

FEUILLE D'EXERCICES n° 2

Équations, polynômes

1. ÉQUATIONS, DÉRIVATION, INTÉGRATION

Pour résoudre une équation, ou un système d'équations, on peut utiliser la commande `solve`. Attention ! Pour cette fonction, comme pour la plupart des fonctions de Sage, il faut d'abord définir les variables.

1) Essayer la commande

```
solve([x-1==0],x)
```

Ok, ça marche. Maintenant essayer

```
solve([t-1==0],t)
```

La variable `x` est une exception. Toute autre variable doit absolument être déclarée au préalable. `t=var('t')`

```
solve([t-1==0],t)
```

```
parent(t)
```

La variable ainsi définie appartient donc à l'ensemble `Symbolic Ring`, qu'on peut noter `SR`.

Si on fait `parent(x)`, on voit que la variable `x` est par défaut dans `Symbolic Ring`.

Taper

```
reset('t')
```

puis

```
parent(t)
```

On a effacé la variable `t`.

2) Dans Aide, Sage Tutorial, *Basic algebra and calculus*, faire les exercices des paragraphes *Solving equations exactly*, *Solving equations numerically* puis *Differentiation, integration etc.*

Remarque. La commande `solve` résout les équations dans \mathbb{C} . Testez par exemple

```
solve(x^2 == -1,x).
```

Pour résoudre des équations dans un corps fini, il faut procéder différemment (on verra ci-dessous comment trouver les racines d'un polynôme dans un corps fini).

2. ANNEAUX ET CORPS

Sage connaît par défaut de nombreux anneaux et corps. Regardez par exemple les anneaux et corps suivants (en tapant leur nom et en validant)

`CC`

`RR`

```

QQ
ZZ
Integers(7)
Integers(10)
GF(5)
GF(25)

```

La commande `GF` permet de définir un corps fini. Elle prend en argument une puissance d'un nombre premier. La syntaxe pour définir un corps fini est `K.<a> = GF(25)`. Quand vous écrivez ça, sage définit K comme étant le corps fini à 25 élément, et a est un générateur de K . En particulier, sage représente tous les éléments de K sous la forme $x + a*y$ où x et y vivent modulo 5. Si on avait pris un corps fini à 5^k éléments, les éléments seraient des polynômes de degré $k - 1$ en a .

1) Par défaut, sage ne choisit par d'anneaux/corps précis pour un élément, tant qu'il n'a pas besoin de le faire. Testez par exemple

```

1 in CC
1 in ZZ
1 in Integers(10)
1 in GF(25)

```

Par contre

```

1.5 in RR
1.5 in QQ
1.5 in Integers(5)
K.<a> = GF(25)
a in K
a in CC

```

2) On peut forcer une variable x à appartenir à un anneau R en faisant `R(x)`. C'est une façon de préciser à sage qu'on ne s'intéresse qu'à cet anneau. La variable n'appartient alors plus forcément à tous les autres anneaux/corps comme avant. Testez

```

ZZ(1) in CC (la réponse est True car  $\mathbb{Z}$  est inclus dans  $\mathbb{C}$ )
CC(1) in ZZ (la réponse est False car  $\mathbb{C}$  n'est pas inclus dans  $\mathbb{Z}$ )

```

Bon, ça peut sembler logique, mais la logique s'arrête assez vite... Testez les 3 égalités entre `GF(3)(1)`, `CC(1)` et `ZZ(1)`. La conclusion, c'est qu'il faut faire attention quand vous commencez à mélanger des variables de différents anneaux.

L'intérêt de forcer une variable à être dans un anneau précis devient plus clair quand on commence à faire des calculs

```

ZZ(2)^3
(Integers(3)(2))^3

```

Remarque. Les anneaux `ZZ` et `QQ` contiennent des éléments exacts. Si vous faites des calculs sur ces éléments, sage va augmenter la taille des éléments pour toujours les représenter de façon exacte (il n'y a pas de limite sur la taille des entiers). C'est pratique, mais ça peut devenir très lent si vous commencez à manipuler de gros entiers, ou des rationnels avec un

gros numérateur et/ou dénominateur. Si vous n'avez pas besoin de précision exacte, vous pouvez utiliser `RR`, qui ne fera les calculs qu'avec 53 bits de précision, mais plus rapidement.

3) Écrivez une fonction `puissance(x,k,n)` qui renvoie la valeur de $x^k \bmod n$. Mesurez le temps de votre fonction sur $(x, k, n) = (7, 2 \cdot 10^8, 13)$ en faisant

```
time puissance(7,2*10^8,13).
```

Si votre fonction prend plus que quelques ms, essayez de comprendre pourquoi et améliorez-la.

