CorrigeExamen2022-2023

December 8, 2022

0.0.1 Exo 1

[1]:
```
# question 3)

def DES1(P,Q):
    a=P.degree(); b=Q.degree()
    if a<b :
        return([P])
    # Cherche plus petite puissance de 2 tq 2^p deg(B) > deg(A)
    k=1
    while k*b <= a:
        k=2*k
    Quo,Rem=P.quo_rem(Q**(k//2))
    LC=DES1(Quo,Q)
    LR=DES1(Rem,Q)
    # on a deg(R)< (k//2)deg(B), donc LR de taille <= k//2. On complete LR avec␣
 ↪des zéros au début
    # pour qu'elle soit de taille exactement k//2 avant de la concaténer à LQ␣
 ↪(sinon y a un souci de taille totale de liste).
    for i in range(k//2-len(LR)):
        LR.insert(0,0)
    LC.extend(LR)
    return(LC)

K.<X>=GF(11)[]
l=8
Q=K.random_element(degree=4)
P=K.random_element(degree=4*l-1)

LC=DES1(P,Q)
LC
P/Q**l==sum(LC[i]/Q**(i+1) for i in range(len(LC)))
```

[1]:
```
True
```

[2]:
```
### Question 4) b)

K.<X>=QQ[]
```

```
def SommeInverses(LQ):
    k=len(LQ)
    if k==1:
        return([1,LQ[0]])
    [N1,M1]=SommeInverses(LQ[:k//2])
    [N2,M2]=SommeInverses(LQ[k//2:])
    return(N1*M2+N2*M1,M1*M2)

k=64
K.<X>=GF(next_prime(1000))[]
LQ=[]
for i in range(k):
    LQ.append(K.random_element(degree=4))
F=K.fraction_field()
S=sum(1/LQ[i] for i in range(k))

%time [N,M]=SommeInverses(LQ)
print(S==N/M)
print(N.parent())
```

```
CPU times: user 258 µs, sys: 38 µs, total: 296 µs
Wall time: 299 µs
True
Univariate Polynomial Ring in X over Finite Field of size 1009
```

[3]:
```
### Question 6

def MultiModules(P,LQ):
    return [P%LQ[i] for i in range(len(LQ))]

def DES(P,LQ,l):  # LQl=[Q_1,...,Q_k],  Ll=[l1,...,lk]
    LQl=[LQ[i]**l[i] for i in range(len(LQ))]
    R=SommeInverses(LQl)[0]
    LR=MultiModules(R,LQl)
    LRinv=[LR[i].inverse_mod(LQl[i]) for i in range(len(LR))]
    LP=MultiModules(P,LQl)
    LCl=[LP[i]*LRinv[i]%LQl[i] for i in range(len(LP))]
    LC=[]
    for i in range(len(LCl)):
        LC.append(DES1(LCl[i],LQ[i]))
    return LC

def Tirage(k,d,p):
    K.<X>=GF(next_prime(p))[]
    F=K.fraction_field()
    LQ=[]
```

```
    l=[]
    for i in range(k):
        cherche=True
        while cherche==True:
            Q=K.random_element(degree=d)
            j=0
            while j<len(LQ):
                if gcd(Q,LQ[j])!=1:
                    break
                else:
                    j=j+1
            if j==len(LQ):
                cherche=False
                LQ.append(Q)
                l.append(randint(1,10))
    n=sum(l[i]*LQ[i].degree() for i in range(k))
    P=K.random_element(degree=n-1)  # deg P=n-1
    return(P,LQ,l)

P,LQ,l=Tirage(256,4,next_prime(100))

%time LC=DES(P,LQ,l)

A=P/prod(LQ[i]**l[i] for i in range(len(LQ)))
B=sum(sum(LC[i][j]/LQ[i]**(j+1) for j in range(l[i])) for i in range(len(LQ)))
A==B
```

```
CPU times: user 81.8 ms, sys: 351 µs, total: 82.2 ms
Wall time: 82.1 ms
```

[3]:
```
True
```

0.0.2 Exo 2

1) Primalité

[4]:
```
## a)
n=2**(2**4)+1
if power_mod(3,n-1,n)==1 and power_mod(3,(n-1)//2,n)!=1:
    print(True)
else:
    print(False)
```

```
True
```

[5]:
```
## b)
```

```
def TestLucas(n):
    a=randint(2,n-1)
    if power_mod(a,n-1,n)!=1:
        return ("assurément non premier")
    P=(n-1).prime_divisors()
    for p in P:
        if power_mod(a,(n-1)//p,n)==1:
            return "possiblement non premier"
    return "assurément premier"

TestLucas(n)
```

