
fpylll Documentation

Release 0.5.2dev

Martin R. Albrecht

Sep 14, 2020

CONTENTS

1	fpyll	1
1.1	Requirements	1
1.2	Online	2
1.3	Getting Started	2
1.4	Multicore Support	5
1.5	Contributing	5
1.6	Attribution & License	6
2	Modules	7
2.1	fplll Modules	7
2.2	Python Algorithms	38
3	Indices and Tables	39
Python Module Index		41
Index		43

CHAPTER
ONE

FpyLLL

A Python (2 and 3) wrapper for `fplll`.

```
>>> from fpylll import *

>>> A = IntegerMatrix(50, 50)
>>> A.randomize("ntrulike", bits=50, q=127)
>>> A[0].norm()
3564748886669202.5

>>> M = GSO.Mat(A)
>>> M.update_gso()
>>> M.get_mu(1, 0)
0.815748944429783

>>> L = LLL.Reduction(M)
>>> L()
>>> M.get_mu(1, 0)
0.41812865497076024
>>> A[0].norm()
24.06241883103193
```

The basic BKZ algorithm can be implemented in about 60 pretty readable lines of Python code (cf. `simple_bkz.py`). For a quick tour of the library, you can check out the [tutorial](#).

1.1 Requirements

`fpylll` relies on the following C/C++ libraries:

- [GMP](#) or [MPIR](#) for arbitrary precision integer arithmetic.
- [MPFR](#) for arbitrary precision floating point arithmetic.
- [QD](#) for double double and quad double arithmetic (optional).
- [fplll](#) for pretty much everything.

`fpylll` also relies on

- [Cython](#) for linking Python and C/C++.

- `cysignals` for signal handling such as interrupting C++ code.
- `py.test` for testing Python.
- `flake8` for linting.

We also suggest

- `virtualenv` to build and install fpyll in
- `IPython` for interacting with Python
- `Numpy` for numerical computations (e.g. with Gram-Schmidt values)

1.2 Online

fpyll ships with Sage. Thus, it is available via `SageMathCell` and `CoCalc` (select a Jupyter notebook with a Sage kernel). You can also fire up a `dply.co` virtual server with the latest fpyll/fplll preinstalled (it takes perhaps 15 minutes until everything is compiled).

1.3 Getting Started

Note: fpyll is also available via `PyPI` and `Conda-Forge` for `Conda`. In what follows, we explain manual installation.

We recommend `virtualenv` for isolating Python build environments and `virtualenvwrapper` to manage virtual environments. We indicate active virtualenvs by the prefix (`fpylll`).

Automatic install

1. Run `bootstrap.sh`

```
$ ./bootstrap.sh  
$ source ./activate
```

Manual install

1. Create a new virtualenv and activate it:

```
$ virtualenv env  
$ ln -s ./env/bin/activate ./  
$ source ./activate
```

2. Install the required libraries - `GMP` or `MPIR` and `MPFR` - if not available already. You may also want to install `QD`.

3. Install fplll:

```
$ (fpylll) ./install-dependencies.sh $VIRTUAL_ENV
```

Some OSX users report that they required `export CXXFLAGS="-stdlib=libc++ -mmacosx-version-min=10.7"` and `export CXX=clang++` (after installing a recent clang with `brew`) since the default GCC installed by Apple does not have full C++11 support.

4. Then, execute:

```
$ (fpylll) pip install Cython  
$ (fpylll) pip install -r requirements.txt
```

to install the required Python packages (see above).

5. If you are so inclined, run:

```
$ (fpylll) pip install -r suggestions.txt
```

to install suggested Python packages as well (optional).

6. Build the Python extension:

```
$ (fpylll) export PKG_CONFIG_PATH="$VIRTUAL_ENV/lib/pkgconfig:$PKG_CONFIG_PATH"
$ (fpylll) python setup.py build_ext
$ (fpylll) python setup.py install
```

7. To run **fpylll**, you will need to:

```
$ (fpylll) export LD_LIBRARY_PATH="$VIRTUAL_ENV/lib"
```

so that Python can find fplll and friends.

