Université de Montréal
Département d'informatique et de recherche opérationnelle

IFT3395/6390
Fondements de l'apprentissage machine

Quelques principes et techniques de base d'optimisation

Professeur: Pascal Vincent

LISA Laboratoire d'Informatique des Systèmes d'Apprentissage

Optimisation?

- La phase d'entraînement d'un algorithme d'apprentissage est souvent une optimisation
- Il s'agit de trouver une valeur des paramètres θ d'une fonction f_θ qui minimisent (ou maximisent) un objectif $J(\theta)$
- ex d'objectif: le risque empirique (ou «l'erreur moyenne») sur l'ensemble d'entraînement:

$$J(\theta) = \hat{R}(f_\theta, D_n)$$

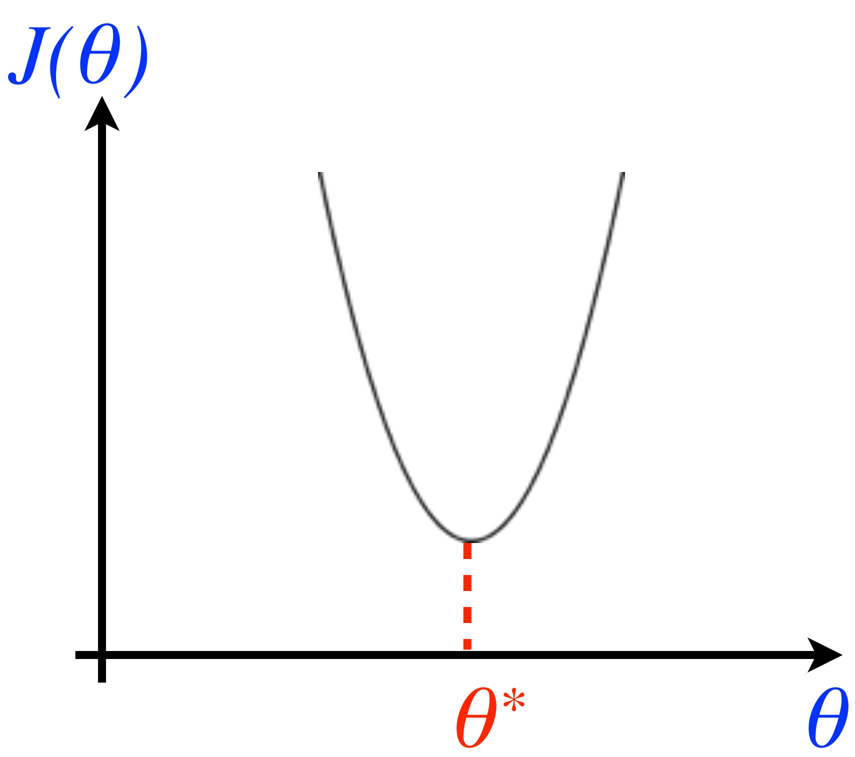
Le problème d'optimisation

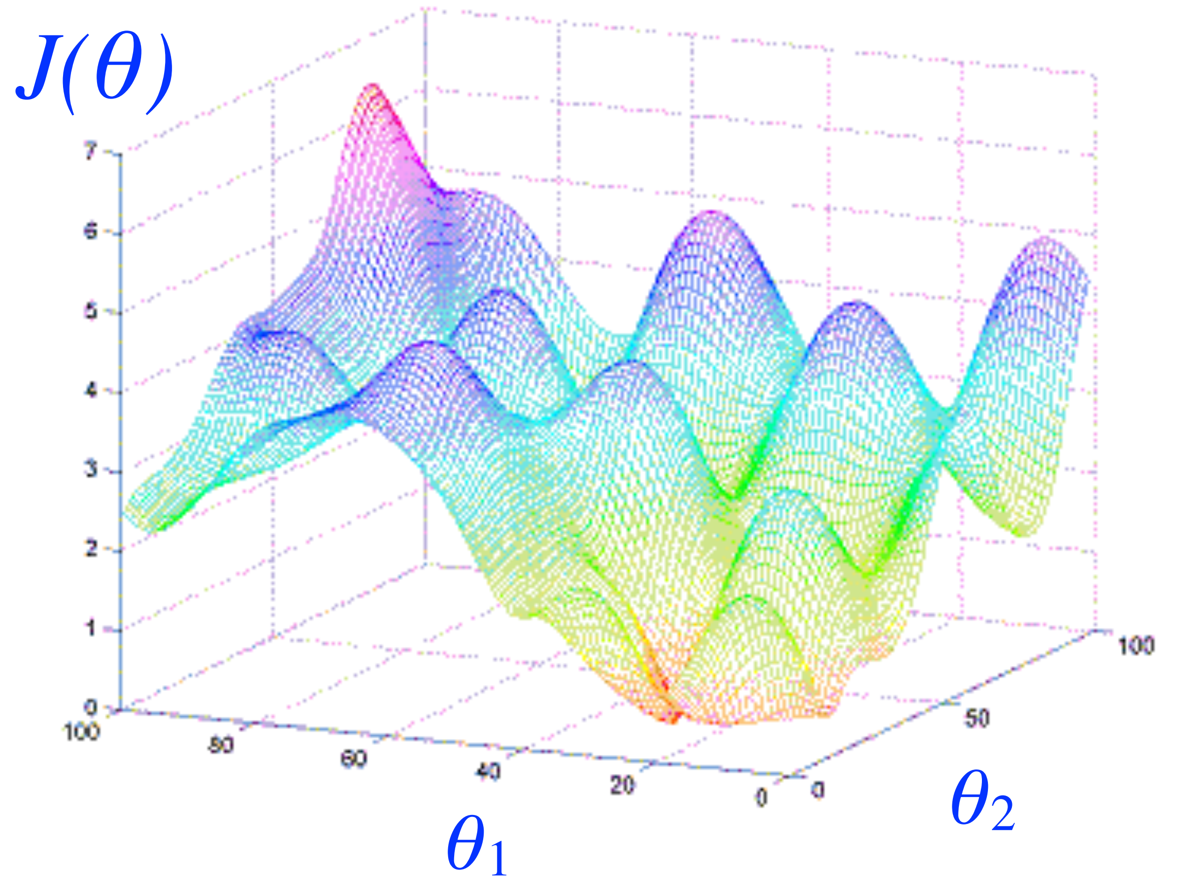
$$\theta^* = \arg\min_{\theta} J(\theta)$$

la valeur optimale
des paramètres

Comment la trouver ???

