TP2 : Les polynômes

Le but de ce TP est d'apprendre à manipuler des polynômes formellement avec `Scilab`. Rappelons que `Scilab` est un logiciel de calcul numérique sur des tableaux et non un logiciel de calcul formel. On va donc pallier cette difficulté en représentant les polynômes sous formes de tableaux.

Dans ce TP, nous allons coder à la main des fonctions qui existent déjà en `Scilab`. Je vous donnerai à chaque fois la fonction `Scilab` correspondante. L'intérêt de ce TP est surtout d'apprendre à coder des fonctions simples en utilisant les outils que nous avons vus au TP précédent. En particulier les branchements *if* et boucles *while* et *for*.

On introduira aussi en fin de TP la notion de *complexité* qui sert à mesurer le temps de calcul d'un algorithme (en terme de nombre d'opérations) en fonction de la taille de l'objet auquel l'algorithme est appliqué.

1 Préliminaires

1.1 Boucles et branchement

1.1.1 Branchement *if*

La syntaxe `Scilab` suivante permet d'effectuer les `instructions1` si la `condition1` est vérifiée, et de ne rien faire sinon.

```
if condition1 then
  instructions1
end
```

On peut aussi choisir d'effectuer les `instructions2` si la `condition1` n'est pas vérifiée. Voilà ce que cela donnerait :

```
if condition1 then
  instructions1
else instructions2
end
```

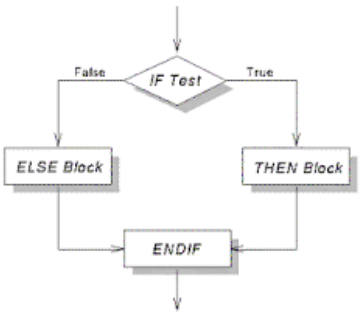

FIGURE 1 – `if then else` Schéma (en anglais) de ce qu'il se passe

Il peut parfois aussi être utile de donner une `condition2` à vérifier lorsque la `condition1` n'est pas vérifiée :

```
if condition1 then
  instructions1
elseif condition2
```

```
  instructions2
else instructions3
end
```

Évidemment, on peut utiliser autant de `elseif` que l'on veut.

La `condition` doit être une proposition logique. Voici un tableau des symboles mathématiques que vous serez probablement amenés à utiliser et de leur traduction en `Scilab`.

Maths	`Scilab`
et	`&`
ou	`|`
non	`~`
$=$	`==`
$\neq$	`~=`
$\geq$	`>=`
$\leq$	`<=`
$>$	`>`
$<$	`<`
Vrai	`%T`
Faux	`%F`

1.1.2 Boucle *for*

Il est souvent utile de répéter plusieurs fois une même instruction. Pour cela on utilise ce qu'on appelle des *boucles*. Il y a deux types de boucles. Les boucles *for* sont utilisées lorsqu'on sait exactement le nombre de fois que l'on doit répéter l'opération. (Sinon, on utilisera des boucles *while*).

```
for i=I do
  instructions
end
```

`I` est un vecteur. `i` va prendre successivement (dans l'ordre) les valeurs des composantes de `I`. Les `instructions` peuvent faire intervenir `i`.

Par exemple, le script suivant renvoie la somme des 37 premiers entiers.

```
somme=0;
for i=1:37 do
  somme = somme + i;
end
somme
```

Question 1 Écrire un script qui renvoie la somme des entiers pairs compris entre 0 et 42.

Question 2 Soit `T` le vecteur $(8, 54, -3, 7, -9, 42)$ En utilisant une boucle *for* et un branchement *if*, écrire un script qui renvoie le maximum de `T`.

Il pourrait être intéressant de créer une fonction qui prenne un vecteur en entier et renvoie son maximum.
Cette fonction existe déjà en `Scilab` : `max(T)` renvoie le maximum du tableau `T`.
On va la recréer à la main.
Je vais vous donner un exemple (qui vaut mieux qu'un long discours).
Il faut d'abord ouvrir un nouveau fichier Scinote dans lequel on va écrire notre fonction.
Ici, on crée une fonction qui prend en argument un entier `n` et renvoie la somme des `n` premiers entiers.
Je dois choisir un nom pour ma fonction, ici : `masa` (ma somme arithmétique)

Ma première fonction :

Vous prendrez soin de ne pas mettre de `clear` au début d'une fonction, contrairement aux `scripts`.

