Maple TD 12 **Mai 2005**

Le but de ce TD est de vous rappeler des notions essentielles qu'il faut absolument maîtriser. Nous vous proposons ensuite une introduction au *calcul numérique* qui vous fera (re)découvrir le calcul approché d'intégrales. Une autre partie, tout aussi importante concerne quelques applications de l'algorithme du pivot de Gauss.

1 Révisions (important)

1.1 Développements limités

La fonction `taylor` doit vraiment être maîtrisée. Un de ses défauts majeurs est qu'il faire bien attention à avoir suffisamment de termes dans le DL, et surtout bien faire la différence entre les deux notation de Landau : $o()$ pour "négligeable devant" (ce que l'on souhaite obtenir quand on demande un DL en mathématiques) et $O()$ pour "majorée par" (ce que la fonction `taylor` de Maple nous renvoie). Il ne faut pas hésiter donc à pousser le développement plus haut que le degré demandé.
D'autre part, il faut bien avoir en tête le résultat de Taylor-Young, i.e. que le premier terme du développement est la limite de la fonction au point considéré et que le $n^{\text{ème}}$ est de la forme $\frac{f^{(n)}(a)}{n!}(t-a)^n$.
Donnez les DL et les limites à l'ordre et au point demandé :

- $\left(\frac{e^t+1}{2}\right)$ (à l'ordre 4 en 0)
- $(t^2-1)\ln(1+t)$ (à l'ordre 5 en 0)
- $tan(t)^{sin(t)}$ (à l'ordre 6 en $\frac{\pi}{4}$)
- $\sqrt{1-t^2}\arctan(t)$ (à l'ordre 7 en 0)

1.2 Suites récurrentes

Nous abordons dans cette partie les problèmes algorithmiques liés aux suites définies par une relation de récurrence. Il existe essentiellement deux types de méthodes pour coder ce type de suite.

La première, la plus élégante, consiste à utiliser une programmation récursive, l'avantage de cette méthode est sa simplicité, son inconvénient principal est sa lenteur - la méthode recalcule beaucoup de termes plusieurs fois. Un autre atout est que la programmation se calque exactement sur la définition "par récurrence" de la suite. Observons en effet un exemple simple (la factorielle) :

$$\forall\, n \in \mathbb{N},\ v_n = \begin{cases} 1 \text{ si } n = 0 \text{ - } \textit{initialisation} \\ n * v_{n-1} \text{ sinon - } \textit{hérédité} \end{cases}$$

Le programme associé est semblable :

```
factorielle:=proc(n)
if n=0 then RETURN(1);
else RETURN(n*factorielle(n-1));
fi;
end;
```

La procédure s'appelle elle-même dans sa définition, c'est une procédure récursive.

Exercice 1 (récursive) *Programmez la suite définie par la relation* $u_0 = 4, \quad u_1 = 5$, *et* $\forall n \geq 0, u_{n+2} = 5 \cdot u_{n+1} - 2 \cdot u_n$.

La seconde méthode est l'utilisation d'une boucle itérative, et donc de variables locales que l'on réaffecte. Cette méthode est souvent rapide et efficace, cependant, il est nécessaire de la manier avec soin.
Pour cela, il est très important de bien comprendre le fonctionnement de la *réaffectation d'une variable locale* - une méthode utilisée dès les premiers tds. Supposons que vous ayez une variable u, et que vous voulez que u soit réaffectée de sorte que u contienne la valeur de $\phi(u)$ où ϕ agit sur u, ceci vous amène à taper :

`u :=`ϕ`(u);`

Ce type d'instructions peut paraître étrange car il suppose des *principes d'autoréférence* qui s'avèrent délicats à assimiler. Ainsi pour calculer la puissance $n^{\text{ème}}$ d'un réel a, il suffit d'écrire :

```
u :=1; - initialisation
for i from 1 to n - itération
do u :=u*a; od;
RETURN(u)
```