Paysage de coût ou d'erreur:



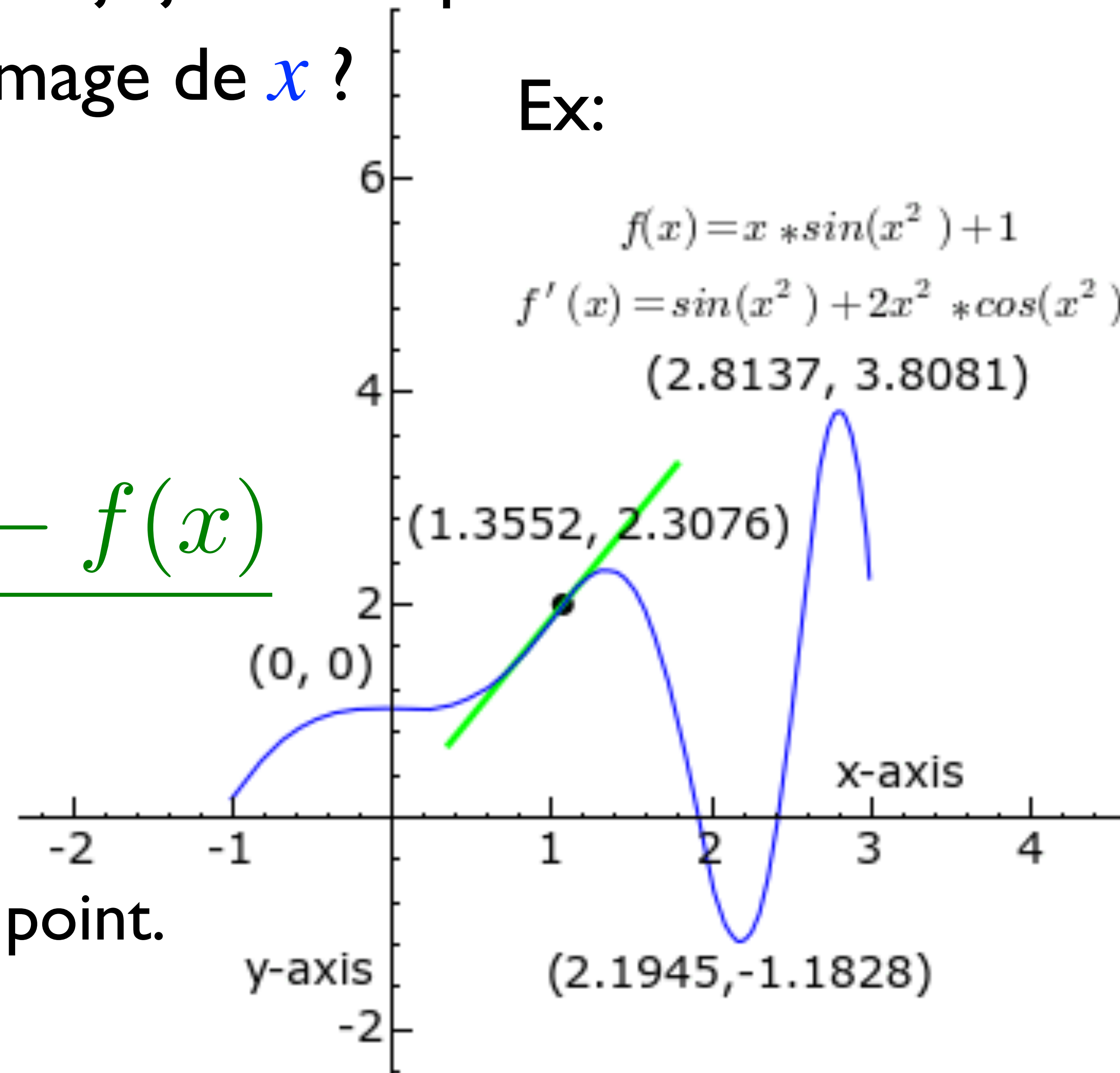


Rappel: dérivée

- Mesure la **sensibilité** d'une fonction f à un changement de son entrée (en un point x).

- Soit $f : \mathbb{R} \to \mathbb{R}$ En un point x si j'ajoute un petit ε à x, de combien de ε cela fait bouger l'image de x ?

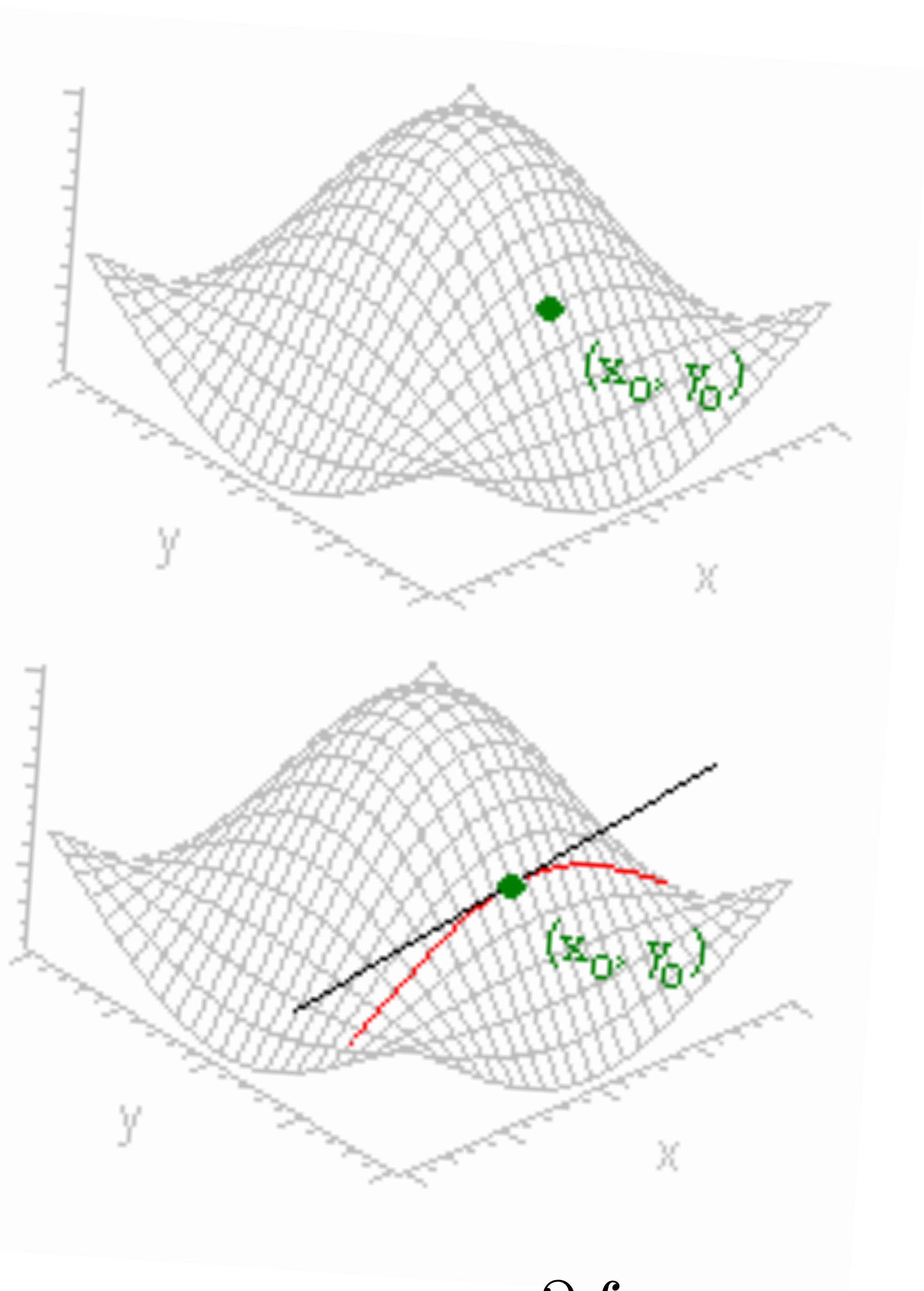
Ex:



$$
\begin{aligned}
f'(x) &= \frac{\partial f}{\partial x}(x) \\
&= \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}
\end{aligned}
$$

- Géométriquement: la **pente** (inclinaison) de la courbe en chaque point.

