Thème 3 : Algèbre linéaire et ajustements

Préambule

Ce TE est dédié à l'algèbre linéaire, et sera l'occasion d'utiliser les bibliothèques BLAS & LAPACK. Elles sont installées dans des répertoires non-standard, repérés par la variable d'environnement `REP_LAPACK` qui dépend de l'architecture utilisée (machine virtuelle 32 bits ou serveur `sappli1` 64 bits). Les fichiers pour l'utilitaire `make` feront donc référence à cette variable (syntaxe `${REP_LAPACK}`) pour être utilisables sur les deux architectures, mais objets et exécutables devront être reconstruits à chaque changement d'architecture. Pour faciliter l'écriture des appels aux procédures de ces bibliothèques, il est prudent de recopier en commentaire les prototypes fournis dans les fichiers d'entête (rechercher [1] dans `${REP_LAPACK}/include/*.h` en C), ou la documentation dans les deux langages (voir guide LAPACK et liens). Le TE comporte trois parties :

1. la première est consacrée à des manipulations élémentaires : construction de matrices, produit par différentes méthodes et comparaison des temps de calcul.
2. la seconde est consacrée à l'étude des modes propres d'une chaîne d'oscillateurs par diagonalisation.
3. la troisième porte sur l'application des méthodes d'optimisation et d'ajustement à un spectre de raies.

1 Manipulation de matrices

Dans un fichier `matelem.{c,f90}`, écrire un programme principal qui réalise l'interface utilisateur pour saisir le profil des matrices et créer les tableaux qui seront passés comme paramètres aux différentes procédures. En fortran, on utilisera des tableaux 2D alloués dynamiquement dans le programme principal. En C, on pourra utiliser les tableaux 2D automatiques du C99. Ce programme principal appellera des procédures de calcul, d'affichage et de sauvegarde sur disque.

1.1 Affichage de matrice

Dans le fichier `io.{c,f90}`, écrire une procédure `affmat` permettant d'afficher une matrice de réels de profil arbitraire. On pourra lui passer une chaîne de caractères servant de titre à l'affichage.

1.2 Construction de matrices

Dans le fichier `matrices.{c,f90}`, écrire les 3 procédures de construction des matrices de réels suivantes :

`matpow` : matrice rectangulaire A, $m \times n$ dont les éléments sont $\mathsf{A}_{ij} = i^j$;

`matrand` : matrice rectangulaire R $m \times n$ de coefficients aléatoires répartis de façon uniforme sur $[-1, 1]$, et qui sera utilisée au § 1.4 ci-dessous. Utiliser la fonction `rand()` en `C` et le sous-programme `random_number` en `fortran`.

`matl1d` : matrice carrée L_{1D}, obtenue par la discrétisation de l'équation de Laplace *à une dimension* utilisée au § 2 (*Suggestion* : utiliser une technique de balayage des diagonales avec une seule boucle par diagonale).

$$\mathsf{A} = \underbrace{\begin{bmatrix} 1^1 & 1^2 & 1^3 & \cdots & 1^n \\ 2^1 & 2^2 & 2^3 & \cdots & 2^n \\ 3^1 & 3^2 & 3^3 & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ m^1 & m^2 & \cdots & \cdots & m^n \end{bmatrix}}_{n} \Bigg\} m \qquad \mathsf{L}_{1D} = \underbrace{\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix}}_{n} \Bigg\} n$$

FIGURE 1 – *Forme des matrices* A *rectangulaire et* L_{1D} *carrée à construire*

1.3 Stockage des matrices en 1D ColMajor

L'utilisation des bibliothèques nécessite en C le stockage des matrices sous forme « aplatie ». On construit donc des outils pour transformer les matrices de tableau 2D en tableau 1D ColMajor (`aplatcolmaj`) et réciproquement (`deploiecolmaj`). Écrire à cet effet la fonction `indcolmaj` de calcul de l'indice 1D en fonction

1. Utiliser par exemple `grep -i -A 3 'LAPACKE_.sysv(' ${REP_LAPACK}/include/lapacke.h` pour afficher le prototype des variantes `ssysv` (`float`) et `dsysv` (`double`) de `sysv` ; `-A 3` affiche les 3 lignes qui suivent celle du motif.

