Calcul Formel et Sciences de la Matière — Janvier 2006
Durée 3 heures — Documents et calculatrices interdits

Prenez le temps de lire ce sujet. Les questions sont indépendantes pour la plupart. L'énoncé est un peu long mais il est noté sur 26 points.

1 Le pivot de Gauss (13 points)

Cette série d'exercices consiste à comprendre et à améliorer les fonctions MAPLE suivantes. On suppose le paquetage *LinearAlgebra* chargé en mémoire.

La fonction suivante applique le pivot de Gauss à « matrice ».

```
pivot_Gauss := proc (matrice)
    local M, n, m, lig, col, lbar;
    M := Copy (matrice);
    n := RowDimension (M);
    m := ColumnDimension (M);
    col := 1;
    lig := 1;
    while lig <= n and col <= m do
        lbar := cherche_pivot (M, lig, col);
        if lbar <> lig then
            permute_lignes (M, lig, lbar)
        fi;
        for lbar from lig+1 to n do
            soustrait_lignes (M, lbar, lig, M[lbar,col]/M[lig,col])
        od;
        lig := lig + 1;
        col := col + 1
    od;
    M
end;
```

La fonction suivante cherche un élément non nul sur la colonne col *de* M *à partir de l'indice de ligne* lig*. Cette fonction suppose qu'un tel élément existe. Elle retourne l'indice du premier élément non nul rencontré.*

```
cherche_pivot := proc (M, lig, col)
    local lbar;
    lbar := lig;
    while M [lbar, col] = 0 do
        lbar := lbar + 1
    od;
    lbar
end;
```

La fonction suivante soustrait k fois la ligne d'indice lig *de M à la ligne d'indice* lbar. *La matrice M passée en paramètre est modifiée.*

```
soustrait_lignes := proc (M, lbar, lig, k)
    ...
end;
```

La fonction suivante permute les lignes d'indices lig *et* lbar *de M. La matrice M passée en paramètre est modifiée.*

```
permute_lignes := proc (M, lig, lbar)
   ...
end;
```

1.1 Connaissance du logiciel

Question 1 [2 pts]. Donner une suite d'instructions qui affecte à une variable M la matrice ci–dessous et qui lui applique la fonction *pivot_Gauss*. Utiliser la notation à base de «$<$», «$>$», «,» et «$|$».

$$M = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 7 & 3 \end{pmatrix}$$

Solution.

```
M := <<1, 2> | <2, 7> | < 2, 3>>;
pivot_Gauss (M);
```

1.2 Compréhension de code

Question 2 [2 pts]. Combien de fois la fonction *cherche_pivot* est–elle appelée sur l'exemple ci–dessus ? Pour chacun des appels faits à *cherche_pivot*, expliciter tous les paramètres effectifs de l'appel.

Solution. Deux fois. Premier appel :

$$M = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 7 & 3 \end{pmatrix}, \quad \ell = 1, \quad c = 1$$

Second appel :

$$M = \begin{pmatrix} 1 & 2 & 2 \\ 0 & 3 & -1 \end{pmatrix}, \quad \ell = 2, \quad c = 2$$

Question 3 [1 pt]. Quelle est la valeur retournée par *pivot_Gauss* sur l'exemple ci–dessus ?

Solution.

$$\begin{pmatrix} 1 & 2 & 2 \\ 0 & 3 & -1 \end{pmatrix}$$

1.3 Programmation

Il est normalement interdit dans les fonctions MAPLE de modifier un paramètre formel. Il y a une exception à cette règle dans le cas où le paramètre est une matrice M. Dans ce cas, toute affectation dans la fonction appelée, faite sur une entrée $M[\ell, c]$, est répercutée sur le paramètre effectif dans la fonction appelante.

Question 4 [2 pts]. Écrire en MAPLE la fonction *soustrait_lignes*. Les spécifications et l'entête de cette fonction sont données en haut de la feuille. Votre fonction doit modifier la matrice M qui lui est passée en paramètre.

