tp_Berlekamp(1)

December 11, 2019

1 Corps finis

1.1 I- Définition et éléments

En Sage, pour définir un corps fini du type $\mathbf{Z}/p\mathbf{Z}$, avec p premier, on dispose de la commande `IntegerModRing`.

```
In [1]: A=IntegerModRing(13); A
```

```
Out[1]: Ring of integers modulo 13
```

En donnant un nom à l'anneau, on peut alors utiliser l'application quotient canonique $\mathbf{Z} \to \mathbf{Z}/p\mathbf{Z}$.

```
In [2]: A(54)
```

```
Out[2]: 2
```

Les "modèles" pour les corps finis à p^n éléments sont des quotients de l'anneau de polynômes $\mathbf{Z}/p\mathbf{Z}[X]$. En Sage, on ne peut pas introduire une nouvelle indéterminée sans la définir.

```
In [3]: 3*X+5
```

```
    ---------------------------------------------------------------------------

    NameError                                 Traceback (most recent call last)

    <ipython-input-3-32f7bd0dc849> in <module>()
----> 1 Integer(3)*X+Integer(5)


    NameError: name 'X' is not defined
```

Pour définir l'anneau de polynômes $\mathbf{Z}/p\mathbf{Z}[X]$, on procède comme suit:

```
In [4]: R.<X> = PolynomialRing(A);R
```

```
Out[4]: Univariate Polynomial Ring in X over Ring of integers modulo 13
```

Ou bien comme cela:

```
In [5]: R1 = PolynomialRing(A,'X'); R1
```

```
Out[5]: Univariate Polynomial Ring in X over Ring of integers modulo 13
```

```
In [6]: 3*X+5
```

```
Out[6]: 3*X + 5
```

On dispose des fonctions usuelles pour l'arithmétique des polynômes: `%`, `//`, `gcd`, `xgcd`. Normalement, vous devriez pouvoir les recoder en quelques lignes et minutes...

```
In [7]: def pgcd(P,Q):
            if (Q.degree()==0):
                return Q;
            elif (Q.degree()<0):
                return P;
            else:
                return pgcd(Q,P%Q)
```

```
In [8]: pgcd(X*(X+1),X*(X-1))
```

```
Out[8]: 2*X
```

On peut alors définir le quotient de R par un idéal (quelconque).

```
In [9]: P=2*X^2+3*X+1; P
```

```
Out[9]: 2*X^2 + 3*X + 1
```

```
In [10]: RR=QuotientRing(R,ideal(P)); RR
```

```
Out[10]: Univariate Quotient Polynomial Ring in Xbar over Ring of integers modulo 13 with modul
```

```
In [11]: RR(5*X^6+4*X^5+5)^-1
```

```
Out[11]: Xbar + 12
```

Noter que la construction précédente est très générale: P n'est pas supposé irréductible!

Vérifiez que vous êtes capables de coder vous mêmes la fonction qui teste si un élément est inversible dans RR et le cas échéant en renvoie son inverse.

Au fait, le P précédent est-il irréductible?

```
In [12]: def inverse_dans_RR(Q):
             a=xgcd(P,Q)[1];
             if a[0]==1:
                 return a[2]
             else:
                 print("Element non inversible.")
```

Comme P est de degré 2, on teste facilement s'il est irréductible en testant s'il a des racines.

```
In [13]: for i in A:
             if P.subs(i)==0:
                 print(i)

6
12
```

Il admet deux racines: il n'est donc pas irréductible!!

En Sage, il existe un raccourci pour définir un corps fini à p^n éléments, il s'agit de la fonction `GF()` (comme Galois Field).

```
In [14]: R2=GF(8); R2

Out[14]: Finite Field in z3 of size 2^3
```

Sage a défini notre corps à 8 éléments comme un quotient de $\mathbf{Z}/2\mathbf{Z}[x]$ par un certain polynôme irréductible de degré 3 qu'il a choisi tout seul (avantage ou inconvénient?). Notez qu'il a aussi choisi tout seul le nom du générateur...

