TP 1 : Prise en main de Sage

Le but de cette séance est de s'exercer à utiliser Sage pour lui faire faire des calculs.

Pour lancer Sage, ouvrez un terminal et tapez-y la commande `sage -n jupyter` puis ouvrez l'URL http://localhost:8080/ dans un navigateur Web. Ou bien utilisez l'application Sage téléchargée sur votre ordinateur personnel au préalable ou connectez-vous sur Cocalc.

On peut lancer l'aide sur un sujet particulier en entrant un nom de commande suivi d'un point d'interrogation, par exemple `expand?` ou `factor?` pour obtenir de l'aide respectivement sur la commande `expand` ou `factor`.

1 Expressions

Comme les calculatrices usuelles, Sage peut en premier lieu effectuer des calculs faisant intervenir des opérateurs mathématiques (`+`, `*`, `^`, etc.) et des fonctions prédéfinies (exponentielle avec `exp`, logarithme avec `ln`, sinus avec `sin`, etc...) et des constantes (e avec `e`, π avec `pi`, i avec `i`, ∞ avec `infinity`, etc...).
Mais il s'agit surtout d'un outil de **calcul formel** dont les possibilités vont bien au-delà de ces manipulations de nombres entiers, de flottants ou de fonctions. En effet, Sage permet aussi de manipuler des expressions, qui peuvent ainsi faire intervenir des indéterminées ou des paramètres non assignés. Diverses fonctions de Sage permettent par conséquent de calculer des dérivées (`diff`), des intégrales ou des primitives (`integrate`), des sommes de séries (`sum`), etc... On peut aussi développer ou factoriser des expressions (`expand`, `factor`) mais aussi y substituer des variables (`subs`) ou les simplifier (`simplify`).

Exercez-vous à utiliser ces possibilités en essayant par exemple de :

- Calculer $42!$;
- Calculer la dérivée de quelques fonctions usuelles (par exemple cos, arctan, ln, $x \mapsto e^{-x^2}$) ;
- Calculer une primitive de quelques fonctions usuelles (par exemple les fonctions ci-dessus) ;
- Calculer $\int_{-\infty}^{\infty} e^{-x^2}\,\mathrm{d}x$;
- Factoriser $4x + 4y + 2x^2y + 9x^2 + 2x^3 + 9xy$;
- Afficher des décimales de π. On pourra par exemple utiliser `N` qui convertit une expression en un nombre flottant ;
- Calculer $\sum_{n=1}^{\infty} \frac{1}{n^2}$;
- Trouver des intégrales ou des séries que Sage ne sache pas "calculer".

Une différence entre Sage et d'autres logiciels de calcul formel est que pour utiliser une expression dans laquelle intervient une indéterminée, il faut **déclarer préalablement** cette indéterminée. Ainsi, pour manipuler une expression en x, il faut d'abord exécuter la commande `x=var('x')`. On peut déclarer deux variables d'un coup par `k,n=var('k','n')` ou `k,n=var('k,n')`. Il ne faut pas confondre ces symboles avec des variables (au sens informatique du terme) qui sont des objets auxquels on a donné un nom à l'aide de l'opérateur `=`. Ce dernier opérateur ne doit pas non plus être confondu avec l'opérateur `==` (qui renvoie "vrai" ou "faux" suivant que l'égalité testée est vérifiée ou non). Vous pouvez testez, dans l'ordre, les commandes suivantes `a=10`, `a+1` et `a==3` pour sentir un peu mieux ces distinctions.

2 Boucles et fonctions

La syntaxe de `Sage` est basée sur celle du langage `Python`. L'**indentation** du code joue un rôle **fondamental** dans cette syntaxe. Ainsi, le nombre d'espaces au début de chaque ligne a une influence sur la manière dont le code sera interprété.

Pour calculer la somme des carrés des entiers de 1 à 100 en utilisant une boucle, on peut utiliser la syntaxe suivante :

```
x = 0
S = 0
while x <= 100:
    S = S + x^2
    x = x + 1
print(S)
```

C'est par l'indentation que l'on délimite le bloc exécuté dans la boucle `while`. Le même principe vaut notamment dans la définition de fonctions et les tests `if`.

Pour définir la fonction f telle que $f(x) = 0$ si $x < 0$ et x^2 si $x \geqslant 0$, on peut faire par exemple (à nouveau bien prendre garde à **l'indentation**) :

```
def f(x):
    if x < 0:
        return 0
    else:
        return x^2
```

L'instruction `if` sert à exécuter une portion de programme seulement si une certaine condition est vérifiée. Pour exprimer cette condition, on peut recourir aux opérateurs de comparaison `==`, `>`, `<`, `<=`, `>=`, `!=` signifiant respectivement $=, >, <, \leqslant, \geqslant, \neq$. On peut également combiner logiquement ces opérateurs en utilisant `or` ou `and` et définir à l'aide du mot `else` une portion à exécuter seulement si la condition n'est pas vérifiée. On peut enfin combiner plusieurs conditions avec la commande `elif`.

