Compétences acquises en séance 1

— Prendre ses repères dans l'aide. (Tutoriel, aide sur mots clefs, menus, ...)
— Différences symboles/fonctions. Ex : si la variable `R` contient le symbole `(a+b)/(1-a^2)` comment créer les fonctions $a \mapsto \frac{a+b}{1-a^2}$ et $(a,b) \mapsto \frac{a+b}{1-a^2}$ à partir de `R` ? (cf. `unapply`)
— Libérer une variable pour pouvoir l'utiliser comme symbole. (cf. `purge` ou `del`)
— Premières matrices.
— Quelques dessins.

Modes Xcas et Python

— ⚠ En 2019 de nombreuses versions d'Xcas ont des problèmes lorsque la configuration par défaut est en mode Python. Il faut donc **choisir le mode Xcas lors du premier lancement**, même si l'on souhaite utiliser une syntaxe comme en python. Si au lancement vous êtes en mode python, allez dans le menu `Cfg>config du Cas` changez le mode vers xcas, **sauvez** la config et **quittez** Xcas puis relancez le.
— Pour utiliser un symbole dans une fonction (par exemple `X`) il vaut mieux [2] faire

(giac/xcas)

```
local X;
purge(X)
```

— Dans Xcas, commencer une cellule par le symbole `#` bascule son contenu en mode python quelle que soit la configuration globale (et commencer par `//` bascule en mode Xcas).

(giac/xcas)

```
# La cellule sera en mode python, ceci donnera donc 0 (le quotient de 3 par 4)
3//4
A=5   # on peut alors utiliser l'affectation comme en python. NB := marche aussi.
```

(giac/xcas)

```
// La cellule sera en mode Xcas, ceci donnera donc 3 suivi du commentaire 4
3//4
A:=5; // ici l'affectation doit se faire avec :=
```

— ⚠ **Mais attention** à ne pas valider une cellule qui ne contient que `#` car cela peut planter [3] Xcas immédiatement.
— NB : Dans une fonction definie par def on est automatiquement en mode python. L'exemple ci dessous donnera toujours 0.

(giac/xcas)

```
def test():
    return 3//4
```

Exercice I: listes, modulaires

Dans xcas (à partir de 1.4.9) et sage, on peut créer des listes automatiquement avec une syntaxe type python suivante :

```
[ 1/k for k in range(1,11) ]
```

1. `http://www.imj-prg.fr/~han/agreg`
2. certaines "anciennes" versions déclarerait automatiquement toutes les variables comme locales dans une fonction python, et leur affecterait 0, d'où le besoin de faire purge/del pour avoir un symbole.
3. semble OK avec la vieille version 1.4.9 sur clefagreg 2019

dans xcas on pourrait aussi faire de manière équivalente : `[seq(1/k, k=1..10)]` mais attention, cette fonction a de nombreuses options qui retournent des choses différentes [4] pour des raisons de compatibilité avec d'autres syntaxes. Si vous l'utilisez tester toujours votre façon de l'utiliser.

1) Créez une liste `L1` des éléments non nuls de $\mathbb{Z}/17\mathbb{Z}$.

(giac/xcas)

```
3>>  L1:=[ k mod 17 for k in range(1,17)];  // mode xcas ou python
[1 % 17,2 % 17,3 % 17,4 % 17,5 % 17,6 % 17,7 % 17,8 % 17,-8 % 17,-7 % 17,-6 % 17,-5 % 17,-4 %
4>>  L1:=[ k % 17 for k in range(1,17)];  // mode xcas uniquement
[1 % 17,2 % 17,3 % 17,4 % 17,5 % 17,6 % 17,7 % 17,8 % 17,-8 % 17,-7 % 17,-6 % 17,-5 % 17,-4 %
```

2) Créez la liste des inverse des éléments de `L1`

(giac/xcas)

```
5>>  [ 1/k for k in L1];
[1 % 17,-8 % 17,6 % 17,-4 % 17,7 % 17,3 % 17,5 % 17,-2 % 17,2 % 17,-5 % 17,-3 % 17,-7 % 17,4 %
```

3) En créant un ensemble, donnez la liste des éléments du groupe engendré par 2 dans $(\mathbb{Z}/17\mathbb{Z})^\times$.