On peut accéder à ce polynôme et aux éléments:

```
In [15]: R2.modulus()

Out[15]: x^3 + x + 1

In [16]: R2.gen()

Out[16]: z3
```

Ici, `gen` **veut bien entendu dire générateur. En quel sens z_3 est-il un générateur?**

C'est un générateur comme algèbre: on obtient tous les éléments du corps comme des sommes de puissances de z.

```
In [17]: for i in R2:
             print (i)

0
z3
z3^2
z3 + 1
z3^2 + z3
z3^2 + z3 + 1
z3^2 + 1
1
```

On peut bien spécifier le nom du générateur et le polynôme:

```
In [18]: R3.<y>=GF(13^4,modulus=x^4 + 3*x^2 + 12*x + 1); R3
```

```
Out[18]: Finite Field in y of size 13^4
```

```
In [19]: R3.modulus()
```

```
Out[19]: x^4 + 3*x^2 + 12*x + 1
```

Il est utile d'avoir accès aux fonctions suivantes:

```
In [20]: R3.cardinality()
```

```
Out[20]: 28561
```

```
In [21]: z=R3.multiplicative_generator(); z
```

```
Out[21]: 8*y^3 + 12*y^2 + 9*y + 9
```

```
In [22]: z.minimal_polynomial()
```

```
Out[22]: x^4 + 12*x^3 + x + 11
```

```
In [23]: y.multiplicative_order()
```

```
Out[23]: 595
```

```
In [24]: z1=z.polynomial(); z1.coefficients()
```

```
Out[24]: [9, 9, 12, 8]
```

Il y a aussi une fonction renvoyant l'ordre additif d'un élément. Qu'en pensez-vous?
Ce n'est pas très utile: dans un corps fini, l'ordre additif d'un élément est toujours p.
Codez vos propres fonctions `ordre` **et** `generateur` **pour R3.**

```
In [25]: def ordre(a):
             i=1;
             b=a;
             while(b<>1):
                 b=b*a;
                 i=i+1
             return i
```

```
In [29]: ordre(y)
```

```
Out[29]: 595
```

```
In [30]: def generateur():
             for x in R3:
                 if x<>0:
                     if ordre(x)==R3.cardinality()-1:
                         return x
```

```
In [31]: generateur()
```

```
Out[31]: y + 8
```

Coder votre fonction `pol_min` en utilisant les fonctions d'algèbre linéaire suivantes pour définir des matrices, en calculer le rang et le noyau.

```
In [45]: M=Matrix(A,3,3,[i for i in range(9)]); M
```

```
Out[45]: [0 1 2]
         [3 4 5]
         [6 7 8]
```

```
In [46]: M.rank()
```

```
Out[46]: 2
```

```
In [47]: V=M.right_kernel(); V
```

```
Out[47]: Vector space of degree 3 and dimension 1 over Ring of integers modulo 13
         Basis matrix:
         [ 1 11  1]
```

```
In [76]: Z=V.gen(); A(1/2)*Z
```

```
Out[76]: (7, 12, 7)
```

L'idée est la suivante: trouver un polynôme annulateur d'un élément z c'est trouver une combinaison linéaire non triviale qui est nulle entre les puissances de y. Les puissances de z sont des vecteurs dans l'espace vectoriel qu'est notre corps fini, qu'on représente par leurs coordonnées dans la base canonique $(1, y, ..., y^n)$.

```
In [41]: def pol(z): # fonction qui transforme un élément d'un corps fini en le vecteur (de bo
             return [z.polynomial()[k] for k in range(4)]
```

```
In [95]: def pol_min(z):
             RR.<x>=PolynomialRing(A)
             l=[[1,0,0,0]];# On initialise la liste avec z^0=1
             M=matrix(A,l);
             k=1;
             while(M.rank()==k):
                 l=l+ [pol(z^k)];
                 k=k+1;
                 M=matrix(A,l);
             V=M.transpose().right_kernel().gen();
             V=V*A(1/V[k-1]) # On le normalise pour que le polynôme minimal soit unitaire
             return sum([RR(V[r])*x^r for r in range(k)])
```

```
In [97]: pol_min(z^2+1)
```

```
Out[97]: x^4 + 8*x^3 + 7*x^2 + 9*x + 5
```

```
In [98]: (z^2+1).minimal_polynomial()
```

```
Out[98]: x^4 + 8*x^3 + 7*x^2 + 9*x + 5
```

1.2 II- Factorisation de polynômes et algorithme de Berlekamp

Sage sait factoriser les polynômes à coefficients dans un corps fini, grâce à la commande `factor`.

```
In [99]: R1.<X>=PolynomialRing(Zmod(7)); R1
```

```
Out[99]: Univariate Polynomial Ring in X over Ring of integers modulo 7
```

```
In [100]: P=X^6 + X^4 + 5*X^3 + 4*X^2 + 6*X + 3
```

```
In [101]: factor(P)
```

```
Out[101]: X^6 + X^4 + 5*X^3 + 4*X^2 + 6*X + 3
```

