Etude d'article : Proximal Algorithms in Statistics and Machine Learning (Polson 2015)

Khalil Bergaoui, Mohamed Amine Hachicha

18/12/2020

Dans ce rapport, nous allons réaliser une analyse de l'article intitulé "Proximal Algorithms in Statistics and Machine Learning (Polson 2015)".

Nous allons commencer par présenter la problématique traitée et réaliser une synthèse des notions introduites. Dans un second temps, nous analyserons l'algorithme proximal dans le cadre de la théorie du point fixe. Enfin, nous allons reproduire quelques résultats expérimentaux du papier en vue de les analyser.

1 Synthèse

Dans cet article, les auteurs présentent l'algorithme proximal appliqué aux problèmes de minimization qui apparaissement fréquement dans le cadre des statistiques et machine learning. Ces problèmes prennent la forme suivante :

$$min_{x \in R^d} l(x) + \phi(x) \tag{1}$$

Ainsi, en prenant $l(x) = \frac{1}{2} \|y - Ax\|_2^2$ et $\phi(x) = \lambda \|x\|_1 ; \lambda \geq 0$, nous pouvons reconnaître le problème classique de régression linéaire avec pénalisation de type Lasso, où (x,y) représentent les observations et A la matrice des régresseurs.

1.1 Notations

Dans la suite, on considère que l est une fonction l.s.c convexe et différentiable, alors que ϕ est aussi l.s.c convexe mais pas nécessairement différentiable, et plus spécifiquement, on s'intéressera aux problème de la forme :

$$min_{x \in R^n} \sum_{i=1}^{n} l(y_i - a_i^T x) + \lambda \sum_{j=1}^{k} \phi([Bx - b]_j)$$

avec $\forall 1 \leq i \leq n, y_i$ les variables à prédire, a_i les variables prédictives, $B \in R^{k*d}$ une matrice de contraintes, b un vecteur de R^k et $\lambda > 0$ le paramètre de régularization.

1.2 L'opérateur proximal et l'enveloppe de Moreau

Les auteurs commencent par quelques définitions de bases qui sont l'enveloppe de Moreau, définie pour toute fonction f par:

$$f^{\gamma}(x) = inf_z \left\{ f(z) + \frac{1}{2\gamma} \|z - x\|_2^2 \right\}$$

et l'opérateur proximal défini par:

$$prox_{\gamma f}(x) = argmin_z \left\{ f(z) + \frac{1}{2\gamma} \|z - x\|_2^2 \right\}$$

Les auteurs citent ensuite quelques propriétés de ces deux concepts dont peut être la plus importante est celle ci qui reli l'enveloppe de Moreau à l'opérateur proximal:

$$prox_{\gamma f}(x) = x - \gamma \partial f^{\gamma}(x)$$

Donc l'opérateur proximal peut être vu comme l'application d'une étape de gradient à l'enveloppe de Moreau.

L'opérateur proximal peut être aussi vu comme une généralization de la projection Euclidienne puisque dans le cas où $f(x) = \iota_C(x)$ on obtient $prox_f(x) = argmin_z \|z - x\|_2^2$ qui correspond à la projection Euclidienne de x sur l'ensemble C.

1.3 Methodes

L'article présente plusieurs méthodes de gradient proximal qu'on va résumer ci dessous:

1.3.1 Le gradient proximal et quelques exemples

Les auteurs annoncent le principe de la méthode de gradient proximal pour résoudre le problème

$$min_x \; F(x) = l(x) + \phi(x)$$

où l est différentiable alors que ϕ ne l'est pas. Cette méthode consiste simplement en deux itérations: une étape de gradient et une application de l'opérateur proximal de ϕ. Concrètement, à chaque itération on calcule:

$$\begin{cases} v^t = x^t - \gamma \nabla l(x^t) \\ x^{t+1} = prox_{\gamma\phi}(v^t) \end{cases}$$

où γ est un pas constant.

L'algorithme de gradient proximal peut être vu comme un algorithm MM (Majorize/Minimize) et les auteurs prouvent ce point de vue dans l'article.

Les auteurs traitent ensuite quelques exemples simples sur des familles de fonctions dans le cadre du proximal Newton et du Iterative Shrinkage Thresholding.

La méthode d'accélération de Nesterov qui introduit une variable de moment z.

1.3.2 Séparation de variables, ascente de dual et lagrangien augmenté

Ces méthodes sont détaillées dans la section 4 de l'article. L'idée est d'effectuer une séparation de variables dans le problème original de manière à se ramener au problème équivalent:

$$min_{x,z}\ l(x) + \phi(z)$$

$$\text{telle que } z - x = 0$$

L'ascente de dual consiste donc à partir de la fonction duale $g(\lambda) = inf_{x,z} L(x, z, \lambda)$ avec $L(x, z, \lambda)$ le lagrangien du problème primal, et de résoudre le problème dual avec une ascente de gradient en remarquant que

$$\nabla g(\lambda) = \nabla_\lambda L(\hat{x}_\lambda, \hat{z}_\lambda, \lambda)$$

avec

$$\hat{x}_\lambda, \hat{z}_\lambda = argmin_{x,z} L(x, y, \lambda)$$

Le lagrangien augmenté correspond à

$$L_{aug}(x, z, \lambda) = L(x, z, \lambda) + \frac{\gamma}{2} \|x - z\|_2^2$$

qui correspond au lagrangien du problème

$$min_{x,z}\ l(x) + \phi(z) + \frac{\gamma}{2} \|x - z\|_2^2$$

$$\text{telle que } z - x = 0$$

qui a la même valeur optimale que le problème original grâce à la condition $x - z = 0$

Les itération de l'ascente de gradient sont alors:

$$\begin{cases} (x^{t+1}, z^{t+1}) = argmin_{x,z} L(x, z, \lambda^t) \\ \lambda^{t+1} = \lambda^t + \gamma(x^{t+1} - z^{t+1}) \end{cases}$$

L'une des difficultés majeures de cette approche est la recherche simultanée d'un x et z optimaux qui minimisent $L(x, z, \lambda^t)$ lors de la première itération, d'où l'intérêt de l'algorithme **ADMM**.

