MT10 – TP3

Antoine COLLAS, Tran Thi Son Nu

13 juin 2017

1 Construction de corps finis :

1.1 Dénombrement des polynômes irréductibles et unitaires de $\mathbb{F}_p[X]$

1.1.1 Factorisation de $X^q - X$ dans $\mathbb{F}_p[X]$

Question 1 : Pour vérifier cette factorisation, nous commençons par chercher tous les polynômes irréductibles, unitaires dont le degré divise n. Pour chaque degré i allant de 1 à n nous commençons par tester si $i \mid n$. Puis pour chaque polynôme de degré i appartenant à $\mathbb{F}_p[X]$ nous testons s'il est unitaire et irréductible. Pour tester le caractère unitaire d'un polynôme nous utilisons les fonctions **factor** et **unit** :

```
factor(polynome).unit()
```

factor(polynome).unit() renvoie 1 si le polynôme est unitaire.
Ensuite nous utilisons **polynome.is_irreducible()** qui renvoie 1 si le polynôme est irréductible. Nous obtenons le programme suivant :

```
def q1(p,n):
    res=[]
    R.<X>=GF(p)['X']
    for degre in range(1,n+1):
        if n%degre==0:
            for polynome in R.polynomials(of_degree=degre):
                if (factor(polynome).unit()==1) and polynome.is_irreducible():
                    res.append(polynome)
    return res
```

Pour $p = 2$ et $n = 6$:

```
q1(2,6)
```

$[X, X+1, X^2+X+1, X^3+X+1, X^3+X^2+1, X^6+X+1, X^6+X^3+1, X^6+X^4+X^2+X+1, X^6+X^4+X^3+X+1, X^6+X^5+1, X^6+X^5+X^2+X+1, X^6+X^5+X^3+X^2+1, X^6+X^5+X^4+X+1, X^6+X^5+X^4+X^2+1]$

Pour vérifier la factorisation donnée dans l'énoncer, nous multiplions tous les polynômes obtenus à l'aide du programme suivant :

```
def q1suite(factorisation):
    res=1
    for i in range(len(factorisation)):
        res=res*factorisation[i]
    return res
```

```
q1suite(q1(2,6))
```

Nous obtenons $X^{64} + X = X^{64} - X$ dans $\mathbb{F}_2[X]$

Question 2 : Pour tester que les 100 premiers termes de $\mu(n)$ appartiennent à $-1, 0, 1$. Nous utilisons la fonction **moebius** pour chaque n et renvoyons **false** si la valeur n'est pas dans $-1, 0, 1$.

```
def moebiusTest(n):
    for i in range(1, n+1):
        if moebius(i) not in [-1, 0, 1]:
            return False
    return True
```

```
moebiusTest(100)
```

True

Donc $\mu(n) \in \{-1, 0, 1\}$ pour les 100 premiers entiers naturels.

Ensuite nous vérifions la formule d'Euler pour la fonction Möbius. Si $n = 1$ alors la fonction vérifie que la fonction de Möbius en 1 vaut bien 1. Sinon pour $n > 1$, nous calculons la fonction de Möbius pour chaque $r \leq n$ en appliquant la formule donnée dans l'énoncé. Nous vérifions à chaque fois que la fonction d'Euler vaut bien 0, sinon le programme renvoie False.

```
def euler(n):
  if n==1:
    return moebius(1)==1
  for r in range(2,n+1):
    res=0
    for d in range(1,r+1):
      if r%d==0:
        res=res+moebius(d)
    if res!=0:
      print r
      return False
  return True
```

```
euler(100)
```

True

Pour calculer $\phi(100)$ nous utilisons la formule d'inversion de Möbius : $\phi(n) = \sum_{d|n} \mu(\frac{n}{d}) \times n$.

```
def MobiusInverse(n):
  res=0
  for i in range(1,n+1):
    if n%i==0:
      res=res+i*moebius(n/i)
  return res
```

```
MobiusInverse(100)
```

40
Nous vérifions notre résultat à l'aide de la commande **euler_phi** :

```
euler_phi(100)
```

40
Question 3 : Pour calculer $Irr_p(n)$, nous appliquons simplement la formule donnée dans l'énoncé :

```
def Irr(p,n):
  res=0
  for d in range(1,n+1):
    if(n%d)==0:
      somme=somme+moebius(n/d)*(p^d)
  somme=(1/n)*somme
  return somme
```

```
def tab():
  for n in range(1,11):
    for p in [2,3,5]:
      print "%6d" % Irr(p,n)  ,
    print "\n",
```