des indices 2D, ainsi qu'une procédure `affmatapl` d'affichage d'une matrice *stockée* en tableau 1D ColMajor. Vérifier le bon fonctionnement de ces outils sur les matrices rectangulaires construites en 2D (*cf* 1.2, p. 1).
NB. : par la suite, on ne travaillera que sur les matrices aplaties en ColMajor pour ne pas dupliquer le stockage inutilement.

```
aplatcolmaj(int m, int n, float tab2d[m][n], float tab1d[m*n]);
deploiecolmaj(int m, int n, float tab1d[m*n], float tab2d[m][n]);
int indcolmaj(int m, int n, int i, int j);
affmatapl(int m, int n, float tab1d[m*n]);
```

1.4 Produit de matrices

Il s'agit ici de comparer sommairement les différentes méthodes disponibles : la méthode « naïve » qui réalise de façon élémentaire la formule du produit matriciel, la méthode fondée sur la procédure `gemm` de la bibliothèque BLAS (soit `GEMM` en fortran et `cblas_?gemm` en C) et, pour le fortran, la fonction intrinsèque `matmul`.

1.4.1 Codage de la méthode « naïve »

Écrire, dans le module `matop` fourni, une procédure `matmul_naive` qui implémente la méthode naïve. Vérifier avec des matrices aléatoires R de profil rectangulaire arbitraire, et pour des dimensions faibles ($\leqslant 5$), que les différentes méthodes donnent bien le même résultat, contrôlé par un calcul dans `Scilab4`. Effectuer en particulier le produit d'une matrice 4×5 par une matrice 5×3, qui donne une matrice 4×3.
En C, la procédure de multiplication respectera l'interface suivante :

```
void matmul_naive(int m, int n, int p, real a[m*n], real b[n*p], real prod[m*p]);
```

avec des matrices « aplaties » ordonnées en `ColMajor`, en vue de la compatibilité avec les bibliothèques.

1.4.2 Évaluation des performances

On ne cherchera pas à reconstruire de façon complète les courbes présentées en cours.
Évaluer le temps de calcul d'un produit de matrices aléatoires carrées donné, pour des tailles $n = 300$ et 1000, avec les différentes méthodes.
Les temps de calcul seront mesurés simplement en utilisant `time` sur la ligne de commande [2], à condition :

1. que les durées des entrées-sorties soient réduites au strict minimum (lectures par redirection d'entrée) ;
2. et que le temps mis à construire les matrices soit négligeable devant celui des multiplications.

Calculer ensuite les rapports des temps mesurés pour les 2 valeurs de n et comparer au rapport théorique attendu. Évaluer enfin le gain de performances apporté par BLAS pour $n = 1000$ par exemple.

2 Étude physique de L_{1D} : chaîne d'oscillateurs

2.1 Position du problème

On considère une chaîne linéaire formée de n masselottes mobiles le long d'une tige définissant l'axe x. Ces masselottes, de même masse m, et dont les positions au repos sont équidistantes de pas a, sont reliées entre elles, ainsi qu'aux extrémités fixes, par $n+1$ ressorts de raideur k (*cf* figure 2, p. 3). Cela constitue un modèle à une dimension des phonons dans un cristal à un atome par maille.
En l'absence de frottements, les écarts x_i à la position d'équilibre vérifient le système d'équations différentielles linéaires couplées :

$$\begin{array}{lcll} \text{à l'intérieur} & \ddot{x}_i + \Omega^2(2x_i - x_{i+1} - x_{i-1}) = 0 & & \text{pour} \quad i = 2, \cdots n-1 \\ \text{aux bords} & \ddot{x}_1 + \Omega^2(2x_1 - x_2) = 0 \qquad \text{et} & \ddot{x}_n + \Omega^2(2x_n - x_{n-1}) = 0 & \end{array}$$

où $\Omega^2 = k/m$. En introduisant le vecteur $\mathsf{X} = (x_1, \cdots x_n)$, ce système différentiel s'exprime grâce à la matrice L_{1D} (*cf* fig. 1) :

$$\ddot{\mathsf{X}} = \Omega^2 \, \mathsf{L}_{1D} \cdot \mathsf{X} \, , \tag{1}$$

2. Pour les méthodes rapides, il faudra éventuellement multiplier le nombre de multiplications matricielles à effectuer pour obtenir une durée mesurable. Pour une mesure de durée plus précise, et **seulement s'il vous reste du temps**, vous pourrez utiliser les fonctions de chronométrage fournies dans les modules `ustiming` (lire `us` comme μs).