Dérivée partielle



- Quand une fonction dépend de plusieurs entrées ou paramètres: $f : \mathbb{R}^m \to \mathbb{R}$
- On peut l'évaluer en un certain point x: $f(x) = f(x_1, \ldots, x_m)$
- Sa valeur est plus ou moins sensible au changement de *chaque* paramètre/entrée séparément (considérant les autres fixes)

$$f'_{x_k}(x) = \frac{\partial f}{\partial x_k}(x)$$ dérivée partielle de f par rapport à x_k, évaluée au point x

$$= \lim_{\epsilon \to 0} \frac{f(x_1, \ldots, x_k + \epsilon, \ldots, x_m) - f(x_1, \ldots, x_k, \ldots, x_m)}{\epsilon}$$

Gradient

- Le gradient est le vecteur de ces dérivées partielles:

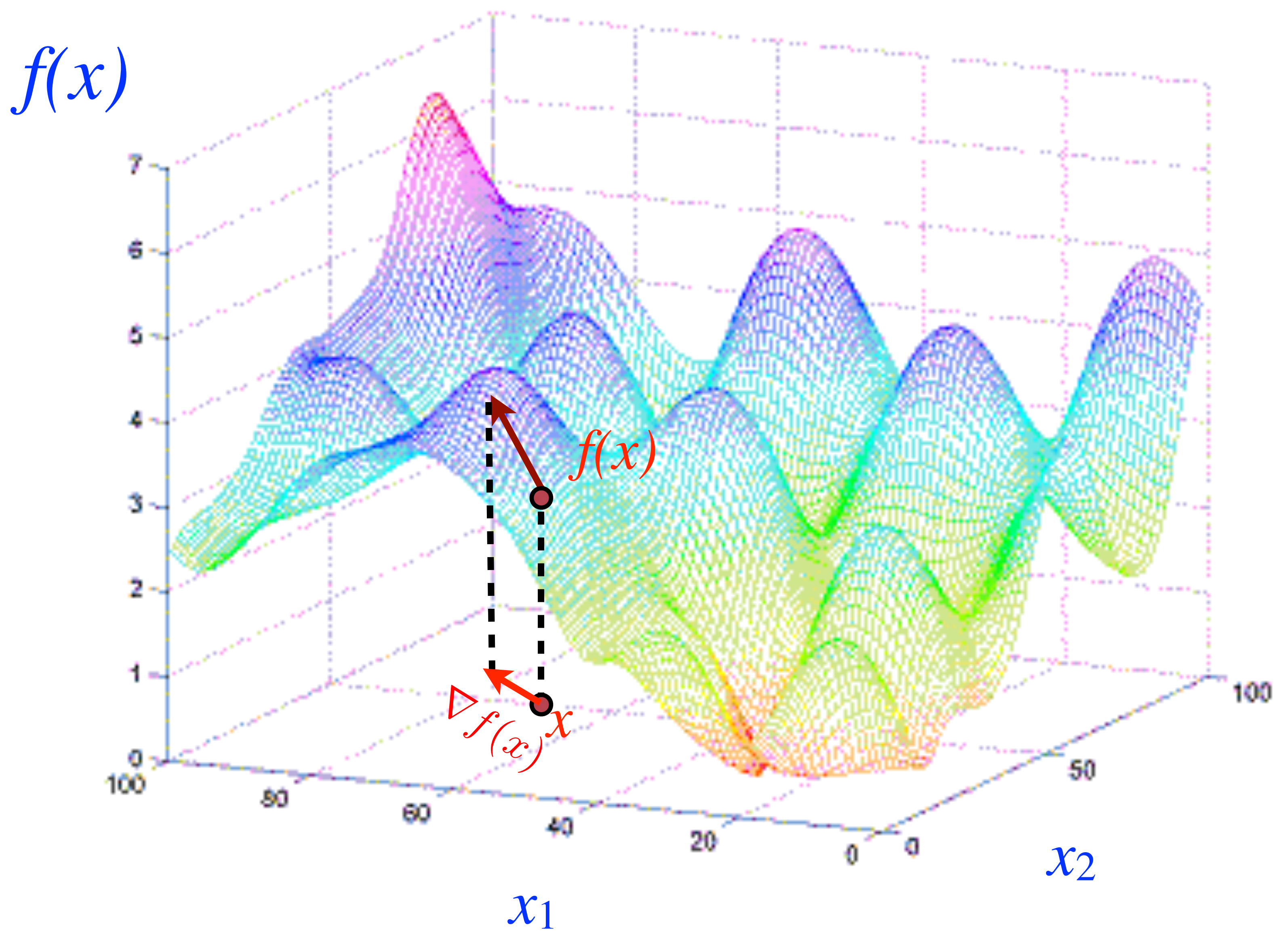
$$\nabla f(x) = \frac{\partial f}{\partial x}(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{pmatrix}(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_m}(x) \end{pmatrix}$$

si $f : \mathbb{R}^m \to \mathbb{R}$

alors $\nabla f : \mathbb{R}^m \to \mathbb{R}^m$

Gradient: géométriquement

- Le gradient pointe dans la direction de plus forte pente (ascendante)