3. POLYNÔMES

Dans sage, toutes les valeurs sont définies en temps qu'éléments d'un ensemble bien précis, que l'on retrouve grâce à la commande `parent`, ou bien `type`.¹ Nous avons vu l'exemple des entiers, des listes, des nombres rationnels, des réels, des corps finis, de $\mathbb{Z}/n\mathbb{Z}$.

Dans cet exercice, on va voir une nouvelle structure : les anneaux de polynômes.

1) Tester

```
p=t^2-2; p
```

Ça ne marche pas : il faut définir `t`. Comme ci-dessus, si l'on met `x` plutôt que `t`, cela va fonctionner, mais `p` ne sera pas vraiment considéré comme un polynôme en `x` et les fonctions adaptées aux polynômes ne fonctionneront pas de la même manière. En effet, `x` est par défaut un élément de `Symbolic Rings`.

2) On va définir l'anneau de polynômes $\mathbb{Q}[x]$, et explorer quelques fonctions définie sur cet anneau.

Expérimentez les commandes suivantes.

```
QQ (C'est le corps Q des nombres rationnels)
```

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

Cette commande définit `Qx` comme étant l'anneau de polynômes $\mathbb{Q}[x]$ et la variable `x` devient la variable x de cet anneau.

```
Qx
```

```
parent(x)
```

```
p=x^2-2; p
```

```
p=Qx([-2,0,1]); p (c'est une autre façon de définir le même polynôme)
```

```
p.degree()
```

```
p.is_irreducible()
```

```
factor(p)
```

```
p.roots()
```

```
p.roots? (pour demander le mode d'emploi de la fonction)
```

```
p.roots(ring=RealField())
```

```
p(2); p[2]
```

```
p.coefficients()
```

¹Ce qui est un peu contradictoire avec le fait que 1 soit à la fois dans `CC`, `ZZ` et `GF(3)`, mais bon, c'est quand même bien pratique de ne pas avoir des notations différentes pour 1 dans chaque anneau...

```
p.coefficients(sparse=false)
```

```
list(p) (comme au dessus mais en plus court)
```

On peut aussi définir $\mathbb{R}[x]$:

```
Rx.<x>=PolynomialRing(RR)
```

Si on veut considérer p comme un polynôme de $\mathbb{R}[x]$, on fait :

```
p=Rx(p)
```

Alors, la commande `p.roots()` donnera les racines de p dans son corps de base, c'est-à-dire \mathbb{R} .

Remarque. Plus généralement, si l'on a défini un objet A , qui est un certain ensemble ou structure, et si a est un objet susceptible de définir un élément de A , alors on définit cet élément par la commande $A(a)$.

Par exemple, si $A = \mathbb{Z}/n\mathbb{Z}$ et si a est un entier, alors $A(a)$ est l'élément de $\mathbb{Z}/n\mathbb{Z}$ défini par a . Nous avons déjà utilisé cette construction, par exemple quand on a posé ci-dessus `p=Qx([-2,0,1])`, nous avons défini un élément de $\mathbb{Q}x$ (c'est-à-dire $\mathbb{Q}[x]$) à partir d'une liste. De même, `list(p)` permet de définir une liste à partir du polynôme p . Nous retrouverons ce principe à maintes reprises.

3) Il y a d'autres façons de définir les anneaux de polynômes. Dans `Aide, Sage tutorials`, faire les exercices de `Polynomials`.

4) Expérimenter les commandes suivantes

```
PR=PolynomialRing(QQ,3,'a');PR
```

```
p=a0+2*a1
```

Problème! Il faut donner un nom aux variables.

```
v=PR.gens();v
```

```
p=v[0]+2*v[1];p
```

5) Nous allons ici calculer toutes les solutions polynomiales de degré inférieur ou égal à 2 de l'équation différentielle

$$(1) \quad (3t - 2)y'' + 2t^2y' - 2(2t - 1)y = 0$$

en utilisant un polynôme à coefficients indéterminés.

Nous allons ici utiliser la commande `solve`. Lorsque nous aurons vu (ou revu) le calcul matriciel sur Sage, nous pourrons refaire le même exercice en utilisant les matrices.

Il y a différentes manières de faire. Par exemple :

```
SRt.<t>=PolynomialRing(SR) (c'est l'anneau des polynômes de variable t sur l'anneau Symbolic Ring)
```

```
a0,a1,a2 = var('a0 a1 a2')
```

```
p = a0+a1*t+a2*t^2
```

```
p1 = diff(p,t)
```

```
p2 = diff(p1,t)
```

```
q = (3*t-2)*p2+2*t^2*p1-2*(2*t-1)*p
```

```
l = q.coefficients()
```

```
solve(l,[a0,a1,a2])}
```

C'est une possibilité, mais comment faire si l'on veut automatiser ce calcul en une fonction qui prendrait le degré maximal du polynôme recherché en paramètre ? Cela ne semble pas évident...

