14, 130]$ (attention ce sont des degrés...) que l'on fixe une par une, on calcule de manière approchée la solution θ_v à l'équation précédente et on stocke sa valeur dans une liste. Il ne reste qu'à tracer.

Exercice 6
Implémenter ceci à l'aide d'une fonction qui retourne la liste des θ_v. Pour calculer une valeur approchée de θ_v étant donné θ_m, on utilisera la fonction **brentq** du module **scipy.optimize**, qui est plus stable que la méthode de Newton dans notre cas pratique.