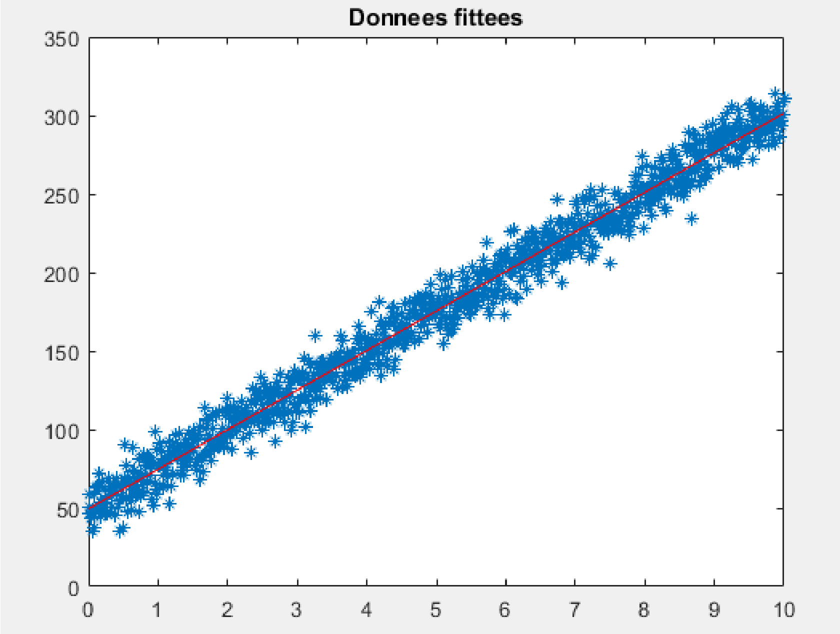
Etude de la convergence d’algorithmes d’optimisation

	A pas constant (p=0,1)	A pas optimal	Newton
Nbre itérations x^2	34	1	1
Résultat x^2	0,0032	0	0
Nbre itérations Rosenbrock	6904	6075	8
Résultat Rosenbrock	(1.0011,1.0022)	(1.0004,1.0009)	(1,1)
Temps de calcul	11.8s	5.7s	0.025225

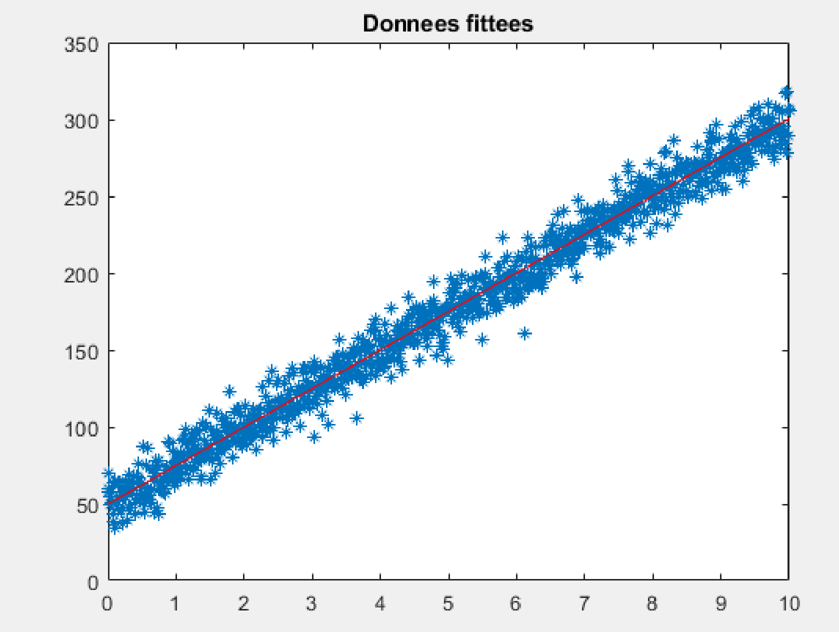
Pour des cas simples, les algorithmes plus complexes sont plus rapides et précis. Cela est dû au pas qui est calculé plutôt que donné. Si l’on prend pas= 0,5, le pas constant obtient les mêmes résultats que les autres algorithmes

Pour Rosenbrock, nous avons choisi un départ à (-1,2) pour tous les algorithmes. Les méthodes utilisant fminbnf et fminsearch obtiennent des résultats similaires. Pour cet exemple, Newton obtient de bien meilleurs résultats. Nous utilisons pour chaque itération la Hessienne et le gradient pour calculer la prochaine itération.

