Projet Maple
ULIN 405 2005/2006

Rodolphe Giroudeau

Lirmm
Université de Montepllier II UMR 5056 CNRS 161, rue Ada
34392 MONTPELLIER Cedex 5
TEL: 04-67-41-85-40
Mail : rgirou@lirmm.fr

Préliminaires : Le projet doit être réalisé et rédigé en binôme ou, le cas échéant, par une seule personne. Le soin de la rédaction est un point important de l'évaluation du projet.

- Le compte-rendu du projet est une (ou plusieurs) *feuille(s) de travail Maple* qui est annotée (soit directement en tant que commentaires insérés dans la feuille de travail, soit à l'aide de feuilles manuscrites jointes à la feuille Maple).
- La note attribuée au projet tiendra compte de l'usage de Maple en fonction des problèmes posés (par exemple, il peut être intéressant de signaler lorsque plusieurs solutions sont possibles en Maple ou bien lorsque Maple présente des faiblesses). Notons que l'utilisation de tout savoir mathématique est permise pour peu que ce dernier soit explicitemement rappelé lorsqu'il sort des connaissances considérées comme classiques.
- Les fonctions (procédures) devront être systématiquement *illustrées à l'aide d'exemples* couvrant les différentes situations possibles.
- Les exercices sont numérotés et à priori indépendants les uns des autres. Veuillez référencer correctement les numéros des exercices que vous traitez. Si vous ne trouvez pas la solution d'un exercice donné, vous pouvez à défaut donner les voies infructueuses que vous avez suivies en utilisant Maple. **Indiquer clairement en tête de votre projet quels sont les exercices que vous avez traité**. Notez que tout exercice entamé sera noté sur sa totalité. minimum).
- La liste des participants devra être rendu avant la fin janvier au secrétariat.

Les projets devront être rendus au plus tard le 5 mai 2006. Ils seront déposés à la scolarité et une version éléctronique me sera envoyer.

Exercice 1 : Somme des puissances d'entiers et polynômes de Bernoulli
Pour n, p entiers naturels on pose $S(n,p) = \sum_{k=0}^{n-1} k^p = 0^p + 1^p + \cdots + (n-1)^p$. Les formules suivantes sont bien connues :

$$S(n,0) = n, \qquad S(n,1) = \frac{n(n-1)}{2}, \qquad S(n,2) = \frac{n(n-1)(2n-1)}{6}.$$

De manière générale on démontre que $S(n,p)$ est un polynôme de degré $p+1$ en n que l'on notera Q_p. L'objet de cet exercice est l'étude des polynômes Q_p ainsi que de leurs dérivées, $B_p = Q'_p$. Les polynômes B_p sont appelés *polynômes de Bernoulli* et interviennent dans de nombreuses formules d'analyse ; les *nombres de Bernoulli* sont définis par $b_p = B_p(0)$.

1. **Observation des polynômes Q_p.**
   Faire calculer par la machine les sommes $S(n,p) = Q_p(n)$ pour $p \leq 10$, les mettre sous forme factorisée et faire tracer les courbes des polynômes Q_p pour x variant entre 0 et 1. On fera le maximum de conjectures au vu des résultats. Ces conjectures seront (peut-être) démontrées formellement dans les questions suivantes.
2. **Calcul formel des Q_p.**
   Il est possible de calculer les polynômes Q_p sans faire appel aux formules préprogrammées dans Maple, il suffit de revenir à la définition des $S(n,p)$ et d'admettre provisoirement que Q_p est un polynôme de degré $p+1$. On a :

$$\forall\ n \in \mathbb{N},\ \mathrm{Q_p(n)} = \sum_{\mathrm{k=0}}^{\mathrm{n-1}} \mathrm{k^p}$$

   soit :

$$Q_p(0) = 0 \quad \text{et} \quad \forall\ n \in \mathbb{N},\ \mathrm{Q_p(n+1) - Q_p(n) = n^p}. \qquad (*)$$

   Il suffit donc de donner des coefficients indéterminés à Q_p, de calculer formellement $Q_p(n+1) - Q_p(n)$ et d'identifier le résultat avec n^p. Les instructions suivantes calculent Q_7, les recopier et les faire exécuter, puis les adapter pour calculer $Q_0, \ldots, Q_{10}$.

```
p := 7;
i := 'i';
Q := sum('a.i*n^i',i=1..p+1);
R := collect(subs(n=n+1,Q) - Q - n^p,n);
inconues  := {seq(a.i,i=1..p+1)};
equations := {seq(coeff(R,n,i)=0,i=0..p+1)};
solution  := solve(equations, inconues);
Q.p := subs(solution,Q);
```

   Remarque : méfiez-vous de `solve`, les algorithmes que cette fonction met en jeu sont inconnus et non fiables, il est possible que `solve` ne fournisse pas toutes les solutions ou, plus rarement, qu'elle fournisse des non-solutions. On admettra que `solve` fournit un résultat exact pour un système d'équations linéaires ou pour une unique équation algébrique de degré inférieur ou égal à 4 car il existe des algorithmes démontrés valides pour ces cas et on peut supposer que Maple les utilise ; dans tous les autres cas, un résultat fourni par `solve` n'a aucune valeur probante.