[5]:
```
'possiblement non premier'
```

[6]:
```
## d)

L=[[73,17],[12,11],[32,96],[23,52],[89,113]]
for [k,l] in L:
    T=TestLucas(2**k*3**l+1)
    c=1
    while T=="possiblement non premier":
        T=TestLucas(2**k*3**l+1)
        c=c+1
    print([T,c])
```

```
['assurément premier', 2]
['assurément premier', 3]
['assurément premier', 3]
['assurément non premier', 1]
['assurément non premier', 1]
```

[7]:
```
for [k,l] in L:
    print(is_prime(2**k*3**l+1))
```

```
True
True
True
False
False
```

2) Méthode rho de Pollard

[8]:
```
# a)

def f(x,n):
    return mod(x**2+1,n)

def Pollard(n,x0):
```

```
    x1=f(x0,n)
    y1=f(f(x0,n),n)
    g=gcd(x1-y1,n)
    while g==1:
        x1=f(x1,n)
        y1=f(f(y1,n),n)
        g=gcd(x1-y1,n)
    return g
```

[9]:
```
# b)

n=10**(20)+67
%time print(Pollard(n,0))
n.factor()
```

```
158021683
CPU times: user 383 ms, sys: 3.87 ms, total: 387 ms
Wall time: 386 ms
```

[9]:
```
158021683 * 632824547249
```

0.0.3 Exo 3

1)

[10]:
```
# d)

F.<X>=GF(3)[]
K.<X>=PolynomialQuotientRing(F,X**3+2*X+1)

print('K un corps ?', is_field(K))
print('Cardinal K:', K.cardinality())

# X générateur de K* ssi X d'ordre 26, ssi X**26=1 et X**p!=1 pour tout␣
 ↪diviseur premier de 26.
print(X**26, X**3, X**13)

k=1
while X**2+X!=X**k:
    k=k+1
print('k=',k)
X**10==X**2+X
```

```
K un corps ? True
Cardinal K: 27
1 X + 2 2
k= 10
```

[10]: True

2)

[11]:
```
# c)

F.<X>=GF(2)[]
Q=X**4+X+1
K=GF(4)
P=Q.change_ring(K)
P.factor()
```

[11]: (X^2 + X + z2) * (X^2 + X + z2 + 1)

4)

[12]:
```
# b)  On cherche une partie stable stricte de Z/8Z contenant 3 entiers␣
 ↪consécutifs.

def PartieStable(L,n,q):
    Sigma=L
    R=IntegerModRing(n)
    for k in L:
        a=mod(k*q,n)
        while a!=k:
            if a not in Sigma:
                Sigma.append(int(a))
            a=q*a
    Sigma.sort()
    return Sigma

Sigma=PartieStable([1,2,3],8,3)
print(Sigma)
B.<z>=GF(3**2)
A.<X>=GF(3)['X']
alpha=B.primitive_element()
g=prod(X-alpha**(Sigma[i]) for i in range(len(Sigma)))
print(g)
print('dimension du code:', 8-g.degree())
print(alpha**(3**8-1))
```

```
[1, 2, 3, 6]
X^4 + 2*X^3 + 2*X + 2
dimension du code: 4
1
```

[13]:
```
# c)
## Un mot m de F_3[X]/(X^8-1) est un mot du code ssi m%g=0.
## Les colonnes d'une matrice de contrôle sont donc les m%g (polynomes de␣
 ↪degrés 3) où m parcourt une base de
## F_3[X]/(X^8-1). En considérant m=1,X,X^2,...,X^7, on obtient :

def MatControle(g):
    M=[]
    for i in range(8):
        Ri=X**i%g
        M.append(list(Ri)+[0]*(4-Ri.degree()-1))  # ajout de 0 pour s'assurer␣
 ↪d'avoir un vecteur de longueur 4
    return matrix(M).transpose()

MatControle(g)
```

[13]:
```
[1 0 0 0 1 1 1 2]
[0 1 0 0 1 2 2 0]
[0 0 1 0 0 1 2 2]
[0 0 0 1 1 1 2 1]
```

La distance minimale est >= 4 par construction. En particulier, chaque triplet de colonne est libre. Or il existe 4 colonnes liées (ex : C0+C1+C3=C4). Donc la distance minimale est exectement 4 (cf Lemme 5 du CM5)

0.0.4 Exo 4

[14]:
```
## 1)c)

RR2.<x,y,z>=PolynomialRing(QQ,'x,y,z')
P=x*y**2-y+1
Q=2*x*y*z-z+y**2
R=P.resultant(Q,y)
R
```

