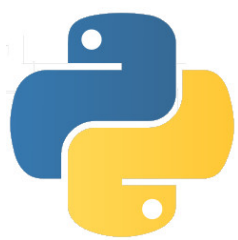
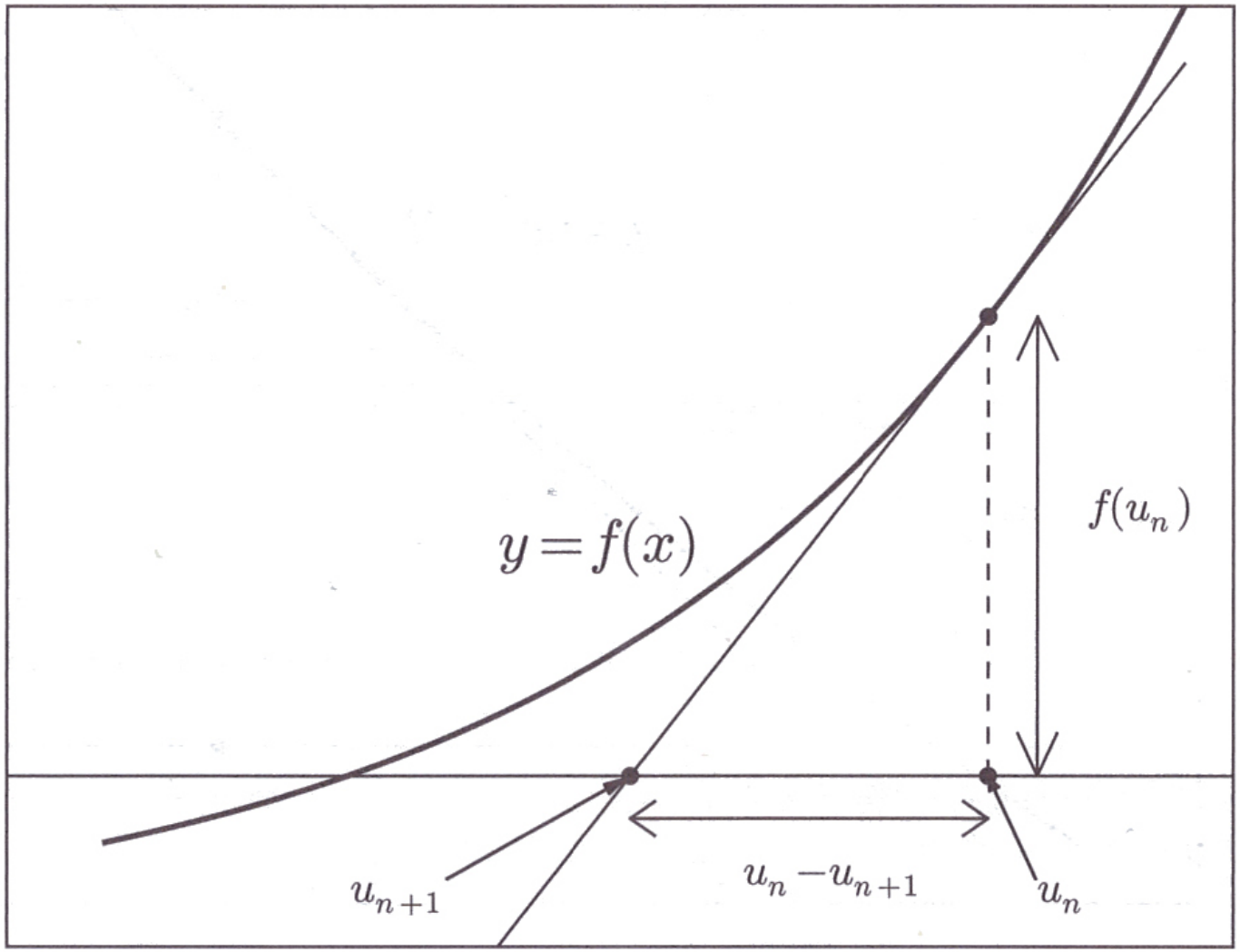
Méthode de Newton

Présentation de la méthode Newton

- Méthode de calcul d'une solution approchée d'une équation $f(x) = 0$
- Aussi appelée « Méthode de la tangente »
- Nécessite que f soit continue et dérivable.
- Nécessite de connaitre l'expression de f mais aussi de f'

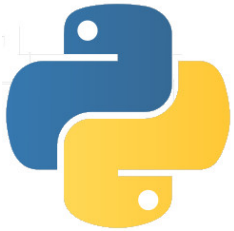




Écriture en Python


```
def newton(f,Df,x0,e):
    """

    Entrees:
    f
    Df
    e
    Sortie:
    v
    """
    u=x0
    v=u-f(u)/Df(u)
    while abs(v-u)>e:
        u,v=v,v-f(v)/Df(v)
    return v
```



