Université Paris 13 Année 2021–2022

M2 Mathématiques des données

Codes correcteurs – Aide d'utilisation de `sagemath`

5 janvier 2022

Table des matières

1 Présentation

Introduction. Sagemath est un logiciel libre de calcul formel (plus largement dédié aux mathématiques) dont l'interface est basée sur le langage `python`. Sagemath s'appuie sur quantité de logiciels libres et de bibliothèques existantes, parmi lesquels `Maxima` et `Singular` (logiciels de calcul formel génériques), `PARI/GP` et `NTL` (pour la théorie des nombres), `LinBox` (pour l'algèbre linéaire et le calcul matriciel). Il s'interface également avec d'autres logiciels, libres ou non, comme `GAP`, `Magma`, `Mathematica`, `Maple`, etc.

La page d'accueil de Sagemath, sur laquelle vous pourrez trouver différentes versions du logiciel en téléchargement libre ainsi que de la documentation, est la suivante :

`https://www.sagemath.org/fr/`

Vous trouverez notamment un manuel d'utilisation en français et sous licence libre *Creative Commons* à l'adresse :

`https://www.sagemath.org/sagebook/french.html`

Modes d'utilisation. On peut utiliser `sagemath` sous deux modes différents.

1. En **ligne de commande** (souvent couplé avec un éditeur de texte). L'invite de commande débute alors par

```
sage:
```

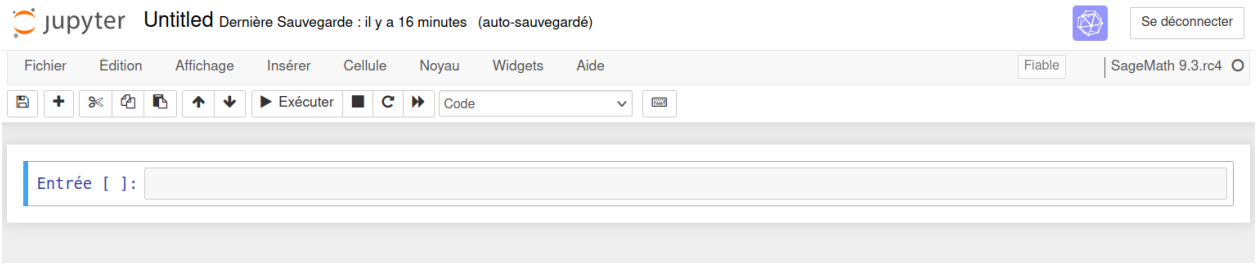
   Vous pouvez y taper des commandes, et la réponse vous est donnée directement :

```
sage: 1+1
2
```

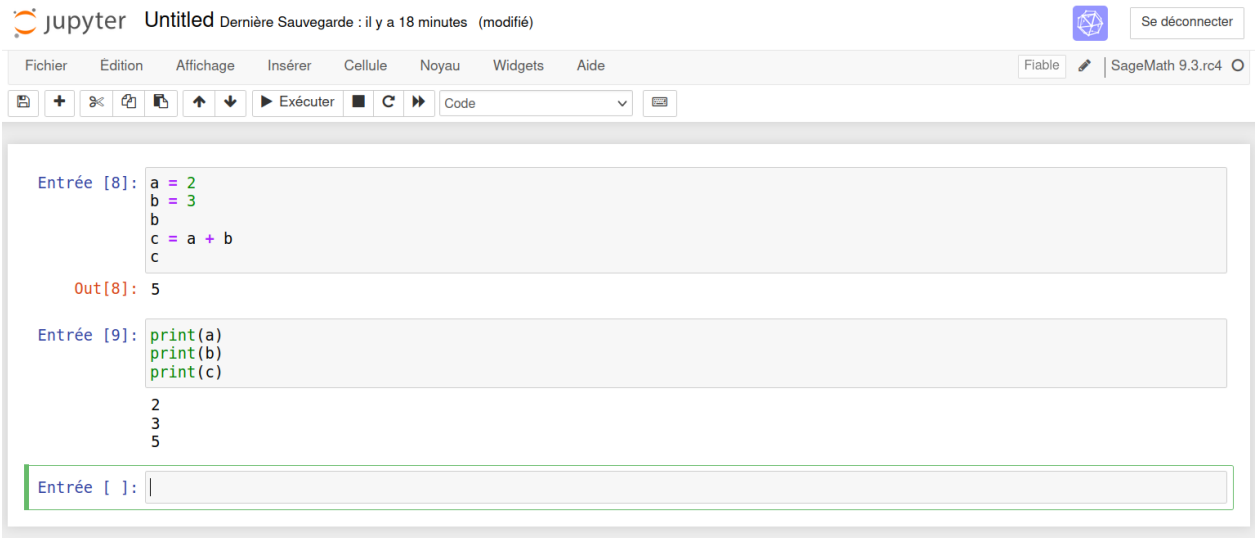
   Après avoir édité un script sage (par exemple nommé `script.sage`), vous pouvez aussi l'exécuter en ligne de commande :

```
sage: load("script.sage")
```