Les boucles `for` utilisent la syntaxe `for i in liste :` où `liste` est une liste. Par exemple, `range(10)` est la liste $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$. Pour parcourir les entiers de x à y, on utilise la syntaxe `range(x,y+1)`.

Testez les deux portions de code suivantes pour vous convaincre de l'importance de l'indentation.

```
if (2>1):
    print("Tout va bien!")
else:
    print("Etonnant!")
    print("Il faut reprendre le calcul.")
```

et

```
if (2>1):
    print("Tout va bien.")
else:
    print("Etonnant!")
print("Il faut reprendre le calcul.")
```

- Définir deux fonctions calculant la factorielle d'un entier n, d'une part en définissant une fonction récursive [1] et d'autre part en faisant une boucle.
- On rappelle que la suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour tout $n \geqslant 2$. Définir une fonction `fibonacci` à nouveau de deux manières, la première à l'aide d'une boucle et la seconde à l'aide d'une fonction récursive. Que se passe-t-il si vous essayez de calculer F_{40} ? Pourquoi ? Recommencer en utilisant la commande `@cached_function`.

3 Exponentiation rapide

- Définir une fonction `exponentiation_rapide(a,n)` renvoyant a^n. La comparer avec un algorithme naïf que vous implémenterez également.

`Sage` classifie les objets selon leur type. La fonction `type` renvoie le type d'un objet (essayez avec certains des objets rencontrés jusqu'ici).

Il est possible de stocker un type dans une variable. Par exemple, on définit l'anneau des entiers modulo N en utilisant la syntaxe `IntegerModRing(N)`. Pour définir un représentant d'une classe $a \in \mathbf{Z}/N\mathbf{Z}$, on peut utiliser la commande `IntegerModRing(N)(a)`. Ainsi on peut faire, par exemple :

```
A = IntegerModRing(31)
x = A(2)
```

La variable x contient alors 2 vu comme élément de $\mathbf{Z}/31\mathbf{Z}$. On peut constater que `Sage` considère x comme étant différent de 2 en tapant `type(x)`. Que se passe-t-il si vous tapez `x==2` ? Vous pouvez aussi vérifier que si l'on pose `a=IntegerModRing(31)(3)` et `b=IntegerModRing(17)(3)`, alors pour Sage `a` et `b` sont différents en tapant `a==b`.

- Calculer l'inverse de 3 modulo 62. *Quid* de l'inverse de 31 ? On pourra utiliser la commande `/`.
- Calculer $3^{2^{2^{2^2}}}$ modulo 42.
- *Quid* de $3^{2^{2^{2^2}}} \in \mathbb{Z}$?
- Afficher la liste des 3^n modulo 42 pour $n \in \{1, \ldots, 50\}$. Que constatez-vous ? En déduire une autre manière de calculer $3^{2^{2^{2^2}}}$ et l'implémenter.
- Définir de nouveau une fonction `fibonacci` en utilisant votre fonction d'exponentiation rapide appliquée à une matrice carrée de taille 2 et directement en utilisant l'expression exacte de F_n pour tout $n \in \mathbb{N}$. Que constatez-vous ? Comparez les vitesses d'exécution de vos différentes implémentations.

4 $\mathbf{Q}(\sqrt{5})$

Une façon de définir le corps $K = \mathbf{Q}(\sqrt{5})$ des nombres réels de la forme $a + b\sqrt{5}$ avec $(a, b) \in \mathbf{Q}^2$ est la suivante :

```
K.<u> = NumberField(x^2-5)
```

La variable u représente alors l'élément $\sqrt{5}$.

- Calculer $\frac{1}{u}$ et u^3.
- En utilisant K, donner une autre implémentation de la fonction `fibonacci`. Comparez l'efficacité de différentes versions et afficher la liste des trente premiers termes de la suite.

1. On rappelle qu'une fonction récursive est une fonction qui fait appel à elle-même.

5 Listes et matrices

Une suite ordonnée d'expressions s'appelle une **liste**. Par exemple, `s=[4,3,3,6]`. La commande `s[i]` appelle le i-ème élément de la liste `s`, sachant qu'**attention** le premier élément est numéroté **zéro**! Ainsi, dans l'exemple ci-dessus `s[1]=3`. La longueur de la liste `s` s'obtient grâce à la commande `len(s)`. Pour obtenir la liste des entiers entre 1 et n, on peut utiliser `range(1,n+1)` ou `[1..n]`. Pour construire une liste obtenue en faisant varier un paramètre, l'opérateur `for` est très utile. Par exemple, pour générer la liste des entiers $[0, 1, 4, 9, 16]$, on peut utiliser la syntaxe `[i^2 for i in range(5)]`.

