sage -n jupyter

# 1 Premiers contacts

## 1.1

L'opération (M,v) :-> Mv est de complexité $n^2$ (n multiplications par lignes), et (M,N) :-> MN est de complexité $n^3$ (on fait l'opération précédente sur les n colonnes de N).

In [3]:

```
def mult_vec_dense(M,v):
    return vector([sum(M[i,j]*v[j] for j in range(v.length())) for i in range(M.nrows
())])
```

In [4]:

```
myM = random_matrix(ZZ,4,3); myv = vector(random_matrix(ZZ,1,3));myM,myv
```

Out[4]:

```
(
[  0 526  -1]
[-89   2 -40]
[  1  -2  -1]
[  2  -8   0], (1, -1, 2)
)
```

In [5]:

```
myM*myv, mult_vec_dense(myM,myv)
```

Out[5]:

```
((-528, -171, 1, 10), (-528, -171, 1, 10))
```

In [6]:

```
def mult_mat_dense(M,N):
    return matrix([mult_vec_dense(M,c) for c in N.columns()]).transpose()
```

In [7]:

```
myN = random_matrix(ZZ,3,5); myN
```

Out[7]:

```
[ 0 -1  0 -3  7]
[-1  1 -1 -1  5]
[ 2  2 -1  1  1]
```

In [8]:

```
myM*myN, mult_mat_dense(myM,myN)
```

Out[8]:

```
(
[-528  524 -525 -527 2629]  [-528  524 -525 -527 2629]
[ -82   11   38  225 -653]  [ -82   11   38  225 -653]
[   0   -5    3   -2   -4]  [   0   -5    3   -2   -4]
[   8  -10    8    2  -26], [   8  -10    8    2  -26]
)
```

# 1.2

La densité dans Sage est par rapport au nombre de coefficients dans chaque ligne et chaque colonne.

In [9]:

```
matrix.random(ZZ,10,10,density=.2)
```

Out[9]:

```
[  0   0   0   0   1   0   0   0   0   0]
[  0  -1   0   0   0   0   0  -2   0   0]
[  0   0   0   0   0  -1   0   0   3   0]
[-10   0   0   0   0   0   0   0  -5   0]
[  1  -2   0   0   0   0   0   0   0   0]
[  0   1   0   0  12   0   0   0   0   0]
[  0   0   2   0   0   0   0   0   0  -1]
[  0   0   0  -1   0   0 -22   0   0   0]
[  0   0   0   0   0   0   0  -1   0  -1]
[  0   0   0   0   0   2  -1   0   0   0]
```

In [10]:

```
matrix.random(GF(9),10,10,density=.1)
```

Out[10]:

```
[       0        0        0        0        0        0        0        0
0     2*z2]
[       0        0        0        0        0        0        0        0
0        0]
[       0        0   z2 + 1        0        0        0        0        0
0        0]
[       0        0        0        0        0        0        0        0
0        0]
[       0        0        0        0        0        0        2        0
0        0]
[       0        0        0        0        0        0        0        0
0        0]
[       0        0        0        0        0        0   z2 + 2        0
0        0]
[       0        0   z2 + 2        0        0        0        0        0
0        0]
[      z2        0 2*z2 + 1        0        0        0        0       z2
0        0]
[       0        0        0 2*z2 + 1        0        0        2        0
0        0]
```

## 1.3

v :-> Mv est de complexité φ(n)n (φ(n) sur chaque ligne) et N :-> MN est donc de complexité φ(n)n² (on répète l'opération précédente pour chaque colonne de N)

## 1.4

In [11]:

```
def gen_generic(A,n,lignes,colonnes,coeffs):
    return [A, n, vector(ZZ,lignes), vector(ZZ,colonnes), vector(A, coeffs)]
```

In [12]:

```
gen_generic(ZZ,6,[0,1],[1,4],[-2,1/2])
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/structure/sequenc
e.py in __init__(self, x, universe, check, immutable, cr, cr_str, use_sage
_types)
    444                 try:
--> 445                     x[i] = universe(x[i])
    446                 except TypeError:

/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/structure/parent.
pyx in sage.structure.parent.Parent.__call__ (build/cythonized/sage/struct
ure/parent.c:9337)()
    899             if no_extra_args:
--> 900                 return mor._call_(x)
    901             else:

/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/rings/rational.py
x in sage.rings.rational.Q_to_Z._call_ (build/cythonized/sage/rings/ration
al.c:32481)()
   4158         if not mpz_cmp_si(mpq_denref((<Rational>x).value), 1) == 0
:
-> 4159             raise TypeError("no conversion of this rational to int
eger")
   4160         cdef Integer n = Integer.__new__(Integer)

TypeError: no conversion of this rational to integer

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call las
t)
<ipython-input-12-9374c8844e15> in <module>
----> 1 gen_generic(ZZ,Integer(6),[Integer(0),Integer(1)],[Integer(1),Inte
ger(4)],[-Integer(2),Integer(1)/Integer(2)])

<ipython-input-11-953d083bb2ba> in gen_generic(A, n, lignes, colonnes, coe
ffs)
      1 def gen_generic(A,n,lignes,colonnes,coeffs):
----> 2     return [A, n, vector(ZZ,lignes), vector(ZZ,colonnes), vector(A
, coeffs)]

/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/modules/free_modu
le_element.pyx in sage.modules.free_module_element.vector (build/cythonize
d/sage/modules/free_module_element.c:6359)()
    558             sparse = False
    559
--> 560     v, R = prepare(v, R, degree)
    561
    562     if sparse:

/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/modules/free_modu
le_element.pyx in sage.modules.free_module_element.prepare (build/cythoniz
ed/sage/modules/free_module_element.c:6971)()
    661         except TypeError:
    662             pass
--> 663     v = Sequence(v, universe=R, use_sage_types=True)
    664     ring = v.universe()
    665     if not is_Ring(ring):
```

```
/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/structure/sequenc
e.py in Sequence(x, universe, check, immutable, cr, cr_str, use_sage_type
s)
    259         return PolynomialSequence(x, universe, immutable=immutable
, cr=cr, cr_str=cr_str)
    260     else:
--> 261         return Sequence_generic(x, universe, check, immutable, cr,
cr_str, use_sage_types)
    262
    263

/opt/sagemath-9.2/local/lib/python3.7/site-packages/sage/structure/sequenc
e.py in __init__(self, x, universe, check, immutable, cr, cr_str, use_sage
_types)
    446                 except TypeError:
    447                     raise TypeError("unable to convert {} to an el
ement of {}"
--> 448                                     .format(x[i], universe))
    449         list.__init__(self, x)
    450         self._is_immutable = immutable

TypeError: unable to convert 1/2 to an element of Integer Ring
```