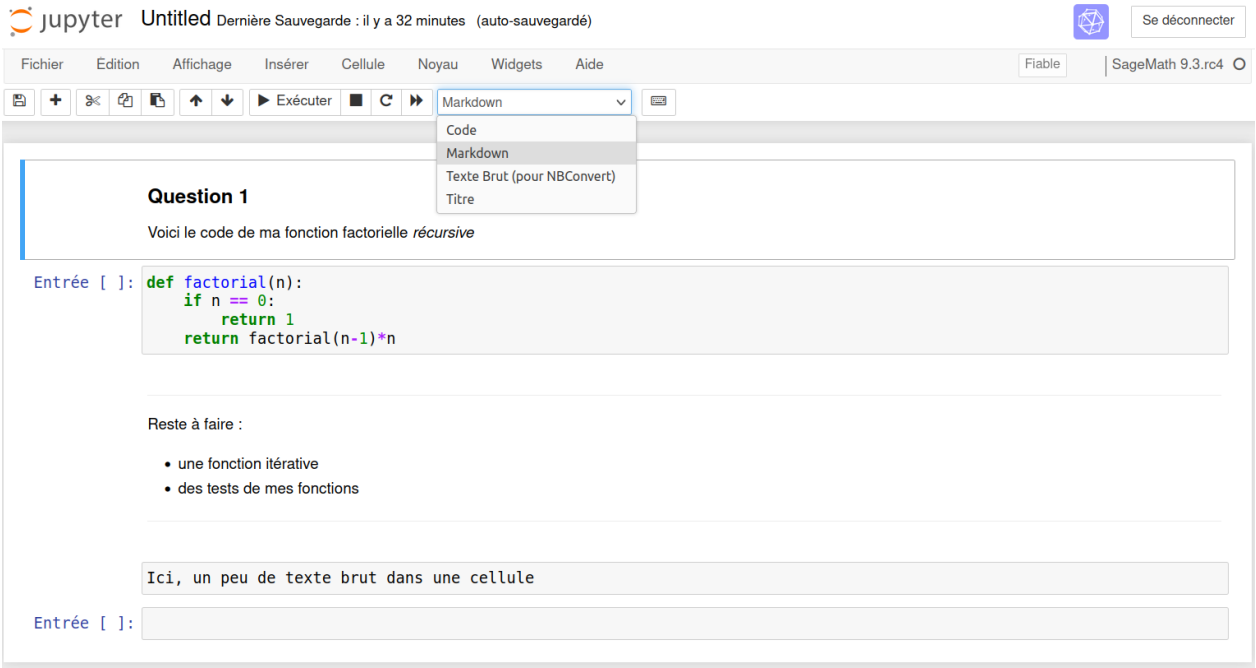
2. En mode **bloc-note**, dans un *notebook Jupyter*. Dans ce cas, une page de navigateur s'ouvre :



Vous entrez vos commandes dans des *cellules*, que vous pouvez ensuite éxécuter globalement. La valeur de sortie de la dernière commande de la cellule est affichée dans une sous-cellule. Vous prouvez forcer l'affichage d'autres valeurs par la commande `print()`.



Vous pouvez également préciser des sections, commentaires, etc. en format `Markdown` (pensez à exécuter les cellules pour avoir le rendu) ou en texte brut.



Exécution locale ou en ligne. Ces deux modes d'utilisation peuvent être utilisés :

1. Ou bien **directement sur votre machine personnelle, sans accès internet**. Dans ce cas vous devez avoir préalablement installé une version de `sagemath` sur votre machine. L'installation peut être effectuée avec les sources du logiciel, ou bien directement avec le binaire exécutable. Voir

   `https://www.sagemath.org/fr/telecharger.html`

   pour plus de détails.
2. Ou bien **en ligne, sur des serveurs distants**. Dans ce cas, l'installation du logiciel n'est pas nécessaire, bien qu'elle puisse être utile. Il existe en effet un service en ligne permettant d'utiliser des serveurs de calculs à distance (gratuitement sous certaines limites). Ce service est disponible à l'adresse suivante :

   `https://cocalc.com/`

   Pour des scripts de quelques lignes ou des calculs rapides, vous pouvez également utiliser :

   `https://sagecell.sagemath.org/`

Objectif du document. Vous trouverez dans ce document un index d'utilisation de `sagemath`, focalisé sur l'implantation d'algorithmes issus de la théorie des codes.

2 Objets et fonctions sage utiles en théorie des codes

2.1 Corps finis

Il existe deux fonctions permettant d'instancier un corps fini à q éléments :

```
sage: q = 16
sage: F = GF(q)
sage: G = FiniteField(q)
sage: F == G
True
```

Si $\mathbb{F}_q = \mathbb{F}_p(u)$ où p est premier, on peut instancier en même temps l'élément $u \in \mathbb{F}_q^\times$ d'ordre $q-1$ (choisi arbitrairement) par `F.<u> = GF(q)` :

```
sage: q = 16
sage: F.<u> = GF(q)
sage: min([i for i in range(1, 16) if u**i == 1])
15
```

La plupart des fonctions utiles sont listées dans l'index de la Table 1.

commande/code	description
`F.gen()`	retourne un élément algébrique $a \in \mathbb{F}_q$ qui définit $\mathbb{F}_q = \mathbb{F}_p(a)$ où $\mathbb{F}_p$ est le sous-corps premier de $\mathbb{F}_q$
`F.multiplicative_generator()`	retourne un générateur du groupe multiplicatif $\mathbb{F}_q^\times$
`F.cardinality()`	retourne le cardinal q du corps fini
`F.zero(), F.one()`	retourne respectivement 0 et 1, vus comme des éléments de $\mathbb{F}_q$
`F.list()`	retourne la liste des éléments de $\mathbb{F}_q$
`F(a)`	essaie de transformer l'objet `a` en un élément de $\mathbb{F}_q$
`for z in F`	itération sur les éléments du corps fini
`F.cardinality()`	retourne la valeur q
`F.prime_subfield()`	retourne le sous-corps premier $\mathbb{F}_p$ de $\mathbb{F}_q$
`F.extension(n)`	retourne l'extension de degré n de $\mathbb{F}_q$, c'est-à-dire $\mathbb{F}_{q^n}$
`x.order()`	retourne l'ordre de x dans le groupe additif $(\mathbb{F}_q, +)$
`x.multiplicative_order()`	retourne l'ordre de x dans le groupe multiplicatif $(\mathbb{F}_q^\times, \times)$
`F.random_element()`	retourne un élément aléatoire de $\mathbb{F}_q$

TABLE 1 – Commandes principales pour l'utilisation des coprs finis. La variable `F` représente le corps fini $\mathbb{F}_q$ et `x` un de ses éléments. Les opérations de base (`+`, `-`, `*`, `/`, `**`, `log(x, a)`) sont naturellement définies sur les éléments de `F`. Prendre garde néanmoins que `log(x, a)` est coûteux lorsque q est grand.

Enfin, finissons par donner un exemple d'utilisation, avec certaines valeurs de retour données en commentaire.

```
q = 16
F = GF(q)
a = F.gen()
print(a**(q-1))                          # valeur d'affichage = 1
print(a.order())                         # valeur d'affichage = 2
print(a.multiplicative_order())          # valeur d'affichage = 15
K = F.prime_subfield()
p = K.cardinality()
print(p*a == 0)                          # valeur d'affichage = True
```

Listing 1 – Illustration d'un script sage autour des corps finis

2.2 Algèbre linéaire

Sagemath permet de manipuler des structures linéaires comme (entre autres) les espaces vectoriels de dimension finie. Voici comment les instancier :

```
sage: F = GF(17)
sage: n = 4
sage: VectorSpace(F, n)
Vector space of dimension 4 over Finite Field of size 17
```

On fournit ci-dessous une table de commandes usuelles pour manipuler des espaces vectoriels ou leurs éléments.

commande/code	description
`VectorSpace(F, n)`	retourne l'espace vectoriel $\mathbb{F}^n$
`V.base_field()`	retourne le corps sur lequel est défini l'espace vectoriel `V`
`V.dimension()`	retourne la dimension de `V` sur son corps de base
`V.zero()`	retourne l'élément nul d'un espace vectoriel `V`
`V(L)`	retourne le vecteur associé à la liste `L` de longueur n
`V.random_element()`	retourne un élément aléatoire d'un espace vectoriel `V`
`V.basis()`	retourne une base de `V`
`V.basis_matrix()`	retourne une matrice dont les lignes forment une base de `V`
`V + W` ou `V.sum(W)`	retourne la somme des espaces vectoriels `V` et `W`
`V.insersection(W)`	retourne l'intersection des espaces vectoriels `V` et `W`
`V.is_subspace(W)`	teste si `V` est un sous-espace vectoriel de `W`
`x in V`	teste si le vecteur `x` appartient à l'espace vectoriel `V`
`for x in V`	lorsque `V` est fini, parcourt tous les éléments de l'espace vectoriel `V`

TABLE 2 – Commandes principales pour l'algèbre linéaire sur un corps `F` — espaces vectoriels. La variable `V` représente un espace vectoriel sur `F` de dimension `n`.

Notons que les éléments d'un espace vectoriel sont des objets de type `vector`, et peuvent être construits individuellement comme suit :

```
sage: F = GF(17)
sage: L = [1, 0, 4, -2, 341]
sage: x = vector(F, L)
sage: x
(1, 0, 4, 15, 1)
sage: x.parent()
Vector space of dimension 5 over Finite Field of size 17
```

Observons enfin que dans le script précédent, les entiers de la liste `L` ont été « castés » comme des éléments du corps fini `F`.

Sagemath permet aussi une utilisation avancée des matrices. Parmi les méthodes (nombreuses) permettant de déclarer une nouvelle matrice, relevons-en deux :

```
sage: F = GF(7)
sage: m, n = 2, 3
sage: L = [1, 2, 3, 4, 5, 6]
sage: A = matrix(F, m, n, L)
sage: A
[1 2 3]
[4 5 6]
sage: T = [[F(1), F(2), F(3)], [F(4), F(5), F(6)]]
sage: B = matrix(T)
sage: A == B
True
```

Au-delà des opérations usuelles (addition `+`, multiplication `*`, passage à la puissance `**` ou à l'inverse `.inverse()` lorsque c'est possible), la table suivante donne quelques commandes utiles à la manipulation de matrices. Observons que l'on peut multiplier des matrices par des objets de type `vector`.

commande/code	description
`matrix(F, n, m, L)`	retourne la matrice de taille $n \times m$ sur le corps `F`, dont les coefficients sont dans la liste `L` de taille nm (d'abord les coefficients de la première ligne, puis la deuxième, etc.)
`matrix(M)`	retourne la matrice correspondant à la liste de listes `M`, où la i-ème liste de `M` correspond à la i-ième ligne de la matrice
`G.nrows(), G.ncols()`	retourne le nombre de lignes et le nombres de colonnes d'une matrice `G`
`G.rank()`	retourne de le rang de la matrice `G`
`G.rref()`	retourne la forme échelonnée réduite (par lignes) de la matrice `G`
`G.pivots()`	retourne les pivots de la matrice `G`
`G.stack(H)`	retourne la matrice $\begin{pmatrix} G \\ H \end{pmatrix}$ où G et H ont même nombre de colonnes
`G.augment(H)`	retourne la matrice $\begin{pmatrix} G & H \end{pmatrix}$ où G et H ont même nombre de lignes
`G[i]`	retourne le vecteur correspondant à la i-ème ligne de `G`
`G[a:b,:]`	retourne la sous-matrice de `G` constituée des lignes entre a (inclus) et b (non-inclus)
`G[:,c:d]`	retourne la sous-matrice de `G` constituée des colonnes entre c (inclus) et d (non-inclus)
`G[I,J]`	retourne la sous-matrice de `G` dont les coefficients sont dans $I \times J$, où $I \subseteq [1, \texttt{G.nrows()}]$ et $J \subseteq [1, \texttt{G.cols()}]$
`G.row_space(), G.column_space()`	retourne l'espace vectoriel engendré par les lignes/colonnes de `G`
`G.is_singular()`	retourne `False` si la matrice carrée `G` est inversible, `True` sinon
`G.right_kernel(), G.left_kernel()`	retourne le noyau à droite/gauche de `G`, comme un objet de type `VectorSpace`
`G.right_kernel_matrix()`	retourne une matrice dont les lignes forment une base du noyau à droite de `G`
`c/M` ou `M.solve_left(c)`	retourne (si elle existe) une solution x de $xM = c$ où c est un vecteur
`M\c` ou `M.solve_right(c)`	retourne (si elle existe) une solution x de $Mx = c$ où c est un vecteur

TABLE 3 – Commandes principales pour l'algèbre linéaire sur un corps `F` — matrices.

Enfin, on propose un exemple d'utitlisation des ces fonctionnalités.

```
F = GF(7)
G = matrix(F, [[1, 2, 3], [4, 5, 6]])
r = G.rank()
n, m = G.ncols(), G.nrows()
print(r, m, n)                          # valeur d'affichage = 2 2 3
V = G.row_space()
B = V.basis()
H = G.right_kernel_matrix()
print(G*H.transpose() == 0)             # valeur d'affichage = True
print(G.echelon_form())                 # valeur d'affichage = [1 0 6]
                                        #                      [0 1 2]
v = vector([1, 4])
print(G.solve_right(v))                 # valeur d'affichage = (1, 0, 0)
```

Listing 2 – Illustration d'un script `sage` maniant des fonctions d'algèbre linéaire

2.3 Polynômes

On instancie l'anneau des polynômes à une variable (affichée `X`) sur un corps `F` par :

```
sage: R = PolynomialRing(F, "X")
```

Les opérations courantes sur les polynômes (`+`, `-`, `*`, `**`) sont déjà implantées. Comme pour les entiers, l'opération `/` représente la division exacte tandis que `//` représente la division avec reste. Notons que la division exacte peut éventuellement produire une fraction rationnelle (et non un polynôme).

commande/code	description
`R.base_ring()`	Retourne l'anneau des coefficients des `R`
`R.gen()`	Retourne le polynôme X
`R.random_element(d)`	Retourne un polynôme aléatoire de degré exactement d
`R(L)` avec une liste `L` d'élements de $\mathbb{F}_q$	Retourne le polynôme dont les coefficients sont donnés par la liste `L` (dans l'ordre de degré croissant).
`P.quo_rem(Q)`	Retourne le quotient et le reste de la division euclidienne de P par Q
`P.divides(Q)`	Teste si P divise Q
`P.coefficients(sparse=False)`	Retourne la liste des coefficients du polynôme P
`P(a)`	Évalue le polynôme P en a.
`P.derivative()`	Retourne le polynôme dérivé de P.
`P.roots()`	Retourne la liste des racines de P dans l'anneau des coefficients de `R`, listées avec multiplicité.
`P.degree()`	Retourne de degré de P.
`P[n]`	Retourne le coefficient de degré n de P.

TABLE 4 – Quelques commandes pour l'utilisation des polynômes. La variable `R` représente un anneau de polynôme dont les coefficients appartiennent à un corps `F`.

```
F = GF(101)
R = PolynomialRing(F, "X")
X = R.gen()
P = X**6 - 5*X**5 + 3*X**4 +15*X**3 - 28*X**2 + 18*X - 4
print(P)       # valeur d'affichage = X^6 + 96*X^5 + 3*X^4 + 15*X^3 + 73*X^2 + 18*X + 97
L = P.roots()
print(L)           # valeur d'affichage = [(99, 1), (1, 3)]
print(P(-2) == 0)  # valeur d'affichage = True
Q = X+2
A, B = P.quo_rem(X-7)
print(Q.divides(P) and (A*(X-7) + B == P))      # valeur d'affichage = True
print((P/(X-7)).parent()) # valeur d'affichage = Fraction Field of Univariate Polynomial
                          #                      Ring in X over Finite Field of size 101
```

Listing 3 – Illustration d'un script `sage` autour des polynômes

2.4 Autres outils

Combinatoire. Les classes suivantes pourraient vous êtres utiles.

— La classe `Subsets` instanciée par `Subsets(n, k=None)` où `k` est optionnel. C'est la classe qui gère les sous-ensembles de $\{1,\dots,n\}$ (de cardinal k si `k` est spécifié en paramètre). On peut notamment lister les éléments (`.list()`) ou les parcourir les uns après les autres (`.first()`, `.next()`, etc.).

—

Graphiques. `sagemath` permet d'utiliser la bibliothèque `matplotlib` de `python`. Par exemple, pour tracer sur un même grahique un nuage de points `L` sous la forme `[[x1,y1], [x2,y2], ..., [xn,yn]]` en rouge, ainsi que le fonction $\sin(x)$ entre 0 et 2π en bleu, on peut utiliser

```
sage: P = list_plot(L, color='red')
sage: P += plot(sin(x), xmin=0, xmax=2*pi, color='blue')
sage: P.show()
```