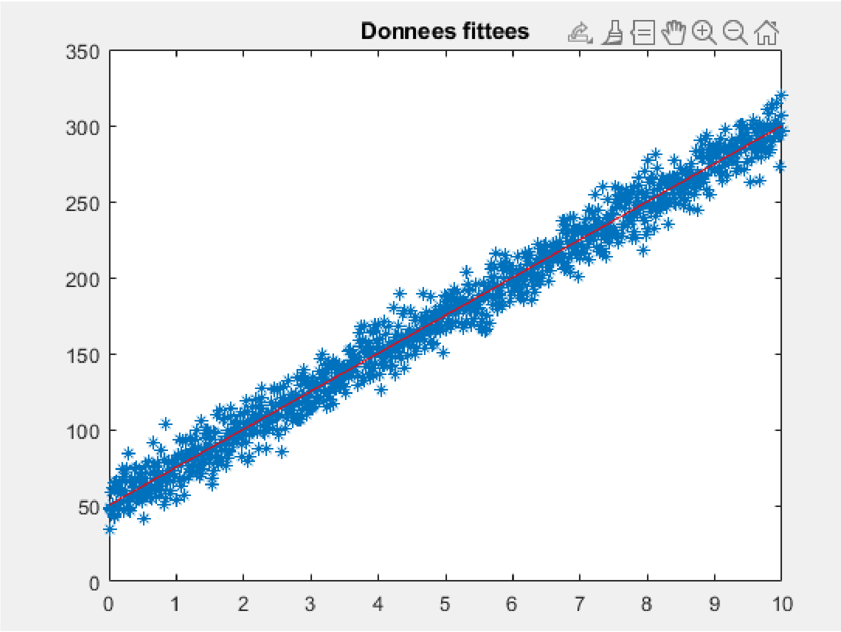
Régression linéaire



Gradient classique (pas= 0.000001; 11000 itérations)
24.9717*x+49.7728



Gradient stochastique (pas = 0.0001 et mu=0.000000001)
25000 itérations
24.9334*x+49.0735



Méthode MiniBatch(pas = 0.001 et mu=0.00001)
200000 itérations
24.9757x+50.1997

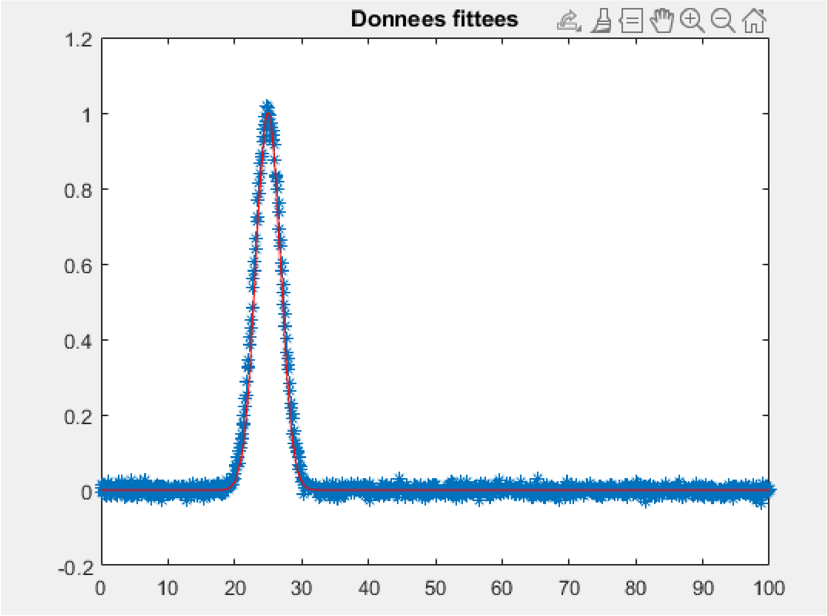
Le gradient stochastique nécessite plus de puissance de calcul, pour les cas simples (linéaire), il ne semble pas adapté.

La méthode MiniBatch va créer un échantillon sur les données (ici 20) et moyenner les gradients.

