Université Lyon 1 Année 2020-2021
M1G - UE Logiciels

TP5 – Bases de Sage 1

1 Pour commencer

1.1 Présentation du logiciel

Sage est un logiciel libre de mathématiques sous licence GPL qui combine la puissance de nombreux programmes libres dans une interface commune basée sur le langage de programmation **Python**.

Sage permet de faire des mathématiques générales et avancées, pures et appliquées. Il couvre une vaste gamme de mathématiques, dont l'algèbre, l'analyse, la théorie des nombres, la cryptographie, l'analyse numérique, l'algèbre commutative, la théorie des groupes, la combinatoire, la théorie des graphes, l'algèbre linéaire formelle, etc.

Sage a deux modes d'utilisation :

— un mode interactif (ou notebook), dont l'interface est un navigateur web disponible par exemple à l'adresse suivante :
`https://jupyter.mecanique.univ-lyon1.fr`
et se connecter en utilisant son identifiant et son mot de passe Lyon 1 habituels. Il suffit ensuite d'aller dans l'onglet `New` et de choisir `SageMath 8.7`.
↪ *L'installation de Sage n'est pas obligatoire pour le mode bloc-note, le logiciel fonctionnant alors en mode client serveur. Ce mode permet de partager ou publier simplement feuilles de calcul, figures et graphiques.*

— un mode ligne de commande après installation du logiciel à l'adresse suivante :
`http://www.sagemath.org/download.html`

Sage est un outil très puissant pour faire une grand variété des calculs formels très simples. Pour un.e enseignant.e, il peut se révéler un outil particulièrement utile pour concevoir des exercices !

1.2 Aide

La documentation en ligne peut être trouvée à l'adresse suivante : `http://www.sagemath.org/doc/index.html`

On peut accéder de manière interactive à la description d'une fonction, sa syntaxe et des exemples d'utilisation en la faisant suivre d'un point d'interrogation (exemple : `sin?`).

En cas de doute sur l'orthographe de la fonction, la touche tabulation <Tab> à la suite d'un début de mot montre quelles sont les commandes commençant par ces lettres.

1.3 Remarques

— Sage utilise des classes, c'est à dire que chaque objet est défini par ses attributs et un certain nombre de *méthodes*, c'est à dire de fonctions spécifiques à cet objet. Par exemple, le type `vector` est défini par un tableau d'entiers, de réels ou de complexes, et possède une méthode `degree` qui est appelée de la manière suivante :

```
v=vector([1,2,3])
v.degree()
```

Une fonction non liée à un objet spécifique sera appelée de manière classique

```
v=vector([1,2,3])
show(v)
```

— Une même méthode a souvent une variante spécifique adaptée à chaque type.
Exemple : `expand_trig()` est une version de la méthode `expand()` spécifique aux expressions trigonométriques.

— ATTENTION ! La numérotation des indices des vecteurs commence à 0.

1.4 Bibliographie

Un livre de référence pour débuter : *Calcul mathématique avec Sage* de P. Zimmerman *et al.*. Une version électronique est disponible librement à l'adresse suivante :

`http://dl.lateralis.org/public/sagebook/sagebook-web-20130530.pdf`

Une version physique peut être achetée sur Amazon.fr pour une dizaine d'euros.

2 Exercices préliminaires

Pour tous les exercices suivants, écrire les commandes suivantes et observer les réponses du logiciel.

Exercice 1 - Calculatrice, scalaires, vecteurs

Les opérations sous Sage sont simlaires à celles sous Python... avec des différences notables !

```
5
2+5
2345/34578
2345./34578
1/3+2
1./3+2
```

Que fait l'ajout du . ?

```
pow(5,1/3)
5**(1/3)
5^(1/3)
5.^(1/3)
5^(1./3)
```

Contrairement à Python, Sage utilise le symbole `^` pour l'exposant.

```
a=10
a
a+5
b=a+5
b

c=[1,-2,7,0,10,8,3]
2*c
c=vector([1,-2,7,0,10,8,3])
2*c
c[3]
c[0]+c[1]+c[2]+c[3]+c[4]
c[0]=0
c
c.degree()
```

Que renvoie la méthode `degree()` ?

Exercice 2 - Fonctions usuelles

Sage comporte beaucoup de fonctions natives, ce qui ne nécessite pas d'importer de package.

```
factorial(100)
binomial(4,2)
sqrt(2)
sqrt(2.)
cos(2*pi)
exp(1)
```

Sauf demande explicite de la part de l'utilisateur, les opérations sont faites dans $\mathbb{C}$

```
sqrt(-1)
2+3*I
x=-1; y=2; z=3.4;
abs(x)
max(x,y)
min(x,y)
exp(x)
ln(y)
sin(y)
arctan(x)
ceil(z)
floor(z)
sqrt(y)
sqrt(y).n()
```

Que fait la dernière commande ?

