Introduction à l'optimisation – TP 2 : Minimisation de la fonction log barrière

Stéphane Canu

Septembre 2017, ASI, INSA Rouen

Le but du TP est de résoudre le même problème d'approximation que la semaine dernière, mais en minimisant non plus la somme des carrés des erreurs mais la fonction log barrière. La méthode consiste à estimer la fonction f supposée inconnue par une fonction connue $\hat{f}$ (ici un polynôme de degré p) dont on va trouver les coefficients $\alpha_j, j = 1, p$ en minimisant la somme des log barrière des erreurs sur les points $(x_i, y_i), i = 1, n$ dont nous disposons. Mathématiquement le problème s'écrit, pour un c donné, comme :

$$\min_{\alpha} \; L_b(\alpha) = \sum_{i=1}^{n} -c \log(1 - \frac{r_i^2}{c^2}) \qquad \text{avec} \qquad r_i = \sum_{j=0}^{p} \alpha_j x_i^j - y_i \tag{1}$$

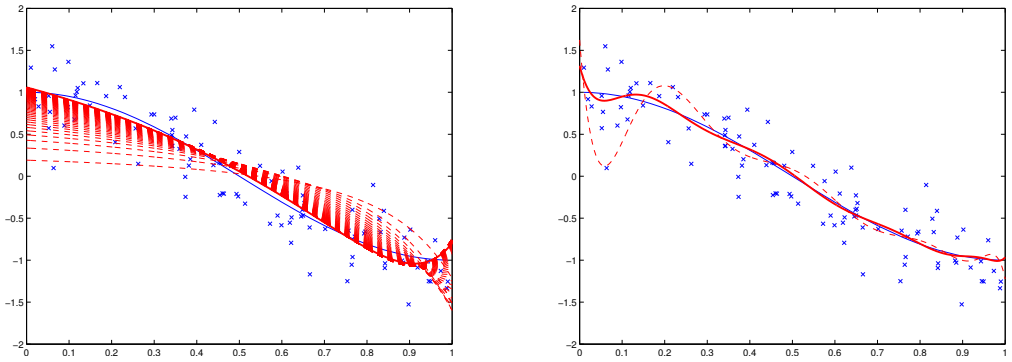

FIGURE 1 – Résultat du TP2

Ex. 1 — Minimisation de la fonction log barrière

On commencera par reprendre tout le TP de la semaine dernière

1. Essayez de résoudre ce problème à l'aide de CVX pour $c = 1,55$; qu'en pensez vous ?

```
cvx_begin
  variables ac(size(X,2),1)
      minimize( -sum(log(1-(X*ac - y).^2/c^2)) )
cvx_end
```

2. Écrire une fonction `cout` ← `cout_logB(`α`,X,y,c)`, qui, à partir des données du problème d'approximation de la semaine dernière, calcule le cout log barrière $L_b(\alpha)$

```
r = X*a-y;
cout = -c*sum(log(1-r.^2/c^2));
```

3. Calcul du gradient
   a) Écrire une fonction `g` ← `grad_logB(`α`,X,y,c)`, qui calcule le gradient cout log barrière $\nabla_\alpha L_b(\alpha)$

$$\frac{\partial L_b(\alpha)}{\partial \alpha_j} = \sum_{i=1}^{n} c \frac{2r_i}{c^2 - r_i^2} x_i^j$$

```
r = X*a-y;
df = 2*c*r./(c^2-r.^2);
g = X'*df;
```

b) Vérifier que la fonction calcule bien le gradient en proposant une autre manière de calculer les dérivées partielle, par exemple en passant par la définition. Pour cela il nous faudra choisir un α et un c. Nous prenons $\alpha = 0$ et c suffisamment grand pour ne pas avoir de problèmes numériques, car le log n'existe que pour des valeurs positives.

```
a = zeros(size(X,2),1);
err = X*a-y;
c = max(abs(err))+.01;

fc = cout_logB(a,X,y,c);
grad = grad_logB(a,X,y,c);

[n,p] = size(X);
chouia = sqrt(eps);
for i=1:p
    d = zeros(p,1);
    d(i) = 1;
    fd = cout_logB(a+chouia*d,X,y,c);
    e(i) = (grad(i) + ((fc - fd)/chouia))/grad(i);
end
```

