Documentation for `case q 3.py`

Francesca Bianchi

September 10, 2015

Abstract

The following documentation aims to give an overview of what different operations can be done with the file `case q 3.py`. These include operations among some special infinite upper triangular matrices, also in the case in which there are some unknown upper diagonals. In particular, the file introduces two different classes, `N` and `NX`, and defines functions which take arguments from both or either of the classes.

A quick reference

This part provides a quick guide on how to use the file. For a more detailed explanation refer to the later sections.
The file allows to do operations with elements of a particular group. Consider

$$G =< x_0, ..., x_{12} | x_i^3 = \mathrm{Id}, x_i x_{i+1} x_{i+4} = \mathrm{Id} >,$$

where Id denotes the identity element and subscripts are taken modulo 13 (see [3]).
There is a faithful representation of G in the group of finite band upper triangular infinite matrices with entries in $M(3, \mathbb{F}_3)$, invertible entries on the main diagonal, and entries on the diagonals with periodicity 3 ($g_{ij} = g_{i+3,j+3}$ for all $i, j \geq 1$ for any g in this group).
Each element in G may thus be identified with an infinite matrix of this type.
The generators are all built-in and can be called by `x0`,..., `x12`.
A diagonal can be described equivalently by a 3×9 matrix with entries in $\mathbb{F}_3$ or by a 3-tuple of non-negative numbers, each less than or equal to 19683. Indeed, if the first 3 entries on an upper diagonal are $a_1, a_2, a_3 \in M(3, \mathbb{F}_3)$, the 3×9 matrix $[a_1, a_2, a_3]$ will describe the diagonal entirely, because of the periodicity 3. Moreover, for $k = 1, 2, 3$, the matrix $((a_k)_{ij})_{1 \leq i,j \leq 3}$ can be represented by the number $A_k = 3^8(a_k)_{11} + 3^7(a_k)_{12} + 3^6(a_k)_{13} + 3^5(a_k)_{21} + 3^4(a_k)_{22} + 3^3(a_k)_{23} + 3^2(a_k)_{31} + 3(a_k)_{32} + (a_k)_{33}$. Therefore, $[A_1, A_2, A_3]$ describes the same upper diagonal as $[a_1, a_2, a_3]$.
For instance,

```
>>> x0
N ([[9613, 9613, 9613], [5859, 5859, 5859], [2970, 2970, 2970], [5859, 5859, 5859],
[2970, 2970, 0], [5859, 0, 0]])
>>> print x0
N([matrix([[1, 1, 1, 1, 1, 1, 1, 1, 1],
        [0, 1, 2, 0, 1, 2, 0, 1, 2],
        [0, 0, 1, 0, 0, 1, 0, 0, 1]]), matrix([[0, 2, 2, 0, 2, 2, 0, 2, 2],
        [0, 0, 1, 0, 0, 1, 0, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]), matrix([[0, 1, 1, 0, 1, 1, 0, 1, 1],
        [0, 0, 2, 0, 0, 2, 0, 0, 2],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]), matrix([[0, 2, 2, 0, 2, 2, 0, 2, 2],
        [0, 0, 1, 0, 0, 1, 0, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]), matrix([[0, 1, 1, 0, 1, 1, 0, 0, 0],
        [0, 0, 2, 0, 0, 2, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]), matrix([[0, 2, 2, 0, 0, 0, 0, 0, 0],
```

```
        [0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]])])
```

means that x_0 is the infinite upper triangular matrix whose diagonal is described by the 3-tuple $[9613, 9613, 9613]$ and whose subsequent five upper diagonals are described, in order, by $[5859, 5859, 5859]$, $[2970, 2970, 2970]$, $[5859, 5859, 5859]$, $[2970, 2970, 0]$, $[5859, 0, 0]$. All other upper diagonals are zero.
We can multiply and take powers. For example, $x_3x_6^{-1}x_5^2$ would be

```
>>> x3*(x6**(-1))*(x5**2)
N ([[11773, 2848, 1192], [4401, 845, 163], [5022, 924, 4631], [5238, 15317, 2417],
[14742, 3006, 920], [9477, 5942, 8389], [5265, 15733, 12955], [9612, 5260, 0], [14904, 0, 0]])
```

We can also work with elements of which only some upper diagonals are known. For instance,

```
>>> x=NX([9613, 9613, 9613])
>>> x
N ([[9613, 9613, 9613]],?)
>>> x.conj(x2)
N ([[9613, 12970, 6931]],?)
>>> x0.comm(x)
N ([[6643, 6643, 6643]],?)
```

Here `x.conj(x2)` and `x0.comm(x)` give $x^{-1}x_2x$ and $x_0^{-1}x^{-1}x_0x$ respectively. Finally, we can truncate elements in the following way:

```
>>> c=x5**2
>>> c
N ([[11773, 9933, 6949], [9693, 13672, 83], [14823, 871, 598], [3699, 6547, 531],
[19062, 6851, 0], [7776, 0, 0]])
>>> c.trunc(2)
N ([[11773, 9933, 6949], [9693, 13672, 83]],?)
```

A more detailed guide

1 Useful procedures

The first part of the code defines some useful operations.

`inversematrix3(A)`: Let $A \in M(3, \mathbb{F}_3)$ be invertible. Then `inversematrix3(A)` returns the inverse of A modulo 3.

`inversebigmatrix3(B)`: Let B be an upper triangular square matrix of arbitrary dimension, whose entries are 3×3 matrices over $\mathbb{F}_3$ and whose diagonal entries are invertible modulo 3. Then `inversebigmatrix3(B)` returns the inverse of B modulo 3.

`transfmn3(M)`: Let $M = (M_{ij})_{1\le i,j\le 3}$ be a 3×3 matrix with entries in $\mathbb{F}_3$. Then `transfmn3(M)` returns the integer $3^8M_{11} + 3^7M_{12} + 3^6M_{13} + 3^5M_{21} + 3^4M_{22} + 3^3M_{23} + 3^2M_{31} + 3M_{32} + M_{33}$.

`transfnm3(n)`: Given an integer $1 \le n \le 19683$, `transfnm3(n)` returns the unique 3×3 matrix M with entries in $\mathbb{F}_3$ such that `transfmn3(M) == n`.

`extract(k,M)`:
Let $M = (M_{ij})_{1\le i\le 3,\ 1\le j\le 9}$ be a 3×9 matrix.
If k is an integer such that $1 \le k \le 3$, `extract(k,M)` returns the 3×3 matrix $(M_{ij})_{1\le i\le 3,\ 3k-2\le j\le 3k}$.
For all other choices of k, the function returns:
`'You entered a value of k out of range:  k must be an integer between 1 and 3'`.

`comb(U1,U2,U3)`:
Given the 3×3 matrices U_1, U_2, U_3, `comb(U1,U2,U3)` returns the unique 3×9 matrix U such that `extract(i,M) == Ui`, for $1 \le i \le 3$.

`numm3(ss)`:
The function `numm3` takes a list ss of three non-negative integers less than or equal to 19683 and returns the 3×9 matrix `comb(transfnm3(ss[0]),transfnm3(ss[1]),transfnm3(ss[2]))`.

`matt3(M)`:
The function `matt3` is the inverse of `numm3`: it takes a 3×9 matrix, reduces it modulo 3, and returns the corresponding 3-tuple of integers.

Example 1.

```
>>> A=matrix([[2,2,1],[0,1,0],[2,1,2]])
>>> Ainv=inversematrix3(A)
>>> Ainv
matrix([[1, 0, 1],
        [0, 1, 0],
        [2, 1, 1]])
>>> transfmn3(A)
18329
>>> transfnm3(_)
matrix([[2, 2, 1],
        [0, 1, 0],
        [2, 1, 2]])
>>> B=matrix([[0, 2, 2, 0, 2, 2, 0, 2, 2],
        [0, 0, 1, 0, 0, 1, 0, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> extract(1,B)
matrix([[0, 2, 2],
        [0, 0, 1],
        [0, 0, 0]])
>>> comb(Ainv,extract(3,B),A)
matrix([[1, 0, 1, 0, 2, 2, 2, 2, 1],
        [0, 1, 0, 0, 0, 1, 0, 1, 0],
        [2, 1, 1, 0, 0, 0, 2, 1, 2]])
>>> numm3([1,271,55])
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 1, 0, 0, 2],
        [0, 0, 1, 0, 0, 1, 0, 0, 1]])
>>> matt3(_)
[1, 271, 55]
```

