Corrige

January 26, 2024

1 Corrigé examen Calcul Formel 2023-2024

1.1 Exo 1

1.1.1 3. a)

[1]:
```
def Inversion(G,N): # deg G < N, G(0)=1
    if G(0)!=A(1):
        return False
    if N==1:
        return 1
    F=Inversion(G,ceil(N/2))
    NF=F+(1-G*F)*F%X^N
    return NF
```

[2]:
```
# test
A.<X>=IntegerModRing(8)['X']
N=15
G=A.random_element(13)
G=X*G+1
F=Inversion(G,N)
print(G)
print(F)
print(G*F%X^N)
```

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

1.1.2 3. b)

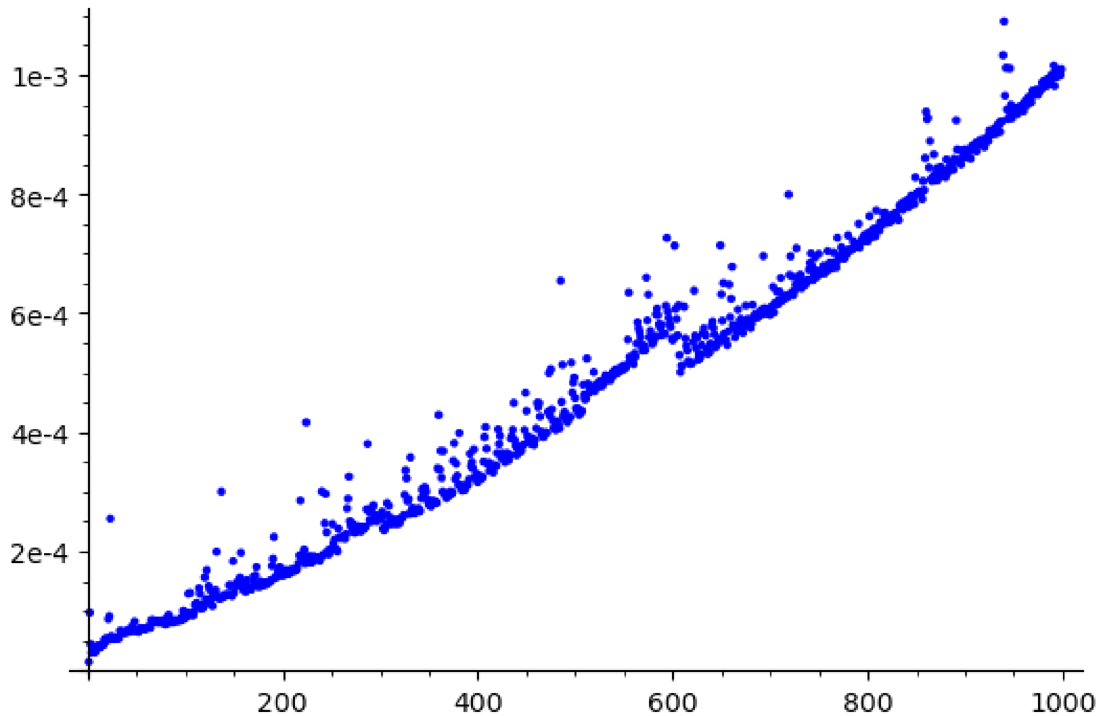
[3]:
```
from time import *

T=[]
```

```
for N in range(1,1000):
    G=A.random_element(N-2)
    G=X*G+1
    t0=time()
    F=Inversion(G,N)
    T.append([N,time()-t0])
```

[4]:
```
point(T)
```

[4]:



1.1.3 3. c)

[5]:
```
def InverseNaif(G,N):
    F=sum((1-G)**i%X**N for i in range(N))
    return F
```

[6]:
```
A.<X>=IntegerModRing(8)['X']
N=1000
G=A.random_element(13)
G=X*G+1
t0=time()
F=Inversion(G,N)
t1=time()
F1=InverseNaif(G,N)
```

```
t2=time()
print("Meme resultat :", F==F1)
print("Temps rapide :", t1-t0)
print("Temps naïf :", t2-t1)
```

```
Meme resultat : True
Temps rapide : 0.00023436546325683594
Temps naïf : 1.1456284523010254
```