Voici une autre manière de faire, qui semble plus facile à automatiser. Commençons par définir les indéterminées. Pour cela, on définit un anneau de polynômes à plusieurs variables (ces variables seront les variables de notre polynôme en t).

```
PR=PolynomialRing(QQ,3,'b');PR
```

PR est l'anneau des polynômes à coefficients dans \mathbb{Q} et à trois indéterminées b_0 , b_1 , b_2 . Si on tape `b0`, on voit qu'il y a un problème, comme on l'a déjà vu dans la question 4. Il faut renommer les variables.

```
v=PR.gens();v;v[1]
```

C'est mieux, mais les listes sont plus maniables que les n -uplets, c'est pourquoi je préfère écrire

```
v=list(PR.gens());v
```

La commande `list('***')` transforme `'***'` en liste, conformément à la remarque générale de la fin du 2.

Reste à définir l'anneau des polynômes en t .

```
PRt.<t>=PolynomialRing(PR);PRt
```

PRt est donc l'anneau $\mathbb{Q}[b_0, b_1, b_2][t]$, et chaque indéterminée b_i correspond ici à `v[i]`. On cherche les polynômes de la forme

$$P = b_2 t^2 + b_1 t + b_0$$

qui sont solutions de (1). On pose

```
p=v[2]*t**2+v[1]*t+v[0]
```

Puis on fait comme ci-dessus

```
p1 = diff(p,t);
```

```
p2 = diff(p1,t)
```

```
q = (3*t-2)*p2+2*t^2*p1-2*(2*t-1)*p
```

```
l = q.coefficients()
```

```
solve(l,v)
```

Ah! Cela ne marche pas! C'est que dans `solve`, il faut que les `v[i]` soient des éléments de `Symbolic Ring`.

Pour cela, on peut faire :

```
for i in range(3):
```

```
    v[i]=SR(v[i])
```

```
for i in range(3):
```

```
    l[i]=SR(l[i])
```

et ensuite seulement :

```
solve(l,v)
```

(c'est sur la boucle `for` ci-dessus que le fait que `v` soit une liste plutôt qu'un triplet est utile : si `v` est un triplet, on ne peut pas modifier ses composantes).

Cette dernière commande affiche alors les coefficients des solutions de l'équation. Cela peut nous suffire, mais on peut aussi affiner en affichant les polynômes solutions eux-mêmes. Pour cela, on peut utiliser la commande

```
S=solve(1,v,solution_dict=True)
```

Alors, **S** est la liste des solutions. Ici, il n'y en a qu'une : **S[0]**, qui est un dictionnaire dont les entrées sont numérotées par les **v[i]**. Ainsi, les solutions sont données par

```
sum([S[0][v[i]]*t**i for i in range(3)])
```

6) Écrire une fonction qui, étant donnés a , b et c dans $\mathbb{Q}[t]$ et un entier d , donne les polynômes de degré inférieur ou égal à d solutions de l'équation

$$ay'' + by' + cy = 0.$$

4. ÉVALUATION ET INTERPOLATION

1) Écrire une fonction qui prend en entrée un polynôme $P \in K[X]$ et un élément $x \in K$ et renvoie $P(x)$ en effectuant au plus $\deg(P)$ multiplications et $\deg(P)$ additions dans K . [Indice : si vous n'avez pas d'idées, aller voir "Horner's method" sur wikipedia.]

2) Écrire une fonction `multi_eval(P,liste)` qui prend en entrée un polynôme P et une liste de valeurs `liste = [x1,...,xk]` et renvoie la liste des évaluations $[P(x_1), \dots, P(x_k)]$.

3) En utilisant les polynômes interpolateurs de Lagrange $L_i = \prod_{j \neq i} \frac{X-x_j}{x_i-x_j}$, écrire une fonction `interpolation(liste_points, liste_valeurs)` qui prend en entrée une liste `liste_points = [x1, ..., xk]` de points distincts ($x_i \neq x_j$ pour tout $i \neq j$), et une liste de valeurs `liste_valeurs = [y1, ..., yk]` et renvoie un polynôme P de degré $k-1$ tel que $P(x_i) = y_i$ pour tout i .

[Note : pour récupérer la taille d'une liste, vous pouvez utiliser la commande `len(liste)`]

4) Vérifiez que vos fonctions `multi_eval` et `interpolation` sont correctes en les utilisant l'une après l'autre et en vérifiant que vous retombez bien sur le polynôme de départ. Faites des tests sur \mathbb{R} et d'autres sur \mathbb{F}_p . Que se passe-t-il si $\deg(P) \geq p$?