Pour l'ADMM on écrit

$$L(x, z, \lambda) = L(x, z, u) = l(x) + \phi(z) + \frac{\gamma}{2}(\|x - z + u\|_2^2 - \|u\|_2^2$$

où $u = \frac{\lambda}{\gamma}$ est la variable duale remise à l'échelle. Les itérations du ADMM sont alors:

$$\begin{cases} x^{t+1} = argmin_x \left\{ l(x) + \frac{\gamma}{2} \|x - z^t + u^t\|_2^2 \right\} = prox_{\frac{l}{\gamma}}(z^t - u^t) \\ z^{t+1} = argmin_z \left\{ \phi(z) + \frac{\gamma}{2} \left\|x^{t+1} - z + u^t\right\|_2^2 \right\} = prox_{\frac{\phi}{\gamma}}(x^{t+1} + u^t) \\ u^{t+1} = u^t + x^{t+1} - z^{t+1} \end{cases}$$

On voit alors l'intérêt de l'ADMM qui est de séparer les variables x et z dans les itérations de l'ascente de gradient.

Les auteurs proposent aussi la méthode de **Diviser pour régner** qui applique l'ADMM au cas où on veut minimiser $\Sigma_{i=1}^{N} l_i(A_i x) + \phi(x)$. Il s'agit donc de généraliser la séparation de variables en écrivant le problème comme:

$$min_{z_i} \ \Sigma_{i=1}^{N+1} l_i(z_i)$$

$$\text{telle que } z_i - A_i x = 0$$

où $A_{N+1} = Id$ et $l_{N+1} = \phi$

1.3.3 Méthodes d'enveloppe

Les méthodes d'enveloppe sont issues des enveloppes de Moreau et donnent lieu à des algorithmes de gradient proximal analogues.

L'enveloppe **Forward Backward (FB)** par exemple a été présentée par les auteurs et s'installe dans le contexte de minimisation de $F = l + \phi$ où l est une fonction fortement convexe et à gradient Lipschitzien, et ϕ est une fonction convexe, propre (i.e $dom(\phi) \neq \emptyset$) et semi-continue inférieurement (c'est à dire $lim\ inf_{x_n \to x)} \geq f(x)\ , \forall x$). Elle est donc définie par:

$$F_\gamma^{FB}(x) = min_v \ \left\{ l(x) + \nabla l(x)^T (v - x) + \phi(v) + \frac{1}{2\gamma} \left\| v - x \right\|^2 \right\}$$

Cette enveloppe est assez critique puisque ses points stationnaires définissent les solutions du problème original de minimisation, où on a:

$$x = prox_{\gamma\phi}(x - \gamma \nabla l(x))$$

Les auteurs définissent aussi l'enveloppe de **Douglas Rachford (DR)** qui est liée à l'enveloppe FB par la relation suivante:

$$F_\gamma^{DR}(x) = F_\gamma^{FB}(prox_{\gamma l}(x))$$

Le gradient de cette enveloppe produit aussi un algorithme de gradient proximal qui converge vers le minimum de notre fonction F. Quelques travaux ont étudié la convergence du gradient proximal avec l'enveloppe de Douglas Rachford, comme c'est le cas dans [3] où les auteurs ont montré que dans le cas où l est convexe quadratique, c'est à dire $\forall x,\ l(x) = \frac{1}{2} x^T Q x + q^T x$, on obtient:

$$F_\gamma^{DR}(x_k) - F_\gamma^{DR}(\tilde{x}) \leq \frac{1}{k} \frac{1 + \nu\gamma}{2\gamma(1 - \gamma\nu)} \left\| x_0 - \tilde{(x)} \right\|^2$$

où ν est la constante de Lipschitz de ∇l et x_k sont les points obtenus par les itérations du gradient proximal de Douglas Rachford comme données dans l'article. Ce résultat prouve donc à une convergence linéaire du gradient de l'enveloppe de Douglas-Rachford.

Les enveloppes **semi-quadratiques (ou Half-Quadratic, HQ)** sont aussi communément utilisées surtout dans le cas où $l(x) = \|Ax - y\|$ et ϕ est une fonction non convexe. Sans rentrer dans les détails de l'algorithme proximal induit par cette enveloppe, on se contente de retenir qu'elle utilisée dans le cas où la fonction objectif est de la forme

$$F(x) = \frac{1}{2}\|Ax - y\|^2 + \Sigma_{i=1}^{d}\phi((B^T x - b)_i)$$

où

$$\phi(x) = F^{HQ}(x) = inf_v \ \{Q(x, v) + \Psi(v)\}$$

avec Q une fonction semi-quadtratique et Ψ est le conjugé convexe d'une certaine fonction.

L'article [5] étudie ces enveloppes avec plus de détails et donne pas mal de cas d'usages, notamment pour l'inférence de loi logit-binomiales où $(y|x) \sim Binom(m, w(x))$ avec $m \in N$ et w est la fonction sigmoïd. Les auteurs prouvent alors que la log-vraisemblance de x est proportionnelle à une enveloppe quadratique en x.

1.3.4 Méthodes proximales pour les fonctions composites

Les auteurs explorent quelques méthodes de gradient proximal appliquées au cas où notre fonction objective a la forme:

$$F(x) = l(x) + \phi(Bx)$$

. Les méthodes exposées passent généralement par l'ajout d'une variable latente z au niveau du second terme avec la contrainte $z = Bx$.

Plusieurs méthodes sont possibles pour la résolution d'un tel problèmes. Par exemple, en écrivant le Lagrangien du problème, on peut se ramener facilement à une résolution par ADMM à l'aide de la séparation des variables z et x.

Une autre approche peut consister à remplacer l'un des termes de la fonction objectif par une enveloppe (par exemple une enveloppe Douglas Rachford) et utiliser donc les méthodes proximales qui vont avec et qui vont utiliser naturellement l'opérateur proximal.

Les auteurs donnent des exemples de résolution par une approche primal-dual et de résolution d'un problème de minimization d'enveloppe quadratique. On ne va pas détailler les calculs de ces exemples ici.

1.4 Expérimentations

Les auteurs présentent finalement une série d'expériences donc la plupart sont faites avec des données synthétisées. Le but des expériences était justement d'essayer d'appliquer quelques méthodes citées dans l'article. Nous avons choisi de reproduire l'expérience 7.1 dans la section 3 de notre analyse.