4. Calcul de la matrice hessienne. Écrire une fonction `H` $\leftarrow$ `Hess_logB(`α`,X,y,c)`, qui calcule la matrice hessienne du cout log barrière $H_\alpha L_b(\alpha)$

$$\frac{\partial L_b^2(\alpha)}{\partial \alpha_j \alpha_j'} = \sum_{i=1}^{n} 2c \frac{c^2 + r_i^2}{(c^2 - r_i^2)^2} x_i^j x_i^{j'}$$

```
r = (X*a-y);
d2f = 2*c*(c^2+r.^2)./((c^2-r.^2).^2);
D = diag(d2f);
H = X'*D*X;
```

5. La méthode de gradient à pas fixe
   a) Résoudre le problème de minimisation en utilisant une méthode de gradient à pas fixe, où l'on choisira le pas (ni trop grand, ni trop petit !)

```
g = 1;
k = 1;
nbitemax = 1000;
fprintf(1,'--------------------------\n');
fprintf(1,' nb ite cout \n');
fprintf(1,'--------------------------\n');

while ((norm(g) > 0.005) && (k < nbitemax))

    cout = cout_logB(a,X,y,c);
    fprintf(1,'%8d %12.4f \n',k, cout);

    g = grad_logB(a,X,y,c);
    a = a - pas*g;

    if mod(k,10) == 0 ,
        plot(xt,Xt*a,'--r');
    end;
    k = k+1;
end

plot(xt,Xt*a,'r','LineWidth',2);
```

   b) Comment s'assurer que la méthode fonctionne ?
   c) Que se passe t'il si le pas est trop grand ?
   d) Que se passe t'il si le pas est trop petit ?

6. La méthode de gradient à pas variable
   a) Résoudre le problème de minimisation en utilisant une méthode de gradient à pas variable en suivant le méthode d'Armijo pour $\alpha = 0,15$ et $\beta = 2$.

```
while ((norm(g) > tol) && (k < nbitemax))

  cout = cout_logB(a,X,y,c);
  fprintf(1,'%8d %12.5f %12.8f \n',k, cout, pas);

  if (cout < cout_old)
    pas = (1+alpha)*pas;
    cout_old = cout;
  else
    a = a + pas*g;
    pas = pas/beta;
  end

  g = grad_logB(a,X,y,c);
  a = a - pas*g;

  if mod(k,10) == 0 ,
    plot(xt,Xt*a,'--r');
  end;
  k = k+1;

end
```

   b) Que se passe t'il si le pas initial est est trop grand ou trop petit ?
7. Résoudre le problème de minimisation en utilisant une méthode de gradient à pas optimal, où le pas est calculé à partir de la matrice hessienne. On ajoutera juste les instructions suivantes permettant, à chaque itération, de calculer le pas optimal.

```
   H = Hess_logB(a,X,y,c);
   pas = (g'*g) / (g'*H*g);
```

8. Résoudre le problème de minimisation en utilisant une méthode de Newton régularisée

```
lambda = sqrt(eps);
I = eye(p);

while ((norm(g) > err) && (k < nbitemax))

   cout = cout_logB(a,X,y,c);
   fprintf(1,'%8d %12.4f \n',k, cout);

   g = grad_logB(a,X,y,c);
   H = Hess_logB(a,X,y,c);

   dir = (H+lambda*I)\g;
   a = a - dir;

   if mod(k,10) == 0 ,
      plot(xt,Xt*a,'--r');
   end;
   k = k+1;

end

plot(xt,Xt*a,'r','LineWidth',2);
```

   a) Que se passe t'il si l'on ne régularise pas ($\lambda = 0$)

9. Quelle est la méthode que vous préconisez pour résoudre la problème d'optimisation (1) et pourquoi ? Simplicité, facilité de mise en œuvre, efficacité, robustesse... ? Justifiez votre réponse à l'aide d'expériences menées à partir du code ci-dessus.