L’algorithme de notre programme suit pas à pas les étapes décrites dans l’énoncé du TP. Dans un premier temps nous récupérons les données du fichier ‘*slump.data*’ grâce à la fonction *read* :

*Z = read('slump.data',-1,11);*

Une fois que nous avons récupéré le contenu du fichier dans la matrice Z, nous avons supprimé la première colonne (-1) et normalisé les données afin que toutes les variables prennent leur valeur dans le même intervalle. Cette étape est primordiale, car elle évite de ne prendre en compte que les variables qui prennent de grandes dans le problème de minimisation. Voici la fonction que nous avons créée pour normaliser notre matrice par colonne :

*function Z=normaliser(Z,A,B)*

*taille = size(Z);*

*for j = 1:taille(2)*

*max\_j = max(Z(:,j));*

*min\_j = min(Z(:,j));*

*for i = 1:taille(1)*

*Z(i,j) = (Z(i,j)-min\_j)\*(B-A)/(max\_j-min\_j) + A;*

*end*

*end*

*endfunction*

Notre fonction ‘normaliser’ prend en paramètre, en plus d’une matrice Z (entrée/sortie) les bornes respectivement inférieure et supérieure de l’intervalle de normalisation. Nous avons choisi de normaliser sur l’intervalle [0 ; 1].

Ensuite, nous avons construit la matrice des données X (7 premières colonnes) et les vecteurs de résultat ‘y8’, ‘y9’ et ‘y10’ (la colonne 8, 9 et 10).

Pour calculer les solutions de l’équation, nous avons créé la fonction ‘approxMC’ qui à partir des paramètres (X,y) calcul la solution s et le résidu r par la méthode des moindres carrés. Pour ce faire, nous avons utilisé la notion de pseudo-inverse vue en cours, qui permet de calculer la solution s. D’après le théorème 3.2.1 du cours, le problème des moindres carrés admet pour solution où est la pseudo-inverse de Moore-Penrose. Le résidu se déduit simplement de la solution trouvée par :

Pour construire la pseudo-inverse de Moore-Penrose, nous devons utiliser la décomposition en valeur singulière de notre matrice X, c’est-à-dire décomposer notre matrice sous la forme avec la matrice diagonale des valeurs singulières de X. Cette fonction étant déjà implémentée en scilab, nous n’entrerons pas plus dans les détails.

Lorsque la décomposition est faite, il faut encore inverser la matrice avant de pouvoir calculer la pseudo-inverse. L’inversion d’une telle matrice ne peut pas se faire directement à l’aide de la fonction ‘inverse’ de scilab car notre matrice des valeurs singulières n’est pas carrée. Nous avons donc créé une autre fonction qui permet d’inverser une matrice diagonale non carrée.

Voici le code obtenu pour les deux fonctions que nous venons de décrire :

*function [solution\_1,residu\_1]=approxMC(X,Y)*

*[U,D,V]=svd(X);// ‘svd’ est la fonction qui décompose en*

*Valeur singulière une matrice.*

*X\_inv = V\*(inverse(D))\*(U'); // ‘inverse’ est la fonction*

*qui permet d’inverser une*

*matrice diagonale rectangulaire*

*solution\_1 = X\_inv\*Y; // Calcul de la solution*

*residu\_1 = Y - X\*solution\_1; // Calcul du résidu*

*endfunction*

*function res=inverse(M)*

*[ligne,colonne] = size(M);*

*res = zeros(colonne,ligne);*

*for i = 1:ligne*

*for j = 1:colonne*

*if M(i,j) ~= 0 then*

*res(j,i) = 1/M(i,j);*

*end*

*end*

*end*

*endfunction*

Pour le calcul de l’erreur, nous avons simplement calculé la norme du résidu.

e8 = norm(residu8);

Ceci étant fait, il ne restait plus qu’à tracer le graph des résidus.

xset("window",1);

plot2d(1:103,residu8,style=[-5]);

Pour la neuvième étape du TP (estimer la troisième sortie en fonction des deux premières), nous avons utilisé les mêmes fonctions que celles décrites précédemment.

Par curiosité, étant donné que les résultats pour les deux premières variables de sortie étaient très proches lors de la première étape, nous avons aussi appliqué la méthode des moindres carrés pour estimer ‘y8’ en fonction de ‘y9’. Nous reviendrons sur les résultats obtenus dans la dernière partie.