2 Généralisation de la fonction objective

Il est clair que les auteurs de l'article ont voulu être le plus exhaustif que possible en donnant un nombre considérable de méthodes. Les auteurs se sont limités surtout au cas où la fonction objective prend la forme

$$F(x) = l(x) + \phi(x)$$

. L'enveloppe de Douglas Rachford peut mener à une généralisation au cas où

$$F(x) = \Sigma_{i=1}^{d} f_i(x)$$

où les fonctions $f_i \in \Gamma_0(\mathcal{H})$, c'est à dire propres, convexes et semi-continues inférieurement (lower semi-continuous) sur des espaces de Hilbert $\mathcal{H}$.

C'est le cas de l'algorithme PPXA exposé dans l'article [6] qui est une extension de l'algorithme proximal de Douglas Rachford. Dans le cas où $\mathcal{H} = R^k$ on commence par initialiser $x_0 = \Sigma_{i=1}^{d} w_i u_{i,0}$ avec $\Sigma_{i=1}^{d} w_i = 1$ et $u_{i,0} \in R^k$.

Les itérations du PPXA sont ensuite données par:

$$\forall l \geq 0 \begin{cases} \forall\ 1 \leq i \leq d,\ p_{i,l} = prox_{\frac{\gamma f_i}{w_i}}(u_{i,l}) \\ p_l = \Sigma_{j=1}^{d} w_j u_{j,l} \\ \forall\ 1 \leq i \leq d,\ u_{j,l+1} = u_{j,l+1} + \lambda_l(2p_l - x_l - p_{j,l}) \\ x_{l+1} = x_l + \lambda_l(p_l - x_l) \end{cases}$$

et on repète ces itérations jusqu'à convergence de l'algorithme. La séquence $(x_n)_{n\geq 0}$ converge alors vers le minimum de F si celui si existe.

Il faut aussi que les constantes employées satisfassent les conditions suivantes:

$$\begin{cases} \forall\ l \geq 0,\ 0 < \lambda_l < 2 \\ \Sigma_{n\in N} \lambda_n(2 - \lambda_n) = +\infty \\ F \text{ est coercive, c'est à dire } lim_{\|x\|\to+\infty} F(x) = +\infty \end{cases}$$

L'article [6] donne aussi une version accélérée du PPXA. L'avantage ultime de cet algorithme est qu'on peut paralléliser chaque itération sur plusieurs calculateurs, il est donc adapté aux problèmes d'optimisations à grande échelle où d est grand.

L'article [4] établit une généralisation assez intéressante de l'algorithme PPXA, dénotée PPXA+ où la fonction objective est de la forme:

$$F(x) = \Sigma_{i=1}^{d} f_i(L_i x)$$

où les L_i sont dans $\mathcal{B}(\mathcal{G}, \mathcal{H}_i)$ où $\mathcal{B}$ désigne l'ensemble des opérateurs linéaires bornés, et $\mathcal{G}$ et les $\mathcal{H}_i$ sont des espaces de Hilbert.

Le PPXA+ correspond donc à une généralisation du cas $F(x) = l(x)+\phi(Bx)$ traité dans la section 6 de l'article.

3 Algorithme proximal dans le cadre de la théorie du point fixe

Dans la section 2.1 de l'article, les auteurs font réference au lien entre l'algorithme proximal et la théorie du point fixe, en se limitant à une description qualitative : *"Finally, there is a close connection between proximal operators and fixed-point theory,[..] All three of these ideas—taking gradient-descent steps, projecting points onto constraint regions, and finding fixed points of suitably defined operators— arise routinely in many classical optimization algorithms. It is therefore easy to imagine that the proximal operator, which relates to all these ideas, could also prove useful."*

Nous proposons dans cette section de développer ce lien, en étudiant les résultats existants sur la convergence des algorithmes de type :

$$x_{n+1} = Tx_n$$

L'algorithme proximal s'obtient en prenant $T = prox_{\gamma\phi}o(Id - \gamma\nabla l)$, en utilisant les notations du problème d'optimisation (1).

Dans la suite nous baserons notre étude sur les réferences [2] et [1].

Nous avons vu dans le Chapitre 3 du cours que l'algorithme de descente de gradient, pour une fonction differentiable f à gradient k-Lipschitzien et vérifiant la condition d'ellipticité avec un constante α, converge vers un minimum local. En effet, sous ces deux conditions, on montre que l'application

$x \mapsto (Id - \gamma\nabla f)(x)$ est une contractante pour un paramètre de descente $0 < \gamma < \frac{2\alpha}{k^2}$ et le résultat de convergence découle du théorème suivant:

Théorème du point fixe pour une application contractante:

Soient E un espace métrique complet (non vide) et f une application k-contractante de E dans E. Il existe un point fixe unique $x*$ de f . De plus, toute suite d'éléments de E vérifiant la récurrence $x_{n+1} = f(x_n)$ vérifie la majoration $d(x_n, x^*) \leq \frac{k^n}{1-k}\ d(x_0, x_1)$ donc converge vers x*.

En pratique, en particulier dans les problèmes d'optimisation rencontrés en statistiques et machine learning, et en restant dans le cadre d'optimisation convexe abordée dans cet article, il est intéressant de traiter une catégorie plus générale d'applications, pour lesquelles la condition de contraction n'est pas nécessairement satisfaite et le théorème précedent ne s'applique pas.

Nous aurons recours à la théorie des applications non expansives (i.e 1-Lipschitzienne) et en particulier à la classe des opérateurs α-moyennés , $\alpha \in [0, 1]$ qui constitue un sous ensemble de l'ensemble des applications 1-Lipschitziennes, et dans laquelle peut s'inscrire l'étude de convergence de l'algorithme proximal.

Cela se justifie par le fait que l'opérateur proximal associé à une fonction convexe, l.s.c et propre (i.e dom(g) non vide) est α -moyenné avec $\alpha = \frac{1}{2}$ [2].

