TP4

September 26, 2022

1 TP 4 : factorisation des polynômes sur les corps finis

1.0.1 Exo 1

[1]:
```
K=GF(27)
```

[2]:
```
K.multiplication_table()
```

[2]:
```
 *  aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw ax ay
az ba
  +-----------------------------------------------------------------------------
----
aa| aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa
ab| aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw ax ay
az ba
ac| aa ac ab ag ai ah ad af ae as au at ay ba az av ax aw aj al ak ap ar aq am
ao an
ad| aa ad ag aj am ap as av ay af ai ac ao ar al ax ba au ah ab ae aq ak an az
at aw
ae| aa ae ai am aq al ay at ax ao ap ak ba as aw ac ad ah az au av ab af ag an
ar aj
af| aa af ah ap al an av ba at ax az as ac ae ag ar ak am aq aj ao aw ay au ab
ad ai
ag| aa ag ad as ay av aj ap am ah ae ab az aw at aq an ak af ac ai ax au ba ao
al ar
ah| aa ah af av at ba ap an al aq ao aj ab ai ad aw au ay ax as az ar am ak ac
ag ae
ai| aa ai ae ay ax at am al aq az av au an aj ar ab ag af ao ak ap ac ah ad ba
aw as
aj| aa aj as af ao ax ah aq az ap ay ag al au ac an aw ae av ad am ba ai ar at
ab ak
ak| aa ak au ai ap az ae ao av ay ah ar ax ad an at ac aj am aw af al as ab aq
ba ag
al| aa al at ac ak as ab aj au ag ar az ai aq ay ah ap ba ad ao aw af an av ae
am ax
am| aa am ay ao ba ac az ab an al ax ai aw ah ak ag aj av at ae aq ad ap as ar
au af
```

```
an| aa an ba ar as ae aw ai aj au ad aq ah al av am az ac ak ax ag ay ab ao af
ap at
ao| aa ao az al aw ag at ad ar ac an ay ak av ai as af aq ab am ba aj ax ah au
ae ap
ap| aa ap av ax ac ar aq aw ab an at ah ag am as au ai ao ba af al ak az ae ad
aj ay
aq| aa aq ax ba ad ak an au ag aw ac ap aj az af ai am at ar av ab ae al ay as
ah ao
ar| aa ar aw au ah am ak ay af ae aj ba av ac aq ao at ag ai an as az ad al ap
ax ab
as| aa as aj ah az aq af ax ao av am ad at ak ab ba ar ai ap ag ay an ae aw al
ac au
at| aa at al ab au aj ac as ak ad aw ao ae ax am af av an ag az ar ah ba ap ai
ay aq
au| aa au ak ae av ao ai az ap am af aw aq ag ba al ab as ay ar ah at aj ac ax
an ad
av| aa av ap aq ab aw ax ar ac ba al af ad ay aj ak ae az an ah at au ao ai ag
as am
aw| aa aw ar ak af ay au am ah ai as an ap ab ax az al ad ae ba aj ao ag at av
aq ac
ax| aa ax aq an ag au ba ak ad ar ab av as ao ah ae ay al aw ap ac ai at am aj
af az
ay| aa ay am az an ab ao ac ba at aq ae ar af au ad as ap al ai ax ag av aj aw
ak ah
az| aa az ao at ar ad al ag aw ab ba am au ap ae aj ah ax ac ay an as aq af ak
ai av
ba| aa ba an aw aj ai ar ae as ak ag ax af at ap ay ao ab au aq ad am ac az ah
av al
```

[3]:
```
c=K.random_element()
c
```

[3]:
```
2*z3^2 + 2*z3
```

[4]:
```
K.inject_variables()
```

```
Defining z3
```

[5]:
```
z3**8
```

[5]:
```
2*z3^2 + 2
```

[6]:
```
K.<a>=GF(27)
```

[7]:
```
a**8
```

[7]: 2*a^2 + 2

[8]:
```
a==z3
```

[8]: False

[9]:
```
K.modulus()
```

[9]: x^3 + 2*x + 1

[10]:
```
F.<x>=PolynomialRing(GF(3))
P=x**3+2*x+2
L=GF(3**3,'a',modulus=P)
print(L.random_element())

# sinon

M=F.quotient_ring(P)
print(M.random_element())
```

```
a^2 + 2
xbar^2 + xbar + 2
```