Les calculs effectués *prouvent* donc formellement l'existence, l'unicité et le caractère polynomial de Q_p pour $p \leq 10$ puisque les relations $(*)$ définissent à p fixé une et une seule suite. Pour montrer l'existence, l'unicité et le caractère polynomial de Q_p pour p quelconque il suffit de prouver que le système d'équations soumis à `solve` admet une unique solution, donner une justification mathématique de ceci.

3. **Polynômes d'Euler.**
   En vous inspirant de ce qui précède, prouvez que les sommes alternées : $A(n,p) = \sum_{k=0}^{n-1}(-1)^k k^p$ sont de la forme $A(n,p) = (-1)^n R_p(n) + K_p$ où R_p est un polynôme de degré p et K_p une "constante" fonction de p et calculer par la méthode des coefficients indéterminés les premiers polynômes R_p.
4. **Polynômes et nombres de Bernoulli.**
   On note $B_p = Q'_p$. Calculer les premiers polynômes B_p et trouver expérimentalement une relation entre les familles (B_p) et (Q_p). Pour prouver formellement que la relation précédente a lieu pour tout p, faire calculer et simplifier $B_p(n+1) - B_p(n)$, justifier mathématiquement les résultats observés et terminer la démonstration.
5. **Formule de récurrence.**
   Comme la famille (B_p) satisfait au théorème des degrés étagés, c'est une base de $R[x]$ et donc en particulier les monômes x^p peuvent s'exprimer comme combinaisons linéaires des B_k. Faire effectuer la décomposition par Maple pour les premières valeurs de p. On formera une combinaison linéaire des B_k à coefficients indéterminés et on identifiera cette combinaison à n^p grâce à l'incontournable `solve`. Comme pour les questions précédentes, conjecturer une formule simple puis la démontrer. En déduire une relation de récurrence sur les nombres de Bernoulli, $b_p = B_p(0)$ et programmer le calcul des b_p à l'aide de cette relation.

Exercice 2 : Résolution itérative d'un système linéaire

Soit $(S) \longleftrightarrow AX = B$ un système d'équations linéaires avec $A \epsilon \mathcal{M}_n(\mathbb{R})$ supposée inversible et $X, B \epsilon \mathcal{M}_{n,1}(\mathbb{R})$. La méthode du pivot de Gauss permet de résoudre ce système de manière exacte, à la précision des calculs près, mais est inutilisable en pratique pour n grand car le nombre d'opérations à effectuer croît comme n^3, et les erreurs d'arrondi cumulées font chuter la précision du résultat final. On étudie dans cet exercice des méthodes de résolution *itératives* qui ont un temps de calcul de l'ordre de n^2, et pour lesquelles les erreurs d'arrondi ne se cumulent pas.

Vous utiliserez la bibliothèque `linalg` ainsi que les fonctions suivantes qui tirent respectivement une matrice carrée et un vecteur aléatoires :

```
# mathasard(n,a,b) tire une matrice n*n au hasard
# avec des coefficients compris entre a et b
mathasard := proc(n::integer,a::numeric,b::numeric)
local M,i,j;
  M := matrix(n,n);
  for i from 1 to n do for j from 1 to n do
```



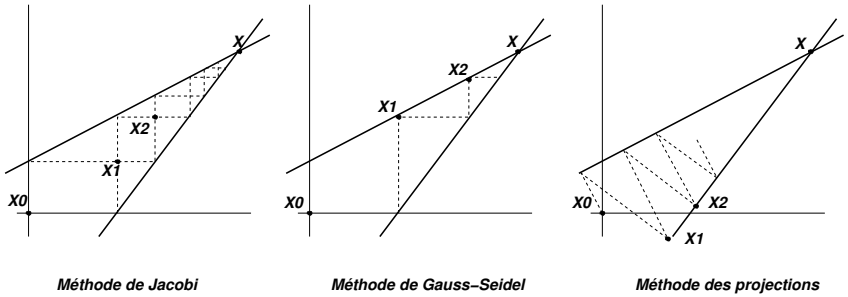
```
    M[i,j] := a + (b-a)*rand()*Float(1,-12);
  od od;
  evalm(M);
end:

#vecthasard(n,a,b) tire un vecteur n au hasard
#avec des coefficients compris entre a et b
vecthasard := proc(n::integer,a::numeric,b::numeric)
local V,i;
  V := vector(n);
  for i from 1 to n do V[i] := a + (b-a)*rand()*Float(1,-12); od;
  evalm(V);
end:
```