Exercice 3 - Commandes usuelles

Variable symbolique

Les variables symboliques sont déclarées à l'aide de la commande `var`.
En regardant l'aide, donner toutes les manières de déclarer des variables muettes x, y, z.

Sommes

```
var('i,n')
sum(i^3,i,1,20)
sum(i^3,i,1,n)
sum(i^3,i,1,n).factor()
sum(1/i^4,i,1,oo)
```

Limites

```
var('x')
((1+1/x)^x).limit(x=oo)
(sin(x)/x).limit(x=0)
(1/x).limit(x=0, dir='right')
```

Dérivées, primitives, intégrales

```
var('x')

sqrt(x).diff(x)
sqrt(x).diff(x,2)

sin(x).integral(x)
sin(x).integral(x,0,1)
exp(x^2).integral(x,-1,2)

var('x y')
P=x^2+3*x*y^2+y^3
P.diff(x)
P.diff(x,y)
P.diff(x,y,y)
```

Exercice 4 - Graphiques

Le tracé de graphiques est simplifié par rapport à Python car l'appel à un package externe comme `matplotlib` n'est pas nécessaire.

Courbe 2D classique

```
var('x y t')
plot(sin(x),x,-pi,pi)
plot([x,x^2],x,0,1)
```

Les mêmes options de tracé que sur Python sont disponibles (couleur, type de trait, etc.), voir l'aide.

Courbes paramétriques

```
p1=parametric_plot((cos(x),sin(x)),(x,0,2*pi))
p2=parametric_plot((cos(x),sin(x)^2),(x,0,2*pi))
p3=parametric_plot((cos(x),sin(x)^3),(x,0,2*pi))
show(p1+p2+p3, axes=false)
```

Courbes polaires

```
var('t');
e, n=0.5, 5
g1=polar_plot((1+e*cos(n*t)),(t,0,2*pi),
          plot_points=5000)
g1.show(aspect_ratio=1);
```

Des tracés plus complexes sont facilement accessibles

Tracé des points

```
p=point((4,3))+point((5, 7))
p+=point((10, 10))+point((11, 14))
p.set_aspect_ratio(1)
p.show()
```

Tracé de segments

```
A=(1,0)
B=(3,2)
C=(-1,1)
L=line([A,B], color="red", thickness=2)
L+=line([B,C], color="red", thickness=2)
L+=line([C,A], color="red", thickness=2)
plot(L)
```

Tracé de figure pleine

```
L=[[-1+cos(pi*i/100)*(1+cos(pi*i/100)),
2*sin(pi*i/100)*(1-cos(pi*i/100))]
for i in range(200)]
polygon(L).show()
```

La visualisation 3D n'est pas spécialité de Sage, on peut avoir des problèmes de visualisation.

```
p=plot3d(x^2 + y^2, (x,-2,2), (y,-2,2))
p.show()
```

Exercice 5 - Expressions

Il est possible d'afficher une expression `expr` de manière "mathématique" (LaTeX compilé) en utilisant les commandes `expr.show()` ou `show(expr)`. Il est également possible d'exporter l'expression en écriture LaTeX.

```
P=x^2+3*x*y^2+y^3
P
P.show()
latex(P)
```

Développement, factorisation, rangement

```
var('x y')
m=(x^2-1)*(x+3)
m.expand()
m.factor()
m.factor_list()

n=(x+y)*(x+1)^2
n.collect(x)
n.expand().collect(x)
n.expand().collect(y)
n.expand().collect(y).collect(x)

((x+y+sin(x))^2).expand().collect(sin(x))
```

La manière dont la factorisation est effectuée n'est pas toujours évidente...