On écrit ceci dans Scinote :

```
function x=masa(n)
x=0 ;
for i = 1 : n do
x = x+i ;
end
x
endfunction
```

Ensuite, je dois exécuter ce code. Il suffit de cliquer sur l'icône *play avec une disquette*. Scilab veut alors que j'enregistre le fichier. Je peux l'enregistrer où je veux par contre je dois absolument choisir le nom `masa.sci`.

Une fois ce script exécuté, je peux appeler la fonction : `masa(37)` renvoie 703.

Un autre exemple :

On veut créer une fonction V(a,n) qui renvoie le terme v_n de la suite de premier terme $v_0 = a$ et telle que $\forall n \in \mathbb{N}$, $v_{n+1} = 3v_n + 2$.

On écrit ce qui suit dans Scinote :

```
function v = V(a,n)
v=a ;
for i = 1 : n do
v = 3*v+2 ;
end
endfunction
```

Question 3 Calculer à la main v_3 lorsque a = 2, puis vérifier le résultat grâce à la fonction V.

Question 4 Concours ATS 2016 :

La suite $(v_n)_{n\in\mathbb{N}}$ vérifie : $\forall n \in \mathbb{N}$, $v_{n+2} = v_{n+1} + 2v_n$.

Ecrire une procédure informatique ATS (c'est ce que j'appelle une fonction) ayant pour variables d'entrée deux réels x et y et un entier naturel n et qui renvoie le terme v_n de la suite telle que $v_0 = x$ et $v_1 = y$.

On pourra utiliser une boucle. ($n \geq 2$)

Indication :

ça commence comme ça :

```
function v = ATS(x,y,n)
v = x ;
w = y ;
for i = 2 : n do
```

Question 5 A vous de jouer : à partir de votre travail de la question 2, créez une fonction `monmax` qui prend en entrée un vecteur `T` quelconque et renvoie son maximum. Puis, appliquer la au vecteur `T` précédemment défini pour vérifier que cela fonctionne. (Haha !)

1.1.3 Boucle *while*

La boucle *while* permet elle aussi de répéter une instruction. L'avantage qu'elle présente sur la boucle *for* est que l'on peut l'utiliser même si on ne connaît pas le nombre d'opérations à effectuer. La syntaxe est la suivante :

```
while condition do
  instructions
end
```

La `condition` prend la même forme que dans un branchement *if* et les `instructions` seront exécutées tant que la `condition` est vérifiée.

Remarque 1.1. Danger : si vous faites une erreur, il est possible que la `condition` soit vérifiée pour toujours et que donc la machine ne s'arrête jamais de calculer. Si cela vous arrive, allez dans l'onglet contrôle de la `Console` et cliquez sur Abandonner. Vous pouvez aussi utiliser le raccourci clavier `Ctrl-C` puis taper la commande `abort`.

Question 6. J'ai replanté l'année dernière (2014) mon sapin de Noël qui mesurait 1,2m. Il grandit de 30cm par an. J'ai décidé de le couper quand il dépassera 8m. En quelle année vais-je le couper ?

Utilisez les variables `a` qui représente l'année (initialisée à 2014) et `s` qui représente la taille du sapin en m (initialisée à 1,2), ainsi qu'une boucle *while* pour répondre au problème.

Les boucles et branchement (On parle de *structures de contrôle*.) vont nous être utiles dans la suite pour manipuler les polynômes.

1.2 Représentations des polynômes en `Scilab`

On travaille avec des polynômes à coefficients réels. L'ensemble des polynômes est noté $\mathbb{R}[X]$. C'est un $\mathbb{R}$-espace vectoriel. Pour $n \in \mathbb{N}$, on note $\mathbb{R}_n[X]$ l'ensemble des polynômes de degré *au plus* n. C'est un espace vectoriel de dimension $n+1$.

On considère la fonction $\Psi : \left| \begin{array}{ccc} \mathbb{R}^{n+1} & \to & \mathbb{R}_n[X] \\ (a_1, a_2, \cdots, a_n, a_{n+1}) & \mapsto & a_1X^n + a_2X^{n-1} + \cdots + a_nX + a_{n+1} \end{array} \right.$.

Ainsi, le vecteur $(a_1, a_2, \cdots, a_n, a_{n+1})$ représente le polynôme $a_1X^n + a_2X^{n-1} + \cdots + a_nX + a_{n+1}$.

Question 7. On considère le polynôme X. Quelle est sa représentation dans $\mathbb{R}_1[X]$? Et dans $\mathbb{R}_{37}[X]$?

2 Opérations élémentaires

2.1 Addition et soustraction

Vous savez additionner et soustraire des polynômes.