(giac/xcas)

```
7>>  L3:=seq( (2 mod 17 )^k, k=0..16);
1,2 % 17,4 % 17,8 % 17,-1 % 17,-2 % 17,-4 % 17,-8 % 17,1 % 17,2 % 17,4 % 17,8 % 17,-1 % 17,-2 %
8>>  L4:=[ (2 mod 17)^k for k in range(17)]
[1,2 % 17,4 % 17,8 % 17,-1 % 17,-2 % 17,-4 % 17,-8 % 17,1 % 17,2 % 17,4 % 17,8 % 17,-1 % 17,-2
9>>  G:=set[L3]; // la reponse de seq n'a pas de crochets
set[1,2 % 17,4 % 17,8 % 17,-1 % 17,-2 % 17,-4 % 17,-8 % 17]
10>>  G:=set[op(L4)]; // ou bien, on enleve les crochets avec op
set[1,2 % 17,4 % 17,8 % 17,-1 % 17,-2 % 17,-4 % 17,-8 % 17]
print(
```

Exercice II: (Petits programmes)

Dans le tutoriel de votre logiciel vous trouverez les syntaxes usuelles de programmation. Sous Xcas dans une entrée normale on fait Shift+Return pour aller à la ligne sans évaluer. Mais pour un programme plus long, il est plus confortable d'insérer un éditeur de programme via le menu : `Programme>Nouveau programme`
Sous Sage, on fait Shift+Return pour évaluer et Return pour aller à la ligne sans évaluer

1) Faire l'exercice V de la séance 1.

(giac/xcas)

```
sq2:=approx(sqrt(2),10000);

newt(n,d):={
  local u,j;
  u:=approx(2,d);
  for(j:=0;j<n;j++){
    u:=u/2+1/u;
  }
  return u;
}
```

(giac/xcas)

```
54>> newt(5,10);
1.41421356237
/* Il faut augmenter la precision pour que la difference avec sqrt(2) soit non
nulle. Asymptotiquement le nombre de chiffres exacts est multiplie par 2 d'une
iteration sur l'autre.

De plus, lorsque la precision est trop faible on peut tomber sur une erreur du
type:
"Exponent overlflow". Pourquoi? Car l'erreur d'arrondi fait que l'on demande un
```

4. Ex : réponse entre crochets ou pas

```
log de quelque chose de tres proche de 0.
*/
55>> seq(floor(log10(abs(newt(k,5000)-sqrt(2)))),k=0..12);
-1,-2,-3,-6,-12,-25,-49,-98,-196,-392,-784,-1568,-3136
// Time 0.43
56>> [ floor(log10(abs(newt(k,5000)-sqrt(2))))  for k in range(13) ];
[-1,-2,-3,-6,-12,-25,-49,-98,-196,-392,-784,-1568,-3136]
```

2) Définir une fonction $g : x \mapsto \sin(x) - x/2$, dessinez son graphe et créez un programme `dicho(f,a,b,epsi)` qui cherche une racine d'une fonction f entre a et$b$ avec $f(a)f(b) < 0$ à la précision `epsi`.

Fonction dicho(f, a, b, eps)
tantque $b - a > eps$ **faire**
  $c \leftarrow \dfrac{a+b}{2}$;
  si [a] $f(a)f(c) \leq 0$ **alors**
    $b \leftarrow c$;
  sinon
    $a \leftarrow c$;
  fsi
ftantque
retourner a, b;

a. mettre une inegalité large, au cas où c soit une racine de f.

(giac/xcas)

```
dicho(f,a,b,epsi):={
    local c;
    while(b-a >epsi){
      c=(a+b)/2
      if(f(a)*f(c) <=0){
        b:=c;
      }
      else{
        a:=c;
      }
    }
    return a,b;
};
g(x):=sin(x)-x/2;
dicho(g,-1,2,1e-5); // donne: -1/262144,1/524288
dicho(g,-1,1,1e-5); // donne: -1/131072,0
```

Exercice III: Algèbre linéaire

1) a) Dans xcas les vecteurs [5] n'ont pas un type particulier ce sont juste des listes et se notent entre : `[ ]`. Par exemple, essayez : `v1:=[j for j in range(1,5)]` et `v2:=[a,b,c,d]` et `t*v1+v2`

(giac/xcas)

```
1>> v1:=[seq(j,j=1..4)];
[1,2,3,4]
2>> v2:=[a,b,c,d];
[a,b,c,d]
13>> v1*v2; // le produit scalaire
a+2*b+3*c+4*d
14>> v1+t*v2; // v1 et v2 sont bien des vecteurs
[1+a*t,2+b*t,3+c*t,4+d*t]
```

Dans Xcas, les matrices se notent `M:=[[1,2,3,4],[5,6,7,8]]` ou bien aussi `matrix([[1,2,3,4],[5,6,7,8]])` comme dans Sage. Leur lignes sont `M[0] M[1]` ... et les coefficients `M[i,j]`. Dans Xcas si l'on souhaite des indices commençant à 1 au lieu de 0 on utiliser des `M(1)` au lieu de `M[0]`

5. dans sage, type : vector

Les vecteurs s'affichent en ligne, mais ce sont bien des vecteurs et non des matrices lignes. Remarquez la différence d'affichage entre `[v1]` et `v1`, et copiez [6] collez la sortie (noire) de `[v1]` dans une zone d'entrée pour étudier les `[]`.

(giac/xcas)

```
3>>  M:=[[1,2,3,4],[5,6,7,8]]; // une matrice
[[1,2,3,4],[5,6,7,8]]
4>>  M[0]; // sa premiere ligne, c'est aussi un vecteur pour + et *
[1,2,3,4]
5>>  M
[[1,2,3,4],[5,6,7,8]]
6>>  [v1]; // une matrice ligne
[[1,2,3,4]]
7>>  M*v1;  //est bien un vecteur.
[30,70]
8>>  M*v2;
[a+2*b+3*c+4*d,5*a+6*b+7*c+8*d]
13>> v1*v2; // le produit scalaire
a+2*b+3*c+4*d
```

(sage)

```
sage: a,b,c,d=var('a,b,c,d')
sage: v1=vector([j for j in range(1,5)])
sage: v2=vector([a,b,c,d])
sage: t=var('t')
sage: t*v1+v2
(a + t, b + 2*t, c + 3*t, d + 4*t)
sage: type(v1)
<type 'sage.modules.vector_integer_dense.Vector_integer_dense'>
sage: M=matrix([[1,2,3,4],[5,6,7,8]]) # une liste de liste n'est pas une matrice ds sage
sage: M*v1
(30, 70)
sage: v1
(1, 2, 3, 4)
```

b) Les lois usuelles en algèbre linéaire utilisent encore `+` et `*`. Calculez les images de v_1 et v_2 par M (remarquez que la sortie est bien un vecteur). Calculez $M_2 = M \cdot ({}^tM)$, M_2^2. Comment obtenir rapidement M_2^n ?

c) Que fait `v1*v2` ?

d) Remaquez que dans xcas la fonction opérande : `op` permet d'enlever un niveau de crochets. On ajoute donc une ligne tres facilement sans connaitre les instructions dédiées.

(giac/xcas)

```
9>>  A:=matrix(4,4)+1; // l'identite
matrix[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]
10>>  v:=[seq(1,j=1..4)];
[1,1,1,1]
11>>  [op(A),v]; // on ajoute une ligne facilement
[[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1],[1,1,1,1]]
12>> A*v;
[1,1,1,1]
```

2) Une syntaxe très utile dans Xcas et Sage qu'il faut retenir, est de créer une matrice à partir d'une fonction [7] de deux variables en utilisant `matrix` :

a) Par exemple, créez dans votre logiciel la fonction : $f : (x, y) \mapsto x + 10^y$ puis la matrice suivante :

```
matrix(6,6,f);
```

6. Notez que l'on peut copier une sortie pour en faire une entrée, même si l'aspect semble différent. Ca n'est pas toujours facile de sélectionner une sortie à cause de la structure d'arbre d'une formule. Control a selectionne tout

7. Attention, dans sage, il faut un programme et non une fonction symbolique

(giac/xcas)

```
15>>  f:=(x,y)->x+10^y;
// Succ\'es
// End defining f
 (x,y)->x+10^y
17>>  matrix(3,3,f);
matrix[[1,10,100],[2,11,101],[3,12,102]]
21>>  matrix(3,3,(x,y)->x+10^y);
// Succ\'es
matrix[[1,10,100],[2,11,101],[3,12,102]]
```

(sage)

```
sage: fpb(x,y)=x+10**y  # Objet de type expression pose PB pour matrix
sage: fpb(1,2)
101
sage: matrix(3,3,fpb)  # PB car fpb est de type Expression
[(x, y) |--> 10^y + x         (x, y) |--> 0         (x, y) |--> 0]
[       (x, y) |--> 0 (x, y) |--> 10^y + x         (x, y) |--> 0]
[       (x, y) |--> 0         (x, y) |--> 0 (x, y) |--> 10^y + x]
sage: def f(x,y):
....:     return x+10**y
....:
sage: matrix(3,3,f)
[  1  10 100]
[  2  11 101]
[  3  12 102]
sage: matrix(3,3,lambda x,y: fpb(x,y))
[  1  10 100]
[  2  11 101]
[  3  12 102]
```

b) On souhaite travailler avec un nombre de variables que l'on peut facilement modifier. On aimerait donc des noms du type $x1, x2, \ldots$. Dans sage on peut définir un nouvel anneau de polynomes et injecter les variables si besoin. Sous Xcas, une première solution est d'utiliser plutot des noms du type `x[j]` où `x` est une lettre inutilisée. On peut alors créer la matrice de Vandermonde de taille 6 ainsi :

(giac/xcas)

```
f1(u,v):=x[u]^v;V:=matrix(6,6,f1);
```

(sage)

```
sage: n=6; R=PolynomialRing(QQ,n,x)
sage: matrix(3,3, lambda a,b: R.gen(a)^b)
[   1   x0 x0^2]
[   1   x1 x1^2]
[   1   x2 x2^2]
```

(sage)

```
# NB: pour utiliser les variables ave la definition precedente de R
sage: R.random_element()
x0*x2 + 3/7*x0*x4 + 2*x0*x5
sage: x0+1
...
NameError: name 'x0' is not defined
sage: R.inject_variables()
Defining x0, x1, x2, x3, x4, x5
sage: x0+1
x0 + 1
```

c) Calculez le déterminant de Vandermonde formel en taille 6x6 et factorisez le.

d) Lorsqu'il y a beaucoup de variables et que la taille de la matrice n'est pas trop grande, le calcul du déterminant par la méthode des mineurs s'avère plus efficace (Je ne sais pas si c'est possible sous

Sage). En déduire un moyen d'obtenir le déterminant de Vandermonde en taille 8x8, puis factorisez le. (pour les calculs longs on a intérêt à stocker d'abord le déterminant puis tenter de factoriser.)

(giac/xcas)

```
22>> f2(u,v):=(#("x"+u))^v;
...
 (u,v)->#("x"+u)^v
23>> V8:=matrix(8,8,f2);
matrix[[1,x0,x0^2,x0^3,x0^4,x0^5,x0^6,x0^7],[1,x1,x1^2,x1^3,x1^4,x1^5,x1^6,x1^7],[1,x2,x2^2,x2
// Time 0.01
24>> d8:=det_minor(V8); // 178 fois plus rapide que det(V8)
Done
// Time 0.15
25>> factor(d8); // NB: souvent bien plus rapide en linux 64bits
(x6-x7)*(x5-x7)*(x5-x6)*(x4-x7)*(x4-x6)*(x4-x5)*(x3-x7)*(x3-x6)*(x3-x5)*(x3-x4)*(x2-x7)*(x2-x6
// Time 0.94
```

(sage)

```
sage: n=8; R=PolynomialRing(QQ,n,x)
sage: V=matrix(n,n, lambda a,b: R.gen(a)^b)
sage: %time d8=V.determinant()
CPU times: user 58.7 s, sys: 45.1 ms, total: 58.7 s
Wall time: 59 s
sage: R2=PolynomialRing(ZZ,n,x)   #NB: 7 fois plus rapide avec ZZ que QQ
sage: V=matrix(n,n, lambda a,b: R2.gen(a)^b)
sage: %time d8=V.determinant()  # temps idem avec V.minors(n)
CPU times: user 7.9 s, sys: 244 ms, total: 8.15 s
Wall time: 8.18 s
```

Pour la syntaxe de programmation, regardez le tutoriel dans le menu : Aide/Débuter en Calcul Formel

Exercice IV: blocs, polynômes, matrices nilpotentes

1) (**Matrices par blocs**)

a) Créez une fonction `JJ(n)` qui retourne une matrice de taille $n \times n$ ayant pour seuls termes non nuls des 1 sur la droite $j - i = 1$.

(giac/xcas)

```
fjj:=(ii,j)->if(ii=j-1) then 1 else 0 fi;
JJ(n):=matrix(n,n,fjj);//forme classique d'ordre n.
JJ(5);
```

b) Mini programme. (Sous xcas on pourra utiliser/découvrir l'éditeur de programme, et regarder le tutoriel).