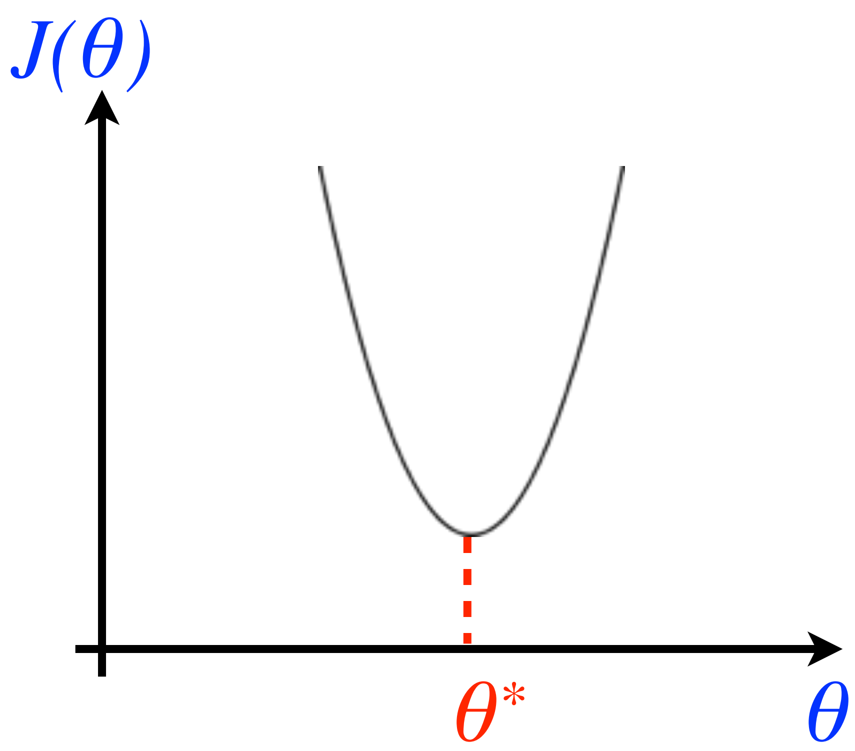
Le problème d'optimisation

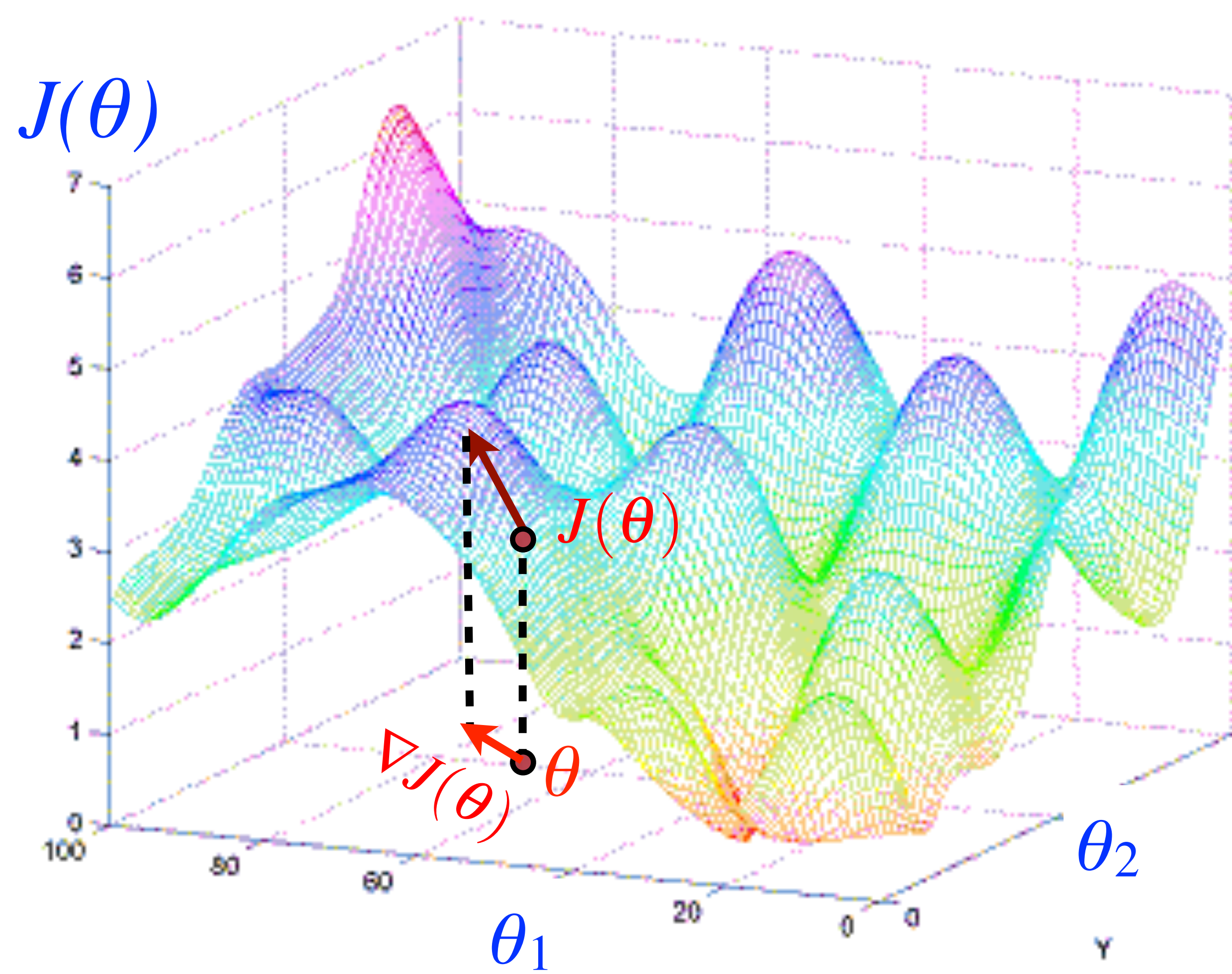
$$\theta^* = \arg\min_{\theta} J(\theta)$$

la valeur optimale
des paramètres

Comment la
trouver ???

Paysage de coût ou d'erreur:





Gradient

- Soit $\theta=(\theta_1, \ldots, \theta_m)$ un vecteur de paramètres.
- Soit $J(\theta)$ une fonction scalaire: l'objectif que l'on veut minimiser.
- Le gradient est le vecteur des dérivées partielles:

$$\nabla J(\theta) = \frac{\partial J}{\partial \theta}(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_m} \end{pmatrix}(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_1}(\theta) \\ \frac{\partial J}{\partial \theta_2}(\theta) \\ \vdots \\ \frac{\partial J}{\partial \theta_m}(\theta) \end{pmatrix}$$

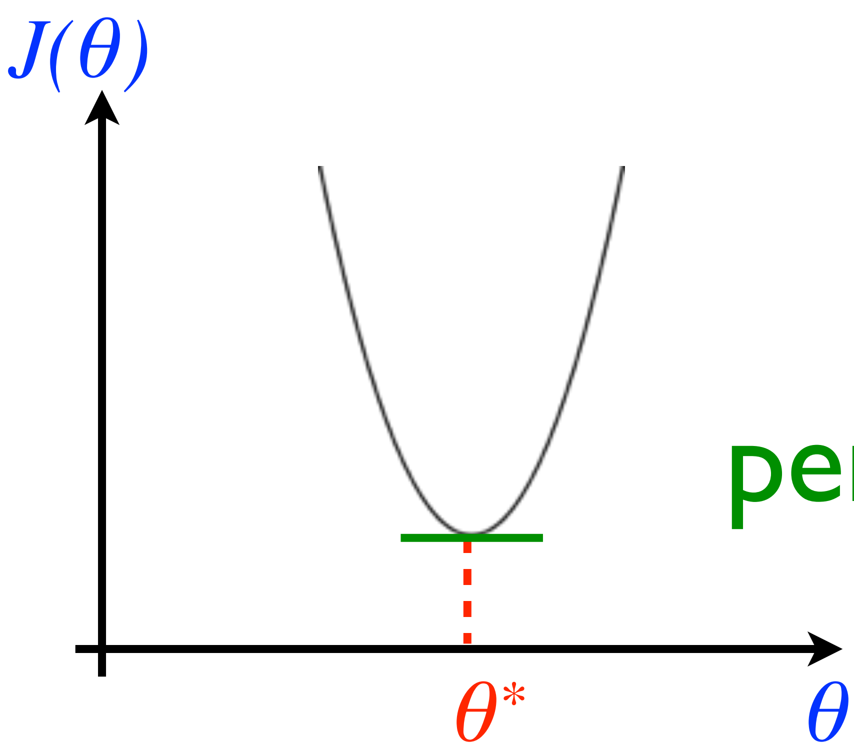
Propriété essentielle

À l'optimum, le **gradient** est nul:

