# TP 3
## Algorithme de descente de gradient

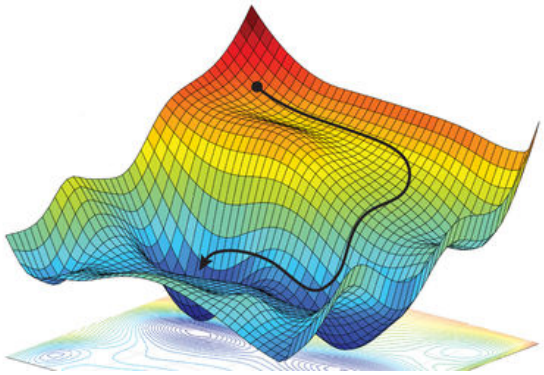
## 1. Rappels théoriques

1.1. **Ce qu'indique le vecteur gradient.** Soit $f : (x, y) \longmapsto f(x, y)$ une application de $\mathbb{R}^2$ dans $\mathbb{R}$ supposée de classe $\mathcal{C}^1$. La fonction $f$ croît le plus vite le long des **lignes de gradient**, c'est-à-dire le long des courbes paramétrées

$$t \longmapsto (x(t), y(t))$$

telles que pour tout $t$ $(x'(t), y'(t)) = \Big(\frac{\partial f}{\partial x}(x(t), y(t)), \frac{\partial f}{\partial y}(x(t), y(t))\Big)$, *i.e.* telles que $v'(t) = \nabla f\big(x(t), y(t)\big)$. De même, la direction de plus rapide décroissance de la fonction sera donnée par $-\nabla f$.
*Voir le cours, paragraphe 4.2.3.*

1.2. **Application : l'algorithme de descente du gradient.** Nous allons utiliser le résultat précédent pour construire un algorithme itératif permettant d'approcher le minimum d'une fonction de une ou plusieurs variables. L'idée est très simple : pour atteindre un minimum il faut que la fonction décroisse, et, pour l'atteindre le plus vite possible, nous allons utiliser l'information portée par le gradient. Pour une fonction $f : \mathbb{R}^2 \to \mathbb{R}$, on espère donc suivre le chemin suivant sur son graphe :



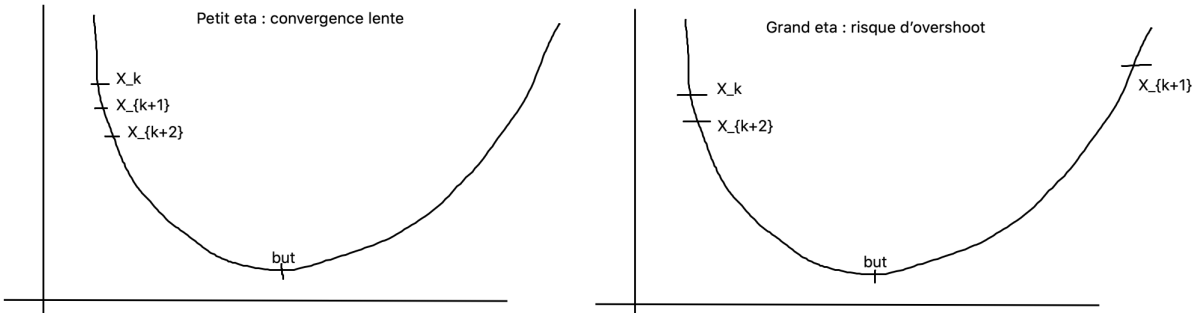
**Algorithme :** on cherche le minimum de $f : \mathbb{R}^n \to \mathbb{R}$ :

(1) Initialiser avec $X_0 \in \mathbb{R}^n$ (au hasard !)
(2) Répéter

$$X_{n+1} = X_n - \eta \nabla f(X_n)$$

(3) jusqu'à convergence

Cela a le mérite d'être très simple, et, en terme de coût de calcul, ce qui est très simple est très efficace. Dans l'étape (2), on notera que le $-$ vient du fait qu'on cherche à minimiser : c'est donc $-\nabla f$ qui indique la direction optimale de descente (on voudrait maximiser, on suivrait $+\nabla f$). Le paramètre positif $\eta$ n'est pas déterminé par la théorie mathématique : c'est juste le paramètre qui précise la taille des "sauts" que fera l'algorithme à chaque itération. Il conviendra de le choisir "intelligemment" : ni trop grand pour éviter de passer "au-dessus" d'un minimum sans le voir, ni trop petit pour éviter que l'algorithme converge trop lentement.