Ici, ϕ est la fonction : $t \mapsto t \cdot a$. Il est très important de comprendre le rôle joué par la variable locale u qui contient à chaque étape de l'itération les différents éléments d'une suite u_i qui calculerait successivement $u_i = a^i$. Seulement il n'est pas nécessaire d'introduire une liste formée de l'ensemble des termes de la suite ($L = [u_1, u_2, ..., u_i]$), car cela est trop coûteux en mémoire ! ! !
La méthode s'adapte à toute situation. En effet nous venons d'observer l'effet de la méthode pour le calcul des termes d'une suite définie par une relation de récurrence d'ordre 1. Le petit miracle de la méthode est qu'elle s'adapte à peu de frais aux relations de récurrence d'ordre supérieure. L'astuce étant que rien ne nous oblige à prendre pour u un réel - on peut très bien prendre une liste de réels - ni pour ϕ une fonction réelle - il est tout à fait possible de choisir une application linéaire par exemple.

Exercice 2 (itérative) *Pour vérifier que vous avez bien compris cette explication, programmez la suite introduite en utilisant la méthode suggérée.*
Réutilisez cette méthode pour programmer la suite définie par la relation de récurrence : $u_0 = 2, u_1 = -1, u_2 = 3$ *et* $\forall n \geq 0, u_{n+3} = u_{n+2} - u_{n+1} + u_n$. *(*Indication *: on pensera à introduire le vecteur* $V_n = (u_{n+2}, u_{n+1}, u_n)$ *et à trouver une relation linéaire* - i.e. *matricielle - entre* V_{n+1} *et* V_n*.)*
Expliquez comment généraliser cette méthode pour une suite définie par une relation de récurrence d'ordre r.

2 Calcul d'intrégrales, Aires, Calculs approchés.

Le but de cette section est de passer en revue les méthodes à connaître pour le calcul d'intégrales.

2.1 Maple comme logiciel de calcul formel

Dans un premier temps nous utiliserons Maple comme un logiciel de calcul formel, grâce à la commande `int`. Vous commencerez donc par lire l'aide à son sujet - en tapant par exemple `"?int"`

Exercice 3 *Calculez les intégrales suivantes :*

$$\int_0^{\pi} \frac{1}{(2+cos(t))^2}dt, \int_0^{\frac{\pi}{4}} \cos(t)^2 cos(3t)\sqrt{cos(2t)}dt, \int_1^{\sqrt{2}} \frac{1}{t\sqrt{-t^4+3t^2-2}}dt$$

Exercice 4 *Définissez et étudiez les fonctions dites* fonction de la borne supérieure *en utilisant la fonction* `int`. *Vous représenterez ces fonctions.*

- $x \mapsto \int_0^{2x^2-3} \frac{1}{t+\sqrt{t^2+5}}dt$
- $x \mapsto \int_3^{3x+\frac{5}{x}} \frac{\ln(\sqrt{3+4t^4})}{1+t^2}dt$

2.2 Calcul approché d'intégrales

L'autre partie concernant les intégrales relève plus de l'algorithmie. Il s'agit de fournir des méthodes de calculs pour donner des valeurs approchées d'intégrales.

Exercice 5 (méthodes des rectangles) *Rappelez le théorème de convergence des sommes de Riemann. Appliquez cette méthode d'approximation par somme de Riemann en considérant :*

$$I_-(f,a,b,n) = \frac{(b-a)}{n}\sum_{k=0}^{n-1} f\left(a+k\frac{b-a}{n}\right), \text{ et } I_+(f,a,b,n) = \frac{(b-a)}{n}\sum_{k=1}^{n} f\left(a+k\frac{b-a}{n}\right)$$

pour parvenir à vos fins, vous écrirez deux procédures prenant en entrée f, a, b, et n et renvoyant la somme. Vous tâcherez d'utiliser le moins de fonctions maple avancées - telles que `sum` *ou autre.*
Testez vos programmes sur des fonctions de votre choix.