Question 8. On considère les polynômes $P_1 = X^2 - 1$ et $P_2 = 2X - 3$. Dans un script (voir la fin de la page 9 du TP1), créer deux tableaux `P1` et `P2` qui représentent respectivement P_1 et P_2. Puis (sans utiliser de boucles, rappelez vous les manipulations sur les tableaux vues au TP1), calculez `S` qui représente $P_1 + P_2$. Calculez aussi `D` qui représente $P_1 - P_2$.

On souhaite maintenant créer une fonction `masomme` qui prend en argument deux représentations de polynômes (pas forcément de même taille) et qui en renvoie la somme.

Question 9. Imaginons que l'on ait déjà créé cette fonction, comment faire pour calculer la différence de deux polynômes via la fonction `masomme` ?

`Scilab` ne peut qu'additionner des vecteurs de même taille. Donc, la première étape pour créer notre fonction `mysomme(P1,P2)` est de trouver un moyen de «remettre `P1` et `P2` à la même taille ».

Question 10. Dans un premier temps, créer une fonction `grandir` qui prend en argument une représentation d'un polynôme `P` et une taille `n` plus grande que la taille du vecteur `P` et renvoie un vecteur `Pn` de taille `n` qui lui aussi représente le même polynôme que `P`.

Question 11. Créer une fonction `taillemax` qui prend en argument deux vecteurs et renvoie la taille du plus grand des deux.

Question 12. À l'aide des deux fonctions précédentes, créer la fonction `mysomme`. Attention : Il faudra inclure dans votre fonction les commandes qui permettent d'exécuter les fonctions que vous avez créées et dont vous vous servez, c'est-à-dire les commandes qui s'affichent dans la console quand vous cliquez sur le bouton `play+disquette`.

Vous avez probablement remarqué que les vecteurs $(0, 1, 2, 3)$ et $(1, 2, 3)$ représentent le même polynôme $X^2 + 2X + 3$ de degré 2.

Question 13. Montrer que tout polynôme admet une *unique* représentation qui ne commence pas par un zéro.

Écrire une fonction `jolipoly` qui prend en argument la représentation d'un polynôme et renvoît son unique représentation qui ne commence pas par un zéro. (Indice : *while*)

Question 14. Modifier la fonction `mysomme` pour s'assurer que la représentation renvoyée par cette fonction ne commence pas par un zéro.

Remarque 2.1. Astuce : pour vérifier qu'une fonction fonctionne, testez la sur des exemples simples (avec lesquels vous pouvez vérifier les calculs à la main.)

2.2 Multiplication

Question 15. Écrire une fonction qui prend en argument une représentation d'un polynôme et renvoît son degré. Attention au cas où la représentation commence par un 0.

Si $P = \sum_{n=0}^{d} a_n X^n$ et $Q = \sum_{n=0}^{d'} b_n X^n$ sont deux polynômes, il a été vu en cours que PQ est un polynôme de degré $d + d'$ et qu'il est donné par

$$PQ = \sum_{n=0}^{d+d'} c_n X^n \text{ avec } c_n = \sum_{k=0}^{n} a_k b_{n-k}.$$

Dans notre représentation, les coefficients des polynômes ne sont pas représenté dans le même sens que dans le cours. En effet, le vecteur $(p_1, p_2, \cdots, p_d, p_{d+1})$ représente le polynôme $p_1 X^d + p_2 X^{d-2} + \cdots + p_d X + p_{d+1}$.

Question 16. Soit P le polynôme représenté par $(p_1, p_2, \cdots, p_d, p_{d+1})$. Soit Q le polynôme représenté par $(q_1, q_2, \cdots, q_{d'}, q_{d'+1})$. On désigne par $(r_1, r_2, \cdots, r_{d+d'}, r_{d+d'+1})$. Via un changement de variable dans la formule qui donne c_n, exhiber une formule pour les r_i.

Question 17. Utilisez cette formule pour écrire pour écrire une fonction `myproduit` qui prend en argument deux représentations de polynômes et renvoît une représentation de leur produit. On s'assurera que cette représentation ne commence pas par un 0. On pourra utiliser la fonction `mydeg` précédemment créée.

2.3 Dérivation

Question 18. Soit $P = p_1 X^d + p_2 X^{d-2} + \cdots + p_d X + p_{d+1}$. Que vaut P' ?

Question 19. En déduire une fonction `myderiv` qui prend en argument la représentation d'un polynôme et renvoît une représentation de son polynôme dérivé. On s'assurera que la représentation en question ne commence pas par un 0.