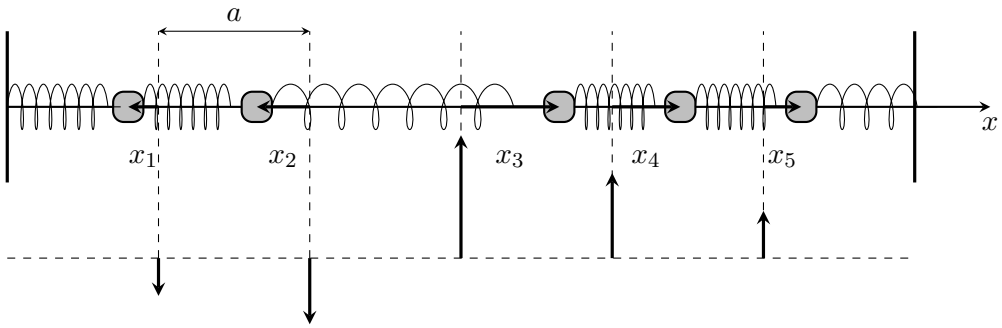

FIGURE 2 – Schéma de la chaîne d'oscillateurs ($n = 5$), et représentation symbolique d'un état quelconque (les écarts à l'équilibre x_i sont indiqués par les flèches verticales sous chaque masselotte à l'équilibre)

dont les solutions sont des fonctions sinusoïdales du temps. Les carrés des pulsations ω_p^2 et les « modes propres » (les vecteurs propres X_p) sont donnés par les valeurs propres et vecteurs propres de la matrice L_{1D}.

2.2 Calcul des éléments propres de L_{1D} par différentes méthodes

Pour déterminer les éléments propres de la matrice L_{1D}, on peut employer différentes méthodes selon les propriétés de la matrice que l'on utilise.

1. On peut utiliser la méthode générale (`GE`) en n'exploitant aucune des caractéristiques de la matrice : c'est alors la fonction `?GEEV`, (soit `LA_GEEV` en fortran et `LAPACKE_?geev` en C) qui est indiquée.
2. À l'inverse, on peut noter que la matrice L_{1D} est en fait une matrice *symétrique*, *tridiagonale*, et que $-L_{1D}$ est *définie positive*. Consulter le guide fourni pour déterminer les fonctions de LAPACK à utiliser.

2.2.1 Méthode générale

Appeler la fonction `?GEEV` directement depuis le programme principal, sans calculer les vecteurs propres. Écrire les valeurs propres ω_p^2 dans un fichier. Puis les lire dans `Scilab4`, les trier par ordre croissant (utiliser la fonction `gsort` de `Scilab`) et tracer les valeurs ω_p en fonction de fonction de p, pour $n = 4$ et $n = 5$ figure `freq.eps`.

2.2.2 Méthode utilisant les propriétés de la matrice L_{1D}

Dans une copie du programme principal, appeler la procédure spécifique[3] sélectionnée au 2.2 ci-dessus et comparer les valeurs propres. Comparer la vitesse d'exécution de ces deux méthodes pour des matrices de taille assez grande (neutraliser l'écriture sur fichier pendant cette analyse).

2.2.3 Application aux phonons

1. Comme on obtient n valeurs propres pour un problème de taille n, le tracé de ω_p en fonction de p n'est pas adapté à la comparaison des résultats. Il faut en fait tracer ω_p en fonction de p/n ou de façon plus physique en fonction du vecteur d'onde $k_p = p\pi/L$ où L est la longueur totale du système, soit $L = (n+1)a$ (on pourra prendre $a = 1$ pour simplifier).
   En superposant ainsi les résultats pour $n = 5$, $n = 10$ et $n = 100$, figure `dispersion.eps`, montrer que l'on obtient à la limite $n \longrightarrow \infty$ une « relation de dispersion » continue $\omega(k)$. Utiliser la fonction `Scilab4` fournie `multiplot.sci` pour superposer les courbes. Quelle est alors la vitesse du son ?
2. Pour visualiser la structure des modes, ajouter le calcul et l'écriture des vecteurs propres dans un fichier. Représenter – à la main – les vecteurs propres obtenus pour $n = 4$ par un schéma analogue à celui de la figure 2. Formater le fichier de sortie de façon à ce que le diagramme puisse être tracé par `Scilab4` à l'aide la fonction `plot2d3` et commenter la structure du mode $p = 30$ pour $n = 200$.

3. Utiliser le stockage spécifique aux matrices tridiagonales qui est décrit dans la documentation, mais pas dans le guide de LAPACK distribué.