1.0.2 Exo 2

[11]:
```
def Racines(Q):
    L=[]
    F=Q.parent().base_ring()
    q=F.cardinality()
    Q=gcd(Q,X**(q)-X)
    for c in F:
        m=0
        while Q(c)==0:
            Q=Q//(X-c)
            m=m+1
        if m>0:
            L.append([c,m])
    return L

F=GF(64)
K.<X>=F['X']

Q=K.random_element(10)

print(Racines(Q))
print(Q.roots())
print()
```

```
P=X**8-X
print(Racines(P))
print(P.roots())
```

```
[[z6^4 + z6^3 + z6 + 1, 1]]
[(z6^4 + z6^3 + z6 + 1, 1)]

[[0, 1], [z6^5 + z6^4 + z6^2 + 1, 1], [z6^4 + z6^2 + z6 + 1, 1], [z6^5 + z6^4 +
z6^2, 1], [z6^5 + z6, 1], [z6^5 + z6 + 1, 1], [z6^4 + z6^2 + z6, 1], [1, 1]]
[(0, 1), (1, 1), (z6^4 + z6^2 + z6, 1), (z6^4 + z6^2 + z6 + 1, 1), (z6^5 + z6,
1), (z6^5 + z6 + 1, 1), (z6^5 + z6^4 + z6^2, 1), (z6^5 + z6^4 + z6^2 + 1, 1)]
```

1.0.3 Exo 3

[12]:
```
def Irreducible(P,q):
    d=P.degree()
    if d==0:
        return True
    Q=power_mod(X,q**d,P)
    if Q!=X:
        return False
    for e in prime_factors(d):
        Q=power_mod(X,q**(d//e),P)-X
        if gcd(Q,P)!=1:
            return False
    return True
R.<x>=GF(17)[]
```

[13]:
```
q=17
R.<X>=GF(q)['X']
P=R.random_element(200)
%time print(Irreducible(P,q))
%time print(P.is_irreducible())
```

```
False
CPU times: user 49.7 ms, sys: 153 µs, total: 49.8 ms
Wall time: 49.1 ms
False
CPU times: user 960 µs, sys: 0 ns, total: 960 µs
Wall time: 962 µs
```

[14]:
```
q=16
R.<X>=GF(q)['X']
P=R.random_element(200)
%time print(Irreducible(P,q))
%time print(P.is_irreducible())
```

```
False
CPU times: user 3.27 s, sys: 0 ns, total: 3.27 s
Wall time: 3.27 s
False
CPU times: user 175 ms, sys: 0 ns, total: 175 ms
Wall time: 175 ms
```

[15]:
```
q=17
R.<X>=GF(q)['X']
P=R.random_element(15)
c=1
while Irreducible(P,q)==False:
    P=R.random_element(15)
    c=c+1
print(P)
print(c)
```