Défintions:

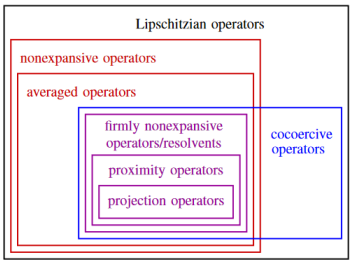


Figure 1: Classes des opérateurs non linéaires [2]

Soient $\mathcal{H}$ un espace de Hilbert, P un fermé de E et $T : P \to P$ une application (non nécessairement linéaire)
- On dit que T est non expansive si $\forall\,(x, y) \in P \times P \qquad \|Tx - Ty\| \leq \|x - y\|$.
- On dit que T est α moyenné, $\alpha \in [0, 1]$ s'il existe une application $R : P \to P$ non expansive telle que $T = (1 - \alpha)Id + \alpha R$ et on a pour $0 < \alpha \leq 1$:

$$\|Tx - Ty\|^2 + \frac{1-\alpha}{\alpha}\|(Id - T)x - (Id - T)y\|^2 \leq \|x - y\|^2$$

Remarques:
- Pour un opérateur α-moyenné $T = (1-\alpha)Id+\alpha R$ l'ensemble de ses points fixes vérifie Fix$T$ =Fix$R$.
En effet, pour $\alpha > 0$, $\tilde{x} \in \text{Fix}T \iff \tilde{x} = T\tilde{x} \iff \tilde{x} = (1-\alpha)\tilde{x} + \alpha R\tilde{x} \iff \alpha\tilde{x} = \alpha R\tilde{x} \iff \tilde{x} \in \text{Fix}R$.
- Toute opérateur 1-moyenné est 1-Lipschitzien.
- Pour toute fonction g $\in \Gamma_0(\mathcal{H})$, convexe, l.s.c et propre (i.e dom(g) non vide), l'opérateur proximal associé $prox_g$ est α -moyenné avec $\alpha = \frac{1}{2}$

Caractérisation des points fixes:
Considérons le problème d'optimisation (1), et notons γ le pas de descente de sorte que l'opérateur correspondant à l'algorithme proximal s'écrit :

$$T = prox_{\gamma\phi}o(Id - \gamma\nabla l)$$

L'ensemble de ses points fixes est caractérisé par

$$FixT = zer(\partial(l + \phi))\ i.e\ x \in FixT \iff 0 \in \partial(l + \phi)(x)$$

où $\partial(l + \phi)(x)$ représente le sous-différentiel de la fonction convexe $l + \phi$ au point x.

D'autre part, pour toute fonction convexe $f : \mathcal{H} \to \mathcal{H}$, nous avons les propriétés suivantes:
- $zer(\partial f) \neq \emptyset$. De plus, $zer(\partial f)$ est un fermé convexe de $\mathcal{H}$.
- Si f est différentiable alors $\forall x \in \mathcal{H},\ \partial f(x) = \{\nabla f(x)\}$. On retrouve donc que x est un minimiseur de f $\iff x \in zer(\partial f)$

Vu que nous considérons des fonction l et ϕ dans $\Gamma_0(\mathcal{H})$, $l + \phi$ est aussi dans ${}_0(\mathcal{H})$, nous déduison que l'opérateur correspondant à l'algorithme proximal $T = prox_{\gamma\phi} o(Id - \gamma\nabla l)$ vérifie bien $\mathbf{FixT} \neq \emptyset$.

De plus, si l est différentiable à gradient k-Lipschitzien alors on peut montrer que (ceci découle de la propriété de compositions des deux opérateurs α-moyennés $prox_{\gamma\phi}$ et $(Id - \gamma\nabla l)$) :

$$\begin{cases} (Id - \gamma\nabla l) \text{ est } \frac{k\gamma}{2}\text{-moyenné } \forall\gamma \in]0, \frac{2}{k}[\\ T\text{est } \alpha \text{ - moyenné avec } \alpha = (2 - \frac{k\gamma}{2})^{-1} \end{cases}$$

avec $\alpha < 1 \iff \gamma < \frac{2}{k}$.
Nous pouvons alors conclure quant à la convergence de l'algorithme proximal, dans ce cas, en utilisant le théorème suivant:

Théorème:
Soit $T : \mathcal{H} \to \mathcal{H}$ un opérateur α-moyenné , $0 < \alpha < 1$ tel que $FixT \neq \emptyset$.
Soit $(\theta_n)_{n\in N}$ une suite dans $]0, \alpha^{-1}[$ telle que $\sum_{n\in N} \theta_n(1 - \alpha\theta_n) = +\infty$
Alors la suite $(x_n)_{n\in N}$ converge (faiblement) vers un point dans $FixT$.

Remarques:
- En prenant $\forall n \in N,\ \theta_n = 1$, (possible car $\alpha^{-1} > 1$ quand $\gamma < \frac{2}{k}$) on retrouve l'algorithme de gradient proximal.
- Sous cette forme, ce théorème permet de justifier les résultats de convergence pour les algorithmes Forward-Backward (gradient proximal étant un cas particulier) et Douglas-Rachford (utile quand les deux fonctions l et ϕ ne sont pas différentiables).
- La caractérisation des points fixes de l'opérateur associé à l'algorithme gradient proximal nous donne une deuxième manière de justifier l'existence du point fixe pour cet opérateur, en plus de la justification qui apparaît dans la section "Appendix A: Proximal Gradient Convergence" de l'article.

4 Etude expérimentale

Dans cette partie, nous allons décrire les résultats du papier que nous avons essayé de reproduire.

Nous avons choisi de reproduire les résultats obtenus sur un cas d'application couramment rencontré en statistiques : la régression logistique avec pénalisation de type Lasso.