2 The class N

2.1 Instances of N

An instance g in this class represents an infinite upper triangular matrix with the following properties:

1. Each entry is a 3×3 matrix over $\mathbb{F}_3$;

2. Each diagonal entry is invertible (in particular, each diagonal entry of a generator has order 3);

3. $g_{ij} = g_{i+3,j+3}$ for all $i, j \ge 1$;

4. There exists $n \geq 1$ such that $g_{ij} = \mathbf{0}$ for all i, j with $j - i \geq n$, where $\mathbf{0}$ denotes the 3×3 zero matrix.

Because of Property 3, we may define an upper diagonal by a 3×9 matrix (see [1]).
We define an element g in `N` in the following way. Let $U_1, ..., U_m$ be 3×9 matrices. Then `g=N(U1,...,Um)` defines the element of `N` with the diagonal described by U_1 and $m-1$ upper diagonals described by $U_2, ..., U_m$. All other upper diagonals are zero. The matrices $U_1, ..., U_m$ may be replaced by the corresponding 3-tuples of numbers.

`print g`:
Assume U_m is not the zero matrix. Then the command `print g` gives `N([U1,...,Um])` and we can call $[U_1, ..., U_m]$ by `g.m`.
If $U_i = 0$ for all $i \geq j$ for some $1 < j \leq m$ and U_{j-1} is non zero, then `print g` gives `N([U1,...,Uj-1])`.
Finally, if we enter `g=N([6643, 6643, 6643]))`, `print g` gives `Id`.

`g`:
Just typing `g` in the shell gives the same output as `print g`, but with the 3×9 matrices replaced by 3-tuples of numbers. It is the same as `print g.transfmn()` (see 2.2). Note, however, that while the latter may be used also in the file, the command `g` gives an output only if typed in the shell.

Example 2.

```
>>> x12
N ([[9613, 2662, 10641], [18495, 19372, 4569], [15606, 3760, 3487], [5859, 11631, 5812],
[10746, 10070, 0], [11448, 0, 0]])
>>> g=N(matrix([[1,0,0,1,0,0,1,0,0],[0,1,0,0,1,0,0,1,0],[0,0,1,0,0,1,0,0,1]]))
>>> g
Id
>>> h=N([7393, 5859, 18329],[0,0,0])
>>> print h
N([matrix([[1, 0, 1, 0, 2, 2, 2, 2, 1],
        [0, 1, 0, 0, 0, 1, 0, 1, 0],
        [2, 1, 1, 0, 0, 0, 2, 1, 2]])])
```

2.2 Operations within the class N

`__eq__()` and `__ne__()`:
We can compare instances in `N` in the obvious way.
For g, h in `N`, `g==h` (resp. `g!=h`) returns `True` (resp. `False`) if g and h represent the same matrix and `False` (resp. `True`) otherwise.

`transfmn()`:
Given an instance `g=N(U1,...,Um)` of `N`, the command `g.transfmn()` returns
`N([[n11,n12,n13],...,[nm1,nm2,nm3]])`, where $0 \leq n_{ij} \leq 19683$ represents the matrix j of U_i.

`ext(n,m)`:
Given g in `N` and n, m positive integers, `g.ext(n,m)` is the $n \times m$ matrix obtained from the first n rows and first m columns of the infinite matrix represented by g.

`__mul__()`:
Given g, h in `N`, `g*h` returns $g \cdot h$.

`inv()`:
`g.inv()` returns the inverse of g. If the precision parameter is large enough, the output of `print g.inv()` is an element in `N`; otherwise, it is an element in `NX` (see Section 3).
The precision parameter is set by default to be equal to 1 and can be modified by overwriting the global

variable `precinv` (see below).

`precinv`:
As mentioned above, the global variable `precinv` controls the maximum number of upper diagonals we allow to be computed in the inverse. Suppose g has m non zero diagonals. When we type `g.inv()`, the function `inv()` will find the exact inverse of g if this has at most $\mathrm{precinv} \cdot m$ non zero diagonals and will return the first $\mathrm{precinv} \cdot m$ upper diagonals of g^{-1} otherwise.
If the program is used to do operations only involving the generators, it is recommendable to set `precinv` to 1.

`__pow__()`:
Let n be an integer (possibly zero or negative). Then `g**n` returns g^n. In particular, note that the inverse of g is returned both if we type `g.inv` or `g**(-1)`.

`conj()`:
`g.conj(h)` returns $g^{-1}hg$ (it may be in `NX` if `g**(-1)` is in `NX`).

`comm()`:
`g.comm(h)` returns $g^{-1}h^{-1}gh$ (it may be in `NX` if `g**(-1)` or `h**(-1)` is in `NX`).
`comm()` also computes higher commutators. That is: `g.comm(y0,...,yj)` returns the higher commutator $[g, y_0, ..., yj] = [[..[g, y_0], ..., y_{j-1}], y_j]$.

`commr()`:
`g.commr(y0,...,yj)` returns the higher commutator $[g, y_0, ..., yj] = [g, [y_0, ..., [y_{j-1}, y_j]...]]$.Note that `g.commr(h)` is the same as `g.comm(h)`.

`trunc(n)`:
`g.trunc(n)` returns the element in `NX` (see Section 3), whose diagonal and first $n-1$ upper diagonals agree with the diagonal and first $n-1$ upper diagonals of g.

Example 3.

```
>>> x0**(-1)
N ([[11773, 11773, 11773], [2241, 2241, 2241], [5859, 5859, 5859], [3699, 3699, 3699],
[4401, 4401, 0], [2970, 0, 0]])
>>> g=N(x0.m[0],x0.m[1],x0.m[2])
>>> g**(-1)
N ([[11773, 11773, 11773], [2241, 2241, 2241], [5859, 5859, 5859]],?)
>>> precinv=5
>>> x0**(-1)
N ([[11773, 11773, 11773], [2241, 2241, 2241], [5859, 5859, 5859], [3699, 3699, 3699],
[4401, 4401, 0], [2970, 0, 0]])
>>> g**(-1)
N ([[11773, 11773, 11773], [2241, 2241, 2241], [5859, 5859, 5859], [1458, 1458, 1458],
[1458, 1458, 1458]])
```

Example 4.

```
>>> a=x0.conj(x1)
>>> a
N ([[9613, 10641, 13891], [18198, 6196, 2999], [11637, 2188, 17037], [5103, 4321, 2235],
[10584, 13793, 840], [1053, 1515, 951], [5130, 1491, 16878], [2970, 11721, 2187],
[14607, 4374, 4374], [2187, 2187, 2187], [4374, 4374, 0], [2187, 0, 0]])
>>> a.trunc(6)
N ([[9613, 10641, 13891], [18198, 6196, 2999], [11637, 2188, 17037], [5103, 4321, 2235],
[10584, 13793, 840], [1053, 1515, 951]],?)
```

```
>>> x0.commr(x1,x2)==x0.comm(x1.comm(x2))
True
>>> x0.comm(x1,x2)==(x0.comm(x1)).comm(x2)
True
>>> x0*x1*x4
Id
```

3 The class NX

3.1 Instances of NX

An instance g of `NX` differs from one of `N` only for the fact that we have information about the first say l upper diagonals of g, but we do not know what the other upper diagonals look like.
We define an element g in `NX` in the following way. Let $V_1, ..., V_l$ be 3×9 matrices. Then `g=NX(V1,...,Vl)` defines the element of `NX` with the diagonal described by V_1 and $l-1$ upper diagonals defined, in order, by $V_2, ..., V_l$. Similarly to `N`, the matrices $V_1, ..., V_l$ may be replaced by the corresponding 3-tuples of numbers.