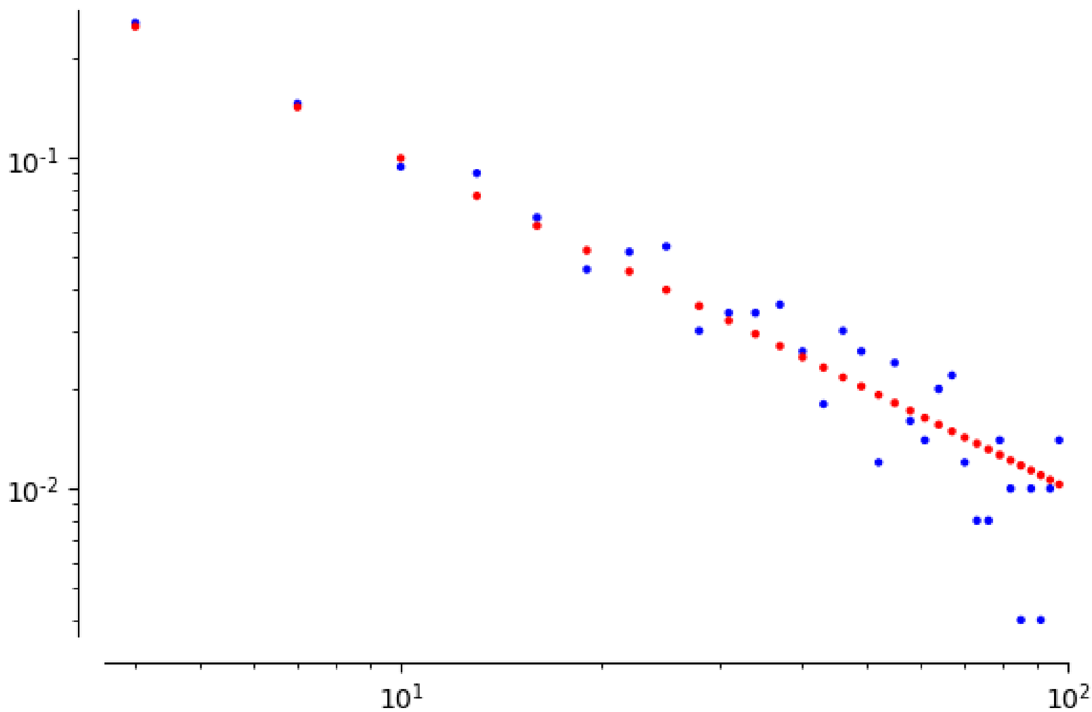
```
15*X^15 + 13*X^14 + 16*X^13 + 6*X^12 + 7*X^11 + 12*X^10 + 2*X^9 + 14*X^8 +
12*X^7 + 12*X^6 + 2*X^5 + 16*X^4 + 10*X^3 + 9*X^2 + 3*X + 8
1
```

[16]:
```
from collections import Counter
np = Counter()  # np[d] = nombre total de polynômes de degrés d tirés
ni = Counter()  # ni[d] = nombre d'irréductibles
```

Les compteurs permettent de capitaliser les calculs déjà effectués. Aussi, si compteur[d] n'existe pas, le compteur retourne 0 et non erreur.

[17]:
```
q=2
R.<x>=GF(q)[]
for d in range(4,100,3):
    for N in range(1,500):
        np[d]+=1
        P=x^d+R.random_element(degree=d-1)
        if P.is_irreducible():
            ni[d]+=1
```

[18]:
```
dessin1 = point2d([(d,ni[d]/np[d]) for d in range(4,100,3)])

# on veut comparer avec les bornes 1/2d et 1/d.

dessin2 = point2d([(d,1/d) for d in range(4,100,3)], color='red')

dessin=dessin1+dessin2

# pour visualiser proprement (linéaire), on passe en log-log

dessin.show(scale='loglog')
```



[19]:
```
P=R.random_element(15)
P.factor()
```

[19]: `x * (x + 1)^4 * (x^10 + x^9 + x^5 + x + 1)`

1.0.4 Exo 4 (algorithme de Berlekamp)

Contruction de la matrice de l'algèbre de Berlekamp

[20]:
```
def Matrice(f):
    d=f.degree()
    q=f.parent().base_ring().cardinality()
    M=matrix(GF(q),d,d)
    for j in range(d):
        g=power_mod(X,j*q,f)-X**j
        for i in range(d):
            M[i,j]=g.monomial_coefficient(X**i)
    return M
```