4.1 Lasso Logit

On considère ici, le problème d'optimization suivant:

$$min_{x\in R^d}\ l(x) + \lambda\phi(x)$$

$$avec \begin{cases} l(x) = \sum_{i=1}^{n} m_i log(1 + exp^{a_i^T x} - y_i * a_i^T x) \\ \phi(x) = \sum_{j=1}^{d} |x_j| \end{cases}$$

où les variables x et y sont générées de la manière suivante:
- $x \sim \mathcal{N}(0,1)$; en gardant seulement 0.1 des coefficients de x non nuls.
- $a_i \sim \mathcal{N}(0,1)$
- $p_i = (1 + exp^{-a_i^T x})^{-1}$
- $yi|pi \sim \mathcal{B}(mi, pi)$

La valeur des m_i n'étant pas spécifiées dans l'article, nous fixons $m_i = 1$ de sorte à se remmetre au cas dans lequel $yi|pi \sim Bernoulli(pi)$, $\forall 1 \leq i \leq n$

En notant A la matrice formée par les vecteurs lignes $a_i^T \forall 1 \leq i \leq n$, nous pouvons écrire:

$$\begin{cases} \tilde{y}_i = m_i * (1 + exp^{-a_i^T x})^{-1} \ , \forall 1 \leq i \leq n \\ \nabla l(x) = A^T(\tilde{y} - y) \\ prox_{\gamma(\lambda\phi)}(x) = (sgn(x_i) * max(x_i) - \gamma\lambda, 0)_{1 \leq i \leq d} \end{cases}$$

Comme dans l'article, nous fixons la borne supérieure du pas de descente γ à 2/k , avec $k = \left\|A^T A\right\|/4 = \sigma_{max}(A)/4$. Pour tout $0 < \gamma < 2/k$, l'algorithme converge. Dans la suite nous fixons $\gamma = 0.99 * \frac{2}{k}$. Le facteur de régularization est fixé à $\lambda = 0.1\sigma_{max}(A)$. Nous obtenons $\sigma_{max}(A) = 2.35$. Nous implementons l'algorithme proximal en utilisant la forme plus générale donnée par l'algorithme Forward-Backward:

$$\forall n \in N \ \begin{cases} y_n = x_n - \gamma \nabla l(x_n) \\ x_{n+1} = x_n + \theta(prox_{\gamma(\lambda\phi)}(y_n) - x_n) \end{cases}$$

En en effet pour $\theta = 1$, on retrouve l'algorithme proximal.

Nous avons remarqué qu'en variant le paramètre θ nous avons pu obtenir des résultats plus similaires à ceux présentés dans le papier.

La version accélerée de l'algorithme, en utilisant l'accéleration de Nesterov est donnée par :

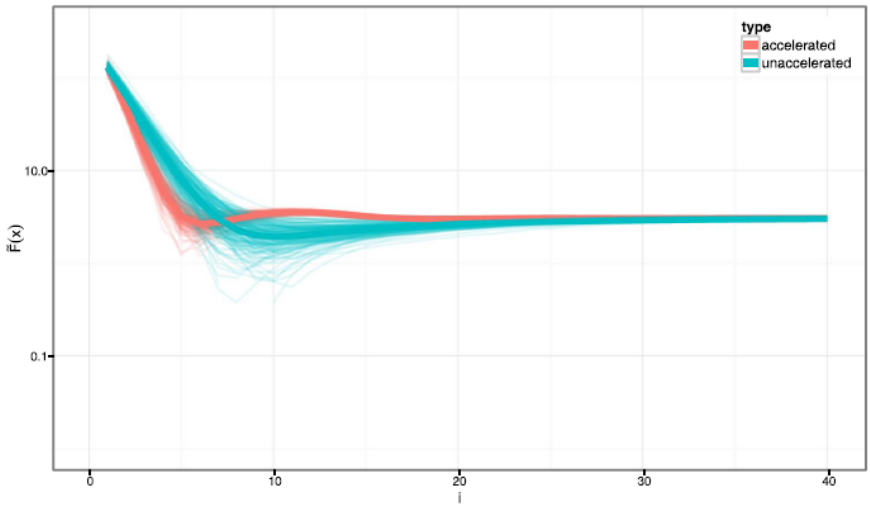
$$\forall n \geq 1 \ \begin{cases} z_n = x_n + \frac{n}{n+2}(x_n - x_{n-1}) \\ y_n = z_n - \gamma \nabla l(z_n) \\ x_{n+1} = x_n + \theta(prox_{\lambda\phi}(y_n) - x_n) \end{cases}$$

où la première iteration reste inchangée par rapport à la version non accélerée de l'algorithme:

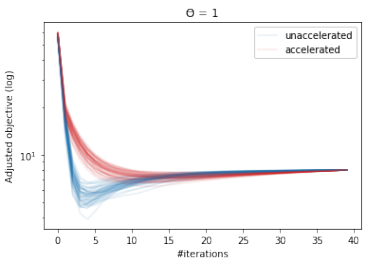
$$n = 0 \ \begin{cases} z_0 = x_0 y_n = z_n - \gamma \nabla l(z_0) \\ x_1 = x_0 + \theta(prox_{\lambda\phi}(y_0) - x_0) \end{cases}$$

Ainsi nous pouvons comparer ces deux versions du l'algorithme. Nous traçons l'évolution de la fonction objectif pour 40 iterations. Nous obtenons les résultats suivants:

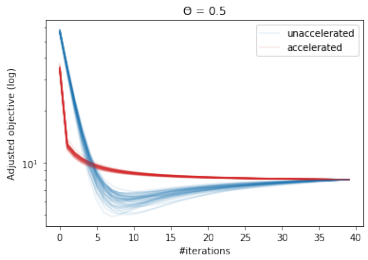
Nous observons dans les figures précedentes que nous retrouvons les mêmes comportements que ceux trouvés par les auteurs. D'une part, nous retrouvons que l'accéleration de Nesterov améliore la vitesse de convergence de l'algorithme.



(a) Résultat du papier.



(b) Notre implémentation : algorithme proximal.



(c) Notre implémentation : algorithme Forward-Backward($\theta = 0.5$)

Figure 2: Reproduction des résultats de la section 7.1 du papier

D'autre part nous retrouvons le caractère non descendant de l'algorithme proximal, en effet la décroissance de la courbe bleue (non accéléré) n'est pas monotone. Notons que ce caractère non descendant de l'algorithme dépend du choix des paramètres λ (régularization) et γ (pas de descente gradient). En effet, comme le démontrent les auteurs dans la section APPENDIX A, la fonction objective F du problème de minimization vérifie:

$$(\lambda - \gamma)\frac{1}{2}\|x_n - x*\| \leq F(x_n) - F(x*)$$

où $x*$ représente la solution finale. Ainsi, pour garantir la décroissance de la fonction objective, il faut choisir $\lambda \geq \gamma$. Ce qui n'est pas le cas dans les résultats

observées. En effet, en reproduisant ces résultats, nous avons trouvé les valeurs suivantes $\gamma = 3.39 > \lambda = 0.24$

Par ailleurs nous avons deux remarques par rapport à cette figure :

- Afin de retrouver la même vitesse de convergence de l'algorithme non accéléré, nous avons implémenté l'algorithme forward-backward avec le paramètre $\theta = 0.5$. Cela donne une résultat plus proche de celui présenté dans le papier (au bout d'une dizaine d'iterations la courbe bleue atteint son minimum)

- Pour un seuil de convergence fixé, l'implémentation de l'accéleration de Nesterov est d'autant plus avantageuse que le paramètre θ est petit.