Régression non linéaire

	Gradient	Stochastique	Mini Batch (N=
Pas	0,0001	10	0,00001
mu	0,0000001	0,00001	0,0001
Itérations	40000 (max)	5317	X
Résultat	24.9952x+ 6.6407	25.0966x +8,0865	X

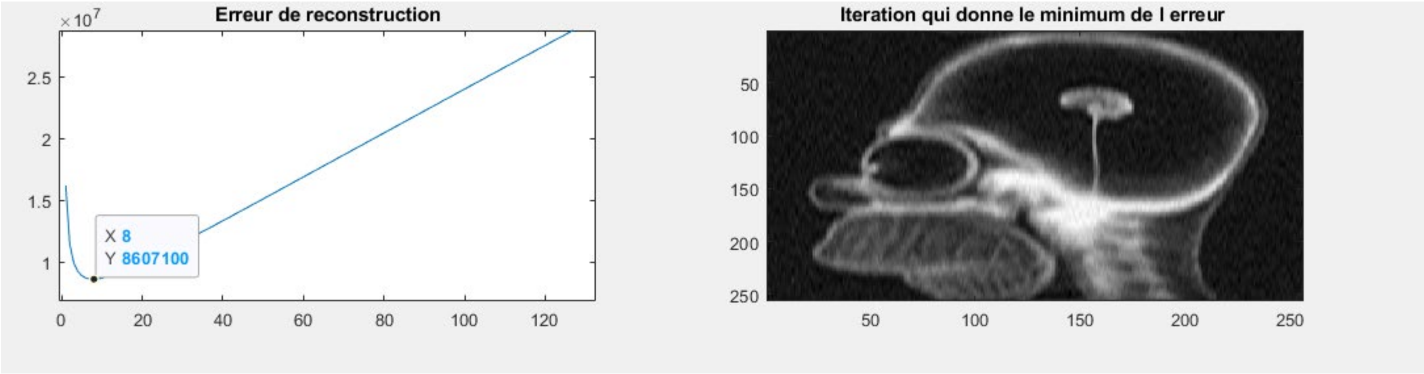
La méthode du gradient classique met trop de temps à converger du fait de la complexité de la courbe (non linéaire). La méthode stochastique semble donc plus adaptée

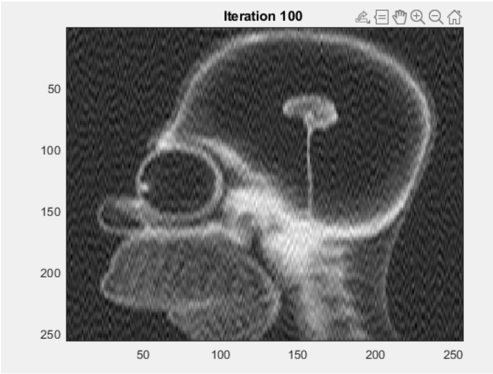


Nous remarquons que quelque soient les paramètres, le minibatch ne nous donne pas un bon résultat, cela est dû au fait que la majorité des données sont en (X,0) et que le minibatch va chercher à faire une régression sur une droite y= 0x +0 . La méthode n'est donc pas adaptée pour cette fonction.