Solution.

```
soustrait_lignes := proc (M, lbar, lig, k)
    local col;
    for col from 1 to ColumnDimension (M) do
        M [lbar, col] := M [lbar, col] - k * M [lig, col]
    od
end;
```

1.3.1 Stratégie alternative de recherche du pivot

La fonction *cherche_pivot* retourne l'indice de ligne du premier élément non nul rencontré sur la colonne *col* de M. Dans le cas d'une matrice à coefficients inexacts, on peut améliorer la précision du résultat en retournant l'indice de ligne du plus grand élément non nul (en valeur absolue) présent sur la colonne *col*.

Question 5 [2 pts]. Modifier la fonction *cherche_pivot* pour qu'elle retourne l'indice de ligne du plus grand élément non nul (en valeur absolue) de la colonne *col*. On suppose que la colonne *col* comporte au moins un élément non nul. On peut utiliser la fonction *abs* de MAPLE.

Solution.

```
cherche_pivot := proc (M, lig, col)
    local lmax, lbar;
    lmax := lig;
    for lbar from lig+1 to RowDimension (M) do
        if abs (M [lbar, col]) > abs (M [lmax, col]) then
            lmax := lbar
        fi
    od;
    lmax
end;
```

1.3.2 Cas des chutes de rang

Toujours dans le cadre de la recherche d'un pivot, on souhaite maintenant prendre en compte la possibilité que tous les éléments de la colonne *col* soient nuls (dans ce cas, il n'y a pas de pivot sur la colonne *col*). Il y a deux modifications à faire : une dans la fonction *cherche_pivot* et une dans la fonction *pivot_Gauss*.

Question 6 [2 pts]. Modifier la fonction *cherche_pivot* pour qu'elle retourne -1 si tous les éléments de la colonne *col* sont nuls. Si au moins un élément de la colonne est non nul, la fonction doit retourner l'indice de ligne du plus grand élément non nul (en valeur absolue).

Solution.

```
cherche_pivot := proc (M, lig, col)
    local lmax, lbar;
    lmax := lig;
    for lbar from lig+1 to RowDimension (M) do
        if abs (M [lbar, col]) > abs (M [lmax, col]) then
            lmax := lbar
        fi
    od;
    if M [lmax, col] <> 0 then
        lmax
    else
        -1
    fi
end;
```

Question 7 [2 pts]. Modifier *pivot_Gauss* pour qu'elle prenne en compte le fait que *cherche_pivot* peut retourner -1. Dans ce cas, il suffit d'incrémenter l'indice de colonne et de passer à l'itération suivante. Ne recopier que la boucle *while* de *pivot_Gauss*.

Solution.

```
    while lig <= n and col <= m do
        lbar := cherche_pivot (M, lig, col);
        if lbar <> -1 then
            if lbar <> lig then
                permute_lignes (M, lig, col)
            fi;
            for lbar from lig+1 to n do
                soustrait_lignes (M, lbar, lig, M[lbar,col]/M[lig,col])
            od;
            lig := lig + 1
        fi;
        col := col + 1
    od;
```

2 Intégration d'équations différentielles (5 points)

On considère l'équation différentielle ordinaire avec condition initiale suivante. On cherche à l'intégrer sur un intervalle de temps $[a,\, b] = [0,\, 1]$.

$$\dot{x} = t\,x + 1, \quad x(a) = 0.$$

2.1 Connaissance du logiciel

Question 8 [2 pts]. Donner une suite d'instructions MAPLE qui affecte l'équation à une variable *edo*, qui l'intègre numériquement (avec le solveur de MAPLE) et qui affecte le résultat à une

variable *sol*. Quelle est la nature de *sol* (une expression, une matrice, une fonction ...) ?