References

[1] H. Bauschke and P. L. Combettes. Convex analysis and monotone operator theory in hilbert spaces,2nd ed, newyork springer, 2019.

[2] Patrick L. Combettes and Jean-Christophe Pesquet. Fixed point strategies in data science., 2020.

[3] Panagiotis Patrinos, Lorenzo Stella, and Alberto Bemporad. Douglas-rachford splitting: Complexity estimates and accelerated variants, 2014.

[4] Jean-Christophe Pesquet and Nelly Pustelnik. A parallel inertial proximal optimization method. *Pacific Journal of Optimization*, 8, 01 2010.

[5] Nicholas G. Polson and James G. Scott. Mixtures, envelopes, and hierarchical duality, 2015.

[6] Nelly Pustelnik, Caroline Chaux, and Jean-Christophe Pesquet. Parallel proximal algorithm for image restoration using hybrid regularization – extended version, 2011.

Reproduction de résultats de la section 7.1 de l'article Proximal Algorithms in Statistics and Machine Learning (Polson 2015)

Khalil Bergaoui , Mohamed Amine Hachicha

In [1]:
```
# Importer bibliotheques utiles

import numpy as np
import matplotlib.pyplot as plt
import scipy
```

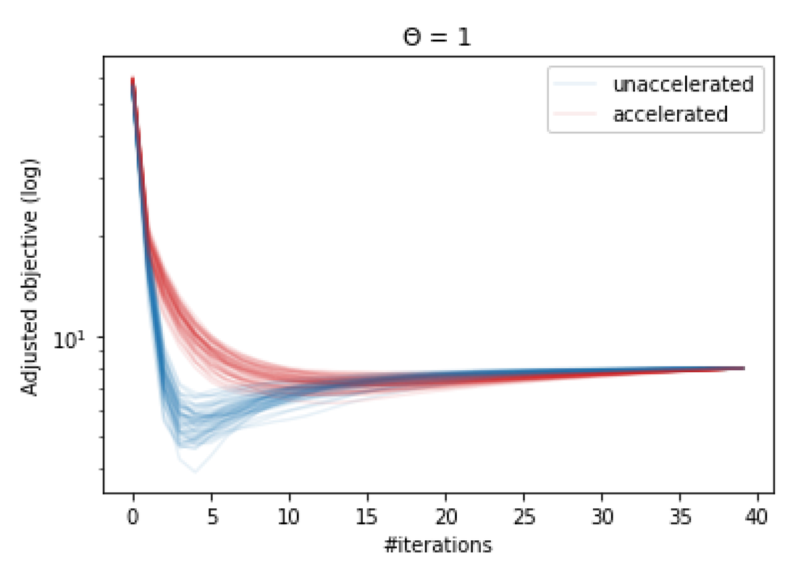
In [2]:
```
def get_x(M):
    """
    Generer ue vecteur x de taille M de la loi normale
    """

    #Comme dans l'article, on méttera 10% des coefficients de x à zero
    nb_non_zero = M//10
    x = np.zeros(M)
    x_non_zero = np.random.normal(loc=0.0,scale=1.0,size=nb_non_zero)
    assert np.where(x_non_zero==0)[0].shape[0] ==0

    indices = np.random.choice(range(M),nb_non_zero,replace=False)
    x[indices] = x_non_zero
    assert np.where(x!=0)[0].shape[0] ==nb_non_zero
    return x
```

In [3]:
```
def get_A(N,M):
    """
    Generer une matrice A de taille NxM de la loi normale
    """
    A = np.random.normal(loc=0.0,scale=1.0,size=(N,M))
    #On normalize les colonnes comme dans l'article
    from sklearn.preprocessing import normalize
    A = normalize(A,axis=1)
    return A
```

In [4]:
```
def get_p(A,x):
    #Calculer les probabilités p_i
    from scipy.special import expit #sigmoid
    p = expit(A@x)
    return p
def get_y(p,m_max):
    #
    m = np.random.randint(m_max,size=(p.shape))+1
    #Generaer y  de la loi binomiale
    y = np.random.binomial(m,p)
    return y,m
```