3 Ajustement linéaire d'un spectre de raies

3.1 Motivation physique : détermination des concentrations

Il est fréquent – notamment pour la physico-chimie de l'atmosphère ou de systèmes extraterrestres, que l'on ait accès à un spectre d'émission ou d'absorption dans le domaine visible ou infrarouge, avec des raies relativement bien identifiées, qui peuvent permettre de déterminer l'abondance relative de différents constituants. Il faut alors faire un ajustement du spectre sur un spectre théorique, où les paramètres ajustables sont simplement l'intensité et la largeur des différentes raies. Cette situation est illustrée par le jeu de données de la figure 3, très analogue à celui que nous allons traiter.

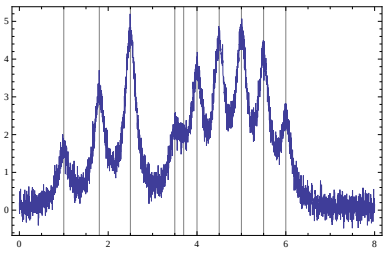

FIGURE 3 – Exemple de spectre

3.2 Modélisation et inversion

Le modèle à ajuster est une somme de m de raies lorentziennes, dont on suppose connues positions et largeurs, et dont les intensités sont les paramètres ajustables $\mathsf{p} = (p_1, \cdots p_m)$. Il est donc *linéaire* à m paramètres :

$$y(x) = \sum_{j=1}^{m} p_j \; F_j(x) \quad \text{où} \quad F_j(x) = \frac{(\gamma_j/2)^2}{(x-c_j)^2 + (\gamma_j/2)^2} \; .$$

Pour déterminer les paramètres p, il faut minimiser « l'erreur » :

$$\mathrm{Err}(\mathsf{p}) = \sum_{i=1}^{n} \left(\frac{y_i - y(x_i, \mathsf{p})}{\sigma_i} \right)^2 = \sum_{i=1}^{n} \left[\frac{y_i - \sum_{k=1}^{m} p_k \, F_k(x_i)}{\sigma_i} \right]^2 .$$

Cela est obtenu en résolvant les « équations normales » : $\mathsf{A} \cdot \mathsf{p} = \mathsf{b}$, où A est une matrice symétrique $m \times m$ et b un vecteur à m composantes ainsi définis :

$$\sum_{j=1}^{m} \mathsf{A}_{kj} \, \mathsf{p}_j = \mathsf{b}_k \quad \text{avec} \quad \mathsf{A}_{kj} = \sum_{i=1}^{n} \frac{F_k(x_i) \, F_j(x_i)}{\sigma_i^2} \quad \text{et} \quad \mathsf{b}_k = \sum_{i=1}^{n} \frac{y_i \, F_k(x_i)}{\sigma_i^2} \; .$$

La matrice C des covariances est alors l'inverse de la matrice A.

3.3 Travail demandé

La liste des positions et des largeurs est donnée dans le fichier `param.txt` (avec $m = 10$), et les spectres mesurés dans les fichiers `data`i`.txt`, qui diffèrent par le nombre de points, le plus petit servant uniquement à la mise au point du code.

Codes fournis On fournit le module `io` pour la lecture des fichiers de données `param.txt` et `data`i`.txt` et l'écriture des résultats, ainsi que le module `modele_lorentz` pour le calcul des lorentziennes et du modèle.

Programmation Le code à écrire aura la structure suivante :

- Un programme principal `lsqmain` qui alloue les tableaux, lit le fichier des paramètres connus c_j et γ_j d'une part, lit les données `x`, `y`, `sig` dans le fichier, puis appelle la procédure `lsqlin` d'ajustement linéaire. Le programme sauvergarde ensuite dans le fichier `ajust.dat` les valeurs des x_i, y_i, $y(x_i, \mathsf{p})$ à visualiser figure `ajust.eps`. Il écrit enfin les paramètres p, la covariance C et l'erreur dans le fichier `param-out.dat`.
- La procédure `lsqlin` doit
  - construire la matrice A et le vecteur b des équations normales. On veillera à organiser les calculs de façon à minimiser le nombre d'évaluations des fonctions $F_j(x_i)$.
  - résoudre les équations normales au moyen de procédures appropriées de BLAS et LAPACK, et renvoyer le vecteur p des paramètres et la matrice des covariances C.
  - calculer la valeur `err` de l'erreur, qui devrait suivre une loi du χ_2.

  Son interface sera :
  - en fortran : `lsqlin(x, y, sig, c, ga, par, covar, err)`,
  - en C99 : `lsqlin(int n, int m, real x[n], real y[n], real sig[n], real c[m], real ga[m], real par[m], real covar[m*m], real *err)` .