Qu'en déduit-on sur P?
Sage dit que P est irréductible...

```
In [105]: K1.<X>=PolynomialRing(GF(7^6));
```

```
In [106]: factor(K1(P))
```

```
Out[106]: (X + 6*z6) * (X + 2*z6^4 + 2*z6^3 + 3*z6^2 + 4*z6 + 6) * (X + z6^5 + 5*z6^4 + 4*z6^3
```

Était-ce prévisible?
Oui: par contruction, K1 est contruit en rajoutant une racine de P à $Z/7Z$... Donc P n'est pas construction plus irréductible sur K1. De plus, la théorie des corps finis nous prédisait qu'un corps qui contenait une racine de P les contient en fait toutes: on a ici les formules pour les autres racines. Noter que le vilain Sage est revenu à une notation z_6 au lieu de X.

```
In [107]: K2.<X>=PolynomialRing(GF(7^3));
```

```
In [108]: factor(K2(P))
```

```
Out[108]: (X^2 + (z3^2 + z3 + 4)*X + 4*z3^2 + z3 + 1) * (X^2 + (2*z3^2 + 2*z3 + 1)*X + 3*z3^2 +
```

Oui: K_1 est un espace vectoriel de dimension 6 sur $Z/7Z$. K_2 est un espace vectoriel de 3 sur $Z/7Z$, comme 3 divise 6, K_2 est un sous-corps de K_1 et K_1 est de dimension 2 sur K_1. P doit donc se factoriser par des polynômes de degré 2.
Était-il prévisible d'avoir 3 facteurs de degré 2?

1.2.1 Exercice

a) Montrer que le polynôme $P = X^3 + 3X + 3 \in \mathbf{F}_5[X]$ est sans facteur carré.
b) Utiliser l'algorithme de Berlekamp pour montrer que P est irréductible.
c) Utiliser cet algorithme pour factoriser les polynômes suivants dans $(\mathbf{F}_5)[X]$: $P_1 := X^5 + 3X^2 + 3X + 1$ et $P_2 = X^6 + 2X^5 + 3X^3 + 4X^2 + 2X + 2$.
d) Ecrire une fonction `Berlekamp(p,P)` qui factorise le polynôme P dans $\mathbf{F}_p[X]$.

```
In [125]: R.<x> = PolynomialRing(IntegerModRing(5));
```

On commence par tester si P est sans facteur carré.

```
In [126]: P=x^3+3*x+3;D=P.derivative()

In [127]: gcd(P,D)

Out[127]: 1
```

P et D étant premiers entre eux, P est sans facteur carré.
On calcule ensuite le noyau de l'application *linéaire* $\Phi : v \mapsto v^5 - v$. On écrit pour cela sa matrice dans la base $1, x, x^2$.

```
In [133]: def coef(z,r): # fonction qui transforme un élément d'un corps fini en le vecteur (d
              return [z[k] for k in range(r)]

In [134]: M=matrix(Zmod(5),3,[coef((x^(5*i)-x^i)%P,3) for i in range(3)]).transpose(); M

Out[134]: [0 4 3]
          [0 3 2]
          [0 2 4]

In [130]: M.rank()

Out[130]: 2
```

M étant de rang 2, P est bien irréductible...
On recommence pour P1 et P2.

```
In [131]: P1=x^5+3*x^2+3*x+1; D1=P1.derivative(); gcd(P1,D1)

Out[131]: 1

In [138]: M1=matrix(Zmod(5),5,[coef((x^(5*i)-x^i)%P1,5) for i in range(5)]).transpose(); M1

Out[138]: [0 4 1 0 1]
          [0 1 1 1 3]
          [0 2 4 3 0]
          [0 0 3 4 1]
          [0 0 4 2 3]

In [139]: M1.rank()

Out[139]: 2
```

A est de rang 2: il y a donc 3 facteurs irréductibles dans $P1$. Pour les trouver, on trouve un élément non constant dans le noyau de Φ.

```
In [143]: K1=M1.right_kernel(); K1.basis()

Out[143]: [
          (1, 0, 0, 0, 0),
          (0, 1, 0, 1, 1),
          (0, 0, 1, 2, 4)
          ]
```

De façon prévisible, le premier vecteur correspond aux polynômes constants. On en choisit un autre, par exemple le second.

```
In [144]: y1=K1.basis()[1]
```

On le reconvertit en polynôme.

```
In [147]: Z1=R(list(y1)); Z1

Out[147]: x^4 + x^3 + x
```

On sait qu'on va trouver un facteur de P1 en parcourant les polynômes Z1-i.

```
In [150]: ll=[gcd(P1,Z1-i) for i in Zmod(5)]; ll

Out[150]: [1, x + 2, 1, x^3 + 2*x^2 + 2*x + 3, x + 1]
```