1.1.4 3. d)

[7]:
```
def Reciproque(F):
    A=F.parent()
    L=list(F)
    L.reverse()
    return A(L)

def Division(F,G): # deg G < N, G unitaire
    n=G.degree()
    m=F.degree()
    if m<n:
        return F
    Gt=Reciproque(G)
    Ft=Reciproque(F)
    Gt_inv=Inversion(Gt,m-n+1)
    P=Ft*Gt_inv%X^(m-n+1)
    Q=Reciproque(P)
    Q=Q*X**(m-n-Q.degree()) # si des fois deg P < m-n, on peut louper des␣
↪coefficients de Q.
    R=(F-G*Q)%X^n
    return (Q,R)
```

[8]:
```
A.<X>=IntegerModRing(8)['X']
m=100000
n=10000
F=A.random_element(m)
G=A.random_element(n-1)+X^n
t0=time()
Q,R=Division(F,G)
print('time :', time()-t0)
t1=time()
print(R.degree()<G.degree(),F==Q*G+R)
```

```
time : 0.4523184299468994
True True
```

1.2 Exo 2

1.2.1 6. a)

[9]:
```
def relation(a,P):
    C=[]
    if a==0: return []
    for p in P:
        c=0
        while a%p==0:
            c+=1
            a//=p
        C.append(c)
    if a!=1: return []
    return C
```

1.2.2 6.b)

[10]:
```
N=4333801
factor(N)
```

[10]: 641 * 6761

[11]:
```
L=[]
P=primes_first_n(9)
N0=ceil(sqrt(N))
for b in range(N0,N0+1000):
    a=b**2-N
    va=relation(a,P)
    if va!=[]:
        L.append([b,va])
print(L)
len(L)
```

```
[[2086, [0, 2, 1, 0, 0, 0, 1, 0, 1]], [2099, [6, 2, 3, 0, 0, 0, 0, 0, 0]],
[2131, [9, 4, 1, 0, 0, 0, 0, 0, 0]], [2147, [5, 1, 0, 0, 0, 2, 1, 0, 0]], [2221,
[10, 2, 1, 0, 0, 1, 0, 0, 0]], [2247, [3, 0, 0, 0, 0, 2, 0, 0, 2]], [2351, [3,
3, 2, 0, 0, 1, 1, 0, 0]], [2477, [9, 2, 0, 0, 0, 0, 1, 0, 1]], [2776, [0, 1, 3,
0, 0, 0, 1, 0, 2]], [2891, [4, 7, 1, 0, 0, 0, 0, 0, 1]]]
```

[11]: 10

1.2.3 6. c) et d)

[12]:
```
V=[l[1] for l in L]
M=matrix(GF(2),V)
V=M.left_kernel().basis()
```

```
ListeI=[]
for v in V:
    I=[]
    for i in range(len(v)):
        if v[i]==1:
            I.append(i)
    ListeI.append(I)

print(M)
print(V)
print(ListeI)
```

```
[0 0 1 0 0 0 1 0 1]
[0 0 1 0 0 0 0 0 0]
[1 0 1 0 0 0 0 0 0]
[1 1 0 0 0 0 1 0 0]
[0 0 1 0 0 1 0 0 0]
[1 0 0 0 0 0 0 0 0]
[1 1 0 0 0 1 1 0 0]
[1 0 0 0 0 0 1 0 1]
[0 1 1 0 0 0 1 0 0]
[0 1 1 0 0 0 0 0 1]
[
(1, 0, 0, 1, 0, 0, 0, 1, 1, 0),
(0, 1, 0, 1, 0, 0, 0, 1, 0, 1),
(0, 0, 1, 1, 0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 1, 0, 1, 1, 0, 1),
(0, 0, 0, 0, 0, 1, 0, 1, 1, 1)
]
[[0, 3, 7, 8], [1, 3, 7, 9], [2, 3, 8], [4, 6, 7, 9], [5, 7, 8, 9]]
```

[13]:
```
ListeI
```

[13]:
```
[[0, 3, 7, 8], [1, 3, 7, 9], [2, 3, 8], [4, 6, 7, 9], [5, 7, 8, 9]]
```

1.2.4 6. e)

[14]:
```
for I in ListeI:
    x=prod(mod(L[i][0],N) for i in I)
    x=ZZ(x)
    vy=sum(vector(L[i][1]) for i in I)/2
    y=prod(mod(P[j]**vy[j],N) for j in range(9))
    y=ZZ(y)
    print("x,y:",x,y)
    print("Factorisation induite :", N ,"=", gcd(x-y,N),"*",gcd(x+y,N))
```

```
x,y: 2257823 2075978
Factorisation induite : 4333801 = 1 * 4333801
x,y: 3453416 2304046
Factorisation induite : 4333801 = 6761 * 641
x,y: 2876502 1457299
Factorisation induite : 4333801 = 1 * 4333801
x,y: 822873 822873
Factorisation induite : 4333801 = 4333801 * 1
x,y: 2666814 3904077
Factorisation induite : 4333801 = 6761 * 641
```

1.3 Exo 3

1.3.1 1. d)

[15]:
```
F.<X>=GF(2)['X']

def ListeIrreductibles(n):
    V=VectorSpace(GF(2),n)
    L=[X^n + F(v.list()) for v in V]
    Lirr=[]
    for Q in L:
        if Q.is_irreducible():
            Lirr.append(Q)
    return Lirr
```

1.3.2 1. e)

[16]:
```
F4=GF(4)['X']

for Q in ListeIrreductibles(4):
    print(F4(Q).factor())
```

```
(X^2 + X + z2) * (X^2 + X + z2 + 1)
(X^2 + z2*X + z2) * (X^2 + (z2 + 1)*X + z2 + 1)
(X^2 + z2*X + 1) * (X^2 + (z2 + 1)*X + 1)
```

1.3.3 2. b)

[17]:
```
F3.<X>=GF(3)['X']
Q=X**10-1
```

[18]:
```
L=prime_factors(Q)
L
```

[18]:
```
[X + 1, X + 2, X^4 + X^3 + X^2 + X + 1, X^4 + 2*X^3 + X^2 + 2*X + 1]
```

[19]:
```
def MatControle(g,n):
    H=[]
    for i in range(n):
        Col=list(X**i%g)
        Col.extend([0 for i in range(g.degree()-len(Col))])
        H.append(Col)
    H=matrix(H).transpose()
    return H


for g in L:
    print("Matrice de controle de", g)
    print (MatControle(g,10))
    print()

print("Matrice de controle de (", L[0],") * (",(L[2]),")")
print (MatControle(L[0]*L[2],10))
print()
```

```
Matrice de controle de X + 1
[1 2 1 2 1 2 1 2 1 2]

Matrice de controle de X + 2
[1 1 1 1 1 1 1 1 1 1]

Matrice de controle de X^4 + X^3 + X^2 + X + 1
[1 0 0 0 2 1 0 0 0 2]
[0 1 0 0 2 0 1 0 0 2]
[0 0 1 0 2 0 0 1 0 2]
[0 0 0 1 2 0 0 0 1 2]

Matrice de controle de X^4 + 2*X^3 + X^2 + 2*X + 1
[1 0 0 0 2 2 0 0 0 1]
[0 1 0 0 1 0 2 0 0 2]
[0 0 1 0 2 0 0 2 0 1]
[0 0 0 1 1 0 0 0 2 2]

Matrice de controle de ( X + 1 ) * ( X^4 + X^3 + X^2 + X + 1 )
[1 0 0 0 0 2 2 1 2 1]
[0 1 0 0 0 1 0 1 2 1]
[0 0 1 0 0 1 2 2 2 1]
[0 0 0 1 0 1 2 1 0 1]
[0 0 0 0 1 1 2 1 2 2]
```

[20]:
```
MatControle(X+ 2,11)
```

[20]:
```
[1 1 1 1 1 1 1 1 1 1 1]
```

[21]:
```
MatControle(X^5 + X^4 + 2*X^3 + X^2 + 2,11)
```

[21]:
```
[1 0 0 0 0 1 2 2 2 1 0]
[0 1 0 0 0 0 1 2 2 2 1]
[0 0 1 0 0 2 1 2 0 1 2]
[0 0 0 1 0 1 1 0 1 1 1]
[0 0 0 0 1 2 2 2 1 0 1]
```

Dans les 4 premiers cas (g irréductible), il existe au moins deux colonnes liées sur $\mathbb{F}_3$. Dans le dernier cas ($g = g_1 g_3$), les colonnes sont deux à deux $\mathbb{F}_3$ linéairement indépendantes. Donc le code produit $g_1 g_3$ est de distance minimale au moins 3 (en fait 4 ici).

1.4 Exo 4

1.4.1 2.

[22]:
```
R1.<t,x,y>=PolynomialRing(QQ,'t,x,y',order='deglex')
G1=x*(1+t**2)**3-8*t**3
G2=y*(1+t**2)**3-(1-t**2)**3

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

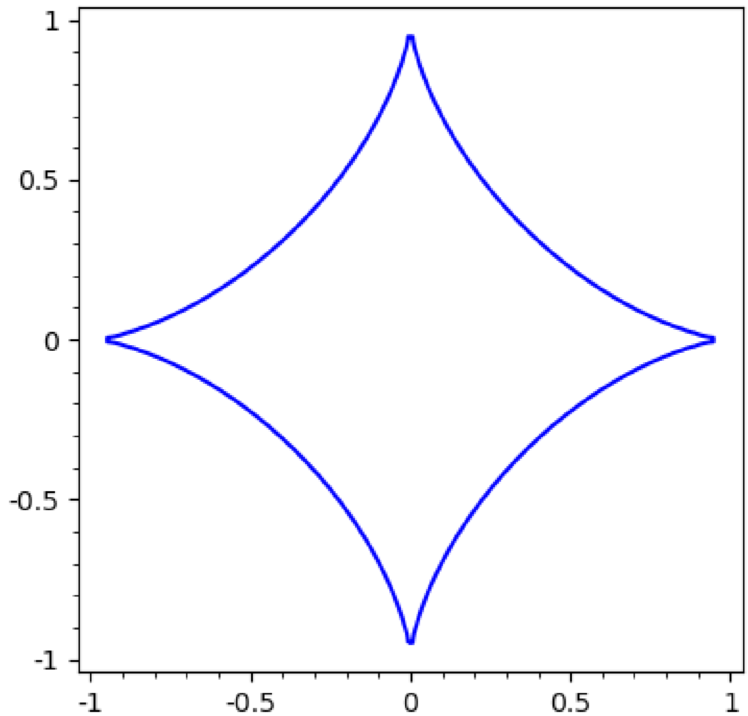
```
262144*x^6 + 786432*x^4*y^2 + 786432*x^2*y^4 + 262144*y^6 - 786432*x^4 +
5505024*x^2*y^2 - 786432*y^4 + 786432*x^2 + 786432*y^2 - 262144
```

[23]:
```
R2.<x,y>=PolynomialRing(QQ,'x,y',order='lex')
f=R2(f)
fx=f.derivative(x)
fy=f.derivative(y)
I=R2.ideal(f,fx,fy)
I.variety(QQbar)
```

[23]:
```
[{y: 0, x: 1},
 {y: 0, x: -1},
 {y: 1, x: 0},
 {y: -1, x: 0},
 {y: -1*I, x: -1*I},
 {y: -1*I, x: 1*I},
 {y: 1*I, x: -1*I},
 {y: 1*I, x: 1*I}]
```

[24]:
```
implicit_plot(f,(-1,1),(-1,1))
```

[24]:



1.4.2 2.

[25]:
```
R3.<x,y,z>=PolynomialRing(QQ,'x,y,z',order='lex')
```

[26]:
```
f=x**2+y**2+z**2+x**2*z-y**2*z-1
```

[27]:
```
fx=f.derivative(x)
fy=f.derivative(y)
fz=f.derivative(z)
```

[28]:
```
I=R3.ideal(f,fx,fy,fz)
I.variety(QQbar)
```

[28]:
```
[{z: 1, y: -1.414213562373095?, x: 0},
 {z: 1, y: 1.414213562373095?, x: 0},
 {z: -1, y: 0, x: -1.414213562373095?},
 {z: -1, y: 0, x: 1.414213562373095?}]
```

[29]:
```
implicit_plot3d(f,(-5,5),(-5,5),(-5,5),color='blue')
```

[29]: Graphics3d Object

Cette fois on observe bien les 4 singularités prévues de la surgace…

1.4.3 3.

[30]:
```
R1.<t,x,y>=PolynomialRing(QQ,'t,x,y',order='deglex')
G1=x*(1+t**4)-(t**3-1)
G2=y*(1+t**4)-(t**2-1)
f=G1.resultant(G2,t)
print(f)
```

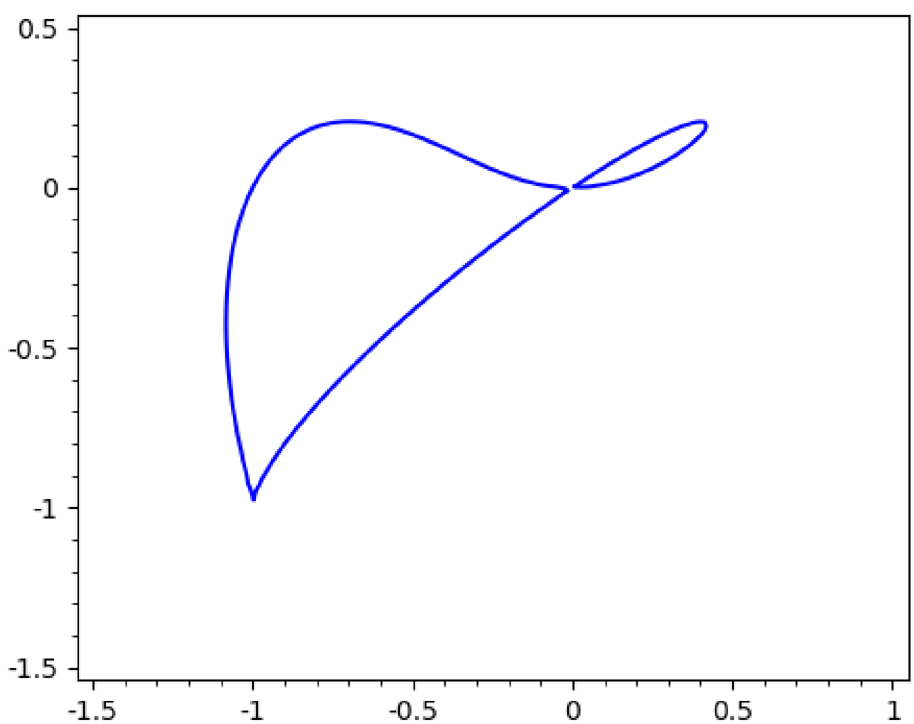
```
4*x^4 - 8*x^3*y + 4*x^2*y^2 + 2*y^4 + 4*x^3 - 10*x^2*y + 4*x*y^2 + 6*y^3 - 4*x*y
+ 6*y^2
```

[31]:
```
R2.<x,y>=PolynomialRing(QQ,'x,y',order='lex')
f=R2(f)
fx=f.derivative(x)
fy=f.derivative(y)
I=R2.ideal(f,fx,fy)
I.variety(QQbar)
```

[31]:
```
[{y: 0, x: 0},
 {y: -1, x: -1},
 {y: -1.333333333333334?, x: -0.6666666666666667?}]
```

[32]:
```
implicit_plot(f,(-1.5,1),(-1.5,0.5))
```

[32]:



Très étrange : le point (y=-1.33, x=-0.66) est censé se trouver sur la courbe, or on ne le voit pas sur le dessin. Un problème d'implicit_plot et de parametric_plot pour les courbes singulières ?

[ ]:

```

```