[14]:
```
4*x^3*z^2 - x^2*z^2 + 4*x*z - z + 1
```

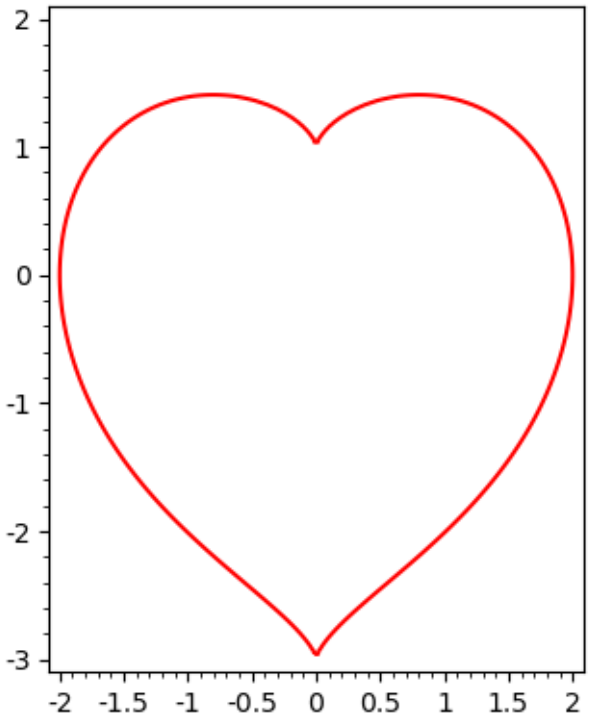
[15]:
```
## 2)a)
RR2.<t,x,y>=PolynomialRing(QQ,'t,x,y',order='lex')
G1=x**2*(1+t**2)-4*(1-t**2)**3
G2=-y*(1+t**2)+4*t-t**2*(1+t**2)

R=G1.resultant(G2,t)
print(R)
```

```
16*x^8 + 128*x^6*y^2 + 512*x^6*y + 1920*x^6 + 256*x^4*y^4 + 2048*x^4*y^3 +
16896*x^4*y^2 + 57344*x^4*y + 103680*x^4 + 24576*x^2*y^4 + 131072*x^2*y^3 +
```

```
114688*x^2*y^2 - 458752*x^2*y - 335872*x^2 + 16384*y^6 + 98304*y^5 + 49152*y^4 -
458752*y^3 - 147456*y^2 + 884736*y - 442368
```

[16]:
```
## 2)b)
RR1.<x,y>=PolynomialRing(QQ,'x,y')
R=RR1(R)
implicit_plot(R,(-2,2),(-3,2),color='red')
```

[16]:



[17]:
```
## 3) a)
RR.<t,x,y,z>=PolynomialRing(QQ,'t,x,y,z',order='lex')
G1=x-t**5
G2=y-t**3
G3=z-t**2

I=RR.ideal(G1,G2,G3)
J=I.groebner_basis()
print(J)
```

```
[t^2 - z, t*y - z^2, t*z - y, x - y*z, y^2 - z^3]
```

L'idéal éliminant t est engendré par les éléments de la base de Gröbner qui vivent dans $\mathbb{Q}[x, y, z]$, c'est à dire ici $J[-1]$ et $J[-2]$.

[18]:
```
R3.<x,y,z>=PolynomialRing(QQ,'x,y,z',order='lex')
P=R3(J[-2])
Q=R3(J[-1])
var('x,y,z')
G1=implicit_plot3d(P,(x,-5,5),(y,-5,5),(z,-2,2),color='blue')
G2=implicit_plot3d(Q,(x,-5,5),(y,-5,5),(z,-2,2),color='red')
G1+G2
```

[18]: Graphics3d Object

[19]:
```
var('t')
parametric_plot3d((t**5,t**3,t**2),(t,-1,1))
```

[19]: Graphics3d Object

[20]:
```
G1=x-t**7
G2=y-t**5
G3=z-t**3

I=RR.ideal(G1,G2,G3)
J=I.groebner_basis()
print(J)
```

```
[t^3 - z, t^2*z - y, t*x - y*z, t*y - z^2, t*z^2 - x, x^2 - y*z^3, x*y - z^4,
x*z - y^2, y^3 - z^5]
```

Donc la courbe est incluse dans la variété $V(x^2 - yz^3, xy - z^4, xz - y^2, y^3 - z^5)$. En particulier dans $V(xz - y^2, y^3 - z^5)$.

[21]:
```
P=R3(J[-2])
Q=R3(J[-1])
var('x,y,z')
G1=implicit_plot3d(P,(x,-5,5),(y,-5,5),(z,-2,2),color='blue')
G2=implicit_plot3d(Q,(x,-5,5),(y,-5,5),(z,-2,2),color='red')
G1+G2
```

[21]: Graphics3d Object

[22]:
```
var('t')
parametric_plot3d((t**7,t**5,t**3),(t,-1,1))
```

[22]: Graphics3d Object

On voit graphiquement que les surfaces bleues et rouges contiennent une extra droite. C'est la droite d'équation $y = z = 0$: tous ces points annulent bien les deux équations $xz - y^2$ et $y^3 - z^5$, mais ne font pas partie de la courbe. On peut en fait montrer que cette courbe ne peut pas être décrite comme l'intersection de seulement deux surfaces algébriques.

[ ]: