TD1 – Prendre en main SageMath pour le calcul formel

M. Bombar, L. Poyeton

Comment lire cette feuille

- Cette première feuille de TD se veut surtout comme une initiation à l'outil Sagemath que nous allons utiliser tout le long du semestre.
- Lorsqu'une fonction SageMath sait déjà faire le calcul demandé dans les exercices, elle peut servir d'*oracle de vérification*, mais sauf exception, elle ne doit pas être appelée à l'intérieur de l'algorithme que l'on vous demande de programmer. Essayez de jouer le jeu.

1 Environnement de travail :

Attention

Depuis cette année, pour lancer une sessions Sagemath, il faut débuter par

```
. activate
```

dans un terminal pour avoir accès à un environnement virtuel qui contient l'executable Sage.

Tutoriel SageMath – une session SageMath

SageMath est un système libre de calcul mathématique qui s'appuie notamment sur Python. Dans un terminal, lancer une session interactive avec

```
sage
```

Quelques réflexes utiles :

- `Ctrl+C` interrompt un calcul ; `Ctrl+D` quitte la session ;
- la touche `Tab` déclenche la complétion ;
- `nom?` affiche l'aide sur un objet ou une fonction, par exemple `factor?` ;
- le caractère `#` introduit un commentaire jusqu'à la fin de la ligne.

Pendant les TP, on privilégiera des fichiers `.sage`, afin de conserver une trace reproductible du travail. Un fichier peut être exécuté depuis un terminal par

```
sage td01.sage
```

ou chargé depuis une session SageMath :

```
load("td01.sage")
```

La commande `attach("td01.sage")` recharge automatiquement le fichier lorsqu'il a été modifié ; elle peut être pratique pour des essais interactifs.

Tutoriel SageMath – VSCode et l'extension SageMath Enhanced

Sur les postes de travail, on pourra utiliser **VSCode** comme éditeur, avec l'extension **SageMath Enhanced**.
Pour démarer l'éditeur vous pouvez cliquer sur son icone, ou bien tout simplement lancer

```
code
```

Afin d'installer l'extension `Sagemath Enhanced`, il suffit de cliquer sur l'onglet `Extensions` à gauche, et de chercher `Sagemath enhanced` dans la barre de recherche, puis de cliquer sur `Install`.
Créer un fichier `td01.sage`, puis vérifier d'abord dans le terminal intégré que la commande suivante fonctionne :

```
sage --version
```

L'extension permet notamment la coloration des fichiers `.sage`, la complétion, l'exécution du fichier SageMath actif, et d'autres joyeuseries. On peut utiliser le bouton d'exécution ou le raccourci (`F5`, par défaut). La sortie apparaît dans le terminal intégré.
Si l'exécution par l'extension n'est pas configurée, il suffit de conserver le même fichier `td01.sage` et de lancer dans le terminal la commande

```
sage td01.sage
```

Enfin, vous pouvez aussi faire un mélange de ces deux approches, en executant `Sage` dans le terminal intégré, et ensuite en chargeant votre fichier via la commande

```
attach("td01.sage")
```

Tutoriel SageMath – Utiliser SageMath dans un notebook Jupyter

Pour les fonctions et algorithmes développés pendant les TP, on privilégiera les fichiers `.sage`. Néanmoins, si vous le souhaitez vous pouvez aussi utiliser SageMath dans un *notebook* Jupyter. Pour cela, après avoir activé l'environnement SageMath, lancer dans un terminal

```
sage -n jupyter
```

Un notebook est composé de cellules que l'on peut exécuter séparément. Il est particulièrement pratique pour mélanger du texte, des formules mathématiques, et des petits calculs SageMath.

Attention cependant : le notebook possède sa propre limite de mémoire interne, que vous pouvez atteindre dans certains cas, et l'ordre d'exécution des cellules compte. Un notebook peut donc afficher un résultat obtenu à partir d'un état qui n'est plus visible en lisant simplement les cellules de haut en bas. En cas de doute, redémarrer le noyau et exécuter toutes les cellules dans l'ordre.

2 Objets, affectations, tests et parents

Tutoriel SageMath – calcul exact et type d'un objet

Exécuter successivement les commandes suivantes et prévoir le résultat avant de valider.

```
1 + 1
a = 8
a
a == 5
a == 8
parent(a)
type(a)
```

```
b = 4/3
b
parent(b)

2 <= 3
1 != 1
1 == 1

2^5
2**5
10 % 3
10 // 3
10 == 3*(10 // 3) + (10 % 3)

floor(2.4); ceil(2.4); round(2.4); round(2.5)
```

Dans les futures séances de TD, n'hésitez pas à abuser de la commande `parent` pour comprendre et vérifier dans quelle structure vivent les objets que vous manipulez! C'est très utile pour identifier et corriger des bugs.

Tutoriel SageMath – complétion et documentation

Exécuter

```
a = ZZ(93)
a.
```

puis utiliser la touche `Tab`. Rechercher en particulier une méthode donnant l'écriture binaire de `a`. Consulter ensuite sa documentation avec `?` et tester :

```
a.bits()
a.nbits()
```

Faire la même expérience après avoir défini

```
F7 = GF(7)
x = F7(3)
```

Que valent `parent(x)` et `type(x)` ?

3 Listes, intervalles et compréhensions

Tutoriel SageMath – les listes Python utilisées par SageMath

On rappelle que les indices des listes Python commencent à 0. Une tranche `L[a:b]` contient les indices $a, a+1, \ldots, b-1$. Exécuter les commandes suivantes. Après chaque modification de `L`, afficher `L` pour vérifier son état.

```
L = [1, 2, 3, 12]
len(L)
L[0]
L[3]
L[-1]
L[1] = 6
L.append(10)
```

```
L.extend([18, 19])
L[2:4]
L[2:]
L[:-1]

list(range(1, 10))
[i^2 for i in range(1, 11)]
[i^2 + 1 for i in range(2, 11, 2) if is_prime(i^2 + 1)]

sum([1, 2, 3, 4])
prod([1, 2, 3, 4])
```

La fonction Python `range` produit un objet itérable ; utiliser `list(range(...))` si l'on veut afficher explicitement la liste correspondante.

Tutoriel SageMath – ensembles

Un ensemble dans Sage ou Python est un objet de type `set`. Il représente un ensemble au sens mathématique du terme, et en particulier il n'y a pas de répartitions, et l'ordre n'importe pas. En pratique, les éléments sont triés.
Tester :

```
A = set([1, 2, 5, 5])
B = set([1, 6, 7, 5])
parent(A)
A | B
A & B
A - B
```

SageMath possède aussi un anneau symbolique. Comparer la nature des objets dans :

```
k, n = var('k n')
s = sum(k^2, k, 1, n)
s
factor(s)
parent(s)
```

Dans la suite du cours, il faudra distinguer une *expression symbolique* d'un élément d'un anneau explicitement construit, par exemple $\mathbb{Q}[X]$.

Tutoriel SageMath – Multiévaluation

La commande `map` permet d'appliquer une fonction à tous les éléments d'une liste, d'un ensemble, d'un n-uplet ... Exécuter les commandes suivantes.

```
L = [1, 2, 3, 12]
f(x) = x^3
list(map(f, L))
```

4 Boucles et fonctions : juste ce qu'il faut de Python

Tutoriel SageMath – fonctions, `for`, `while`, `if`

Dans Python et SageMath, les blocs sont déterminés par l'indentation. Copier le programme suivant dans `td01.sage`, l'exécuter, puis le modifier.

```
def pairs(n):
    v = []
    for i in range(3, n):
        if i % 2 == 0:
            v.append(i)
    return v

print(pairs(20))
```

Les trois structures de contrôle principales sont :

```
if condition:
    # instructions
elif autre_condition:
    # instructions
else:
    # instructions

while condition:
    # instructions

for x in iterable:
    # instructions
```

Écrire ensuite une fonction `valuation2(n)` qui, pour $n > 0$, renvoie le plus grand entier v tel que $2^v \mid n$. Par exemple `valuation2(40)` doit renvoyer 3.

5 Polynômes représentés par des listes

Pour chaque fonction importante, on cherchera désormais à fournir quatre éléments :

1. la spécification des entrées et de la sortie ;
2. un argument de correction, souvent via un invariant ;
3. des tests, y compris sur les cas limites ;
4. un comptage ou un ordre de grandeur du coût.

Dans `Sage`, on peut définir un anneau de polynômes, et utiliser un arsenal de fonctions spécifiques à ces polynômes. Dans cette section, cependant, on n'utilisera pas ces fonctions directement, sauf pour faire vos tests (faites-en!). Le but est de programmer les opérations élémentaires sur les polynômes, lesquels seront représentés par des listes.

Ainsi, une liste

$$[p_0, p_1, \ldots, p_d]$$

représente le polynôme

$$P(X) = p_0 + p_1 X + \cdots + p_d X^d.$$

Les coefficients appartiennent à un anneau R. Pour les tests, on prendra d'abord $R = \mathbb{Q}$.