Nous obtenons le tableau suivant avec p=2,3,5 (colonnes) et n=1,...,10 (lignes) :
2 3 5
1 3 10
2 8 40
3 18 150
6 48 624
9 116 2580
18 312 11160
30 810 48750
56 2184 217000
99 5880 976248
Question 4 : Pour déterminer tous les polynômes irréductibles de $\mathbb{F}_2[X]$ de degré inférieur ou égal à 10 nous utilisons les mêmes fonctions que celles utilisées dans la question 1. L'attribut **max_degree=n** permet d'obtenir tous les polynômes de degré inférieur ou égal à 10. Pour chaque polynôme appartenant à $\mathbb{F}_2[X]$ de degré inférieur ou égal à 10 nous testons s'il est irréductible puis l'ajoutons ou non à un tableau.

```
def q4(p,n):
    res=[]
    R.<X>=GF(p)['X']
    for polynome in R.polynomials(max_degree=n):
        if polynome.is_irreducible():
            res.append(polynome)
    return res
```

```
q4(2,10)
```

Nous obtenons un grand nombre de polynômes irréductibles de $\mathbb{F}_2[X]$ de degré inférieur ou égal à 10.

2 Les codes de Reed et Solomon

2.1 Définition des codes de Reed-Solomon généralisés (GRS)

Question 5 : A partir de maintenant nous manipulerons des éléments de $\mathbb{F}_q[X]$ avec $q = p^n$ où p est premier. Pour cela nous utilisons les commandes suivantes qui permettent de former $\mathbb{F}_q[X]$.

```
q=2**8
c='a'
Fq.<a>=GF(q,c)
c=Fq.gen()
R=PolynomialRing(Fq,'X')
X=R.gen()
```

Afin de générer **v** et **alpha** plus rapidement nous créons deux fonctions. La première, **generate_v** forme un vecteur de taille n (passé en paramètre) d'éléments aléatoires appartenant à $\mathbb{F}_q^*$.

```
def generate_v(n):
  v=[]
  for i in range(n):
    temp=Fq.random_element()
    while temp==0:
      temp=Fq.random_element()
    v.append(temp)
  return v
```

Nous réalisons les mêmes opérations pour alpha en faisant attention à ne pas avoir deux éléments égaux dans le vecteur. A chaque élément aléatoire généré nous vérifions qu'il n'est pas déjà dans le vecteur.

```
def generate_alpha(n):
  alpha=[]
  temp=0
  for i in range(n):
    present=1
    while present==1:
      temp=Fq.random_element()
      present=0
      for j in range(i):
        if temp==alpha[j]:
          present=1
    alpha.append(temp)
  return alpha
```

Pour coder le message **x**, notre fonction **codeGRS** prend en paramètres : **x**, **v**, **alpha** et **q**. Nous commençons par effectuer quelques tests sur les paramètres. Nous testons que **q** est une puissance d'un nombre premier, l'égalité des longueurs de **v** et **alpha**, puis l'inégalité $0 \leq k \leq n \leq q$ et enfin que toutes les valeurs de **alpha** sont différentes. Ensuite nous créons le polynôme **f** comme étant $\sum_{i=0}^{k-1} x_{k-1-i} \times X^i$. Enfin, l'évaluation consiste à retourner un tableau des $v_i \times f(\alpha_i)$.

```
def codeGRS(x, v, alpha, q):
  k=len(x)
  n=len(v)
  if q.is_prime_power()==0:
    print "erreur: q n'est pas la puissance d'un nombre premier"
    return 0
  if len(v)!=len(alpha):
    print "erreur: longueur de v differente de alpha"
    return 0
  if k>n:
    print "erreur: k>n"
    return 0
  if n>q:
    print "erreur: n>q"
  for i in range(n):
    for j in range(i+1,n):
      if alpha[i]==alpha[j]:
        print "erreur: alpha["+str(i)+"]=alpha["+str(j)+"]"
        return 0
  f=0
  for i in range(k):
    f=f+x[k-1-i]*X^i
  res=[]
  for i in range(n):
    res.append(v[i]*f(alpha[i]))
  return res
```

Nous prenons $q = 2^8$, $n = 10$, $k = 5$.

```
x=generate_v(k)
v=generate_v(n)
alpha=generate_alpha(n)
code=codeGRS(x,v,alpha,q)
```