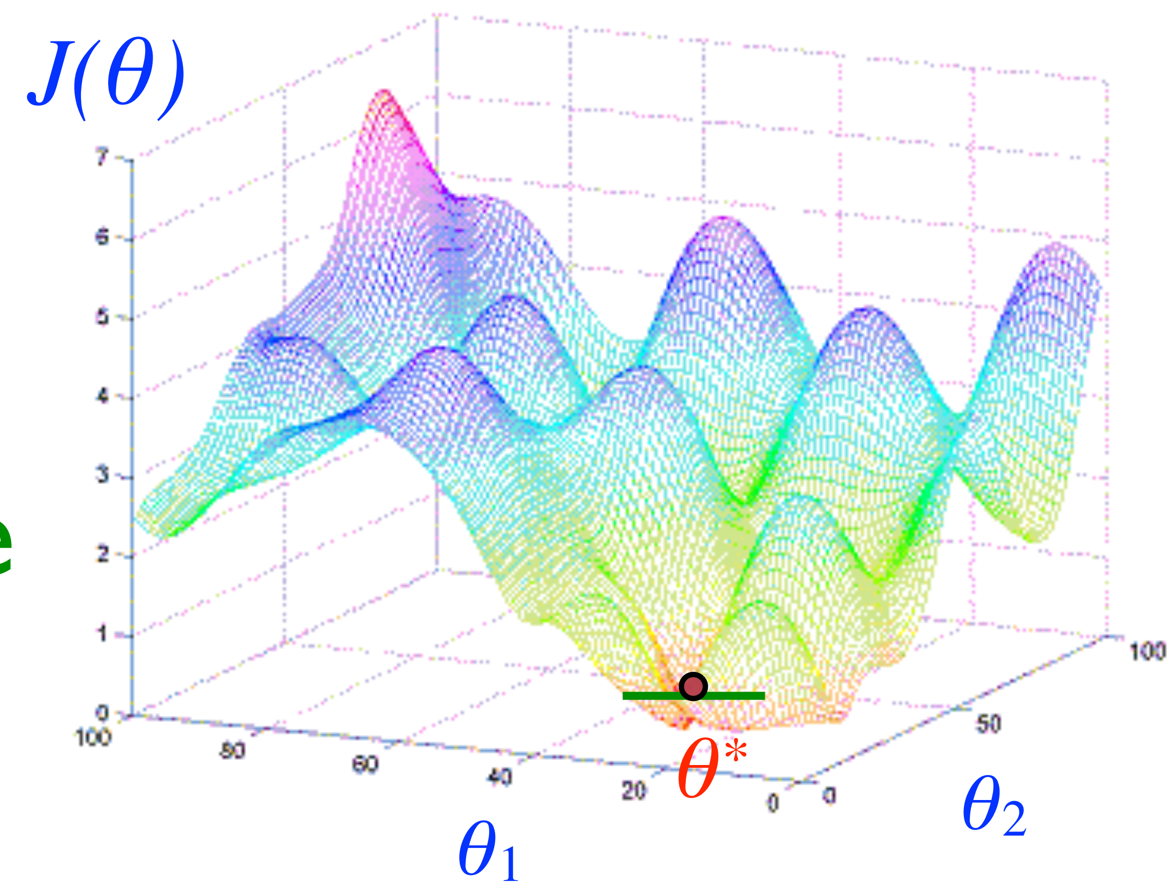
la «pente»

$$\frac{\partial J}{\partial \theta}(\theta^*) = 0$$

Paysage de coût ou d'erreur:



pente nulle



Solution analytique

- **Parfois** on peut résoudre analytiquement l'équation pour trouver le θ optimal:

$$\frac{\partial J}{\partial \theta} = 0 \qquad \begin{cases} \frac{\partial J}{\partial \theta_1} = 0 \\ \frac{\partial J}{\partial \theta_2} = 0 \\ \vdots \\ \frac{\partial J}{\partial \theta_m} = 0 \end{cases}$$

...

θ = ...

- Ex d'optimisations donnant lieu à une solution analytique:
  - maximum de vraisemblance pour trouver les paramètres d'une densité Gaussienne
  - régression linéaire (ou ridge)

Algorithmes de recherche

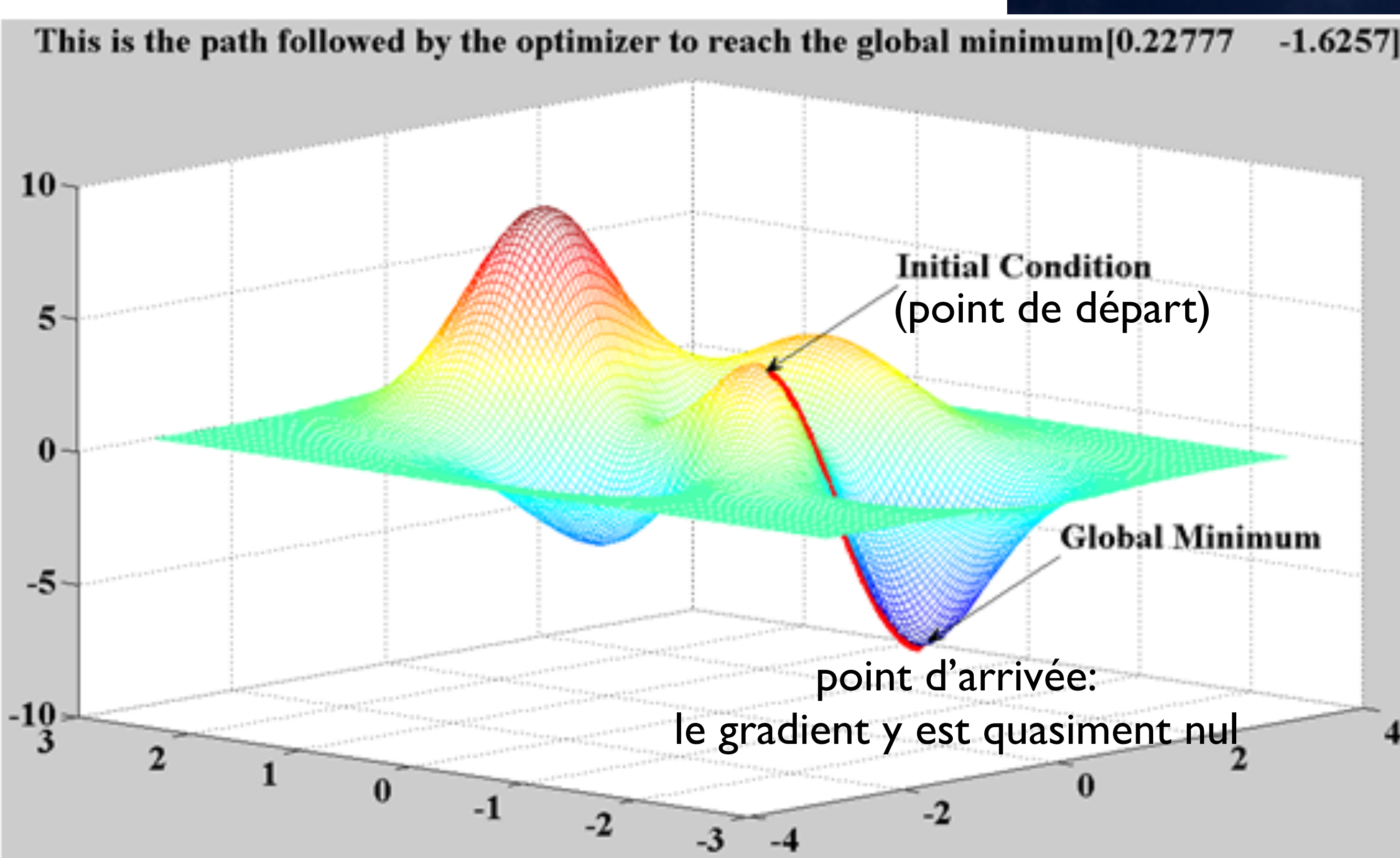
- **Très souvent** on ne peut pas résoudre analytiquement l'équation $\frac{\partial J}{\partial \theta} = 0$
- On va alors utiliser un **algorithme itératif** pour rechercher une solution optimale.
- Solution 1: **recherche exhaustive**
essayer toutes les valeurs possibles de θ et garder la meilleure...
=> IMPRATICABLE POUR θ DE HAUTE DIMENSION (m>4).
- Solution 2: **descente de gradient**