Tutoriel SageMath – SageMath comme oracle pour $\mathbb{Q}[X]$

Construire l'anneau $\mathbb{Q}[X]$ et tester les conversions entre listes et polynômes :

```
Qx.<X> = PolynomialRing(QQ)

p = [1, 2, 0, -3]
P = Qx(p)
P
list(P)
P.parent()
```

Remarque: Pour créer un anneau de polynômes, on a aussi la syntaxe simplifiée suivante

```
Qx.<X> = QQ[]
```

Si `p` et `q` sont des listes de rationnels, les commandes

```
Qx(p) + Qx(q)
Qx(p) * Qx(q)
list(Qx(p) + Qx(q))
list(Qx(p) * Qx(q))
```

serviront d'oracles pour vérifier vos propres fonctions. Elles ne doivent pas être utilisées à l'intérieur de vos implémentations.

Exercice 1 – Représentation canonique et opérations élémentaires

(Q1) Les listes `[1,2]` et `[1,2,0,0]` représentent le même polynôme. Écrire une fonction `nettoie(P)` qui supprime les zéros terminaux, avec la convention que le polynôme nul est représenté par la liste vide `[]`.

(Q2) Écrire une fonction `plus(P,Q)` qui additionne les polynômes P et Q. Si P et Q ont respectivement $m+1$ et $n+1$ coefficients, donner le nombre d'additions *dans l'anneau des coefficients* effectuées par votre implémentation.

(Q3) Écrire une fonction `scalaire(a,P)`, qui représente le polynôme aP, et `decale(P,k)`, qui représente $X^k P$. Quelles opérations dans l'anneau des coefficients sont réellement nécessaires ?

(Q4) Pour chacune des trois fonctions, tester le polynôme nul, les constantes et des entrées contenant plusieurs zéros terminaux. Ce sont souvents des cas de base à vérifier.

Exercice 2 – Multiplication classique, Division

(Q1) Écrire une fonction `fois(P,Q)` calculant le produit de deux polynômes représentés par des listes, sans appeler la multiplication polynomiale de SageMath. On pourra notamment s'inspirer de la multiplication naïve dans $\mathbb{N}$.

(Q2) Si $\deg P = m$ et $\deg Q = n$, montrer que le nombre d'opéations de base nécessaires pour cette multiplication est inférieur ou égal à $2mn + m + n + 1$.

(Q3) Écrire une fonction `divise(P, Q)` (où $Q \neq 0$), qui effectue la division euclidienne de P par Q.

(Q4) Prouver la correction de vos algorithmes à l'aide d'invariants de boucles.

Exercice 3 – Évaluation et schéma de Horner

Soit $P = p_0 + p_1 X + \cdots + p_d X^d$.

(Q1) Écrire une première fonction d'évaluation qui calcule séparément les puissances x^i, puis comparer son coût à celui d'une évaluation mieux organisée.

(Q2) Montrer l'identité

$$P(x) = p_0 + x\big(p_1 + x(p_2 + \cdots + x(p_{d-1} + x p_d)\cdots)\big),$$

et en déduire un algorithme, dû à Horner, pour évaluer un polynôme.

(Q3) Programmer `horner(P,x)` pour un polynôme représenté par une liste.

(Q4) Pour $d \geq 1$, donner le nombre exact d'additions et de multiplications effectuées par Horner.

(Q5) Évaluer $2X^5 - 3X^3 + X - 7$ en $x = 3$ en donnant les valeurs successives de l'accumulateur, puis vérifier avec SageMath.

Il est important de tester vos implémentations

À chaque fois que vous implémentez quelque chose, il est nécessaire de faire des tests, à commencer par des exemples très simples. En particulier, lorsque vous manipulez des algorithmes sur les polynômes, inutile de tester votre fonction sur un polynôme aléatoire de degré 12, évalué en 73/121. Si votre implémentation est bugguée, vous devriez probablement observer le problème sur des exemples très simples que vous pouvez vous même dérouler à la main en suivant exactement votre implémentation. Le bug sera d'autant plus facile à identifier, voire corriger si vous faites les calculs à la main.
Dans un second temps, on pourra également faire des tests aléatoires, notamment grâce à des oracles utilisant les fonctions déjà implémentées dans `Sage`. Cela permet d'éviter des bugs qui n'apparaissent pas si les exemples sont trop simples.

```
Qx.<X> = PolynomialRing(QQ)

for _ in range(100):
    p = [QQ.random_element() for _ in range(8)]
    q = [QQ.random_element() for _ in range(8)]

    assert Qx(plus(p, q)) == Qx(p) + Qx(q)
    assert Qx(fois(p, q)) == Qx(p) * R(q)
```

Compléter les tests par des cas non aléatoires :

- liste vide et polynôme nul ;
- constante ;
- listes de longueurs très différentes ;
- plusieurs zéros terminaux ;
- coefficients nuls à l'intérieur de la liste ;
- pour Horner, $x = 0$, $x = 1$ et quelques rationnels non entiers.

Si vos fonctions passent les tests, ça ne veut bien évidemment pas dire qu'elles sont correctes (sauf si vous avez une preuve de correction), mais c'est déjà un premier pas !