Nous obtenons $x = [a^7+a^5+a^2+a, a^7+a^6+a^5+a^2+1, a^7+a^5+a^2+1, a^5+a^4, a^7+a^6+a^4+a^3+a^2+a+1]$ et $code = [a^7 + a^6 + a^4 + a^2 + a, a^6 + a^2 + a + 1, a^6 + a^3 + a^2 + 1, a^7 + a^6 + a^2 + 1, a^6 + a^5 + a^2 + a + 1, a^6 + a^5 + a^3 + a^2 + 1, a^3 + a^2, a^6 + a^4 + a, a^6 + a^4 + a, a^7 + a^6 + a^5 + a^2]$

2.2 Cas sans erreur : décodage des GRS par interpolation de Lagrange

Question 6 : Pour décoder nous allons avoir besoin de calculer les polynômes de Lagrange $L_i(X)$. Nous commençons donc par définir une fonction qui crée ces polynômes. Il suffit simplement d'appliquer la formule donnée dans l'énoncé : on multiplie les $(X - \alpha_j)$ pour j allant de 0 à $n - 1$, $j \neq i$, n étant la taille du vecteur α.

```
def lagrange(i, alpha):
  n=len(alpha)
  L=1
  for j in range(n):
    if j!=i:
      L=L*(X-alpha[j])
  return L
```

Ensuite pour décoder nous appliquons la formule donnée dans l'énoncé qui va nous permettre de retrouver $f(X)$. **decodeGRS** prend en paramètres le **code**, **v**, **alpha** et **q**. Les $f(\alpha_i)$ sont présents dans le code, il suffit de diviser **code** par **v_i**. Les polynômes sont disponibles grâce à la fonction précédente. Une fois $f(X)$ calculée, il suffit de récupérer ses coefficients pour obtenir le message initial.

```
def decodeGRS(code, v, alpha, q):
  n=len(v)
  if q.is_prime_power()==0:
    print "erreur: q n'est pas la puissance d'un nombre premier"
    return 0
  if len(v)!=len(alpha):
    print "erreur: longueur de v differente de alpha"
    return 0
  for i in range(n):
    for j in range(i+1,n):
      if alpha[i]==alpha[j]:
        print "erreur: alpha["+str(i)+"]=alpha["+str(j)+"]"
        return 0
  f=0
  for i in range(n):
    f=f+(code[i]/v[i])*(((lagrange(i,alpha))(alpha[i]))**(-1))*
    lagrange(i,alpha)
  res=[]
  for i in range(f.degree()+1):
    res.append(f[f.degree()-i])
  return res
```

En reprenant les mêmes valeurs que dans la question précédente, on retrouve le message initial :

```
decodeGRS(code,v,alpha,q)
```

$[a^7 + a^5 + a^2 + a, a^7 + a^6 + a^5 + a^2 + 1, a^7 + a^5 + a^2 + 1, a^5 + a^4, a^7 + a^6 + a^4 + a^3 + a^2 + a + 1]$

Si deux points de α sont égaux : $\alpha_i = \alpha_j$ pour $i \neq j$ alors $L_i(\alpha_i) = 0$, donc $(L_i(\alpha_i))^{-1}$ n'existe pas.

2.3 Simulation d'erreurs de transmission

Question 7 : Nous commençons par créer un tableau **e** contenant n 0. Puis nous changeons **Nb_err** valeurs de **e** de manière aléatoire. Pour cela, nous tirons un nombre aléatoire entre 0 et $n-1$. Si la case de **e** correspondante est égale à 0 alors cela signifie que nous n'avons pas encore mis d'erreur à cet indice. Nous tirons alors un nombre aléatoire dans $\mathbb{F}_q^*$. Nous répétons ces étapes jusqu'à avoir introduit **Nb_err** valeurs différentes de 0 dans **e**. Enfin nous retournons $y + e$.

```
def errTrans(y,q,Nb_err):
  import random
  e=[0]*len(y)
  while Nb_err!=0:
    indice=random.randint(0,len(y)-1)
    if e[indice]==0:
      Nb_err=Nb_err-1
      temp=Fq.random_element()
      while temp==0:
        temp=Fq.random_element()
      e[indice]=temp
  y_prime=[]
  for i in range(len(y)):
    y_prime.append(y[i]+e[i])
  return y_prime
```

```
errTrans(code,q,2)
```

$[a^7+a^6+a^4+a^2+a, a^6+a^2+a+1, a^6+a^3+a^2+1, a^7+a^6+a^2+1, a^6+a^5+a^2+a+1, a^6+a^5+a^3+a^2+1, a^3+a^2, a^7+a^3+a^2+a, a^6+a^4+a, a^5+a^4]$
Nous observons qu'il y a deux erreurs aux positions 7 et 9 par rapport au code précédent.