Pour les essais, prendre une matrice A et un second membre aléatoires. On note $X = (x_1, \ldots, x_n)$ la solution exacte de (S) et $X_0, X_1, \ldots, X_k, \ldots$ des approximations successives de X : $X_k = (x_{k,1}, \ldots, x_{k,n})$. Prendre comme approximation initiale $X_0 = 0$ et faire afficher les vecteurs X_k.

1. **Méthodes de Jacobi et Gauss-Seidel**
   Méthode de Jacobi : $x_{k,i}$ est calculé à partir de l'équation numéro i de (S) dans laquelle on remplace toutes les autres inconnues par les coefficients correspondants de X_{k-1}.
   On a donc : $x_{k,i} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j \neq i} a_{ij} x_{k-1,j}\Big)$.
   Méthode de Gauss-Seidel : Même principe, mais en utilisant les les coefficients de X_k déja calculés à la place de ceux de X_{k-1} dans les équations suivantes :
   $x_{k,i} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij} x_{k,j} - \sum_{j>i} a_{ij} x_{k-1,j}\Big)$.
   Écrire les fonctions : `jacobi(A::matrix, B::vector, X::vector)` et `gaussseidel(A::matrix, B::vector, X::vector)` qui calculent le vecteur X_k en fonction de X_{k-1}.
2. **Analyse de convergence**
   Vérifier qu'en général, les méthodes de Jacobi et Gauss-Seidel divergent...
   On démontre cependant leur convergence dans les deux cas particuliers suivants :

   • A est à diagonale dominante, c'est à dire : $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ pour tout i.
   • (méthode de Gauss-Seidel) A est symétrique définie positive, c'est à dire $A = {}^tA$ et pour tout $X \epsilon \mathbb{R}^n$ non nul, ${}^tXAX > 0$.
   Vérifier cela. Pour obtenir une matrice aléatoire à diagonale dominante, tirer A au hasard, puis ajouter un nombre suffisament grand sur la diagonale. Observer l'effet de ce coefficient sur la vitesse de convergence. Pour obtenir une matrice aléatoire symétrique définie positive, tirer M au hasard, et prendre $A = {}^tMM$. Cette matrice est toujours symétrique positive ; elle est définie positive si M est inversible, ce qui est presque toujours le cas.
3. **Méthode des projections orthogonales**
   Géométriquement, chaque équation de (S) peut être interprétée comme l'équation d'un hyperplan de $\mathbb{R}^n$. Le vecteur X solution de (S) représente le point intersection de n hyperplans. Pour $n = 2$ les dessins suivants illustrent les méthodes de Jacobi, Gauss-Seidel et la méthode des projections présentée ci-dessous.



Méthode de Jacobi *Méthode de Gauss–Seidel* *Méthode des projections*

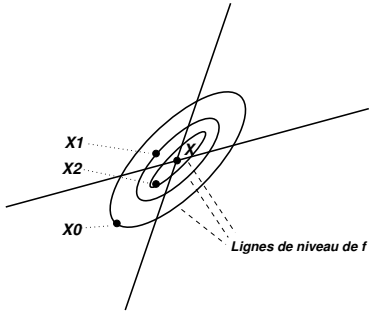
La méthode des projections orthogonales consiste à aller vers un hyperplan en suivant la direction orthogonale à cet hyperplan, et non la direction d'un axe de coordonnée. Autrement dit, pour passer de X_{k-1} à X_k, on projette X_{k-1} sur le premier hyperplan, H_1, puis on projette ce projeté sur H_2, et ainsi de suite jusqu'au projeté sur H_n qui est X_k.
Écrire une fonction `projections(A::matrix, B::vector; X::vector)` qui calcule X_k à partir de X_{k-1} suivant cette méthode. On rapelle que si H est un hyperplan d'équation : $a_1x_1 + \cdots + a_nx_n = b$, alors le projeté orthogonal du vecteur $(x_1, \ldots, x_n)$ sur H pour le produit scalaire usuel de $\mathbb{R}^n$ est :

$$(x_1, \ldots, x_n) - t(a_1, \ldots, a_n) \text{ avec } t = \frac{a_1x_1 + \cdots + a_nx_n - b}{a_1^2 + \cdots + a_n^2}.$$

Bien qu'apparament plus lente que les méthodes de Jacobi et Gauss-Seidel, la méthode des projections converge pour toute matrice A inversible ; pouvez-vous expliquer pourquoi ?