On a déjà trois facteurs, donc comme on en cherchait trois, on a terminé! Vérifions tout de même:

```
In [152]: P1-ll[1]*ll[3]*ll[4]

Out[152]: 0

In [153]: P2=x^6 + 2*x^5 + 3*x^3 + 4*x^2 + 2*x + 2; D2=P2.derivative(); gcd(P2,D2)

Out[153]: x + 2
```

Ici $P2$ n'est pas sans facteur carré: on le divise par $x+2$ et on peut faire comme avant. (En fait, vu qu'on a trouvé $x+2$, (déjà sous forme factorisé!!), on pourrait même simplifier par (x+2)^2).

```
In [162]: Q2=P2//(x+2); Q2

Out[162]: x^5 + 3*x^2 + 3*x + 1

In [163]: M2=matrix(Zmod(5),5,[coef((x^(5*i)-x^i)%Q2,5) for i in range(5)]).transpose(); M2, M

Out[163]: (
          [0 4 1 0 1]
          [0 1 1 1 3]
          [0 2 4 3 0]
          [0 0 3 4 1]
          [0 0 4 2 3], 2
          )
```

Rang 2, donc on cherche 3 facteurs.

```
In [164]: K2=M2.right_kernel(); B=K2.basis(); B
```

```
Out[164]: [
          (1, 0, 0, 0, 0),
          (0, 1, 0, 1, 1),
          (0, 0, 1, 2, 4)
          ]
```

```
In [167]: y2=B[2]; Z2=R(list(y2)); Z2
```

```
Out[167]: 4*x^4 + 2*x^3 + x^2
```

```
In [168]: ll=[gcd(Q2,Z2-i) for i in Zmod(5)]; ll
```

```
Out[168]: [1, 1, x + 2, x^4 + 3*x^3 + 4*x^2 + 3, 1]
```

Là: on n'a trouvé que 2 facteurs. X+2 est visiblement irréductible, mais le second facteur de degré 4 ne l'est pas et il faut recommencer...

Après quelques lignes de calculs, on trouve $Q2 = (x+1)(x+2)^2(x^3+2*x^2+2*x+3)$

On automatise tout ça...

```
In [ ]: def sfc(q): # Renvoie la partie sans facteur carré de P
            u=gcd(q,q.derivative());
            return(u, q//u)
```

```
In [199]: def fact_sfc(p,P): #fonction qui factorise un polynôme sans facteur carré
              R.<x> = PolynomialRing(IntegerModRing(p));
              q=R(P);
              d=q.degree();
              M=matrix(Zmod(p),d,[coef((x^(p*i)-x^i)%q,d) for i in range(d)]).transpose();
              K=M.right_kernel();
              if K.dimension()==1:
                  return [P];
              else:
                  l=[];
                  z=R(list(K.basis()[1]));
                  for i in Zmod(p):
                      g=gcd(q,z-i);
                      if g<>1:

                          l=l+fact_sfc(p,g);
                  return l;
```

```
In [201]: fact_sfc(5,Q2*x^2 +x+1)
```

```
Out[201]: [x^2 + 3, x^3 + 3*x + 3, x + 4, x + 1]
```

```
In [202]: factor(Q2* x^2+x+1)
```

```
Out[202]: (x + 1) * (x + 4) * (x^2 + 3) * (x^3 + 3*x + 3)
```

```
In [ ]:
```

1.3 Sous-corps

```
In [ ]:
```

Exercice : définir en Sage un corps K à 5^4 éléments.

A partir de quel polynôme irréductible est-il construit?

Déterminer un sous-corps K_1 à 5^2 éléments de K: - faire la liste des éléments de K_1, - trouver un générateur z de K_1 comme algèbre - déterminer le polynôme minimal d'un générateur y de K sur K_1. *(Pour cela, on pourra déterminer a priori le degré d de ce polynôme, et utiliser une $\mathbf{F}_p$-base de K de la forme $y^i z^j$ pour trouver une relation linéaire adéquate.)*

```
In [242]: K.<y>=GF(5^4);
```

```
In [243]: K.modulus()
```

```
Out[243]: x^4 + 4*x^2 + 4*x + 2
```