In [5]:
```
from scipy.special import expit


def prox_l1_norm(y, gamma, penalty):
    """
    Calculaer l'opérateur proximal pour la norme l1 avec
    @y: vecteur pour lequel on évalue l'opérateur
    @gamma : réel designant le pas de descente de gradient
    @penalty: coefficient de penalisation Lasso (correspondant à la variable lambda)
    """
    y_int = y.copy()
    idx = np.abs(y_int) - gamma * penalty > 0
    y_int[idx] = (np.abs(y_int[idx]) - gamma * penalty) * np.sign(y_int[idx])
    y_int[~idx] = 0
    return y_int


def objective(A,x,y,m,lamda):
    """
    Fonction objective
    """

    N = y.shape[0]
    return (np.sum(np.multiply(m,np.log(np.ones(y.shape[0])+np.exp(A@x))) - np.multiply(y,A@
x))/np.sum(m)+ lamda*np.sum(np.abs(x)))

def unaccelerated(y,m,A,x,max_iter,x_0,theta='default',penalty=None):
    """
    Implementation de l'algorithme forward backward non accéléré
    """

    #Initialiser une liste pour les valeurs de la fonction objectif
    obj =[]

    #Determine la valeur singuliaire maximale de la matrice A
    eig_max = scipy.linalg.eigh(np.transpose(A)@A)[0].max()
    #Constante de lipschitz de la partie différentiable de la fonction objectif
    v = eig_max/4
    #pas de descente gradient
    gamma = 1.999 / v
    #Pénalisation Lasso
    if penalty == None:
        penalty = 0.1*eig_max
    if theta =='default':
        theta = 0.99 * (2 - gamma * v / 2) # =1


    K = np.transpose(A)@y
    x_n = x_0
    for i in range(max_iter):
        loss= objective(A,x_n,y,m,penalty)
        obj.append(loss)
        x_n_1 = x_n

        target = np.multiply(m,expit(A@x_n))
        #iteration de descente gradient
        y_n = x_n - gamma * (np.transpose(A)@target -K)
        #Iteration utilisant l'opérateur proximal
        x_n = x_n + theta * (prox_l1_norm(y_n, gamma, penalty) - x_n)


    return x_n, obj

def accelerated(y,m,A,x,max_iter,x_0,theta='default',penalty=None):
    """

    Implementation de l'algorithme forward backward
    avec aceeleration de Nesterov

    """
      #Initialiser une liste pour les valeurs de la fonction objectif
    obj =[]

    #Determine la valeur singuliaire maximale de la matrice A
    eig_max = scipy.linalg.eigh(np.transpose(A)@A)[0].max()
    #Constante de lipschitz de la partie différentiable de la fonction objectif
    v = eig_max/4
    #pas de descente gradient
    gamma = 1.999 / v
    #Pénalisation Lasso
    if penalty == None:
        penalty = 0.1*eig_max
    if theta =='default':
        #avec theta = 1 l'algorithme forward backward correspond à l'algorithme proximal
        theta = 0.99 * (2 - gamma * v / 2) # =1


    K = np.transpose(A)@y

    x_prev = x_0
    x_next = x_0
    for i in range(max_iter):
        loss= objective(A,x_next,y,m,penalty)
        obj.append(loss)

        #iteration de Nesterov
        acceleration = i/(i+2)
        z = x_prev + acceleration*(x_next-x_prev)
        target = np.multiply(m,expit(A@z))
        #iteration de descente gradient
        y_n = z - gamma * (np.transpose(A)@target -K)
        #Iteration utilisant l'opérateur proximal
        x_next = x_prev + theta * (prox_l1_norm(y_n, gamma, penalty) - x_prev)

    return x_next, obj
```

In [6]:
```
#Fixer la même random seed
seed = 42
np.random.seed(seed)

#Dimensions des vecteurs et matrice comme dans l'article
M=300
N=100
x = get_x(M)
A = get_A(N,M)
p = get_p(A,x)
#Nombre de réalisations de la variable y|p pour x et A donnés
trials = 50
m_max = 1


adj_param = 8 #ajouter à la fonction objective pour avoir des valeurs positives et tracer en
echelle log


for n in range(trials):
    #Echantillonage aléatoire de la variable y
    y,m = get_y(p,m_max)

    #Initialization
    x_0 = np.ones(x.shape)

    #Resolution sans accéleration
    #on utilise 40 iterations au total
    x_n,obj_un = unaccelerated(y,m,A,x,40,x_0)
    adj_obj_un = np.array(obj_un)-obj_un[-1]+adj_param #Ajuster la fonction objectif

    #Resolution avec accéleration de Nesterov
    #on utilise 40 iterations au total
    x_n,obj_acc = accelerated(y,m,A,x,40,x_0)
    adj_obj_acc = np.array(obj_acc)-obj_acc[-1] +adj_param

    #Plot fonction objectif
    plt.plot(adj_obj_un,color ='tab:blue',alpha=0.1)
    plt.plot(adj_obj_acc,color ='tab:red',alpha=0.1)




plt.yscale('log')#Echelle log

plt.ylabel('Adjusted objective (log)')
plt.xlabel('#iterations')
plt.legend(['unaccelerated','accelerated'],framealpha=10)
```



Remarque : La figure ci-dessus est obtenue en ajustant la fonction tracée de sorte à obtenir le même état final (car chaque courbe tracée correspond à une nouvelle résolution vu que le target y est échantilloné aléatoirement)

In [7]:
```
eig_max = scipy.linalg.eigh(np.transpose(A)@A)[0].max()
v = eig_max/4
gamma = 1.999 / v
penalty = 0.1*eig_max

# Nous vérifions si on a bien penalty < gamma pour expliquer la non decroissance de la courb
e bleue
```

```
3.399524869968754 0.2352093397120373
```

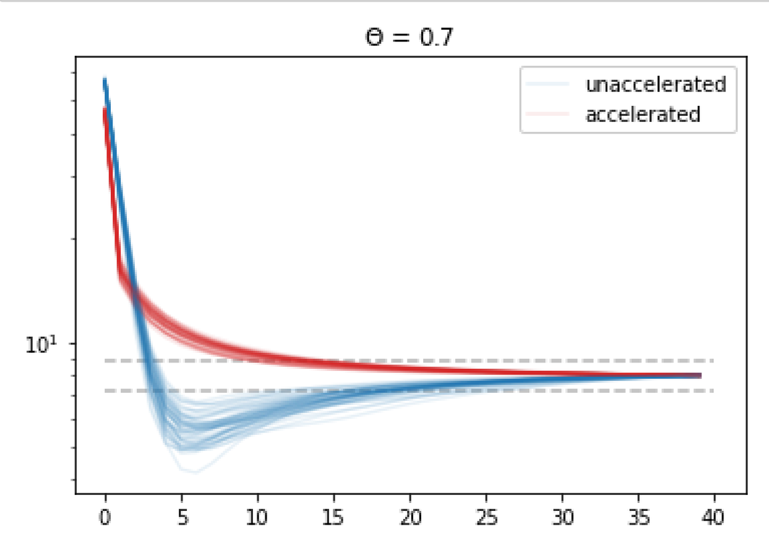
Nous trouvon vien $\gamma = 3.39 > \lambda = 0.24$ ce qui explique le caractère non descendant de cet algorithme.En effet, comme le démontrent les auteurs dans la section APPENDIX A, la fonction objective F du problème de minimization vérifie:

$$(\lambda - \gamma)\frac{1}{2}\|x_n - x*\| \leq F(x_n) - F(x*)$$

où $x*$ représente la solution finale. Donc pour imposer la décroissance il faut imposer$\gamma \leq \lambda$