3 Pour aller plus loin

3.1 Division euclidienne

Le but de cette partie est d'écrire une fonction qui prend en argument deux représentations de polynômes `A` et `B` et renvoît `[Q, R]` où `Q` et `R` représentent respectivement le quotient et le reste de la division euclidienne du polynôme représenté par `A` par celui représenté par `B`.

On rappelle l'algorithme de la division euclidienne :

Le polynôme `A` de degré `n`. Le polynôme `B` de degré `m`. On s'assure que leur représentation ne commence pas par des 0.

`Q`$\leftarrow$`(0,0, ...,0)` dans $\mathbb{R}_{n-m}[X]$

`R`$\leftarrow$`A`

`k`$\leftarrow$`1`

Tant que deg `R` $\geq$ deg `B` faire

  `Q(k)` $\leftarrow$ `R(k)./B(1)`

  `R`$\leftarrow$`R-Q(k).*[B, 0, 0, ..., 0]` (avec `n-m` `0`)

  `k`$\leftarrow$`k+1`

fin faire

Question 20. Choisissez deux polynômes de votre choix, et sur feuille, convainquez-vous que cet algorithme en effectue bel et bien la division euclidienne.

Question 3.1.2.

Implémentez cet algorithme : créer la fonction `mydiveucl`. Appliquez le à votre exemple pour vérifier qu'il fonctionne.

3.2 Exponentiation rapide

Étant donné un exposant $n \in \mathbb{N}$, on souhaite calculer le plus efficacement possible x^n pour $x \in \mathbb{R}$, c'est-à-dire en minimisant le nombre de multiplications, pour gagner en temps d'exécution.

En effet, pour calculer x^8, on peut :

— Calculer $x \times x \times x \times \cdots \times x$, ce qui fait 7 multiplications. La liste des exposants calculés est : 1, 2, 3, 4, 5, 6, 7, 8.

— Calculer : $y = x \times x$, ce qui correspond à x^2, puis $z = y \times y$, ce qui correspond à x^4, enfin $t = z \times z$, ce qui donne le résultat.

On obtient donc le résultat $t = x^8$ en seulement trois multiplications. La liste des exposants est : 1, 2, 4, 8.

L'écart se creuse très rapidement, entre la méthode naïve et les méthodes plus évoluées. Ainsi, pour calculer x^{1024},il faut

— 1023 multiplications, pour la méthode naïve,

— 10 multiplications pour la méthode dite exponentiation rapide.

Wikipedia propose donc l'algorithme suivant :

$$\text{puissance}(x,\, n) = \begin{cases} x, & \text{si } n = 1 \\ \text{puissance}(x^2,\, n/2), & \text{si } n \text{ est pair} \\ x \times \text{puissance}(x^2,\, (n-1)/2), & \text{si } n > 2 \text{ est impair} \end{cases}$$

C'est un algorithme dit *récursif* car il s'appelle lui-même. On rajoute les conditions initiales `puissance(x,0)=1` et `puissance(x,1)=1` pour assurer que l'algorithme s'arrête.

Question 21. Écrire la fonction `puissance`.

3.3 Évaluation d'un polynôme

3.3.1 Évaluation naïve

On va utiliser cette fonction pour évaluer un polynôme P en un réel x, c'est-à-dire calculer $P(x)$.

Question 22. Écrire une fonction qui prend en argument la représentation d'un polynôme P et un réel x et qui calcule $P(x)$ sans utiliser la fonction `puissance`, mais en utilisant `x.^k` ce qui correspond à k multiplications. Si P est de degré n, combien d'additions ont été effectuées pour calculer $P(x)$ (dans le pire des cas) ? Combien de multiplications ?

3.3.2 Algorithme de Hörner

On se propose de trouver un algorithme qui effectue le même calcul en moins d'opérations.
Soit $P = a_1X^n + a_2X^{n-1} + \cdots + a_nX + a_{n+1}$.
L'idée consiste à écrire $P = (((\cdots((a_1X + a_2)X + a_3)X + \cdots)X + a_{n-1})X + a_n)X + a_{n+1}$.

Question 23. Démontrer cette formule par récurrence sur le degré de P.

Question 24. Écrire une fonction `myhorner` qui calcule $P(x)$ via cette méthode. Si P est de degré n, combien d'additions ont été effectuées pour calculer $P(x)$ (dans le pire des cas) ? Combien de multiplications ? Conclure.

Question 25. En utilisant la fonction `myhorner`, et ce qui a été vu au TP1, tracer le graphe du polynôme de votre choix. (script)