In [13]:

```
gen_generic(ZZ,6,[0,1],[1,4],[-2,3])
```

Out[13]:

```
[Integer Ring, 6, (0, 1), (1, 4), (-2, 3)]
```

On définit maintenant les accesseurs (utiles car on n'a pas à tout remodifier ensuite si l'on modifie l'implantation)

In [14]:

```
def get_ring(M):
    return M[0]
```

In [15]:

```
def get_size(M):
    return M[1]
```

In [16]:

```
def get_rows(M):
    return M[2]
```

In [17]:

```
def get_ncoefs(M):
    return matc_get_rows(M).length()
```

In [18]:

```
def get_cols(M):
    return M[3]
```

In [19]:

```
def get_coefs(M):
    return M[4]
```

On passe maintenant aux fonctions usuelles

In [20]:

```
def to_dense(M): #version pleine d'une matrice creuse (utile pour l'impression)
    N = matrix(get_ring(M),get_size(M))
    myr = get_rows(M)
    myc = get_cols(M)
    mycoefs = get_coefs(M)
    for i in range(len(myr)):
        N[myr[i],myc[i]] = mycoefs[i]
    return N
```

In [21]:

```
def gen_zero(A,n):
    return gen_generic(A,n,[],[],[])
```

In [22]:

```
gen_zero(ZZ,3)
```

Out[22]:

```
[Integer Ring, 3, (), (), ()]
```

In [23]:

```
to_dense(gen_zero(ZZ,3))
```

Out[23]:

```
[0 0 0]
[0 0 0]
[0 0 0]
```

In [24]:

```
def gen_id(A,n):
    return gen_generic(A,n,range(n),range(n),[1 for i in range(n)])
```

In [25]:

```
gen_id(ZZ,5)
```

Out[25]:

```
[Integer Ring, 5, (0, 1, 2, 3, 4), (0, 1, 2, 3, 4), (1, 1, 1, 1, 1)]
```

In [26]:

```
to_dense(gen_id(ZZ,6))
```

Out[26]:

```
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
```

In [27]:

```
gen_id(Integers(4),6)
```

Out[27]:

```
[Ring of integers modulo 4,
 6,
 (0, 1, 2, 3, 4, 5),
 (0, 1, 2, 3, 4, 5),
 (1, 1, 1, 1, 1, 1)]
```

In [28]:

```
to_dense(gen_id(Integers(4),6))
```

Out[28]:

```
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
```

In [29]:

```
def mult_vec(M,v):
    # on suppose que les anneaux de définition sont compatibles
    # le résultat n'est pas creux
    myr = get_rows(M)
    myc = get_cols(M)
    mycoefs = get_coefs(M)
    res = [0 for i in range(v.length())]
    for i in range(myr.length()):
        res[myr[i]] += mycoefs[i] * v[myc[i]]
    return vector(res)
```

In [30]:

```
mult_vec(gen_id(ZZ,5),vector(QQ,[1,4/3,6,9,1]))
```

Out[30]:

```
(1, 4/3, 6, 9, 1)
```

In [31]:

```
def mult_mat(M,N):
    # M est creuse, pas N
    return matrix([mult_vec(M,c) for c in N.columns()]).transpose()
```

In [32]:

```
def gen_random(A, n, phi):
    # génère une matrix aléatoire de taille n avec phi(n) = n*phi
    mesrows = []
    mescols = []
    mescoefs = []
    for i in range(n):
        temp = {randint(0,n-1) for j in [1..floor(n*phi)]}
        for j in temp:
            mesrows.append(i)
            mescols.append(j)
            mescoefs.append(A.random_element())
    return gen_generic(A,n,mesrows,mescols,mescoefs)
```

In [33]:

```
myM = gen_random(ZZ,10,.2); to_dense(myM)
```

Out[33]:

```
[  0   0   0   0   0  -1   0   0  -3   0]
[  0   0   0   0  -4   0   0   0   1   0]
[  0   0   1   0   0   0   0   0   0   0]
[  0   0   1   0   0   7   0   0   0   0]
[  0   0   0   0   0 105   0   0   0   0]
[  0   4  12   0   0   0   0   0   0   0]
[  0   0   0  -1   0   0   0   0   0  -1]
[  0   0   0   0   0   0  -3  -1   0   0]
[  2   0   0   0   0   0   0   0   0   0]
[  0   4   0   1   0   0   0   0   0   0]
```

In [34]:

```
myN = random_matrix(ZZ,10); myN
```

Out[34]:

```
[ -2   0   1   1   1   6   1   6  -1   1]
[ -2   1  -1   2   0   0  -1   1   2  -1]
[  2   1  12   0  -1  -1   3  49   0   3]
[ -1 347  -3   0  -1   9   0 -17   4   1]
[  0  -6   0   1   0   2  -4   0   4  -2]
[ -1   2   1  -1   3  -1   1   3   0   2]
[  1   2   2   0   1  -1   0  -1  -1   0]
[  2  -1   1   1   1   1   0 -70  10  -1]
[  1   0  -1  -1   0   6  -1   1  -2  -1]
[  0   1  -2  -3  13  -2  -1   2   1   1]
```

In [35]:

```
mult_mat(myM,myN)
```

Out[35]:

```
[  -2   -2    2    4   -3  -17    2   -6    6    1]
[   1   24   -1   -5    0   -2   15    1  -18    7]
[   2    1   12    0   -1   -1    3   49    0    3]
[  -5   15   19   -7   20   -8   10   70    0   17]
[-105  210  105 -105  315 -105  105  315    0  210]
[  16   16  140    8  -12  -12   32  592    8   32]
[   1 -348    5    3  -12   -7    1   15   -5   -2]
[  -5   -5   -7   -1   -4    2    0   73   -7    1]
[  -4    0    2    2    2   12    2   12   -2    2]
[  -9  351   -7    8   -1    9   -4  -13   12   -3]
```