4. **Méthode du gradient**
   Dans cette méthode, on reformule le problème : *¡¡ trouver X tel que $AX = B$ ¿¿* en : *¡¡ trouver X tel que $\|AX - B\|^2$ soit minimal ¿¿* où $\|\ \ \|$ désigne la norme euclidienne usuelle sur $\mathbb{R}^n$. On introduit donc la fonction $f : X- > \|AX - B\|^2$ de $\mathbb{R}^n$ dans $\mathbb{R}$. Le point X_k est déterminé à partir de X_{k-1} en recherchant le minimum de f sur la droite passant par X_{k-1} et dirigée par le gradient de f en X_{k-1} (ce gradient est orthogonal à la surface de niveau de f passant par X_{k-1}, et indique dans quelle direction f varie le plus vite localement). On montre à l'aide de développements limités que $\nabla f(X) = 2\,{}^tA(AX - B)$ et que $f(X - \lambda U)$ est minimal quand λ décrit $\mathbb{R}$ pour $\lambda = {}^tU\,{}^tA(AX - B)/\|AU\|^2$ (le vérifier).
   Écrire une fonction `gradient(A::matrix, B::vector, X::vector)` qui effectue une itération de la méthode du gradient et vérifier expérimentalement la convergence de cette méthode.

Exercice 3 : Les tableaux et les tris
Un tableau est une srucutre de données qui permet de repèrer ses éléments par des



Méthode du gradient

indices entiers. Ainsi, pour un tableau T à une dimension, $T[i] = k$ désigne le ième élément du tableau. Dans cet exercice, nous travaillerons uniquement sur des tableaux à une dimension de taille n. Ecrire les procédures qui effectue les instructions suivantes :

1. Initialise un tableau aléatoirement avec des valeurs entre 0 et 10. (pour cela utiliser la fonction *rand*) : *Initialise(T,n).*
2. Calcule la moyenne d'un tableau : *Moyenne(T,n).*
3. Recherche l'élément maximal du tableau : *Max(T,n).*
4. Effectue la somme de 2 tableaux dans un 3ème : *Somme (T1,T2,n).*
5. Ecrire une procédure qui fait une permutation cyclique des éléments du tableaux de un pas vers la droite. Si on a un tableau T de taille n, on permute de la manière suivante : pour $i \in \{2..n\}, T[i] = T[i-1]$, pour $i = 1$ $T[i] = T[n]$: *Permutation.*

Maintenant, nous voulons trier un tableau de taille n. Pour cela, deux méthodes se présentent à nous :

1. **Le tri Sélection :**
   Ce tri est basé sur l'idée suivante :
   on recherche le plus petit élément de la liste et on le met dans la première case. Ensuite on recherche le deuxième plus petit élément de la liste et on le place dans la deuxième case... etc
   On a donc l'algorithme suivant (liste de taille n)

   ```
   pour i=1 à n faire :
   Rechercher le plus petit élément parmi les éléments l[i],..l[n]
   Si j est son indice, échanger les éléments en position i et j
   ```

2. **Le tri Insertion :**
   Ce tri est basé sur l'idée suivante :
   on classe les deux premiers éléments de la liste, puis on insère le troisième au sein des dexu éléments classés. Ensuite, on recommence avec le 4ème .. jusqu'au nme.
   On a donc l'algorithme suivant (liste de taille n)

```
Classer les 2 premiers éléments.
pour i=3 à n faire :
insérer le ième élément parmi les (i-1) premiers.
```

1. Programmez la méthode du tri par sélection avec pour paramètres d'entrée le tableau T et la taille n du tableau. Pour cela, aidez-vous d'une fonction auxiliaire *echange* qui echange l'emplacement de deux éléments.
   Remarque : vous ne pouvez-vous faire les modifications directement sur le tableau T passé en paramètre, vous devez le copier dans un tableau déclaré en local.
2. Programmez la méthode du tri par insertion avec pour paramètres d'entrée le tableau T et la taille n du tableau.
   Remarque : vous ne pouvez-vous faire les modifications directement sur le tableau T passé en paramètre, vous devez le copier dans un tableau déclaré en local.
3. Nous voulons comparer les temps d'exécution des deux méthodes. Programmez une fonction *compare* qui compare les temps d'exécution des deux méthodes. La taille du tableau varie entre 100 et 300 avec des palier de 100 (plus si vous pouvez). Tracez sur un même graphe les deux courbes de temps mesurés. Conclusion ?