Question 8 : Montrons sur un exemple que l'interpolation de Lagrange donne n'importe quoi dès qu'il y a une erreur de transmission :

```
code=codeGRS(x,v,alpha,q)
y_prime=errTrans(code,q,1)
decodeGRS(y_prime,v,alpha,q)
```

On obtient $[a^7+a^6+a^3+1, a^7+a^5+a^4+a^2+a, a^7+a^5+a^4+a^3+a^2, a^7+a^5+a^4+a^3+a^2+a, a^7+a^3+a^2+a+1, a^5+a^3+a, a^7+a^6+a^3+a^2, a^4+a^3+a^2+a, a^7+a+1, a^7+a^4+a^2]$ au lieu de $[a^7+a^6+a^5+a^4+a^3+1, a^7+a^3+a^2+1, a^7+a^6+a^5+a^3, a^5+a+1, a^5]$.

3 Correction d'erreurs grâce aux GRS

3.1 Le polynôme syndrome

Question 9 : Pour calculer le polynôme syndrome nous utilisons la fonction Lagrange définie précédemment et appliquons simplement la formule.

```
def Syndrome(y_prime, alpha, v, q, k):
  n=len(v)
  S=0
  r=n-k
  for i in range(n):
    temp=0
    for j in range(r):
      temp=temp+(alpha[i]*X)^j
    S=S+y_prime[i]*((v[i]*((lagrange(i,alpha))(alpha[i])))**(-1))
        *temp
  return S
```

Question 10 : Vérifions maintenant que $y' \in C \iff S(X) = 0$ à l'aide de quelques exemples. Nous commençons par vérifier l'implication. Nous prenons un message que nous codons puis nous calculons $S(X)$. Dans les deux exemples suivants nous obtenons 0.

```
code=codeGRS(x,v,alpha,q)
Syndrome(code,alpha,v,q,len(x))
```

0

Un second exemple :

```
x2=generate_v(6)
v=generate_v(10)
alpha=generate_alpha(10)
code=codeGRS(x,v,alpha,q)
Syndrome(code,alpha,v,q,len(x2))
```

0

Pour vérifier la réciproque nous utilisons la contraposée : $y' \notin C \implies S(x) \neq 0$. Nous codons un message puis nous créons une erreur de transmission et enfin calculons le polynôme syndrome. Dans l'exemple suivant nous obtenons une valeur différente de 0.

```
code=codeGRS(x,v,alpha,q)
y_prime=errTrans(code,q,1)
Syndrome(y_prime,alpha,v,q,len(x))
```

$(a^6 + a^5 + a^4 + a^2 + 1) * X^4 + (a^7 + a^6 + a^5 + a^4 + a^3 + a^2) * X^3 + (a^7 + a^5 + a^4 + a^3 + 1) * X^2 + (a^7 + a^5 + a^2 + a) * X + a^7 + a^2 + a$

3.2 Résolution de l'équation clef par Euclide

Question 11 : Pour écrire la fonction **Clef**, nous avons besoin du polynôme syndrome **S(X)**, de **r** et de **q**. Nous appliquons le **Théorème 4** pour obtenir $\sigma(X)$ et $\omega(X)$ à partir de S(X). Nous obtenons le programme suivant :

```
def Clef(S,r,q):
   reste=[]
   reste.append(X^r)
   reste.append(S)
   u=[]
   u.append(1)
   u.append(0)
   v=[]
   v.append(0*(X^0))
   v.append(X^0)
   j=1
   quotient=[]
   quotient.append(0)

   while (reste[j].degree())>=(r/2):
      reste.append(reste[j-1]%reste[j])
      quotient.append((reste[j-1]-reste[j+1])/reste[j])
      u.append(u[j-1]-u[j]*quotient[j])
      v.append(v[j-1]-v[j]*quotient[j])
      j=j+1

   sigma_tilde=v[j]
   omega_tilde=reste[j]

   sigma=(sigma_tilde(0)**(-1))*sigma_tilde
   omega=(sigma_tilde(0)**(-1))*omega_tilde

   return [sigma,omega]
```

Nous réalisons un premier test avec transmission sans erreur :

```
x=[a^7 + a^6 + a^5 + a^4 + a^3 + 1,
 a^7 + a^3 + a^2 + 1,
 a^7 + a^6 + a^5 + a^3,
 a^5 + a + 1,
 a^5]
code=codeGRS(x,v,alpha,q)
k=len(x)
n=len(alpha)
r=n-k
S=Syndrome(code,alpha,v,q,k)
Clef(S,r,q)
```

[1,0]

Donc $\sigma = 1$ et $\omega = 0$. L'équation clef est vérifiée ($S(x) = 0$). Nous réalisons un second test avec transmission avec une erreur :

```
x=[a^7 + a^6 + a^5 + a^4 + a^3 + 1,
  a^7 + a^3 + a^2 + 1,
  a^7 + a^6 + a^5 + a^3,
  a^5 + a + 1,
  a^5]
k=len(x)
n=len(alpha)
r=n-k
code=codeGRS(x,v,alpha,q)
y_prime=errTrans(code,q,1)
S=Syndrome(y_prime,alpha,v,q,k)
[sigma,omega]=Clef(S,r,q)
```

Nous trouvons $\sigma(X)\times S(X) = (a^5+a^4+a^3)*X^5+a^7+a^5+a^4+a^3+a^2$ et $\omega(X) = a^7+a^5+a^4+a^3+a^2$. Donc l'équation clef est encore vérifiée puisque $X^r = X^5 = 0[X^r]$.

3.3 Localisation et évaluation des erreurs de transmission

Question 12 : Pour calculer l'erreur de transmission **e**, nous commençons par chercher les positions des erreurs à l'aide du polynôme localisateur σ en cherchant les $b \in [\![0, n-1]\!]$: $\sigma(\alpha_b^{-1}) = 0$. A partir des indices b trouvés nous les calculons les $e_b = -\alpha_b\omega(\alpha_b^{-1})(u_b\sigma'(\alpha_b^{-1}))^{-1}$ où $u_b = \frac{1}{v_b\times L_b(\alpha_b)}$. Nous utilisons la méthode **derivative()** pour obtenir la dérivée de σ. Ainsi nous obtenons le programme suivant :

```
def Erreur(sigma,omega,alpha,q,r):
   n=len(alpha)
   indice=[]
   for i in range(n):
      if sigma(((alpha[i]*X^0)**(-1)))==0:
         indice.append(i)

   u=[]
   for i in range(len(alpha)):
      Li=lagrange(i,alpha,q)
      u.append(1/(v[i]*Li(alpha[i])))

   e=[0]*len(alpha)
   for b in indice:
      e[b]=-alpha[b]*omega((alpha[b]*X^0)**(-1))*
      ((u[b]*sigma.derivative()((alpha[b]*X^0)**(-1)))**(-1))
   return e
```

Nous testons ce programme :

```
x=[a^7 + a^6 + a^5 + a^4 + a^3 + 1,
  a^7 + a^3 + a^2 + 1,
  a^7 + a^6 + a^5 + a^3,
  a^5 + a + 1,
  a^5]
code=codeGRS(x,v,alpha,q)
codeErr=errTrans(code,q,1)
S=Syndrome(codeErr,alpha,v,q,k)
[sigma,omega]=Clef(S,r,q)
Erreur(sigma,omega,alpha,q,r)
```

Nous obtenons le code $[a^7+a^6+a^4+a+1, a^7+a^2, a^7+a^5+1, a^7+a^5+1, a^7+a^6+a^5, a^7+a^6+a^5+a, a^7+a^6+a^5+a^4+a^3, a^3+a^2+a, a^7+a^4+a^2+a+1, a^6+a^3+a^2]$

Le code avec une erreur $[a^7+a^6+a^4+a+1, a^7+a^2, a^7+a^5+1, a^7+a^5+1, a^7+a^6+a^5, a^7+a^6+a^5+a, a^7+a^6+a^5+a^4+a^3, a^3+a^2+a, a^7+a^4+a^2+a+1, a^7+a^4+a^3]$

Et l'erreur de transmission $[0,0,0,0,0,0,0,0,0, a^7+a^6+a^4+a^2]$.

$(a^6+a^3+a^2)+(a^7+a^6+a^4+a^2) = a^7+a^4+a^3$. Donc le programme permet bien de retrouver les erreurs de transmissions.