Fractions rationnelles

```
var('x y')
r=(x^3+x^2*y+3*x^2+3*x*y+2*x
  +2*y)/(x^3+2*x^2+x*y+2*y)
show(r.simplify_rational())
show(r.factor())
show(r.factor().expand())

var('x y a b c')
r=((x-1)*x/(x^2-7)+y^2/(x^2-7)
  +b/a+c/a+1/(x+1))
show(r.combine())
```

Que fait la méthode `.combine()` ?

```
r=1/((x+1)^2*(x+2))
show(r.partial_fraction())
show(r.partial_fraction(x))
show(r.partial_fraction(y))
```

Opérations trigonométriques

```
var('x')
(cos(x)^2+sin(x)^2).simplify()
(cos(x)^2+sin(x)^2).simplify_trig()

sin(2*x).expand_trig()
(sin(x)^2).reduce_trig()
```

Autres opérations

```
var('n x')
f=factorial(n+1)/factorial(n)
f.simplify_factorial()

var('a b')
f=(a-b)^2/(2*(sqrt(a)+sqrt(b))^2)
f.canonicalize_radical()

var('x')
f=(e^(I*x)-e^(-I*x))/(I*e^(I*x)+I*e^(-I*x))
f.simplify_rectform()

var('k,n')
ex=sum(abs(-k*k+n),k,1,n)(n=8); ex.show()
ex.expand_sum()
```

↪ *La commande* `simplify_full` *permet d'appliquer les fonctions (dans l'ordre)* , elle peut donc être utilisée par défaut.

- `simplify_factorial`
- `simplify_rectform`
- `simplify_trig`
- `simplify_rational`
- `expand_sum`

```
var('k n')
S=(k*binomial(n,k)-n*binomial(n-1,k-1))
show(S)
show(S.simplify_full())
```

Exercice 6 - Polynômes

Définition

Les polynômes peuvent être définis de deux manières :

- Comme des expressions polynômiales :
  ```
  var('x')
  p=(2*x+1)*(x+2)*(x^4-1)
  ```
- Comme des éléments d'un certain anneau, par exemple $\mathbb{Q}[X]$:
  ```
  x=polygen(QQ,'x')
  p=(2*x+1)*(x+2)*(x^4-1)
  ```

Essayer d'utiliser les fonctions `degree`, `factor`, `expand` et `collect` dans les deux cas.

Anneaux

Il est possible de définir les polynômes dans d'autres anneaux. Essayer les lignes de commandes suivantes en remplaçant successivement `QQ` par `RR` puis `CC`.

```
x=polygen(QQ,'x')
p=(2*x+1)*(x+2)*(x^4-1)
show(p.factor())
```

Exercice 7 - Résolution

Sage permet de résoudre une grande variété de problèmes pour lesquels une méthode générale existe.

```
var('x')
solve(-x^2+14*x-49==0,x)
solve(-x^2+14*x-49<0,x)
solve(-x^2+14*x-49,x)
solve(exp(x)-x-1==0,x)
```

Que se passe-t-il pour la dernière équation ?
Il est également possible de résoudre des systèmes.

```
var('x y z')
solve([x+y==1, 2*x-y==3], x, y)
solve([x+y+z==1, 3*x-2*y-z==2,
      -x+y-z==0], x,y,z)

S=solve([x^2+y^2==1, y^2==x^3+x+1], x, y)
```

Pour ce système, récupérer la valeur de la troisième solution de x. On pourra remarquer que la solution est donnée dans un tableau dont il faut récupérer le troisième élément. Cet élément est constitué d'un couple dont on veut récupérer le premier élément. Cet élément est une égalité dont on souhaite récupérer le membre de droite, ce qui pourra être fait en utilisant la méthode `.rhs()`.

```
solve([x+y==3, 2*x+2*y==6], x, y)
solve([cos(x)*sin(x)==1/2, x+y==0], x, y)
```

Que représentent les inconnues `r1` et `z36` (ou `z60`) qui apparaissent dans les deux dernières réponses du logiciel ?

La résolution de `solve` est toujours faite dans $\mathbb{C}$.

```
solve(x^2+1==0,x)
solve(x^2+1,x)
(x^2+1).roots()
```

Quelle est la différence entre `root` et `solve` ?

```
(x^3+2*x+1).roots(x)
(x^3+2*x+1).roots(x, ring=RR)
(x^3+2*x+1).roots(x, ring=CC)

q=sin(x)+sin(2*x)+sin(3*x)
solve(q,x)
```

Que se passe-t-il pour la dernière équation ? Deux solutions sont possibles dans ce cas :

- Essayer de transformer l'expression pour obtenir une forme plus facile à résoudre :
  ```
  f=q.simplify_trig()
  solve(f,x)
  ```
- Chercher une solution numérique :
  ```
  find_root(q,0.1,pi)
  ```

Exercice 8 - Programmation

La structure des boucles et des fonctions de Python et de Sage sont les mêmes.

1. En utilisant une boucle `for`, programmer une fonction itérative qui calcule la somme des n premiers entiers et qui affiche les sommes intermédiaires.
2. En utilisant une boucle `if`, programmer une fonction récursive qui calcule le n-ième terme de la suite de Fibonacci.