Exercice 4 : Polynômes d'interpolation de Lagrange

Soient $x_0, x_1, \ldots, x_n$, $n+1$ valeurs deux à deux distinctes, on définit le $i$ème polynôme d'interpolation de Lagrange pour la suite de points x_j :

$$L_i(T) = \frac{\prod_{j \neq i}(T - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

1. Montrez que $L_i(x_i) = 1$ et $L_i(x_j) = 0$, si $j \neq i$.
2. Montrez que le polynôme $p(T) = \sum_{i=0}^{n} y_i L_i(T)$ est de degré n et satisfait $p(x_i) = y_i$, pour tout i de 0 à n.
3. Écrivez une procédure `lagrange`(x,y,T), prenant en arguments deux listes x et y de même longueur, le nom de variable T et retournant le polynôme p de la question précédente.
4. Comparez votre fonction avec la fonction `interp`.

Exercice 5 : Encryption

1. Classiquement pour crypter un message, on le transcrit d'abord en un nombre à l'aide de la correspondance suivante entre les lettres de l'alphabet et les nombres de 1 à 26: A=01, B=02, ..., Z=26.
   Ecrivez une procédure qui à une liste, dont les entrées sont les lettres d'un message, renvoie le nombre qui lui correspond. Appliquez cette procédure pour coder "AUSSI" (On pourra utiliser une liste [A,B,..,Z] pour représenter l'alphabet).
2. Ecrire la procédure de décodage qui à un nombre retourne, quand cela est possible, le message correspondant. On pourra noter qu'il s'agit d'une modification légère de l'algorithme qui écrit un nombre en base 100. Appliquer décode au nombre code([A,U,S,S,I]) pour la vérification.

3. **Encryption exponentielle de Pohlig-Hellman**
   Principe de codage: Soient p un nombre premier et e une clef telle que $pgcd(e, p-1) = 1$.
   On transforme un nombre $m < p$ ("message") en $E(m) = m^e \ mod(p)$. Bien sur si le message est plus grand que p, on le découpe en $m_1 m_2 ... m_r$ où m_i (pour i = 1,...r-1) a k lettres ($k < p$) et m_r a au plus k lettres.
   Ecrivez un programme qui a une liste $L1$ retourne son encryption par la méthode de Pohlig-Hellman. On pourra mettre le résultat dans un liste $[m_1 ... m_r]$.
   Utilisez le pour crypter "MEFIEZVOUSDESEVIDENCES" pour p=4578971, e=3317271 et k=2. Vérifier que le programme est correct en décryptant directement le résultat.
4. **Encryption à clef dite publique: RSA**
   Le principe général des systèmes d'encryption à clefs publiques réside en l'utilisation d'une fonction arithmétique f (déterminée par la "clef publique") qu'il est quasiment impossible à inverser sans la connaissance d'une clef secrète.
   Exemple: $f(p, q) = pq$ pour p, q premiers. Il est très difficile de trouver la décomposition pq d'un grand nombre produit de deux nombres premiers (il n'y a pas d'algorithme satisfaisant).
   C'est sur cette fonction qu'est basée l'encryption RSA, qui est une très légère modification de l'encryption de Pohlig-Hellman, rendant très difficile la détermination de la fonction décryptante (i.e. d tel que $m = E(m)^d \ mod(n)$).
   RSA: On choisit un nombre n produit de deux nombres premiers p, q. On choisit également une clef dite publique e telle que $pgcd(e, \phi(n)) = 1$, de sorte que e a un inverse d modulo $\phi(n) = (p-1)(q-1)$. On pose alors $E(m) = m^e \ mod(n)$, pour $m < n$. D'après le théorème d'Euler $x^{\phi(n)} = 1 \ mod(n)$, donc puisque $ed - 1$ est un multiple de $\phi(n)$, $m^{ed} = m \ mod(n)$.

   Supposez que deux utilisateurs utilisent le système RSA pour crypter leurs messages avec le même nombre n = 5038301 et des clefs publiques respectives $e_1 = 787$, $e_2 = 6785$. Supposez qu'ils envoient respectivement un message crypté $m_1 = 21051$ et $m_2 = 4953713$.

   - (a) Vérifier que e_1 et e_2 sont premiers entre eux. Quels sont leurs coefficients de Bezout x et y (donnés par **igcdex**)?
   - (b) Calculer $mm = m_1^x * m_2^y \ mod(n)$, décodez le, puis calculez $mm^{e_1} \ mod(n)$ et $mm^{e_2} \ od(n)$. Commentez.