Cette figure est obtenue pour une valeur de $\theta = 0.99$ du pas dans l'iteration faisant intervenir $prox_{\gamma f}$ pour $\theta = 1$ forward-backward est equivalent à l'algorithme proximal. Nous remarquons que l'allure de cette figure dépend de la valeur de ce paramètre. En effet il est évident que pour des valeurs plus faibles, l'algorithme prend plus d'iterations pour converger. Cependant, l'apport de l'algorithme utilisant l'acceleration de Nesterov devient plus clair. Ci-dessous, nous retraçons la courbe

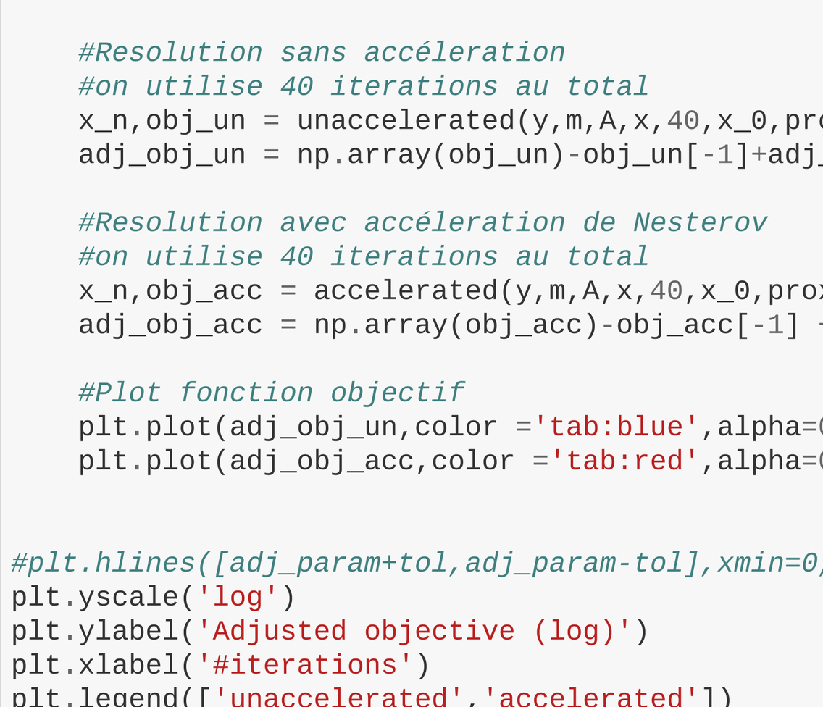
In [8]:
```
#Fixer la même random seed
seed = 42
np.random.seed(seed)

#Dimensions des vecteurs et matrice comme dans l'article
M=300
N=100
x = get_x(M)
A = get_A(N,M)
p = get_p(A,x)
#Nombre de réalisations de la variable y|p pour x et A donnés
trials = 50
m_max = 1
adj_param = 8 #ajouter à la fonction objective pour avoir des valeurs positives et tracer en
echelle log

prox_step = 0.7
for n in range(trials):
    #Echantillonage aléatoire de la variable y
    y,m = get_y(p,m_max)

    #Initialization
    x_0 = np.ones(x.shape)

    #Resolution sans accéleration
    #on utilise 40 iterations au total
    x_n,obj_un = unaccelerated(y,m,A,x,40,x_0,prox_step)
    adj_obj_un = np.array(obj_un)-obj_un[-1]+adj_param #Ajuster la fonction objectif

    #Resolution avec accéleration de Nesterov
    #on utilise 40 iterations au total
    x_n,obj_acc = accelerated(y,m,A,x,40,x_0,prox_step)
    adj_obj_acc = np.array(obj_acc)-obj_acc[-1] +adj_param

    #Plot fonction objectif
    plt.plot(adj_obj_un,color ='tab:blue',alpha=0.1)
    plt.plot(adj_obj_acc,color ='tab:red',alpha=0.1)


tol = 0.1*adj_param
plt.hlines([adj_param+tol,adj_param-tol],xmin=0,xmax=40, alpha=0.3,ls='--')
plt.yscale('log')
plt.legend(['unaccelerated','accelerated'],framealpha=10)
```



Dans ce cas, la fonction objective pour l'algorithme accéléré(en rouge) passe dès les premières iterations sous celle l'algorithme non acceléré (en bleu) et se trouve dans un voisinage de la valeur finale plus rapidement. Par rapport à la figure précedente, nous pouvons conclure que pour notre cas d'usage, combiner 'acceleration de Nesterov et la méthode proximale

Ci-dessous pour $\theta = 0.5$

In [9]:
```
#Fixer la même random seed
seed = 42
np.random.seed(seed)

#Dimensions des vecteurs et matrice comme dans l'article
M=300
N=100
x = get_x(M)
A = get_A(N,M)
p = get_p(A,x)
#Nombre de réalisations de la variable y|p pour x et A donnés
trials = 50
m_max = 1
adj_param = 8 #ajouter à la fonction objective pour avoir des valeurs positives et tracer en
echelle log

prox_step = 0.5
for n in range(trials):
    #Echantillonage aléatoire de la variable y
    y,m = get_y(p,m_max)

    #Initialization
    x_0 = np.ones(x.shape)

    #Resolution sans accéleration
    #on utilise 40 iterations au total
    x_n,obj_un = unaccelerated(y,m,A,x,40,x_0,prox_step)
    adj_obj_un = np.array(obj_un)-obj_un[-1]+adj_param #Ajuster la fonction objectif

    #Resolution avec accéleration de Nesterov
    #on utilise 40 iterations au total
    x_n,obj_acc = accelerated(y,m,A,x,40,x_0,prox_step)
    adj_obj_acc = np.array(obj_acc)-obj_acc[-1] +adj_param

    #Plot fonction objectif
    plt.plot(adj_obj_un,color ='tab:blue',alpha=0.1)
    plt.plot(adj_obj_acc,color ='tab:red',alpha=0.1)


#plt.hlines([adj_param+tol,adj_param-tol],xmin=0,xmax=40, alpha=0.3,ls='--')
plt.yscale('log')
plt.ylabel('Adjusted objective (log)')
plt.xlabel('#iterations')
plt.legend(['unaccelerated','accelerated'])
```



Pour $\theta = 0.5$, on converge ≈ 2.5 fois plus rapidement avec l'accéleration de Nesterov