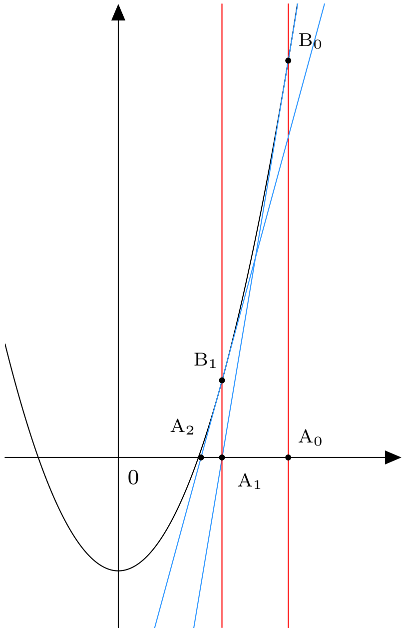
I Méthode de Newton



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 $(\mathrm{O}x)$ 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 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 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 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 Méthode d'Euler

On souhaite utiliser la méthode d'Euler pour résoudre numériquement l'équation $y' = -\mathrm{A}y^2 + \mathrm{B}$ où $\mathrm{A} = 10^{-2}$ et $\mathrm{B} = 10$.

- on découpe l'intervalle $[a, b]$ de résolution en n parties et on pose $h = \frac{b-a}{n}$. En posant $t_k = t_0 + kh$, on a alors $h = t_1 - t_0$ et k prend toutes les valeurs entre 0 et n.
- les valeurs de $y(t_0+kh)$ sont notées y_k et définies par $y_{k+1} = y_k + h \times (-\mathrm{A}y_k^2 + \mathrm{B})$, ce qui correspond exactement à considérer que la courbe de y est égale à sa tangente en t_k sur l'intervalle $[y_k, y_{k+1}]$.

Exercice Retrouver grâce à **numpy.linspace** comment obtenir les $n+1$ valeurs de t_k (avec $t_0 = a, t_n = b$).

Exercice Créer une fonction **euler(y0, T)** qui prend comme paramètres :

- T est la liste $[t_0, \ldots, t_n]$. Ainsi T[0] est la valeur du temps t_0 qui définit notre condition initiale
- $y0$ est la valeur de $y(t_0)$ (l'autre partie de la condition initiale).

La valeur de retour doit être $[y_0, \ldots, y_n]$ la liste des valeurs de y pour chaque temps de T.

Test Tracer la courbe obtenue pour des valeurs de n égales à $10, 50, 100$ sur l'intervalle de $[0, 10]$

Laissons faire les pros Il existe une fonction dans la bibliothèque *scipy.integrate* qui retourne exactement ce que l'on vient de calculer.

```
import matplotlib.pyplot as plt
import scipy.integrate as sci

def f(y, t):
        return -A*y**2 + B  # il faut définir A et B

y0, a, b = 1, 0, 10
# T est le même que pour Euler
Y = sci.odeint(f, y0, T)
# équation différentielle y'(t) = f(y(t), t)
plt.plot(T, Y, linewidth=2, color='black')
```

```
plt.show()
```

Comparer le tracé obtenu. Faire augmenter n pour la méthode d'Euler pour s'approcher du tracé python.

Que venons nous de tracer ? Il s'agit d'une équation différentielle qui peut modéliser l'évolution de la vitesse d'un corps en chute libre soumis à des frottements non linéaires, un modèle plus proche de la réalité que celui que l'on sait résoudre analytiquement.

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 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 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.
Rappel : taper

```
help(fonction)
```

dans la console affiche l'aide pour cette fonction.