Exercice 6 (autres méthodes (facultatif)) *Dessinez le graphe d'une fonction quelconque sur le quadrant $\{(x,y) \in \mathbb{R}^2, x \geq 0$ et $y \geq 0\}$. Représentez alors sur ce graphe les rectangles correspondant à la somme I_- et les points $f(a_k)$ où $a_k = a + k\frac{(b-a)}{n}$.*
Comme vous l'aurez compris, il est possible de choisir d'autres figures géométriques pour effectuer l'approximation de l'aire sous la courbe.

La première alternative est le trapèze*, la méthode s'appelant d'ailleurs* ***méthode des trapèzes****. Elle consiste à calculer l'aire associée à la figure composée des trapèzes $(a_k,0)(a_{k+1},0)(a_{k+1},f(a_{k+1}))(a_k,f(a_k))$. Programmez une fonction* `trap` *prenant en entrée f, a, b et n, et retournant la valeur approché de l'intégrale de f prise entre a et b obtenue selon la méthode des trapèzes.*

La seconde méthode revient à déterminer la parabole passant par les trois points $(a_i, f(a_i)), i = 3k, 3k+1, 3k+2$ puis à calculer l'aire sous la figure ainsi réalisée. On remarque qu'il faut un nombre de points multiple de 3. Vous commencerez d'abord par calculer l'aire sous la figure déterminée pas un groupe de trois points, ensuite vous implémenterez un programme `simp` *reprenant ce principe de calcul. On appelle* ***méthode de Simpson*** *cette manière d'approcher l'aire de f.*

Expliquez comment généraliser au cas où f n'est plus nécessairement positive.

3 Algèbre linéaire

Exercice 7 (calcul d'une base de Im, de Ker) *Au cours d'un précédent td, nous avons mis en oeuvre le calcul de la* matrice réduite de Gauss*, obtenu à l'aide de l'algorithme du pivot de Gauss. En fait cette méthode fournit un moyen commode de calcul du* noyau *et de l'*image *d'une application linéaire. Voici comment : étant donnée une application linéaire u, il est possible de considérer une matrice M associée dans des bases adaptées aux espaces vectoriels source et but de u. Supposons que M soit de taille $n \times p$. Illustrons la méthode avec l'image. On considère la matrice identité de taille p, I_p, puis on applique les opérations élémentaires de l'algorithme du pivot* sur les lignes *- et uniquement sur les lignes - à M, mais aussi à I_p. Une fois M mise sous forme réduite, on peut lire en colonne dans la transformée de I_p une base de l'image de u.*
Analysez bien cette méthode et proposez une méthode similaire pour calculer une base du noyau de u.
Programmez ces deux procédures, `Im` *et* `Ker` *afin qu'elles prennent en entrée une matrice et qu'elles renvoient respectivement une base de l'image et une base du noyau.*

Exercice 8 (déterminant) *Rappelez la formule générale de calcul du déterminant d'une matrice carrée M de taille n. Évaluez le nombre d'opérations à effectuer pour évaluer $Det(M)$ en utilisant cette formule en fonction de n.*
Que pensez-vous de cette estimation ? Est-il raisonnable d'utiliser cette formule pour programmer une procédure de calcul du déterminant ?

Nous allons proposer une méthode différente. Soit $T = (t_{ij})_{1 \leq i,j \leq n}$ une matrice triangulaire supérieure. Calculez $Det(T)$. Combien faut-il d'opérations pour calculer le déterminant d'une matrice trigonale supérieure ? En exploitant cette propriété, quel serait d'après vous une méthode pour calculer $Det(M)$?

En remarquant que les opérations élémentaires n'affectent pas la valeur du déterminant d'une matrice, expliquez comment en utilisant l'algorithme du pivot on peut calculer de manière efficace $Det(M)$. Donnez un majorant de sa complexité.