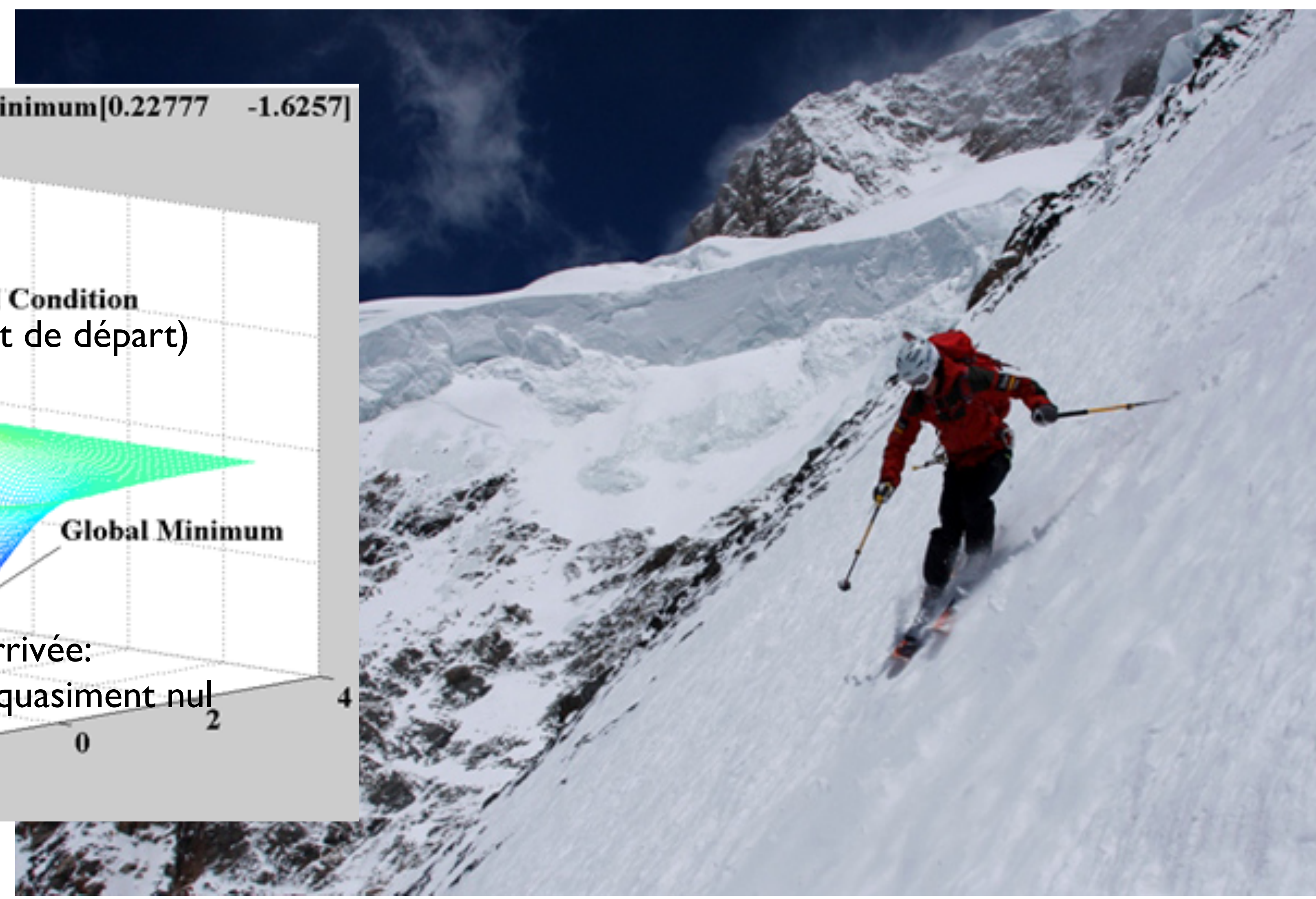
Descente de gradient

- Initialiser θ aléatoirement (selon une heuristique appropriée)
- Répéter, jusqu'à convergence:

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$$

on modifie θ en effectuant un pas dans la direction opposée au vecteur de gradient (la direction de la plus grande pente).





Descente de gradient

taux d'apprentissage

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$$

- η est un réel positif nommé *«taux d'apprentissage»* *«learning rate»* ou parfois *«pas de gradient»*.
  Il contrôle l'échelle de taille des déplacements dans l'espace.

- C'est un hyper-paramètre crucial de la procédure d'optimisation (il faut bien le choisir).

- Souvent on le fait lentement décroître à mesure des itérations.

  Ex: soit t le numéro de l'itération, on peut utiliser un taux d'apprentissage $\eta(t) = \frac{\lambda}{t_0 + t}$

  Dans ce cas t_0 et λ sont des hyper-paramètres à choisir judicieusement.

Descente de gradient
critère d'arrêt

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$$

- On peut arrêter la descente de gradient lorsque θ ne bouge presque plus
- Quand la norme du gradient devient plus petite qu'un petit seuil

$$\left\| \frac{\partial J}{\partial \theta}(\theta) \right\| < \epsilon$$

- On peut aussi, pour d'autres raisons (ex: meilleure généralisation) s'arrêter avant d'avoir atteint un optimum => technique d'arrêt prématuré *(basé sur un autre critère, telle que l'erreur sur un ensemble de validation: nous y reviendrons)*

Descente de gradient
quelle solution atteint-on?