[21]:
```
q=5
R.<X>=GF(q)['X']
f=R.random_element(5)
M=Matrice(f)
print(M)
```

```
print("Noyau :", kernel(M.transpose()).basis())
factor(f)
```

```
[0 3 0 0 2]
[0 2 1 3 2]
[0 0 1 0 3]
[0 1 1 2 2]
[0 2 3 1 0]
Noyau : [
(1, 0, 0, 0, 0)
]
```

[21]:
```
(4) * (X^5 + 3*X^4 + 4*X^3 + 2*X + 2)
```

Approche déterministe

[22]:
```
def Berlekamp(f):
    if f.degree()<=1:
        return [f]
    #if gcd(f,f.derivative())!=1:
    #    print ("non separable")
    #    return()
    M=Matrice(f)
    B=kernel(M.transpose()).basis()
    if len(B)==1:
        return [f]     # f est irreductible (s=1)
    R=f.parent()
    j=0
    g=R(B[0].list()) # R(B[j].list()) transforme le vecteur B[j] en un polynôme
    while g.degree()==0:  # Cherche g non constant
        j=j+1
        g=R(B[j].list())
    for a in R.base_ring():  # a in GF(q)
        f1=gcd(f,g-a)
        if f1.degree()>0:
            f2=f//f1
            return Berlekamp(f1)+Berlekamp(f2)
```

[23]:
```
q=5
R.<X>=GF(q)['X']
f=R.random_element(10)
pretty_print(Berlekamp(f))
pretty_print(factor(f))
```

```
[X, X + 4, X^6 + X^5 + X^4 + 3*X^3 + 4*X^2 + 2, 2*X^2 + 2*X + 4]
```

```
(2) * X * (X + 4) * (X^2 + X + 2) * (X^6 + X^5 + X^4 + 3*X^3 + 4*X^2 + 2)
```

[24]:
```
f=X**q-X
print(Berlekamp(f))
print(factor(f))
```

```
[X, X + 4, X + 3, X + 2, X + 1]
X * (X + 1) * (X + 2) * (X + 3) * (X + 4)
```

Approche probabiliste

[25]:
```
def Cherche_g(f,B):
    R=f.parent()
    q=R.base_ring().cardinality()
    while True:
        g=sum(GF(q).random_element()*R(v.list()) for v in B)
        f1=gcd(f,g)
        if f1.degree()!=0 and f1.degree()!=f.degree():
            return [f1,f//f1]
        f1=gcd(f,power_mod(g,(q-1)//2,f)-1)
        if f1.degree()!=0 and f1.degree()!=f.degree():
            return [f1,f//f1]


def Berlekamp2(f):
    if f.degree()<=1:
        return [f]
    M=Matrice(f)
    B=kernel(M.transpose()).basis()
    if len(B)==1:
        return [f]     # f est irreductible (s=1)
    [f1,f2]=Cherche_g(f,B)
    return Berlekamp2(f1)+Berlekamp2(f2)
```

[26]:
```
q=5
R.<X>=GF(q)['X']
f=X**(5**3)-X
%time Berlekamp(f)
%time Berlekamp2(f)
%time F=factor(f)
print(len(F))
```

```
CPU times: user 1.84 s, sys: 3.94 ms, total: 1.84 s
Wall time: 1.84 s
CPU times: user 1.78 s, sys: 3.91 ms, total: 1.78 s
Wall time: 1.78 s
CPU times: user 913 µs, sys: 0 ns, total: 913 µs
```

```
Wall time: 916 µs
45
```

[27]:
```
p=next_prime(1000000)
R1.<X>=GF(p)['X']
f=R1.random_element(10)

%time Berlekamp(f)
%time Berlekamp2(f)
%time factor(f)
```

```
CPU times: user 843 ms, sys: 3.91 ms, total: 847 ms
Wall time: 845 ms
CPU times: user 4.04 ms, sys: 7 µs, total: 4.05 ms
Wall time: 4.05 ms
CPU times: user 136 µs, sys: 0 ns, total: 136 µs
Wall time: 138 µs
```

[27]:
```
(503364) * (X + 892028) * (X^4 + 816288*X^3 + 721420*X^2 + 347350*X + 321489) *
(X^5 + 799265*X^4 + 936813*X^3 + 685476*X^2 + 500706*X + 617404)
```

Les approches déterministes et probabilistes se valent pour q petit, mais l'approche probabiliste devient nettement meilleure quand q est grand.

1.0.5 Exo 5 (algorithme de Cantor-Zassenhauss)

Factorisation de f en degrés distincts.

[28]:
```
def DistinctDegreeFacto(f):
    q=f.parent().base_ring().cardinality()
    L=[]
    r=1
    while f.degree()!=0:
        h=gcd(power_mod(X,q**r,f)-X,f)
        if h.degree()>0:
            L.append((h,r))
        f=f//h
        r=r+1
    return(L)
```

[29]:
```
R3.<X>=GF(3)['X']
f=X**9-X
L=DistinctDegreeFacto(f)
print(L)
```

```
[(X^3 + 2*X, 1), (X^6 + X^4 + X^2 + 1, 2)]
```

On factorise ensuite chaque facteur en s'inspirant de Berlekamp probabiliste (Lemme 8 du CM).

[30]:
```
def Split(h,r): # h produit de polynômes de degrés r
    R=f.parent()
    q=R.base_ring().cardinality()
    while True:
        g=R.random_element(2*r)
        h1=gcd(h,g)
        if h1.degree()!=0 and h1.degree()!=h.degree():
            return [h1,h//h1]
        h1=gcd(h,power_mod(g,(q-1)//2,f)-1)
        if h1.degree()!=0 and h1.degree()!=f.degree():
            return [h1,h//h1]

def EqualDegreeFacto(h,r):
    if h.degree()==r:
        return [h]
    [h1,h2]=Split(h,r)
    return EqualDegreeFacto(h1,r)+EqualDegreeFacto(h2,r)
```

[31]:
```
(h,r)=(X^6 + X^4 + X^2 + 1, 2)
print(EqualDegreeFacto(h,r))
```

```
[X^2 + X + 2, X^2 + 2*X + 2, X^2 + 1]
```

Finalement, on met bout à bout ces deux factorisations pour obtenir la facto globale de f

[32]:
```
def Zassenhauss(f):
    L=DistinctDegreeFacto(f)
    Res=[]
    for (h,r) in L:
        Res.extend(EqualDegreeFacto(h,r))
    return(Res)
```

[33]:
```
%time Res1=Zassenhauss(X**(3**5)-X); len(Res1)
%time Res2=Berlekamp2(X**(3**5)-X); len(Res2)
```

```
CPU times: user 17.7 ms, sys: 26 µs, total: 17.7 ms
Wall time: 17.5 ms
CPU times: user 9.64 s, sys: 3.97 ms, total: 9.65 s
Wall time: 9.64 s
```

[33]: 51

Racines

[34]:
```
def Racines2(f):
    q=f.parent().base_ring().cardinality()
    f=gcd(f,X**q-X)
    Z=Zassenhauss(f)
```

```
    Res=[-Z[i](0) for i in range(len(Z))]
    return Res
```

[35]:
```
RR.<X>=GF(3)['X']
f=RR.random_element(100)
%time print(Racines(f))
%time print(Racines2(f))
```

```
[]
CPU times: user 713 µs, sys: 0 ns, total: 713 µs
Wall time: 516 µs
[]
CPU times: user 100 µs, sys: 0 ns, total: 100 µs
Wall time: 93 µs
```

[36]:
```
X + 2*z3^2 + z3
```

[36]:
```
X + 2*z3^2 + z3
```

1.0.6 Exo 5 (facteurs multiples)

On écrit $f = ab$ avec $a = g^q$ pour q une puissance de p, b sans facteur irréductible de multiplicité divisible par p et a et b premiers entre eux. On note bb le radical de b.

Etape “traite b” : On calcule bb, que l’on factorise via Zassenhauss. On en déduit la facto de b.