In [36]:

```
to_dense(myM) * myN
```

Out[36]:

```
[  -2   -2    2    4   -3  -17    2   -6    6    1]
[   1   24   -1   -5    0   -2   15    1  -18    7]
[   2    1   12    0   -1   -1    3   49    0    3]
[  -5   15   19   -7   20   -8   10   70    0   17]
[-105  210  105 -105  315 -105  105  315    0  210]
[  16   16  140    8  -12  -12   32  592    8   32]
[   1 -348    5    3  -12   -7    1   15   -5   -2]
[  -5   -5   -7   -1   -4    2    0   73   -7    1]
[  -4    0    2    2    2   12    2   12   -2    2]
[  -9  351   -7    8   -1    9   -4  -13   12   -3]
```

# 1.5

In [37]:

```
M = gen_random(ZZ,200,.01); M
```

Out[37]:

```
[Integer Ring,
 200,
 (0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11,
11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 2
0, 21, 21, 22, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29,
30, 30, 31, 31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 37, 38, 38, 3
9, 39, 40, 40, 41, 41, 42, 42, 43, 43, 44, 44, 45, 45, 46, 46, 47, 47, 48,
48, 49, 49, 50, 50, 51, 51, 52, 52, 53, 53, 54, 54, 55, 55, 56, 56, 57, 5
7, 58, 58, 59, 59, 60, 60, 61, 61, 62, 62, 63, 63, 64, 64, 65, 65, 66, 66,
67, 67, 68, 68, 69, 69, 70, 70, 71, 71, 72, 72, 73, 73, 74, 74, 75, 75, 7
6, 76, 77, 77, 78, 78, 79, 79, 80, 80, 81, 81, 82, 82, 83, 83, 84, 84, 85,
85, 86, 86, 87, 87, 88, 88, 89, 89, 90, 90, 91, 91, 92, 92, 93, 93, 94, 9
4, 95, 95, 96, 96, 97, 97, 98, 98, 99, 99, 100, 100, 101, 101, 102, 102, 1
03, 103, 104, 104, 105, 105, 106, 106, 107, 107, 108, 108, 109, 109, 110,
110, 111, 111, 112, 112, 113, 113, 114, 114, 115, 115, 116, 116, 117, 117,
118, 118, 119, 119, 120, 121, 121, 122, 122, 123, 123, 124, 124, 125, 125,
126, 126, 127, 127, 128, 128, 129, 129, 130, 130, 131, 131, 132, 132, 133,
133, 134, 134, 135, 135, 136, 136, 137, 137, 138, 138, 139, 139, 140, 140,
141, 141, 142, 142, 143, 143, 144, 144, 145, 145, 146, 146, 147, 147, 148,
148, 149, 149, 150, 150, 151, 151, 152, 152, 153, 153, 154, 154, 155, 155,
156, 156, 157, 157, 158, 158, 159, 159, 160, 160, 161, 161, 162, 162, 163,
163, 164, 164, 165, 165, 166, 166, 167, 167, 168, 168, 169, 169, 170, 170,
171, 171, 172, 172, 173, 173, 174, 174, 175, 175, 176, 176, 177, 177, 178,
178, 179, 179, 180, 180, 181, 181, 182, 182, 183, 183, 184, 184, 185, 185,
186, 186, 187, 187, 188, 188, 189, 189, 190, 190, 191, 191, 192, 192, 193,
193, 194, 194, 195, 195, 196, 196, 197, 197, 198, 198, 199, 199),
 (58, 28, 106, 182, 184, 34, 73, 41, 112, 153, 72, 37, 33, 42, 132, 7, 12
4, 6, 40, 153, 40, 121, 170, 101, 160, 195, 112, 17, 0, 138, 147, 159, 19
6, 7, 96, 20, 139, 29, 56, 46, 157, 30, 157, 190, 168, 77, 8, 33, 138, 13
1, 147, 100, 65, 74, 53, 165, 166, 190, 184, 52, 147, 37, 185, 11, 172, 19
7, 80, 124, 162, 38, 116, 149, 144, 182, 127, 79, 157, 119, 49, 134, 96, 4
2, 184, 151, 57, 164, 122, 5, 88, 33, 152, 13, 40, 194, 67, 30, 56, 87, 6
8, 174, 116, 173, 32, 100, 19, 198, 64, 61, 72, 171, 139, 78, 109, 126, 18
1, 110, 90, 55, 98, 84, 13, 15, 197, 159, 21, 141, 8, 36, 9, 29, 94, 151,
158, 86, 74, 108, 119, 71, 12, 39, 27, 62, 77, 94, 153, 130, 0, 30, 93, 16
6, 124, 133, 107, 159, 112, 118, 41, 13, 32, 67, 74, 103, 82, 12, 80, 2, 9
6, 50, 105, 138, 74, 42, 160, 187, 181, 159, 82, 157, 154, 60, 44, 70, 17
1, 94, 152, 16, 188, 61, 90, 82, 133, 15, 185, 53, 46, 78, 172, 20, 75, 8
5, 62, 14, 82, 123, 50, 93, 89, 95, 185, 2, 187, 38, 160, 116, 40, 122, 3
7, 39, 66, 27, 11, 87, 8, 47, 136, 61, 160, 33, 153, 35, 25, 131, 73, 141,
119, 47, 99, 103, 0, 194, 31, 58, 59, 160, 89, 80, 157, 13, 166, 48, 5, 10
4, 118, 40, 44, 132, 70, 120, 153, 25, 37, 41, 162, 1, 5, 96, 2, 17, 133,
170, 68, 141, 167, 160, 103, 83, 164, 148, 6, 48, 87, 53, 175, 46, 31, 56,
19, 17, 154, 42, 69, 153, 187, 100, 28, 72, 4, 159, 7, 45, 133, 191, 47,
3, 51, 187, 70, 100, 135, 61, 78, 96, 140, 16, 37, 99, 165, 169, 4, 106, 1
01, 194, 150, 171, 175, 136, 99, 185, 22, 194, 68, 58, 130, 27, 110, 18, 1
41, 185, 52, 1, 134, 44, 6, 170, 69, 184, 1, 195, 190, 145, 38, 184, 119,
180, 78, 10, 173, 156, 62, 122, 76, 196, 68, 8, 10, 10, 191, 187, 197, 25,
153, 1, 135, 129, 149, 0, 149, 180, 142, 163, 183, 48, 194, 36, 5, 75, 18
1, 136, 127, 73, 156, 51, 140, 119, 95, 61, 54, 168, 119),
 (0, -2, 0, 2, 0, 0, 12, 10, -3, -20, 85, -1, -2, 1, 1, 1, -62, 7, -1, 1,
18, -2, -1, -1, -13, -12, 7, -2, 1, -2, 0, -2, -5, 3, -3, -1, 0, 5, -7, -
9, -1, 0, -2, 1, 0, 0, -327, 0, -2, 3, -1, 19, 3, -1, -4, 0, 0, -3, -1, -
1, 0, 0, -1, 0, 1, 1, 10, -1, 9, 0, 0, -21, 2, -10, 0, 0, 2, -1, 0, -1, -
4, 3, -15, 2, -3, 7, -1, -1, 0, 2, -1, -5, 2, 0, -3, 0, -2, 0, 39, -5, -5,
2, -6, 0, 0, -1, -1, 1, -1, 1, 0, -1, 0, 21, 4, 1, -1, -4, 1, 1, 1, -3, 0,
2, 2, -5, -1, -7, 1, 0, 2, -20, 0, 2, -2, 0, 3, -2, 0, 0, 5, 0, 1, -1, -1,
2, -1, 0, -1, -1, 230, 0, 0, -5, 1, 1, 0, -1, -1, -4, -1, 1, 1, 2, 9, -1,
0, -1, 0, -2, 0, 1, 0, 1, 0, 6, 1, -2, 8, -5, 2, 0, -5, -1, 4, -1, -1, 1,
```

```
1, 0, 0, 0, -1, 1, 0, 0, -4, -4, 10, -1, 1, -3, 0, -1, 0, -7, -1, 0, -2, -
1, 3, 0, -1, 0, 1, -1, 1, 2, 1, 0, 1, -1, 3, 1, -3, 0, 5, 0, 1, -1, -1, -
1, -2, 4, 1, 1, -1, -1, 1, -3, 1, -7, 3, 0, 1, -143, -7, 1, -2, 0, -1, -1,
37, 3, -1, -8, 2, -2, -3, 2, 2, 0, -1, -11, -4, 8, 0, 0, 0, 9, -1, -1, -1,
2, 2, 0, -3, -1, 7, -1, 6, 1, 1, -23, 0, -2, -2, 5, -1, 1, -1, 1, 2, -1, -
1, 0, 2, -3, 3, -2, 0, 4, 2, 5, -1, -1, 0, -1, 72, -2, 1, -10, -2, -3, 0,
2, 1, -1, 4, 1, -5, 0, -2, 1, 12, 1, 8, 0, -3, 1, 14, 1, 1, -1, 3, -3, 1,
4, 0, 1, 1, 1, -5, 0, -5, -1, 0, 4, -1, 21, 19, -1, 0, -1, -1, -1, 3, 0, -
1, 0, 2, -2, -1, -1, 1, 0, -1, 1, 1, -1, -1, -1, -1, 1, 2, -5, 5, 0, -6,
1, 0, 5, 0, 0, 1, -2, 0, 1, 0, 1, 0, 0, -1, 0, 2, 0, 1, -3, -2)]
```

In [38]:

```
Mdense = to_dense(M); N = random_matrix(ZZ,200)
```

In [39]:

```
time(mult_mat(M,N))
```

```
CPU times: user 391 ms, sys: 31 ms, total: 422 ms
Wall time: 454 ms
```

Out[39]:

```
200 x 200 dense matrix over Integer Ring (use the '.str()' method to see t
he entries)
```

In [40]:

```
time(mult_mat_dense(Mdense,N))
```

```
CPU times: user 7 s, sys: 31 ms, total: 7.03 s
Wall time: 7.2 s
```

Out[40]:

```
200 x 200 dense matrix over Integer Ring (use the '.str()' method to see t
he entries)
```

# 1.6

Il faut faire via la méthode naïve, pour garder le caractère creux

In [41]:

```
def puiss(M,n): # M creuse
    if n == 0:
        return identity_matrix(get_size(M))
    return mult_mat(M,puiss(M,n-1))
```

In [42]:

```
def expo_rap(M,n): #M pleine
    if n == 0:
        return identity_matrix(M.nrows())
    N = expo_rap(M,n//2)
    res = mult_mat_dense(N,N)
    if n%2 == 0:
        return res
    return mult_mat_dense(res,M)
```

In [43]:

```
M = gen_random(ZZ,200,.01); M
```

Out[43]:

```
[Integer Ring,
 200,
 (0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11,
11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 2
0, 21, 21, 22, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29,
30, 30, 31, 31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 37, 38, 39, 3
9, 40, 40, 41, 41, 42, 42, 43, 43, 44, 44, 45, 45, 46, 46, 47, 47, 48, 48,
49, 49, 50, 50, 51, 51, 52, 52, 53, 53, 54, 54, 55, 55, 56, 56, 57, 57, 5
8, 58, 59, 59, 60, 60, 61, 61, 62, 62, 63, 63, 64, 64, 65, 65, 66, 66, 67,
67, 68, 68, 69, 69, 70, 70, 71, 71, 72, 72, 73, 73, 74, 74, 75, 75, 76, 7
6, 77, 77, 78, 78, 79, 79, 80, 80, 81, 81, 82, 82, 83, 83, 84, 84, 85, 85,
86, 86, 87, 87, 88, 88, 89, 89, 90, 90, 91, 91, 92, 92, 93, 93, 94, 94, 9
5, 95, 96, 96, 97, 97, 98, 98, 99, 99, 100, 100, 101, 101, 102, 102, 103,
103, 104, 104, 105, 105, 106, 106, 107, 107, 108, 108, 109, 109, 110, 110,
111, 111, 112, 112, 113, 113, 114, 114, 115, 115, 116, 116, 117, 117, 118,
118, 119, 119, 120, 120, 121, 121, 122, 122, 123, 123, 124, 124, 125, 125,
126, 126, 127, 127, 128, 128, 129, 129, 130, 130, 131, 131, 132, 132, 133,
133, 134, 134, 135, 135, 136, 136, 137, 137, 138, 138, 139, 139, 140, 140,
141, 141, 142, 142, 143, 143, 144, 144, 145, 145, 146, 146, 147, 147, 148,
148, 149, 149, 150, 150, 151, 151, 152, 152, 153, 153, 154, 154, 155, 155,
156, 156, 157, 157, 158, 158, 159, 159, 160, 160, 161, 161, 162, 162, 163,
163, 164, 164, 165, 165, 166, 166, 167, 167, 168, 168, 169, 169, 170, 170,
171, 171, 172, 172, 173, 173, 174, 174, 175, 175, 176, 176, 177, 177, 178,
178, 179, 179, 180, 180, 181, 181, 182, 182, 183, 183, 184, 184, 185, 185,
186, 186, 187, 187, 188, 188, 189, 189, 190, 190, 191, 191, 192, 192, 193,
193, 194, 194, 195, 195, 196, 196, 197, 197, 198, 198, 199, 199),
 (147, 119, 193, 114, 25, 101, 138, 151, 82, 173, 178, 159, 113, 28, 16, 1
1, 57, 90, 129, 20, 24, 195, 89, 142, 76, 39, 33, 187, 8, 140, 155, 166, 1
23, 139, 136, 190, 30, 119, 24, 67, 170, 46, 172, 196, 184, 11, 130, 21, 1
63, 31, 56, 112, 96, 196, 89, 173, 107, 172, 33, 109, 64, 61, 33, 98, 176,
3, 49, 179, 153, 138, 113, 118, 93, 142, 0, 125, 106, 43, 111, 56, 109, 7
2, 15, 60, 62, 118, 127, 105, 57, 68, 103, 162, 18, 3, 156, 113, 181, 169,
190, 188, 199, 195, 15, 27, 167, 80, 54, 192, 129, 158, 190, 150, 7, 58, 1
59, 129, 157, 48, 105, 105, 177, 57, 58, 66, 3, 196, 197, 41, 131, 89, 18
5, 130, 131, 19, 173, 132, 173, 194, 74, 168, 179, 13, 134, 73, 118, 146,
197, 50, 14, 144, 32, 154, 143, 0, 2, 37, 159, 17, 111, 19, 151, 20, 134,
115, 23, 184, 129, 168, 9, 185, 78, 41, 75, 82, 42, 160, 125, 176, 67, 43,
62, 67, 126, 20, 31, 53, 175, 0, 66, 148, 15, 73, 154, 81, 119, 16, 37, 18
6, 106, 181, 15, 106, 44, 178, 21, 56, 13, 177, 76, 26, 28, 32, 129, 40, 4
5, 92, 79, 113, 71, 128, 34, 69, 127, 21, 22, 155, 164, 173, 102, 99, 166,
96, 149, 112, 150, 148, 119, 145, 3, 26, 123, 160, 126, 123, 140, 17, 44,
88, 150, 70, 175, 91, 183, 80, 141, 121, 97, 182, 191, 57, 115, 124, 135,
32, 71, 122, 98, 26, 196, 40, 113, 24, 75, 180, 38, 192, 2, 88, 174, 2, 6
7, 16, 29, 41, 71, 197, 22, 74, 86, 153, 53, 89, 6, 192, 147, 72, 96, 112,
67, 114, 189, 163, 109, 130, 31, 24, 199, 69, 63, 155, 196, 130, 118, 34,
18, 138, 163, 40, 49, 0, 76, 4, 94, 76, 165, 136, 99, 1, 2, 115, 150, 190,
30, 72, 151, 100, 12, 173, 46, 21, 37, 160, 89, 179, 5, 117, 111, 139, 18
3, 122, 141, 155, 190, 195, 20, 27, 45, 64, 52, 178, 71, 12, 15, 74, 119,
77, 13, 169, 39, 89, 44, 16, 45, 66, 119, 170, 125, 91, 55, 174, 86, 82, 4
6, 184, 74, 155, 62, 64, 41, 132, 95, 116, 15, 0, 185, 146, 13, 163, 135),
 (-1, 3, -5, 1, 6, 0, 9, 3, -14, 0, -1, -3, 0, 0, -1, 2, -5, 1, 1, -5, 1,
2, 0, -16, 1, -1, 1, 38, 3, 0, 4, -1, 6, 6, 3, 0, 1, 0, -1, 5, -11, 2, 0,
-3, 0, -1, 0, 1, -31, -12, 1, 0, 1, 6, -6, -7, 0, -1, 0, 1, 2, 1, 10, 1, -
114, -1, 0, -4, -7, -1, -1, -61, 0, 1, 2, -7, 4, -5, 12, 1, -1, -1, 1, 16,
0, -2, -2, 1, 6, 0, 0, 3, 0, -3, -1, 1, 1, 0, 1, 9, 0, 2, 3, 0, 1, -2, -1,
5, 1, -13, 7, 1, -1, 0, 3, 0, 6, 0, -9, 1, 4, -1, 0, 0, 0, -2, 1, -3, 4, -
1, 0, 0, -2, -1, 2, 0, -1, 3, -1, 1, -1, 0, 1, 0, 0, 0, 1, -1, 0, -1, 16,
0, -18, -1, 0, 3, -5, 1, 1, -1, 0, 0, -6, 370, -2, 0, -1, -1, 3, -3, 1, -1
3, -1, 1, -10, 0, 0, 1, 6, -1, 0, -12, 0, 0, 0, 0, -3, 0, 8, -4, 2, -1, 0,
```

```
-5, 0, -3, 2, -9, 1, 2, 1, -3, 1, -1, -1, -10, 1, -2, -3, 1, 4, -1, -1, 0,
2, 1, 1, 25, 0, -2, 1, -2, 2, -5, -139, -2, 0, 0, 1, -1, 0, 1, 0, -3, 0, -
1, 0, 0, 1, 3, 0, 0, 1, 2, 1, 1, 0, -3, 0, 1, 2, 9, 0, -3, 0, 2, 1, 1, 3,
1, -2, -2, 0, -11, 1, 1, 5, 2, -1, -1, 2, -1, -1, 7, 0, 1, -2, 2, 3, 1, 0,
-23, -1, 1, 0, 5, -68, -1, 1, 0, -2, 1, 0, 1, 1, -1, 1, 0, -3, -1, 0, 0, -
6, 0, 1, 3, 1, -1, -1, -1, 0, -4, 2, 0, -1, -1, 1, 1, -11, 1, 3, 0, 0, 0,
-1, -22, 12, 0, 0, 8, 1, 0, -9, -3, 1, 1, 0, -7, 1, -1, 0, 0, -12, 2, -1,
1, -1, 1, 1, 1, -1, 2, 0, 1, 2, -1, 17, -1, 1, 0, -1, -1, 20, -1, 1, 2, 2,
0, -1, 0, 0, 555, -11, 11, 1, 0, -4, -41, 3, 3, -1, -1, -2, 1, 4, -1, 8, -
3, 6, -1, 26, -6, 1, 1, 0, -12, -1, 1, -1)]
```

In [44]:

```
Mdense = to_dense(M); Mdense
```

Out[44]:

```
200 x 200 dense matrix over Integer Ring (use the '.str()' method to see t
he entries)
```

In [45]:

```
time(puiss(M,4))
```

```
CPU times: user 922 ms, sys: 15 ms, total: 937 ms
Wall time: 983 ms
```

Out[45]:

```
200 x 200 dense matrix over Integer Ring (use the '.str()' method to see t
he entries)
```

In [46]:

```
time(expo_rap(Mdense,4))
```

```
CPU times: user 50.5 s, sys: 750 ms, total: 51.3 s
Wall time: 1min 24s
```

Out[46]:

```
200 x 200 dense matrix over Integer Ring (use the '.str()' method to see t
he entries)
```

# 2 Intermède polynômes

In [47]:

```
K = QQ
```

In [48]:

```
KX.<X> = PolynomialRing(K)
```

## 2.1

In [49]:

```
def division(A, B): # A, B sont dans KX ; renvoie [quotient, reste]
    assert(B != KX.zero())
    if B.degree() > A.degree(): # le degré du polynôme nul est -1
        return [KX.zero(), A]
    Qloc = A.leading_coefficient() / B.leading_coefficient() * X^(A.degree() - B.degree
())
    respart = division(A -  Qloc * B, B)
    return [Qloc + respart[0], respart[1]]
```

In [50]:

```
division(X^2+1,X)
```

Out[50]:

```
[X, 1]
```

In [51]:

```
division(X^2+1,0)
```

```
---------------------------------------------------------------------------
-
AssertionError                            Traceback (most recent call las
t)
<ipython-input-51-4c1609238d56> in <module>
----> 1 division(X**Integer(2)+Integer(1),Integer(0))

<ipython-input-49-867ba3aa44f9> in division(A, B)
      1 def division(A, B): # A, B sont dans KX ; renvoie [quotient, rest
e]
----> 2     assert(B != KX.zero())
      3     if B.degree() > A.degree(): # le degré du polynôme nul est -1
      4         return [KX.zero(), A]
      5     Qloc = A.leading_coefficient() / B.leading_coefficient() * X**
(A.degree() - B.degree())

AssertionError:
```

In [52]:

```
(X^4+1).quo_rem(X^2+X+2), division(X^4+1,X^2+X+2)
```

Out[52]:

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

## 2.2

In [53]:

```
def euclide(A, B): # A, B dans KX non tous nuls ; renvoie le pgcd unitaire de A et B vi
a l'algorithme d'Euclide
    R0 = A
    R1 = B
    while R1 != KX.zero():
        R = R0
        R0 = R1
        R1 = division(R, R1)[1]
    return R0 / R0.leading_coefficient()
```

In [54]:

```
euclide(X^4-1,X^3+X)
```

Out[54]:

```
X^2 + 1
```

In [55]:

```
gcd(X^4-1,X^3+X)
```

Out[55]:

```
X^2 + 1
```

## 2.3

In [56]:

```
def euclide_etendu(A, B):
    R0 = A
    R1 = B
    U0 = KX.one()
    U1 = KX.zero()
    V0 = KX.zero()
    V1 = KX.one()
    # U0 A + V0 B = R0
    # U1 A + V1 B = R1
    while R1.degree() >= 0:
        d = division(R0, R1)
        R0 = R1
        R1 = d[1]
        U = U0
        V = V0
        U0 = U1
        V0 = V1
        U1 = U - U0 * d[0]
        V1 = V - V0 * d[0]
    # R1 est nul et R0 est constant
    return U0/R0.leading_coefficient(),V0/R0.leading_coefficient()
```

In [57]:

```
myA = KX.random_element(degree=6); myA
```

Out[57]:

```
-1/9*X^6 + 2/3*X^5 - X^2 - X - 1/2
```

In [58]:

```
myB = KX.random_element(degree=6); myB
```

Out[58]:

```
-X^6 + 3*X^5 + 1/10*X^4 + 1/4*X^3 - X^2 + 11/10
```

In [59]:

```
euclide(myA,myB),gcd(myA,myB)
```

Out[59]:

```
(1, 1)
```

In [60]:

```
myU,myV = euclide_etendu(myA,myB); myU,myV
```

Out[60]:

```
(-1385248570671600/4233265297132627*X^5 + 3768011217238320/423326529713262
7*X^4 + 452284816359600/604752185304661*X^3 - 7279314790927932/42332652971
32627*X^2 + 5920964334207900/4233265297132627*X - 3100216569431382/4233265
297132627,
 153916507852400/4233265297132627*X^5 - 880417436583680/4233265297132627*X
^4 - 465630260197720/4233265297132627*X^3 + 1473021833603920/4233265297132
627*X^2 - 127031274843120/4233265297132627*X + 2439233647651760/4233265297
132627)
```

In [61]:

```
myA*myU + myB*myV
```

Out[61]:

```
1
```

# 3 Suites récurrentes linéaires

## 3.2

In [62]:

```
def euclide_etendu_restr(A, m):
    # on fait l'algo d'Euclide à A et X^2m, mais on s'arrête dès que le reste est de de
gré < m
    # on aura à priori A = le U de l'énoncé et m = e
    R0 = A
    R1 = X^(2*m)
    U0 = KX.one()
    U1 = KX.zero()
    V0 = KX.zero()
    V1 = KX.one()
    # U0 A + V0 B = R0
    # U1 A + V1 B = R1
    while R1.degree() >= m:
        d = division(R0, R1)
        R0 = R1
        R1 = d[1]
        U = U0
        V = V0
        U0 = U1
        V0 = V1
        U1 = U - U0 * d[0]
        V1 = V - V0 * d[0]
    # le degré de R1 est < m
    return U1 # c'est le P de l'énoncé
```

## 3.3

In [63]:

```
def trouve_pol_ann_suite(u):
    M = len(u)
    e = M // 2 # on veut 2e-1 ≤ M-1
    return euclide_etendu_restr(sum(u[2*e - 1 - i] * X^i for i in range(0, 2*e)), e)
```

On teste pour la suite de Fibonacci : on a d = 2 donc on doit produire au moins 2e = 4 termes.

In [64]:

```
trouve_pol_ann_suite([0,1,1,2])#,3,5,8])
```

Out[64]:

```
-X^2 + X + 1
```

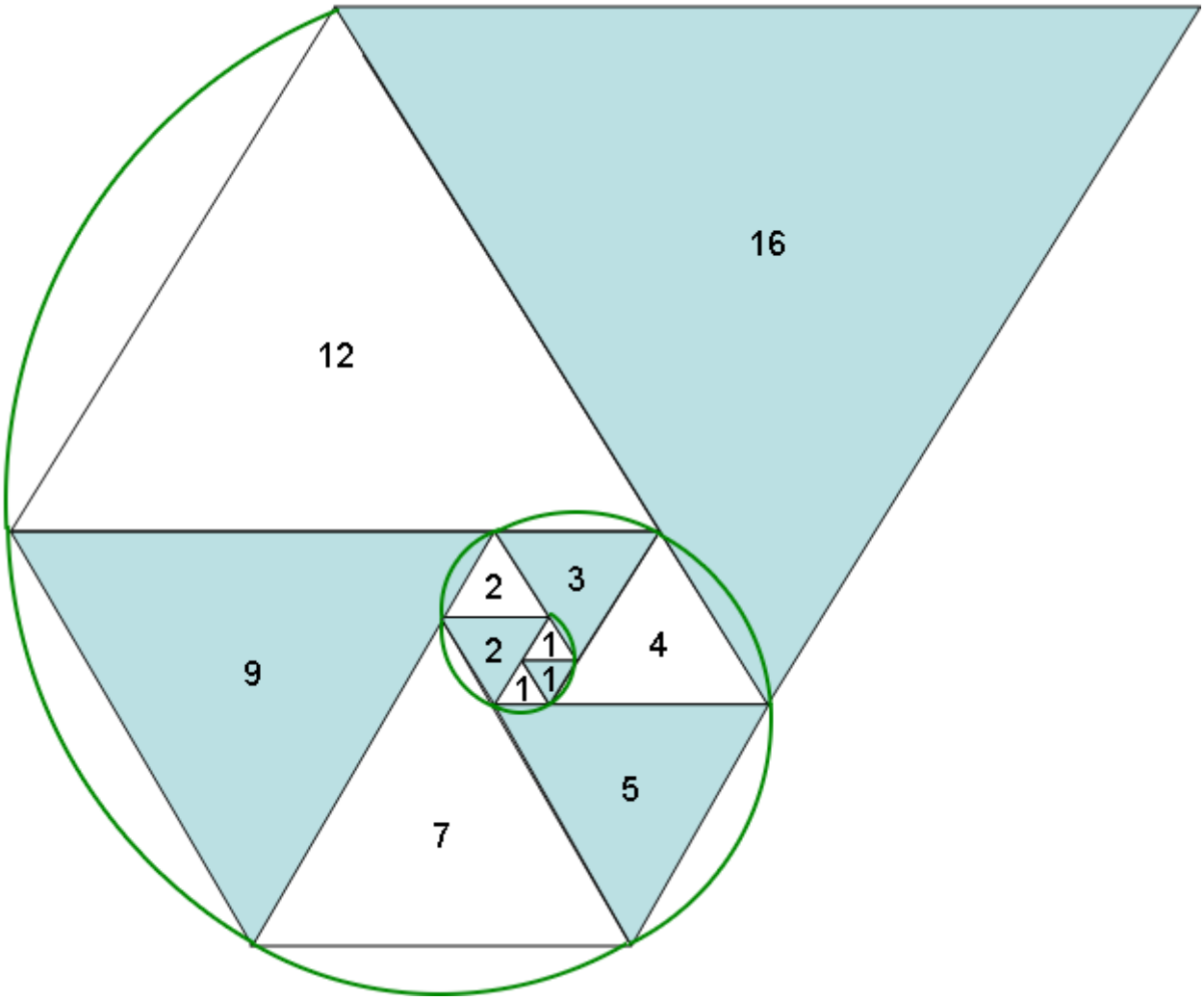
On teste pour la suite de Padovan : on a d = 3 donc on doit produire au moins 2e = 6 termes.

In [65]:

```
trouve_pol_ann_suite([1,1,1,2,2,3])
```

Out[65]:

```
-X^3 + X + 1
```




# 4 Inverse d'une matrice creuse

## 4.2.(a)

Si P(M) = 0 alors l'ensemble des y tels que P(M)y = 0 est un hyperplan, de mesure nulle (car on est sur R ou C). Si pour un y c'est ≠ 0, l'ensemble des x tel que x (P(M)y) = 0 est également un hyperplan donc c'est pareil.

In [66]:

```
def pol_ann_en_inv(P, M):
    #P annulateur de M, renvoie M^-1
    #M est une matrice creuse
    puiss = P.exponents()
    assert puiss[0] == 0 #le polynôme doit posséder un terme constant non nul
    coeff = P.coefficients() #mode creux, comme puiss
    # on doit renvoyer -sum(expo(M, puiss[i] - 1) * coeff[i] for i in range(1,len(coef
f))) / coeff[0], mais de façon intelligeante
    puissM = identity_matrix(get_size(M))
    res = matrix.zero(get_size(M))
    puiss[0] = 1
    for i in range(1, len(puiss)):
        for j in range(puiss[i-1], puiss[i]):
            puissM = mult_mat(M, puissM)
        # puissM est maintenant M^(puiss[i] - 1)
        res = puissM * coeff[i] + res
    return res * (-1 / coeff[0])
```

In [67]:

```
M = gen_random(QQ,10,.6); Mdense=to_dense(M); Mdense,det(Mdense)
```

Out[67]:

```
(
[   0    1    0    0 1/15    1    0    0    2    0]
[ 1/2    0  1/4    0   -2    0    0    0    0    0]
[   0    0 -7/3    0    1    0    0    2    0  -13]
[  -1    0 39/2   -1    0    0    0   -2    1    0]
[ 1/3    0  -11    0   10    0    0    2    0   -1]
[   0    0    0  1/5    0   -2  4/5    0 11/5    0]
[ 1/4    0    0    1    0    0    0    0  7/4    0]
[   0    0    0    0    0    3    0 -1/3    0  1/5]
[   0    1    0    0    0   -1    0    0   -1    0]
[  -2    0    0    0    0    0    0    0    0    3], 4099679/450
)
```

In [68]:

```
pol_ann_en_inv(Mdense.characteristic_polynomial(),M)
```

Out[68]:

```
[  -268785/4099679   3137244/4099679   -340173/4099679    293220/4099679
663258/4099679                  0    293220/4099679    179190/4099679    26
8785/4099679  -1264943/4099679]
[  2631527/8199358  8747384/61495185   323533/40996790     55414/4099679
1046247/40996790                 0     55414/4099679    489384/4099679   5
567831/8199358  2136704/61495185]
[  -377190/4099679   1956324/4099679     66234/4099679    411480/4099679
387156/4099679                  0    411480/4099679    251460/4099679    37
7190/4099679    399302/4099679]
[ -2515410/4099679   1137630/4099679    189324/4099679    135197/4099679
225363/4099679                  0   4234876/4099679   1676940/4099679   251
5410/4099679    783729/4099679]
[  -114345/4099679  -1020988/4099679    -76764/4099679    124740/4099679
214209/4099679                  0    124740/4099679     76230/4099679    11
4345/4099679   -266323/4099679]
[  -320023/8199358 25221164/61495185   919423/40996790    174558/4099679
3281547/40996790                 0    174558/4099679   1473234/4099679
320023/8199358  6143789/61495185]
[ -7659115/8199358 41635877/24598074    692789/8199358  2920967/16398716
2757507/8199358               5/4   -294678/4099679  11938875/8199358   76
59115/8199358  9375989/24598074]
[ -3095235/8199358  16387596/4099679   1382823/8199358   1688310/4099679
6437391/8199358                 0   1688310/4099679   1031745/4099679   30
95235/8199358   4000232/4099679]
[  1475775/4099679  -1098252/4099679    -59589/4099679   -119144/4099679
-223530/4099679                 0   -119144/4099679   -983850/4099679  -14
75775/4099679   -267139/4099679]
[  -179190/4099679   2091496/4099679   -226782/4099679    195480/4099679
442172/4099679                  0    195480/4099679    119460/4099679    17
9190/4099679  1569793/12299037]
```

In [69]:

```
mult_mat(M,_)
```

Out[69]:

```
[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 1]
```

## 4.2.(b)

In [70]:

```
def trouve_pol_ann(M):
    # M est creuse
    # on veut trouver un polynôme annulateur de degré n (la taille de M)
    # ... donc e = n
    e = get_size(M)
    x = vector(QQ.random_element() for i in range(e))
    y = vector(QQ.random_element() for i in range(e))
    My = y
    U = [0 for i in range(2*e)]
    for i in range(2*e):
        U[i] = x * My
        My = mult_vec(M,My)
    return trouve_pol_ann_suite(U)
    # on a bien deg R < e (sinon U serait tout le temps nul)
```

In [71]:

```
def inverse(M):
    return pol_ann_en_inv(trouve_pol_ann(M), M)
```

In [72]:

```
myM = gen_random(ZZ,10,.5); to_dense(myM),det(to_dense(myM))
```

Out[72]:

```
(
[  1  49  -2   0   0   0   0   1   0   0]
[ 21   0   0   0   4   0   0   1   0   3]
[  0   0   0   3   0  -2   0   0  -1   0]
[  0   1   1 -27   4   0   0   0  -1   0]
[  0   0  -1   1   1   0  -9   0   0   0]
[  0   0   0   0   2   0   2   0   0  -1]
[  0   0  -2   0   0   0   0   0   0   0]
[  0   0   0   4   0   0   0  -1   0   0]
[  0  -1   0   0   0   0  -1   0   0   4]
[  1   0   1   0   0   0  -1   2   0   0], -2509584
)
```

In [73]:

```
inverse(myM)
```

Out[73]:

```
[    -127/104566       695/14259              0              0    -1774/
156849     -4801/52283    -2701/627396      887/313698       -127/2134
-2337/104566]
[      305/14938         -1/2037              0              0        2
6/22407          3/7469     -2335/89628       -13/44814          1/2134
-151/14938]
[              0              0              0              0
0               0            -1/2              0              0
0]
[      39/209132       -88/14259              0              0    -1786/
156849       943/52283   88105/1254792   158635/627396         39/4268
27065/209132]
[       69/29876          1/2037              0              0      201
1/22407       3392/7469    -9589/179256     -2011/89628        483/4268
-377/29876]
[     -629/52283       -887/9506            -1/2            1/2    -1834
5/52283   -66707/104566     74411/52283   401295/104566       -191/2134
206155/104566]
[      29/104566         -3/4753              0              0     -535
4/52283      2743/52283    12007/209132     2677/104566         29/2134
1357/104566]
[       39/52283      -352/14259              0              0    -7144/
156849      3772/52283    88105/313698     1786/156849         39/1067
27065/52283]
[    5149/209132        799/4753              0             -1     3490
4/52283     69536/52283 -1102471/418264 -1446545/209132        881/4268  -
743425/209132]
[     541/104566        -4/14259              0              0    -3970/
156849       691/52283     4919/627396     1985/313698        541/2134
75/104566]
```

In [74]:

```
mult_mat(myM,_)
```

Out[74]:

```
[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 1]
```

L'algorithme qui utilise le pivot de Gauss est de complexité $O(n^3)$ (également pour une matrice creuse). Ici pour générer la suite u on calcule successivement My, M(My), $M(M^2y)$,...,M(M^{k-1}y) puis on multiplie ces vecteurs par x. Le calcul M x vecteur a un cout $\varphi(n)n$, on fait ça 2e-1 fois où e est la taille de m donc on obtient $\varphi(n)n^2$. La multiplication par x est est $O(n^2)$ donc ça ne change pas. On doit ensuite trouver un polynôme annulateur à partir de u, là c'est Euclide étendu pour des polynômes de degré 2e (donc de l'ordre de n) donc $O(n^2)$. Pour ensuite évaluer le polynôme annulateur (modifié) en la matrice, on calcule les puissances de la matrice creuse M (donc $\varphi(n)n^2$) jusqu'à la puissance à peu près n, donc $\varphi(n)n^3$, qui donc l'emporte. En fait du coup la complexité est plus grande que le pivot de Gauss, mais si on dit qu'on s'intéresse seulement aux systèmes linéaires (et un seul suffit à inverser M, cf. Mx=y !) alors on a bien $\varphi(n)n^2$ qui est plus petit que les $n^3$ du pivot de Gauss.