On converge vers un point où
le **gradient** est (presque) nul:

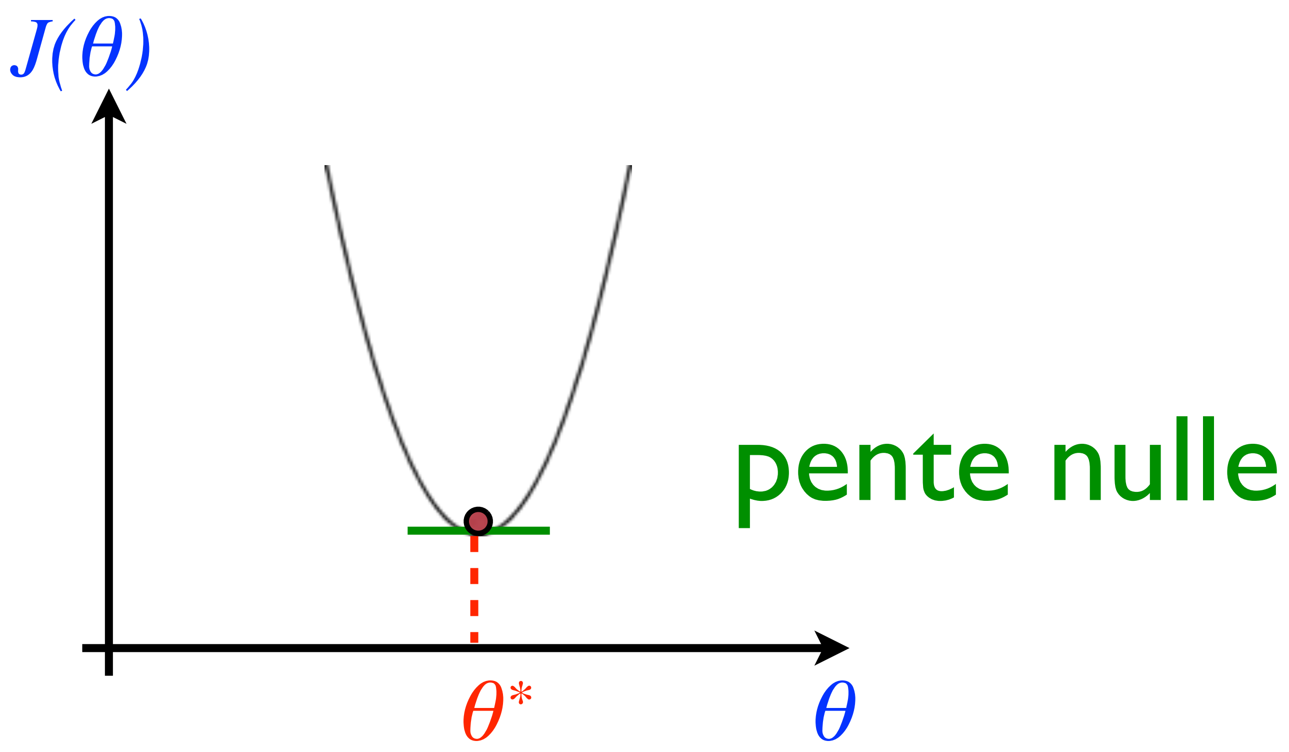
$$\frac{\partial J}{\partial \theta}(\theta) \approx 0$$

Descente de gradient

objectif convexe

On converge vers un point où le **gradient** est (presque) nul: $$\frac{\partial J}{\partial \theta}(\theta) \approx 0$$

Paysage de coût ou d'erreur:

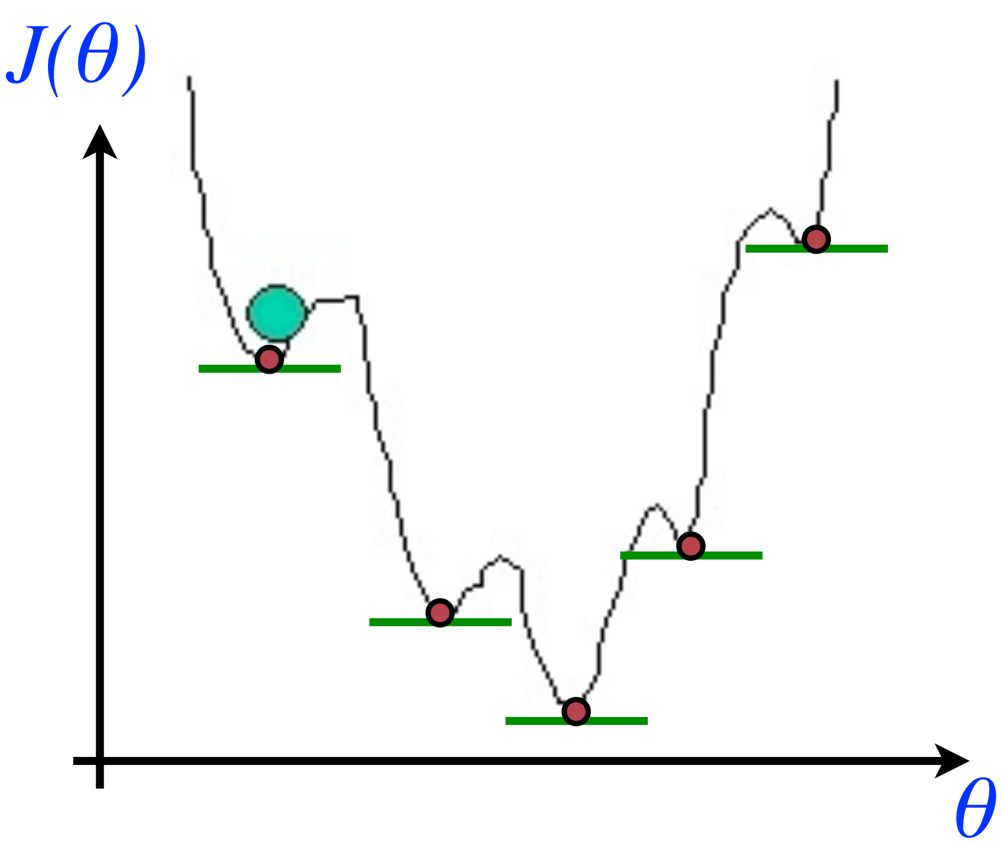


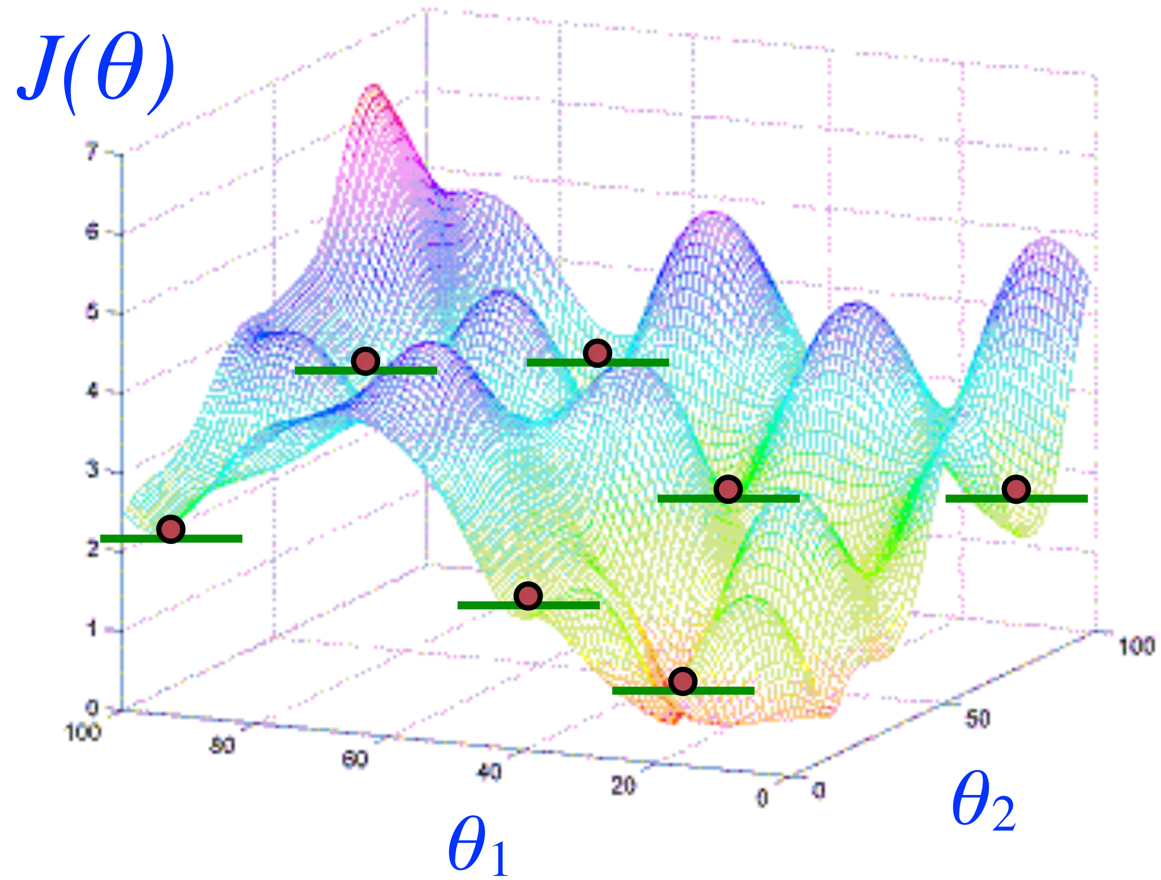
Dans le cas où l'objectif est convexe, on atteint le **minimum global**

Descente de gradient
objectif non-convexe

On converge vers un point où le **gradient** est (presque) nul: $\frac{\partial J}{\partial \theta}(\theta) \approx 0$

Si objectif non convexe il peut y avoir *beaucoup* de tels points: tous les extréma (minima et maxima) *locaux* (et les points de selle).





➠ En pratique on va converger vers **un minimum local** (plutôt que minimum global), qui dépend de notre point de départ.

Objectif à minimiser

typique en apprentissage

- En apprentissage, l'objectif à minimiser est souvent une somme ou une moyenne sur les *n* exemples d'un ensemble d'entraînement $D_n = \{z^{(1)}, \ldots, z^{(n)}\}$ d'une fonction de coût (ou perte) *L*

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(z^{(i)}; \theta)$$

- Remarque: une somme et la moyenne correspondante ont les mêmes minima: les minimiser est équivalent.

Gradient de l'objectif
typique en apprentissage

- Le gradient d'une moyenne est la moyenne des gradients

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(z^{(i)};\boldsymbol{\theta})$$

$$\begin{aligned}\frac{\partial J}{\partial \theta}(\theta) &= \frac{\partial}{\partial \theta}\left(\frac{1}{n}\sum_{i=1}^{n} L(z^{(i)};\theta)\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta} L(z^{(i)};\theta)\end{aligned}$$

Descente de gradient
batch, mini-batch

- La descente de gradient «batch» (par lot) consiste à calculer ce gradient moyen sur tous les n exemples de D_n

$$\nabla(\theta) = \frac{\partial J}{\partial \theta}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \theta} L(z^{(i)};\theta) = \underset{z\in D_n}{\text{moyenne}} \left[\frac{\partial}{\partial \theta} L(z^{(i)};\theta)\right]$$

avant chaque pas de mise à jour des paramètres: $\theta \leftarrow \theta - \eta\nabla(\theta)$

- La descente de gradient par «mini batch» (mini lot) de taille n' consiste à approximer cette moyenne en ne la calculant une moyenne que sur $n'<n$ exemples de D_n

$$\nabla(\theta) = \underset{z\in \text{minibatch}}{\text{moyenne}} \left[\frac{\partial}{\partial \theta} L(z^{(i)};\theta)\right] \approx \frac{\partial J}{\partial \theta}(\theta)$$

(ces n' exemples, différents à chaque fois, constituent le *mini batch*)

Descente de gradient
batch, mini-batch, stochastique, en ligne

Algo:

$$\nabla(\theta) = \underset{z \in \text{minibatch}}{\text{moyenne}} \left[\frac{\partial}{\partial \theta} L(z^{(i)}; \theta) \right] \approx \frac{\partial J}{\partial \theta}(\theta)$$ *Calcul du gradient*

$$\theta \leftarrow \theta - \eta \nabla(\theta)$$ *Mise à jour des paramètres*

- Avant chaque calcul de ∇ les n' exemples du mini-batch pourraient idéalement être tirés aléatoirement parmi D_n.
- Mais pour des raisons d'efficacité, on va souvent plutôt parcourir D_n *séquentiellement* pour obtenir les mini-batchs (d'abord les n' premiers exemples, puis les n' suivants, etc... et on boucle).
- Remarque: en procédant ainsi, avec $n'=n$ on retrouve le cas *batch*.
- Le cas $\boldsymbol{n'=1}$ (on ne visite qu'un exemple avant de faire une mise-à jour des paramètres) correspond à une descente de gradient stochastique ou *en ligne*.

Autres variantes de descente de gradient

Il existe de nombreux algorithmes d'optimisation basés sur la descente de gradient:

- Technique de *momentum*
- Gradient conjugué
- Méthodes de second ordre. Utilisent aussi l'information de «courbure» donnée par les dérivées secondes: la *Hessienne*)
  Ex: méthode de Newton.
- ...

Tous nécessitent des paramètres continus, et le calcul efficace d'un «gradient».

Méthode de Newton

Gradient: vecteur des dérivées premières

$$\nabla_J = \frac{\partial J}{\partial \theta} = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_m} \end{pmatrix}$$

Descente de gradient simple (batch):

$$\theta \leftarrow \theta - \eta \nabla_J(\theta)$$

Méthode de Newton (batch):

$$\theta \leftarrow \theta - \eta H(\theta)^{-1} \nabla_J(\theta)$$

Hessienne: matrice des dérivées secondes

$$H = \frac{\partial^2 J}{\partial \theta^2} = \begin{pmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_m} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_m \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_m \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_m^2} \end{pmatrix}$$

Méthode essentiellement batch.

H^{-1}: coûteux et compliqué à calculer.

Optimisation avec contraintes

- Dans ce que nous avons vu pour l'instant, les paramètres n'étaient pas contraints
- Parfois on veut aussi que les paramètres respectent une ou plusieurs contraintes (ex: être positifs, sommer à 1, ...)
- Problèmes d'optimisation sous contrainte => algorithmes plus complexes.
  Ex: «programmation» linéaire; quadratique; ...