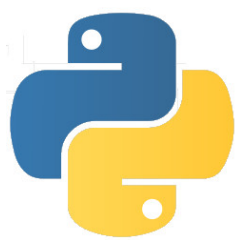
Analyse de l'algorithme

⊖ Méthode plus contraignante que la méthode dichotomique

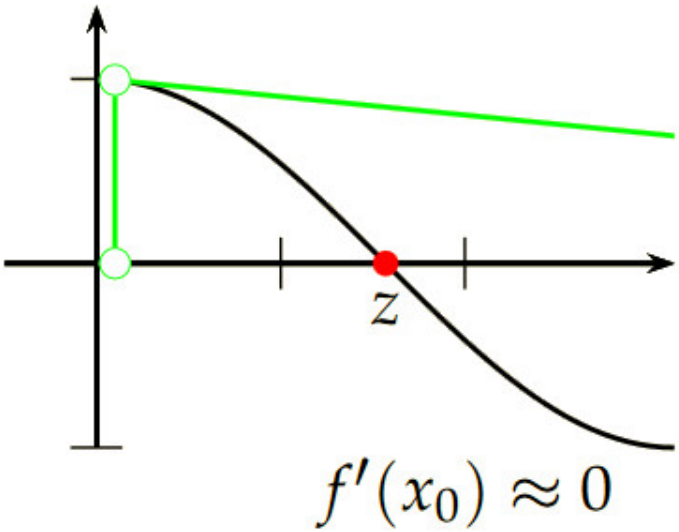
- Il faut connaitre et f et f'

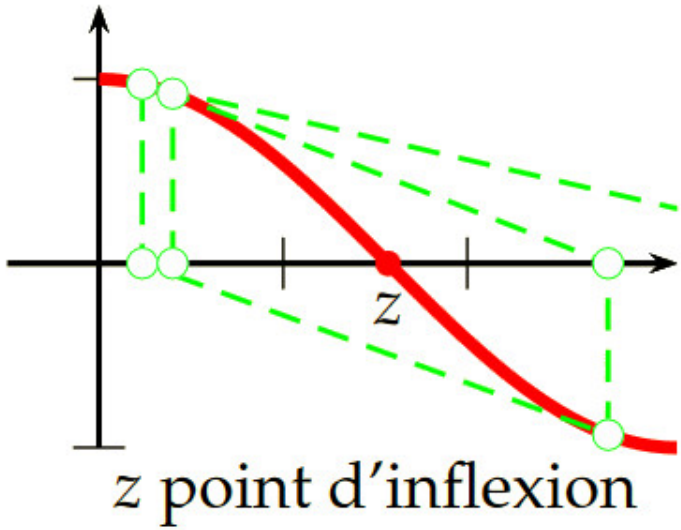
Remarque : La connaissance n'est en fait pas strictement nécessaire, un calcul de taux d'accroissement peut suffire.

```
def newtonTaux(f,x0,e,h):
    u=x0
    Df=(f(u+h)-f(u))/h
    v=u-f(u)/Df
    while abs(v-u)>e:
        u=v
        Df=(f(u+h)-f(u))/h
        v=v-f(v)/Df
    return v
```



- Nécessite d'avoir déjà une idée de la position de la solution (identifier préalablement un intervalle $[a, b]$ où $f(a)f(b) < 0$)
- Même si la fonction est continue, monotone et s'annule sur un intervalle préalablement identifié, la convergence de l'algorithme n'est pas garantie





⊕ Quand ça converge...

On note

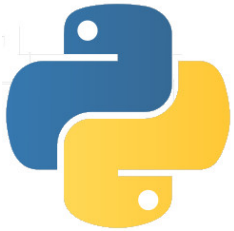
- z la solution exacte de l'équation $f(x) = 0$,
- x_n l'estimation de z à la n^e étape,
- $r_n = x_n - z$ l'erreur à la n^e étape.

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

DL 2 autour de z :

$$f(z) = f(x_n) + (z - x_n) f'(x_n) + \frac{1}{2} f''(x_n)(z - x_n)^2 + o((z - x_n)^2)$$

$$0 = f(x_n) + r_n f'(x_n) + \frac{1}{2} f''(x_n) r_n^2 + o(r_n^2)$$



$$r_n f'(x_n) - f(x_n) = \frac{1}{2} f''(x_n) r_n^2 + o(r_n^2)$$

Donc

$$r_{n+1} = \frac{f''(x_n)}{2f'(x_n)} r_n^2 + o(r_n^2)$$

Si $\left| \frac{f''(x_n)}{2f'(x_n)} \right| < 1$ au voisinage de z alors la convergence est quadratique :

Si $r_n \sim 1 \times 10^{-2} \leftarrow r_{n+1} \sim 1 \times 10^{-4} \leftarrow r_{n+2} \sim 1 \times 10^{-8} \leftarrow r_{n+3} \sim 1 \times 10^{-16}$...

Il y a doublement du nombre de chiffres significatifs à chaque itération !

...ça converge très vite ! !

Rq : La méthode de Newton est déjà programmée dans Python : **scipy.optimize.newton**