Créez une fonction `J(L)` qui associe à une liste décroissante d'entiers naturels non nuls `L`[8], une matrice diagonale par blocs constitués des `JJ(n)` correspondants. On cherchera une instruction pour construire des matrices diagonales par blocs, et l'on vérifiera que la suite est bien décroissante.

(giac/xcas)

```
Jbasic(L):=diag([seq(JJ(u),u=L)]);//u parcourt la liste L
Jbasic(L):=diag([JJ(u) for u in L]);// idem
Jbasic([3,2,2,1]);
J(L):={
  local j,n;
  a:=L[0];
  n=size(L);
  for(j:=0;j<n-1;j++){
    if(L[j]<L[j+1]){return "Erreur";}
  }
```

8. exemple : [2,2,1,1]

```
  return diag([seq(JJ(u),u=L)]);
}
J([3,2,1,1]);
```

c) Etudiez sur divers exemples les rangs et noyaux des puissances de `J(L)` en fonction de L. Conclusion ?

2) (**Matrices et polynômes**) Cette année, nous aurons souvent besoin de trouver des matrices d'applications définies par des polynômes.

a) Créez une fonction `M(P,Q)` qui aux deux polynômes P, Q en la variable x associe la matrice dans la base canonique $(1, x, \ldots, x^{\deg(Q)-1})$ de l'application :

$$\begin{array}{ccc} \mathbb{Q}[x]/(Q) & \longrightarrow & \mathbb{Q}[x]/(Q) \\ R & \mapsto & R \cdot P \end{array}$$

Indications : on cherchera une fonction qui donne à i fixé le coefficient de x^i dans un polynome en x plutôt que de convertir des polynômes en liste. (Pourquoi ? quelle serait la longueur de la liste pour $1 + x^3, 2 + x, \ldots$?)

b) Vérifiez que $M(x + 2, x^4 - 1)$ vaut : $\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{pmatrix}$

3) On considère la matrice $N = M(P, Q)$ où :

$$P = x^5 - 2 \cdot x^4 + x^3 - 2 \cdot x^2 - 2 \cdot x + 4$$

$$Q = \left(x^2 - 1\right)^2 \left(x^2 + 2\right)^5 (x^3 - 5 \cdot x^2 + 8 \cdot x - 4)$$

a) Quel est le polynôme caractéristique de N ? (NB : l'instruction directe est plus efficace que $\det(A - x.I)$. Quel est le polynôme minimal de N ?

b) Comment obtient t'on une base du noyau de N ? Par exemple, comment calculer la somme du second et troisième vecteur de cette base ? Comment obtenir la dimension du noyau de N ? (On cherchera une technique qui fonctionne aussi pour les applications injectives)

c) En demandant au logiciel de calculer des rangs de certaines matrices, déterminer l'unique L pour que $J(L)$ et N puissent être semblable. (On rappelle que l'on impose à L d'être décroissante à valeur dans $\mathbb{N}^*$, et on n'utilisera pas de commande de Jordanisation du logiciel)

d) Donnez une base de $\ker N^4$, et choisissez deux vecteurs a_1, a_2 de $\ker N^5$ indépendants modulo $\ker N^4$. Que devez vous demander au logiciel pour prouver que a_1, a_2 conviennent ?

e) Prenez un vecteur a_3 dans $\ker N^3$ tel que $N^2(a_1), N^2(a_2), a_3$ soient indépendants modulo $\ker N^2$ (Prennez en un au hazard et prouvez avec votre logiciel que votre choix de a_3 est légitime).

f) Prenez deux vecteurs a_4, a_5 de $\ker(N^2)$ tels que $N^3(a_1), N^3(a_2), N(a_3), a_4, a_5$ soient libres modulo $\ker(N)$.

g) On considère la famille de vecteurs

$$N^4(a_1), \ldots, N(a_1), a_1, N^4(a_2), \ldots, N(a_2), a_2, N^2(a_3), \ldots, a_3, N(a_4), a_4, N(a_5), a_5.$$

Vérifiez [9] que c'est une base de $\mathbb{Q}^{17}$. Donnez ma matrice P_N de passage associée, et conjuguez N pour obtenir une matrice de type `J(L)`.

9. vu que l'on a fait les vérifications précédentes, on pourrait démontrer que cette dernière vérification est inutile