Application à la déconvolution d'images

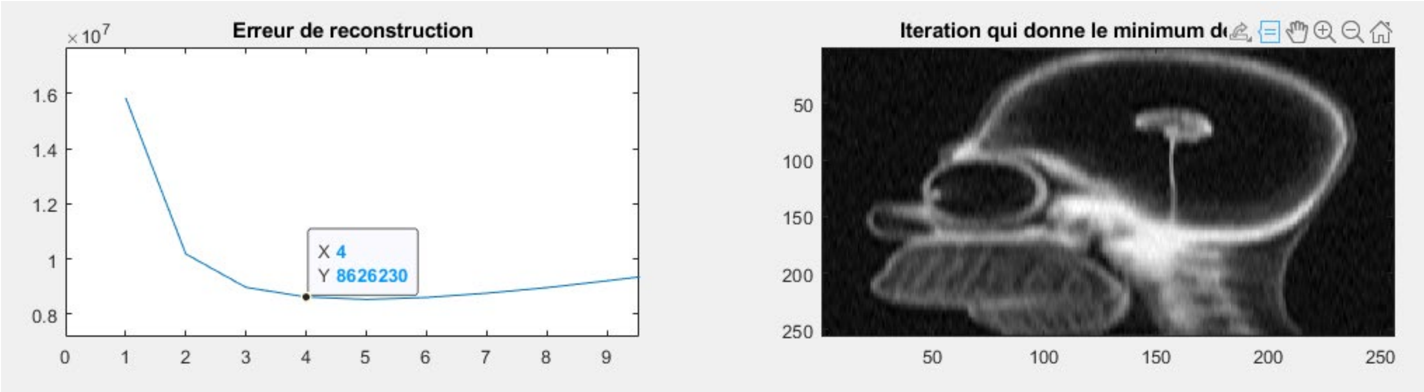
gradient pas cst





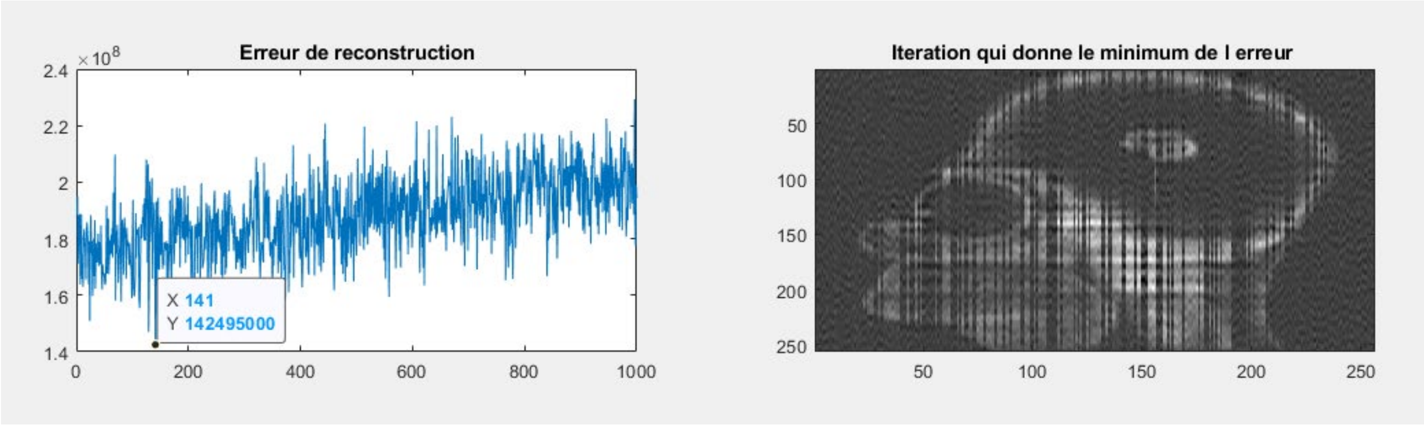
La 7ème itération est celle qui donne le moins d 'erreurs. La 100ème itération rend un résultat dégradé.

recherche unidimensionnelle



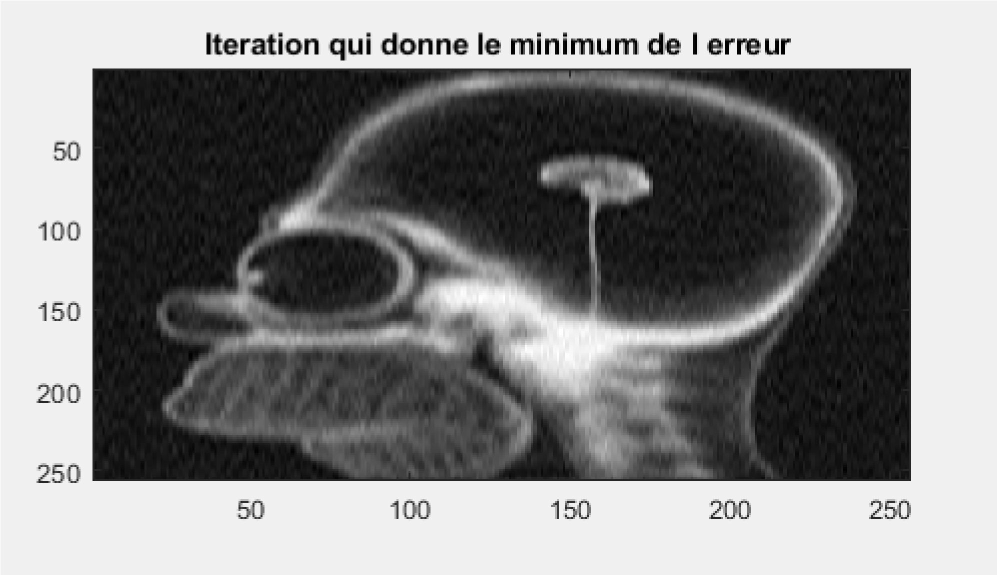
La 4ème itération est celle qui donne le moins d 'erreurs.