Note that you can also patch `activate` to set `LD_LIBRARY_PATH`. For this, add:

```
### LD_LIBRARY_HACK
_OLD_LD_LIBRARY_PATH="$LD_LIBRARY_PATH"
LD_LIBRARY_PATH="$VIRTUAL_ENV/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH
### END_LD_LIBRARY_HACK

### PKG_CONFIG_HACK
_OLD_PKG_CONFIG_PATH="$PKG_CONFIG_PATH"
PKG_CONFIG_PATH="$VIRTUAL_ENV/lib/pkgconfig:$PKG_CONFIG_PATH"
export PKG_CONFIG_PATH
### END_PKG_CONFIG_HACK
```

towards the end and:

```
### LD_LIBRARY_HACK
if ! [ -z ${_OLD_LD_LIBRARY_PATH+x} ] ; then
    LD_LIBRARY_PATH="${_OLD_LD_LIBRARY_PATH}"
    export LD_LIBRARY_PATH
    unset _OLD_LD_LIBRARY_PATH
fi
### END_LD_LIBRARY_HACK

### PKG_CONFIG_HACK
if ! [ -z ${_OLD_PKG_CONFIG_PATH+x} ] ; then
    PKG_CONFIG_PATH="${_OLD_PKG_CONFIG_PATH}"
    export PKG_CONFIG_PATH
    unset _OLD_PKG_CONFIG_PATH
fi
### END_PKG_CONFIG_HACK
```

in the `deactivate` function in the `activate` script.

Running fpylll

1. To (re)activate the virtual environment, simply run:

```
$ source ./activate
```

2. Start Python:

```
$ (fpylll) ipython
```

Manual update of fpylll and fpLLL inside Sagemath 9.0+

The instructions are very similar to the manual ones above.

1. Activate the sage-sh virtualenv:

```
$ sage -sh
```

2. Install the required libraries - [GMP](#) or [MPIR](#) and [MPFR](#) - if not available already. You may also want to install [QD](#).

3. Install fpLLL:

```
$ (sage-sh) ./install-dependencies.sh $SAGE_LOCAL
```

Some OSX users report that they required `export CXXFLAGS="-stdlib=libc++ -mmacosx-version-min=10.7"` and `export CXX=clang++` (after installing a recent clang with brew) since the default GCC installed by Apple does not have full C++11 support.

4. Then, execute:

```
$ (sage-sh) pip3 install Cython  
$ (sage-sh) pip3 install -r requirements.txt
```

to install the required Python packages (see above).

5. If you are so inclined, run:

```
$ (sage-sh) pip3 install -r suggestions.txt
```

to install suggested Python packages as well (optional).

6. Build the Python extension:

```
$ (sage-sh) export PKG_CONFIG_PATH="$SAGE_LOCAL/lib/pkgconfig:$PKG_CONFIG_PATH"  
$ (sage-sh) python3 setup.py build_ext  
$ (sage-sh) python3 setup.py install  
$ (sage-sh) exit
```

7. Verify the upgrade went well:

```
$ sage  
sage: import fpylll  
sage: print(fpylll.__version__)
```

The output should match the value of `version_insrc/fpylll/_init_.py`.

1.4 Multicore Support

fpylll supports parallelisation on multiple cores. For all C++ support to drop the GIL is enabled, allowing the use of threads to parallelise. Fplll is thread safe as long as each thread works on a separate object such as `IntegerMatrix` or `MatGSO`. Also, **fpylll** does not actually drop the GIL in all calls to C++ functions yet. In many scenarios using `multiprocessing`, which sidesteps the GIL and thread safety issues by using processes instead of threads, will be the better choice.

The example below calls `LLL.reduction` on 128 matrices of dimension 30 on four worker processes.

```
from fpylll import IntegerMatrix, LLL
from multiprocessing import Pool

d, workers, tasks = 30, 4, 128

def run_it(p, f, A, prefix=""):
    """Print status during parallel execution."""
    import sys
    r = []
    for i, retval in enumerate(p imap_unordered(f, A, 1)):
        r.append(retval)
        sys.stderr.write('\r{} done: {:.2%}'.format(prefix, float(i)/len(A)))
        sys.stderr.flush()
    sys.stderr.write('\r{} done {:.2%}\n'.format(prefix, float(i+1)/len(A)))
    return r

A = [IntegerMatrix.random(d, "uniform", bits=30) for _ in range(tasks)]
A = run_it(Pool(workers), LLL.reduction, A)
```

To test threading simply replace the line `from multiprocessing import Pool` with `from multiprocessing.pool import ThreadPool as Pool`. For calling `BKZ.reduction` this way, which expects a second parameter with options, using `functools.partial` is a good choice.