## 2. Le TP

### 2.1. **Code et premières expérimentations.**

2.1.1. *Coder une méthode de descente pour la minimisation d'une fonction d'une variable réelle.* Faire un code reprenant l'algorithme (1)-(3) dans le cas d'une fonction d'une variable. Évidemment il faut penser à mettre un test de convergence (avec un seuil que vous choisirez) pour arrêter l'algorithme itératif. Le tester sur l'exemple suivant:

$$f(x) = x^2 - x + 1$$

$$\nabla f(x) = \frac{\partial f(x)}{\partial x} = f'(x) = 2x - 1$$

Il n'y a qu'un seul paramètre, la dérivée partielle est égale à la dérivée.

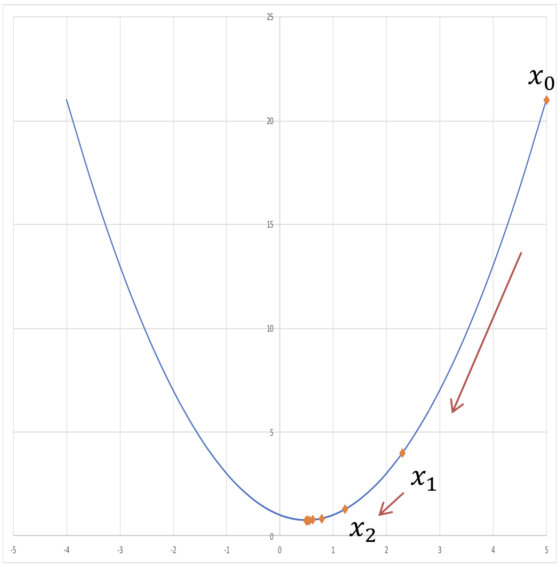
$\eta = 0.3$

| eta | 0.3 |
|---|---|

$x_0 = 5$

| x | f'(x_t) | f(x) |
|---|---|---|
| 5.0000 | | 21.0000 |
| 2.3000 | 9.0000 | 3.9900 |
| 1.2200 | 3.6000 | 1.2684 |
| 0.7880 | 1.4400 | 0.8329 |
| 0.6152 | 0.5760 | 0.7633 |
| 0.5461 | 0.2304 | 0.7521 |
| 0.5184 | 0.0922 | 0.7503 |
| 0.5074 | 0.0369 | 0.7501 |
| 0.5029 | 0.0147 | 0.7500 |
| 0.5012 | 0.0059 | 0.7500 |
| 0.5005 | 0.0024 | 0.7500 |
| 0.5002 | 0.0009 | 0.7500 |
| 0.5001 | 0.0004 | 0.7500 |
| 0.5000 | 0.0002 | 0.7500 |

$$x_{t+1} = x_t - \eta \times \nabla f(x_t)$$




$\ldots x_2 < x_1 < x_0$

Faire varier la valeur de eta ($\eta$) et le point initial $x_0$ : quels effets observe-t-on ?

2.1.2. *Coder une méthode de descente pour la minimisation d'une fonction de deux variables réelles.* Faire un code reprenant l'algorithme (1)-(3) dans le cas d'une fonction de deux variables réelles, $f : \mathbb{R}^2 \to \mathbb{R}$. Le tester avec la fonction

$$f(x, y) = x^2 + 1.8xy + y^2$$

à partir du point initial $X_0 = (x_0, y_0) = (-0.9, 1)$, puis à partir de $X_0 = (x_0, y_0) = (-1, 1)$.
Un commentaire ? (penser à représenter la fonction pour juger de la pertinance des résultats !)
Mêmes questions pour la fonction

$$f(x, y) = \frac{1}{100}(1 - x)^2 + (y - x^2)^2$$

à partir de $X_0 = (x_0, y_0) = (-1, 0)$.