[37]:
```
def Traite_b(f): # factorise la partie non puissance de p
    bb=f//gcd(f,f.derivative())
    if bb.degree()==0:
        return []
    Fac=Zassenhauss(bb)
    Res=[]
    for h in Fac:
        f=f//h
        k=1
        while f%h==0:
            f=f//h
            k=k+1
        Res.append((h,k))
    return Res
```

[38]:
```
Traite_b(X**7)
```

[38]:
```
[(X, 7)]
```

Etape “traite a” : on suppose ici que $f' = 0$. On a donc $f = g^n$ avec $g' \neq 0$ et n une puissance de p. On cherche à calculer (g, n).

Attention, il faudra penser à rappeler l’algorithme global de factorisation sur g ensuite.

[39]:
```
def Traite_a(f):
    p=f.parent().characteristic()
    mult=1
    h=f
    while h.derivative()==0: # donc h(X)=g(X^p)=g(X)^p. On remplace h par g et␣
↪la multiplicité est multipliée par p
        L=h.list()
        h=sum(L[i]*X**(i//p) for i in range(0,len(L),p))
        mult=mult*p
    return (h,mult)
```

[40]:
```
Traite_a(X**9+1)
```

[40]:
```
(X + 1, 9)
```

Om met tout ensemble, sans oublier de rappeler l'algo global sur g.

[41]:
```
def Facto(f):
    Lb=Traite_b(f)
    db=sum(Lb[i][0].degree()*Lb[i][1] for i in range(len(Lb))) # degré de b
    if db==f.degree():
        return Lb
    for (h,m) in Lb:
        f=f//h**m      # f=a
    (g,n)=Traite_a(f)   # f'=0, f=g^n avec g' non nul.
    Lg=Facto(g)
    La=[(Lg[i][0],n*Lg[i][1]) for i in range(len(Lg))]  # les multiplicités des␣
↪facteurs de g doivent être multipliées  par n
    return La+Lb
```

[42]:
```
g=R3.random_element(10)
f=g(X**6)*g(X**2)*g
f
```

[42]:
```
2*X^90 + 2*X^89 + 2*X^88 + X^87 + X^86 + 2*X^82 + X^81 + X^80 + 2*X^79 + 2*X^78
+ 2*X^75 + X^74 + 2*X^72 + 2*X^70 + X^69 + X^68 + 2*X^67 + X^66 + 2*X^64 +
2*X^63 + X^62 + X^60 + X^59 + 2*X^58 + 2*X^55 + X^54 + X^52 + X^51 + X^49 + X^47
+ 2*X^46 + X^45 + 2*X^44 + 2*X^43 + 2*X^38 + 2*X^37 + X^36 + X^35 + X^34 +
2*X^33 + X^32 + 2*X^31 + X^30 + X^29 + 2*X^28 + X^27 + 2*X^24 + X^23 + 2*X^22 +
2*X^21 + 2*X^20 + 2*X^18 + 2*X^16 + 2*X^15 + 2*X^14 + X^13 + 2*X^11 + X^10 +
2*X^9
```

[43]:
```
print(factor(f))
print(Facto(f))
```

```
(2) * (X + 1) * X^9 * (X^2 + 2*X + 2) * (X^2 + 1)^4 * (X^4 + 2*X^2 + 2)^4 * (X^6
+ X^5 + 2*X^4 + X^3 + X^2 + 2) * (X^12 + X^10 + 2*X^8 + X^6 + X^4 + 2)^4
[(X, 9), (X + 1, 1), (X^2 + 1, 4), (X^2 + 2*X + 2, 1), (X^4 + 2*X^2 + 2, 4),
(X^6 + X^5 + 2*X^4 + X^3 + X^2 + 2, 1), (2*X^12 + 2*X^10 + X^8 + 2*X^6 + 2*X^4 +
1, 4)]
```

Magique ! Ca fonctionne.

[ ]:

```

```