- Faire la liste des entiers naturels multiples de 3 et inférieurs à 1000. Tester les commandes `s+s`, `4*s` et enfin `s.append(pi)` suivies de l'affichage de `s` pour `s=[3*i for i in range(10)]`. Comment multiplier tous les termes de `s` par 4? Comment mettre tous les termes de `s` au carré?
- Créer la liste des six premières dérivées de $x \mapsto xe^{-x}$ sous forme factorisée.
- Pour $n = 4$, définir dans `Sage` la matrice carrée de taille n dont le terme (i, j) (pour $0 \leq i, j < n$) est i^j. S'entraîner à calculer son déterminant, extraire sa troisième colonne, sa deuxième ligne, calculer le produit de cette matrice par un vecteur, donner son inverse et son polynôme caractéristique.

6 Un peu d'arithmétique et de théorie des nombres

Un petit avant-goût du TP de la semaine prochaine avec un peu d'arithmétique.

- Étudier la différence entre les commandes `mod(a,m)` et `a%m` qui donnent la classe de `a` modulo `m`. Calculer alors le reste de la division euclidienne de 53 par 22. On pourra utiliser les commandes `%` et `\\`.
- Calculer le pgcd d de 15132 et de 103196 en utilisant la commande `gcd` et déterminer deux entiers u et v tels que $d = 15132u + 103196v$. Retrouver la valeur de d en utilisant la commande `factor`.
- Reprendre les questions précédentes avec les polynômes `-x^8+x^7+x^5-x^3-x+1` et `x^7-x^6-x+1`. Donner le produit des deux polynômes. La commande `X=polygen(QQ,'X')` permet de travailler dans l'anneau de polynômes $\mathbf{Q}[X]$.
- Que font les fonctions `is_prime`, `is_irreducible`, `previous_prime` et `next_prime`? Produire la liste des nombres premiers jumeaux (tels que p et $p + 2$ soient premiers) inférieurs à 1000.

Pour finir, un petit peu de théorie des nombres en bonus pour celles et ceux qui auraient fini le reste du TP en avance!

- Comparer en utilisant `Sage` les fonctions suivantes :

$$x \mapsto \pi(x) = \#\{p \leqslant x \ : \ p \text{ premier }\}, \quad x \mapsto \frac{x}{\ln(x)} \quad \text{et} \quad x \mapsto \mathrm{Li}(x) = \int_2^x \frac{\mathrm{d}t}{\ln(t)}.$$

  Que constatez-vous? Pouvez-vous aboutir à une conjecture similaire concernant les nombres premiers jumeaux?

- Comparez $x \mapsto \displaystyle\sum_{p \leqslant x} \frac{1}{p}$ où la somme porte sur tous les nombres premiers inférieurs à x avec la fonction $x \mapsto \ln(\ln(x))$. Que conjecturez-vous concernant la série $\displaystyle\sum_{p \text{ premier}} \frac{1}{p}$? Et concernant $\displaystyle\sum_{p \leqslant x} \frac{1}{p} - \ln(\ln(x))$?

- Reprendre la question précédente pour $\displaystyle\sum_{\substack{p \text{ premier} \\ p \equiv 1 (\mathrm{mod}\ 4)}} \frac{1}{p}$.

- Que parvenez-vous à conjecturer en utilisant `Sage` concernant $\displaystyle\sum_{p \text{ jumeaux}} \left(\frac{1}{p} + \frac{1}{p+2}\right)$ où la série porte sur tous les nombres premiers jumeaux?

7 Un exercice supplémentaire s'il reste du temps

Soit $u_0 \in \mathbf{N}^*$ donné et $A \geqslant 1$. On définit alors une suite $(u_n)_{n\in\mathbf{N}}$ par récurrence de la façon suivante

$$u_{n+1} = \begin{cases} \frac{u_n}{2} \text{ si } n \equiv 0 \,(\text{mod } 2) \\ Au_n + 1 \text{ sinon.} \end{cases}.$$

On note alors $f(u_0, A)$ la valeur minimale de $n \in \mathbb{N}$ telle que $u_n = 1$ (éventuellement $+\infty$ si la suite n'atteint jamais cette valeur). Calculer avec `Sage` les valeurs de $f(u_0, 3)$ pour $u_0 \in \{2, \ldots, 100\}$. Que pouvez-vous conjecturer ? Que se passe-t-il lorsque $A = 5$?