2.2. **Application : régression linéaire.** Étant donné un nuage de points $(x_i, y_i)_{i=1}^n$, la droite des moindres carrés (ou droite de régression linéaire) est la droite d'équation $y = mx + p$ qui minimise la quantité

$$F(m,p) = \sum_{k=1}^{n} (y_k - mx_k - p)^2.$$

Générer des données, par exemple 100 points de coordonnées dans $[-1, 1]$, non équirépartis (d'où la loi normale ci-dessous) :

```
X = np.linspace(-1, 1, 100) + np.random.normal(0, 0.25, 100)
Y = np.linspace(-1, 1, 100) + np.random.normal(0, 0.25, 100)
```

2.2.1. *La solution analytique.* Il est possible de calculer explicitement les coefficients $m$ et $p$ solution (c'est l'objet de l'exercice 6 des Exercices sur le chapitre 4 du cours). Si $\sum_{k=1}^n (x_k - \bar{x})^2 \neq 0$, on trouve les formules suivantes :

$$m = \frac{\sum_{k=1}^n (x_k - \bar{x})(y_k - \bar{y})}{\sum_{k=1}^n (x_k - \bar{x})^2}$$

où on a noté $\bar{x}$ et $\bar{y}$ les valeurs moyennes respectives de $(x_i)_{i=1,\dots,n}$ et $(y_i)_{i=1,\dots,n}$, et

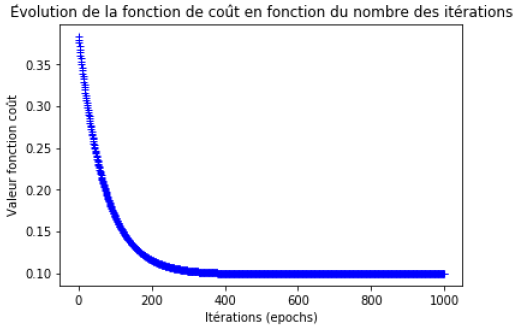
$$p = \frac{1}{n} \sum_{k=1}^{n} (y_k - mx_k).$$

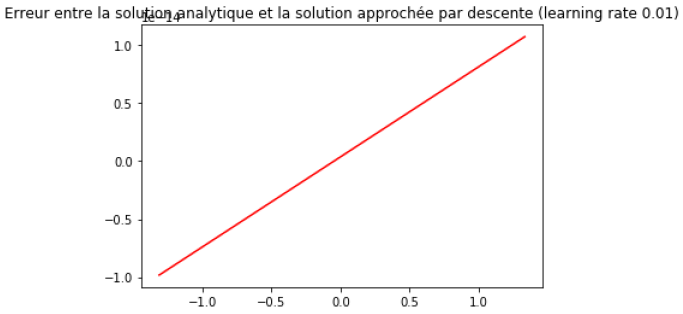
- Afficher le nuage de points et la droite de régression d'équation $y = mx + p$, $m$ et $p$ étant obtenus avec les formules ci-dessus.

2.2.2. *La solution par la méthode de descente de gradient.* Trouver une approximation de $m$ et $p$ obtenue grâce à la méthode de gradient obtenue au paragraphe 2.1.2. On rappelle que la fonction à minimiser est celle qui minimise l'erreur au sens des moindres carrés :

$$(m,p) \mapsto F(m,p) = \sum_{k=1}^{n} (y_k - mx_k - p)^2.$$

Tracer la droite correspondante d'équation $y = m_{app}x + p_{app}$ sur la même figure que le nuage de points et la droite de régression $y = mx + p$. Commentaire ?
Ajouter un affichage de l'erreur entre la droite de régression approchée et la droite de régression en fonction du nombre d'itérations pour rendre plus rigoureuse l'analyse visuelle des résultats.




**Remarque :** L'algorithme de descente de gradient est à la racine de la plupart des algorithmes de deep learning (ce sera l'objet du prochain TP). En IA, on parle d'époque (ou d'epoch en anglais) plutôt que d'itération et de taux d'apprentissage (learning rate) pour le paramètre $\eta$.