`print g`:
The command `print g` gives `N([V1,...,Vl],?)` and we can call $[V_1, ..., V_l]$ by `g.l`.

`g`:
The difference between `print g` and `g` in `NX` is analogous to the difference between the same commands in `N`.

3.2 Operations within the class NX

Except for `==` and `!=`, all the other functions listed in 2.2 can also be used with arguments belonging to `NX`. The output of `*`, `**`, `inv()` `conj()`, `comm()`, will in this case be an element in `NX` (except for when we raise an element to the power 0, which gives `Id`).

Example 5.

```
>>> x=NX(x0.m[0],x0.m[1])
>>> y=NX(x1.m[0],x1.m[1])
>>> z=NX(x4.m[0],x4.m[1])
>>> x*y*z
N ([[6643, 6643, 6643], [0, 0, 0]],?)
>>> x**(-1)
N ([[11773, 11773, 11773], [2241, 2241, 2241]],?)
>>> x.conj(y)
N ([[9613, 10641, 13891], [18198, 6196, 2999]],?)
>>> res=x.comm(z)
>>> res
N ([[6643, 17650, 10871], [5130, 2948, 5946]],?)
>>> res.trunc(10)
N ([[6643, 17650, 10871], [5130, 2948, 5946]],?)
```

4 Operations among classes and comparison operators

As well as multiplying, taking conjugates and commutators of instances of the same class, one can perform these operations with one element in `N` and one in `NX`. The outcome will obviously belong to `NX`.

Example 6.

```
>>> x
N ([[9613, 9613, 9613], [5859, 5859, 5859], [2970, 2970, 2970]],?)
>>> x*x1*x4
N ([[6643, 6643, 6643], [0, 0, 0], [0, 0, 0]],?)
>>> x.comm(x1)
N ([[6643, 2305, 17650], [3699, 1517, 5893], [4401, 762, 3675]],?)
```

Besides, we can compare two elements of `NX` or one element of `N` and one of `NX` with the operators `>`, `<`, `>=`, `<=`. The output is explained in what follows. For an element g of `NX` we denote by $l(g)$ the number of known diagonals.

`__gt__()`:
Let g be an instance of `NX` and h an instance of `N` or `NX`. Then `g>h` returns `True` if all the first $l(g)$ upper diagonals of g agree with the first $l(g)$ upper diagonals of h and, in the case of h in `NX`, $l(g) < l(h)$.

`__ge__()`:
Let g be an instance of `NX` and h an instance of `N` or `NX`. Then `g>=h` returns `True` if all the first $l(g)$ upper diagonals of g agree with the first $l(g)$ upper diagonals of h.

`__lt__()`:
Let g be an instance of `N` or `NX` and h an instance of `NX`. Then `g<h` returns `True` if all the first $l(h)$ upper diagonals of g agree with the first $l(h)$ upper diagonals of h and, in the case of h in `NX`, $l(h) < l(g)$.

`__le__()`:
Let g be an instance of `N` or `NX` and h an instance of `NX`. Then `g<=h` returns `True` if all the first $l(h)$ diagonals of h agree with the first $l(h)$ upper diagonals of g.

Example 7.

```
>>> x=NX(x0.m[0],x0.m[1])
>>> y=NX(x0.m[0],x0.m[1],x0.m[2])
>>> x0<x
True
>>> x<y
False
>>> x>y
True
>>> z=x
>>> (x<z) or (z<x)
False
>>> (x<=z) and (x>=z)
True
```

References

[1] N. Peyerimhoff and A. Vdovina, "Cayley graph expanders and groups of finite width", *Journal of Pure and Applied Algebra* 215, no. 11 (2011): 2780-8.

[2] N. Barker, N. Boston, N. Peyerimhoff and A. Vdovina, "An Infinite Family of 2-Groups with Mixed Beauville Structures", *International Mathematics Research Notices*, (2014).

[3] N. Barker, N. Boston, N. Peyerimhoff and A. Vdovina, "Regular Algebraic Surfaces, Ramification Structures and Projective Planes", *Beauville Surfaces and Groups*, Springer Proceedings in Mathematics and Statistics, Vol. 123, Springer (June 2015).