Solution. Le résultat est une fonction $t \mapsto [t,\, x(t)]$.

```
edo := diff (x(t),t) = t*x(t) + 1;
sol := dsolve ({edo, x(0)=0}, x(t), numeric);
```

2.2 Connaissance de la méthode d'Euler

On souhaite appliquer la méthode d'Euler à cette équation, pour un nombre de pas N.

Question 9 [1 pt]. Quelle est la suite (x_n) construite par la méthode d'Euler (donner la formule) ?

Solution. Avec $h = (b-a)/N$ on a :

$$t_0 = a, \quad x_0 = 0, \quad t_{n+1} = t_n + h, \quad x_{n+1} = x_n + h\,(t_n\, x_n + 1).$$

Question 10 [1 pt]. Donner la liste des couples $(t_n,\, x_n)$ calculés dans le cas où $N = 1$.

Solution. On a $h = 1$.

$$\begin{array}{rcl} t_0 & = & 0 \\ x_0 & = & 0 \\ t_1 & = & 1 \\ x_1 & = & 1 \end{array}$$

Question 11 [1 pt]. Donner la liste des couples $(t_n,\, x_n)$ calculés dans le cas où $N = 2$.

Solution. On a $h = 1/2$.

$$\begin{array}{rcl} t_0 & = & 0 \\ x_0 & = & 0 \\ t_1 & = & 1/2 \\ x_1 & = & 1/2 \\ t_2 & = & 1 \\ x_2 & = & 9/8 \end{array}$$

3 Intégration à pas adaptatif (8 points)

On considère l'équation différentielle ordinaire avec condition initiale suivante. On cherche à l'intégrer sur un intervalle de temps $[a,\, b]$, c'est–à–dire à calculer une suite de points (x_n) qui approxime le graphe de la solution.

$$\dot{x} = f(t,\, x), \quad x(a) = \alpha.$$

La méthode d'Euler étudiée en cours est une méthode à pas fixe, c'est–à–dire que la valeur de h ne change pas au cours de l'intégration. Cette stratégie est en général inefficace. De façon imagée, il vaut mieux utiliser un pas h très petit, là où la courbe intégrale effectue des « virages serrés » et utiliser un pas h grand, là où la courbe est plus lisse.

Mais comment l'intégrateur peut–il savoir s'il est en train d'aborder un virage serré ou non ? L'idée consiste, à partir du point x_n courant, à calculer en même temps deux approximations x_{n+1} et $\hat{x}_{n+1}$ du point suivant. Si l'écart entre les deux valeurs est grand, c'est qu'on aborde un virage serré. S'il est petit, c'est qu'on aborde une zone lisse. Il y a plusieurs façons de mettre cette idée en œuvre. Celle de Fehlberg, qu'on illustre ci–dessous, est très populaire. Elle présente l'avantage de déterminer $\hat{x}_{n+1}$ en réutilisant les évaluations de f déjà employées dans le calcul de x_{n+1}. C'est important parce qu'évaluer f peut être coûteux.

Dans ce qui suit, on calcule x_{n+1} à partir de t_n, x_n et le pas courant h en utilisant la formule de Runge (1895) qui est d'ordre 2.

$$\begin{aligned} k_1 &= h\, f(t_n,\, x_n) \\ k_2 &= h\, f(t_n + h/2,\, x_n + k_1/2) \\ x_{n+1} &= x_n + k_2. \end{aligned}$$

On calcule $\hat{x}_{n+1}$ à partir des nombres k_1 et k_2 précédemment calculés en utilisant une célèbre méthode d'ordre 1.