Mini Batch



La meilleure itération dégrade fortement l'image. En effet, nous prenons des composantes aléatoires de l'images pour les moyenner, ce qui n'est as représentatif de l'image. Cette méthode n'est donc pas compatible avec ce format

Application à la déconvolution d’images - Régularisation



Gamma = 0,02
Alpha = 0,05
231 itération
Algorithme a pas constant

Gamma = 0,02
Alpha = 0,05
203è itération
Algorithme a pas constant régularisé

gamma intervient devant ||x-p||^2 (||x-Ax||^2) et semble influer sur le « bruit » de l’image reconstruite (p et A)

Alpha intervient devant ||y-Hx||^2 et semble influer sur le « flou » de l’image reconstruite (matrice H)

Le kernel A permet d’améliorer les résultats (performance et output), en lissant les lignes et en retirant le bruit (gris)

Application à la déconvolution d’images - Régularisation

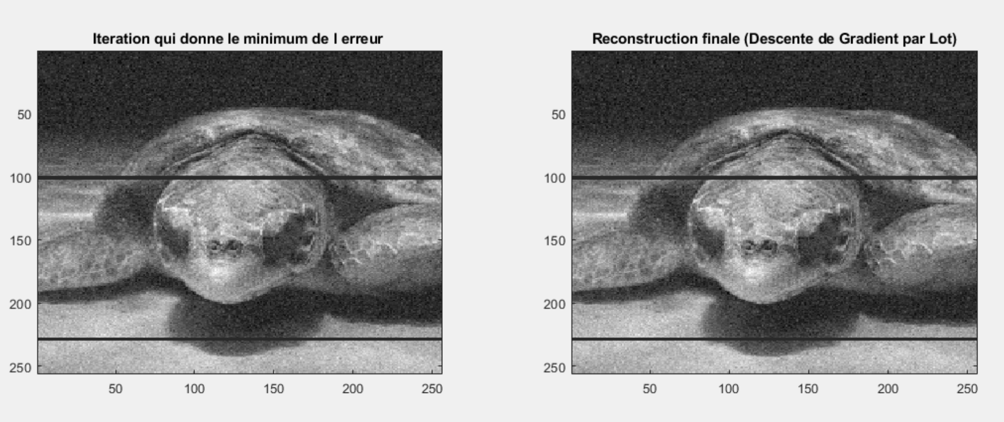
```
% Algorithme de Descente de Gradient par Batch
alpha = 0.01;
nombre_iterations = 1000;


recons_lot = 1/(m*n) * ones(n, m);


erreur_lot = zeros(1, nombre_iterations);

for iter = 1:nombre_iterations
  % Calculer le gradient
  Hx_lot = masque * recons_lot;
  grad_lot = masque' * (Hx_lot - tortue_degradee);
  recons_lot = recons_lot - alpha * grad_lot;
  erreur_lot(iter) = sum(sum((tortue_crop - recons_lot).^2));
end

figure(1); subplot(3, 3, 6); plot(erreur_lot); colormap(gray); title('Erreur de reconstruction (Descente de Gradient par Lot)');
[min_erreur, min_pos] = min(erreur_lot);
subplot(2, 3, 6); imagesc(recons_lot); colormap(gray); title('Reconstruction finale (Descente de Gradient par Lot)');
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



Algorithme de gradient Batch pour optimiser le critère des moindres carrés
(gauche – algo donné, droite – algo crée)

Pour l’erreur avec régularisation, nous changeons la matrice A
Nous pouvons voir que la régularisation permet d’avoir de meilleurs résultats