L'unique sous-corps à 5^2 éléments de K est constitué des racines de $X^{5^2} - X$.

```
In [247]: l=[u for u in K if u^(5^2)==u]; l
```

```
Out[247]: [0,
           y^3 + y^2 + y + 3,
           y^3 + y^2 + y + 1,
           4*y^3 + 4*y^2 + 4*y,
           2*y^3 + 2*y^2 + 2*y + 3,
           4*y^3 + 4*y^2 + 4*y + 3,
           2,
           2*y^3 + 2*y^2 + 2*y + 1,
           2*y^3 + 2*y^2 + 2*y + 2,
           3*y^3 + 3*y^2 + 3*y,
           4*y^3 + 4*y^2 + 4*y + 1,
           3*y^3 + 3*y^2 + 3*y + 1,
           4,
           4*y^3 + 4*y^2 + 4*y + 2,
           4*y^3 + 4*y^2 + 4*y + 4,
           y^3 + y^2 + y,
           3*y^3 + 3*y^2 + 3*y + 2,
           y^3 + y^2 + y + 2,
           3,
           3*y^3 + 3*y^2 + 3*y + 4,
           3*y^3 + 3*y^2 + 3*y + 3,
           2*y^3 + 2*y^2 + 2*y,
           y^3 + y^2 + y + 4,
           2*y^3 + 2*y^2 + 2*y + 4,
           1]
```

K1 contient comme attendu les constantes. Comme il est de dimension 2 sur Z/5Z, tout élément non constant l'engendre comme algèbre. Vérifions le:

```
In [263]: z=l[1]; z

Out[263]: y^3 + y^2 + y + 3

In [264]: [i*z+j for i in Zmod(5) for j in Zmod(5)]

Out[264]: [0,
           1,
           2,
           3,
           4,
           y^3 + y^2 + y + 3,
           y^3 + y^2 + y + 4,
           y^3 + y^2 + y,
           y^3 + y^2 + y + 1,
           y^3 + y^2 + y + 2,
           2*y^3 + 2*y^2 + 2*y + 1,
           2*y^3 + 2*y^2 + 2*y + 2,
           2*y^3 + 2*y^2 + 2*y + 3,
           2*y^3 + 2*y^2 + 2*y + 4,
           2*y^3 + 2*y^2 + 2*y,
           3*y^3 + 3*y^2 + 3*y + 4,
           3*y^3 + 3*y^2 + 3*y,
           3*y^3 + 3*y^2 + 3*y + 1,
           3*y^3 + 3*y^2 + 3*y + 2,
           3*y^3 + 3*y^2 + 3*y + 3,
           4*y^3 + 4*y^2 + 4*y + 2,
           4*y^3 + 4*y^2 + 4*y + 3,
           4*y^3 + 4*y^2 + 4*y + 4,
           4*y^3 + 4*y^2 + 4*y,
           4*y^3 + 4*y^2 + 4*y + 1]

In [235]: l.sort(), l

Out[235]: (None,
           [0,
            1,
            2,
            3,
            4,
            y^3 + y^2 + y,
            y^3 + y^2 + y + 1,
            y^3 + y^2 + y + 2,
            y^3 + y^2 + y + 3,
            y^3 + y^2 + y + 4,
            2*y^3 + 2*y^2 + 2*y,
            2*y^3 + 2*y^2 + 2*y + 1,
            2*y^3 + 2*y^2 + 2*y + 2,
            2*y^3 + 2*y^2 + 2*y + 3,
```

```
            2*y^3 + 2*y^2 + 2*y + 4,
            3*y^3 + 3*y^2 + 3*y,
            3*y^3 + 3*y^2 + 3*y + 1,
            3*y^3 + 3*y^2 + 3*y + 2,
            3*y^3 + 3*y^2 + 3*y + 3,
            3*y^3 + 3*y^2 + 3*y + 4,
            4*y^3 + 4*y^2 + 4*y,
            4*y^3 + 4*y^2 + 4*y + 1,
            4*y^3 + 4*y^2 + 4*y + 2,
            4*y^3 + 4*y^2 + 4*y + 3,
            4*y^3 + 4*y^2 + 4*y + 4])
```

Compte tenu des dimensions, y a un polynôme minimal de degré 2 sur K1. On cherche donc une CL non triviale nulle entre 1, z, y, yz et z^2.

```
In [265]: M=Matrix(Zmod(5), 5,[[1,0,0,0],pol(z),pol(y), pol(y*z), pol(y^2)]).transpose()
```

```
In [266]: k=M.right_kernel().basis(); k
```

```
Out[266]: [
          (0, 1, 3, 4, 1)
          ]
```

```
In [267]: y^2+(4*z+3)*y+z
```

```
Out[267]: 0
```

Le polynôme $X^2 + (4z + 3)X + z$ est donc le polynôme minimal de y sur K_1.

```
In [ ]:
```