$$\hat{x}_{n+1} = x_n + k_1$$

L'erreur est donnée par la formule :

$$\mathit{err} = x_{n+1} - \hat{x}_{n+1} = k_2 - k_1.$$

Connaissant le pas h, l'erreur *err* et une précision $\varepsilon > 0$ qu'on suppose fournie par l'utilisateur, on calcule ce qu'on appelle le « pas optimal » h_{opt} par la formule suivante (on suppose err $\neq 0$) :

$$h_{opt} = 0.9\, h \sqrt{\frac{\varepsilon}{\mathit{err}}}.$$

Pour éviter les trop grandes variations de h, on impose en plus la restriction suivante :

$$0.2\, h \leq h_{opt} \leq 5\, h.$$

L'algorithme suivant applique les raisonnements tenus ci–dessus. Il est rédigé en pseudo–code et constitue un exemple d'intégrateur à pas adaptatif. On dit qu'il s'agit d'un intégrateur d'ordre 2 avec une formule « emboîtée » d'ordre 1. Par comparaison, l'intégrateur par défaut de MAPLE est d'ordre 4 avec une formule emboîtée d'ordre 5. Remarquer que les pas trop grands et les pas trop petits ne sont pas gérés de façon symétrique et que la valeur initiale h_0 du pas est passée en paramètre.

fonction Runge_Fehlberg_2_1 $(f, a, b, \alpha, h_0, \varepsilon)$
début
  initialiser t, h et x comme il se doit
  tant que $t < b$ faire
    calculer k_1, k_2 et *err*
    calculer h_{opt} (on suppose *err* $\neq 0$)
    ne pas oublier d'appliquer la restriction $0.2\, h \leq h_{opt} \leq 5\, h$
    si *err* $\leq \varepsilon$ alors

le pas est accepté : on avance puis on met h à jour
$t := t + h$
$x := x + k_2$ (Runge)
$h := \min(h_{opt},\, b - t)$
sinon
le pas est rejeté : on met h à jour
$h := \min(h_{opt},\, b - t)$
fin si
fait
fin

3.1 Connaissance du cours

Question 12 [1 pt]. Comment s'appelle la méthode utilisée pour calculer $\hat{x}_{n+1}$?

Solution. C'est la méthode d'Euler.

Question 13 [2 pts]. On a dit que la méthode de Runge est d'ordre 2. Qu'est–ce que cela signifie (rappeler la définition de l'erreur et de l'ordre) ?

Solution. Notons $x(t)$ la solution du problème. Définissons $e_n = |x(t_n) - x_n|$. Il existe une constante K telle que $e_n \leq K\, h^2$.

3.2 Programmation

Question 14 [2 pts]. On suppose que h et h_{opt} sont calculés mais qu'on n'a pas encore appliqué la restriction : $0.2\, h \leq h_{opt} \leq 5\, h$. Comment faire pour l'appliquer (on peut modifier h_{opt} mais pas h) ? Donner une suite d'instructions MAPLE (pas une fonction) qui applique la restriction.

Solution. On peut ramener h_{opt} dans l'intervalle souhaité. Si $h_{opt} < 0.2\, h$ alors on affecte $0.2\, h$ à h_{opt}. Si $h_{opt} > 5\, h$ alors on affecte $5\, h$ à h_{opt}.

```
if hopt < 0.2*h then
    hopt := 0.2*h
elif hopt > 5*h then
    hopt := 5*h
fi
```

Question 15 [3 pts]. Coder en MAPLE l'algorithme *Runge_Fehlberg_2_1*. On peut utiliser les fonctions *sqrt* et *min* de MAPLE. Ne pas se soucier du fait que la fonction ne retourne rien.

Solution.

```
Runge_Fehlberg_2_1 := proc (f, a, b, alpha, h0, epsilon)
    local t, h, x, k1, k2, err;
    t := a;
    h := h0;
    x := alpha;
    while t < b do
        k1 := h*f(t,x);
```

```
        k2 := h*f(t+h/2, x+k1/2);
        err := k2 - k1;
        hopt := 0.9*h*sqrt (epsilon/err);
        if hopt < 0.2*h then
            hopt := 0.2*h
        elif hopt > 5*h then
            hopt := 5*h
        fi;
        if err <= epsilon then
            t := t + h;
            x := x + k2;
            h := min (hopt, t - b)
        else
            h := min (hopt, t - b)
        fi
    od
end:
```