1.5 Contributing

fpylll welcomes contributions, cf. the list of open issues. To contribute, clone this repository, commit your code on a separate branch and send a pull request. Please write tests for your code. You can run them by calling:

```
$ (fpylll) PY_IGNORE_IMPORTMISMATCH=1 py.test
```

from the top-level directory which runs all tests in `tests/test_*.py`. We run `flake8` on every commit automatically. In particular, we run:

```
$ (fpylll) flake8 --max-line-length=120 --max-complexity=16 --ignore=E22,E241 src
```

Note that **fpylll** supports Python 2 and 3. In particular, tests are run using Python 2.7 and 3.5. See `.travis.yml` for details on automated testing.

1.6 Attribution & License

fpyll is maintained by Martin Albrecht.

The following people have contributed to **fpyll**

- Eamonn Postlethwaite
- E M Bray
- Fernando Virdia
- Guillaume Bonnoron
- Jeroen Demeyer
- Jérôme Benoit
- Konstantinos Draziotis
- Leo Ducas
- Martin Albrecht
- Michael Walter
- Omer Katz

We copied a decent bit of code over from Sage, mostly from it's fpLLL interface.

fpyll is licensed under the GPLv2+.

MODULES

2.1 fpLLL Modules

The modules in this category in some way represent classes or functions from fpLLL. They are typically implemented in Cython.

2.1.1 Integer Matrices

Dense matrices over the Integers.

```
class fpylll.fpLLL.integer_matrix.IntegerMatrix(arg0, arg1=None, int_type='mpz')
```

Dense matrices over the Integers.

copy(*self*)

Copy this matrix.

getitem()

Select a row or entry.

Parameters **key** – an integer for the row, a tuple for row and column or a slice.

Returns a reference to a row or an integer depending on format of **key**

```
>>> from fpylll import IntegerMatrix
>>> A = IntegerMatrix(10, 10)
>>> A.gen_identity(10)
>>> A[1, 0]
0
```

```
>>> print(A[1])
(0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
```

```
>>> print(A[0:2])
[ 1 0 0 0 0 0 0 0 0 0 ]
[ 0 1 0 0 0 0 0 0 0 0 ]
```

init()

Construct a new integer matrix

Parameters

- **arg0** – number of rows 0 or matrix
- **arg1** – number of columns 0 or None

The default constructor takes the number of rows and columns:

```
>>> from fpylll import IntegerMatrix
>>> IntegerMatrix(10, 10)
<IntegerMatrix(10, 10) at 0x...>

>>> IntegerMatrix(10, 0)
<IntegerMatrix(10, 0) at 0x...>

>>> IntegerMatrix(-1, 0)
Traceback (most recent call last):
...
ValueError: Number of rows must be >0
```

The default constructor is also a copy constructor:

```
>>> A = IntegerMatrix(2, 2)
>>> A[0,0] = 1
>>> B = IntegerMatrix(A)
>>> B[0,0]
1
>>> A[0,0] = 2
>>> B[0,0]
1
```

`__setitem__(self, key, value)`

Assign value to index.

Parameters

- **key** – a tuple of row and column indices
- **value** – an integer

Example:

```
>>> from fpylll import IntegerMatrix
>>> A = IntegerMatrix(10, 10)
>>> A.gen_identity(10)
>>> A[1,0] = 2
>>> A[1,0]
2
```

Arbitrary precision integers are supported:

```
>>> A[0, 0] = 2**2048
```

The notation `A[i][j]` is not supported. This is because `A[i]` returns an object of type `IntegerMatrixRow` object which is immutable by design. This is to avoid the user confusing such an object with a proper vector.:

```
>>> A[1][0] = 2
Traceback (most recent call last):
...
TypeError: 'fpylll.fplll.integer_matrix.IntegerMatrixRow' object does not support item assignment
```

`apply_transform(self, IntegerMatrix U, int start_row=0)`

Apply transformation matrix `U` to this matrix starting at row `start_row`.

Parameters

- **U** (`IntegerMatrix`) – transformation matrix
- **start_row** (`int`) – start transformation in this row

`clear(self)`

classmethod from_file (`type cls, filename, **kwds`)

Construct new matrix from file.

```
>>> import tempfile
>>> A = IntegerMatrix.random(10, "qary", k=5, bits=20)
```

```
>>> fn = tempfile.mktemp()
>>> fh = open(fn, "w")
>>> _ = fh.write(str(A))
>>> fh.close()
```

```
>>> B = IntegerMatrix.from_file(fn)
>>> A == B
True
```

Parameters `filename` – name of file to read from

classmethod from_iterable (`type cls, nrows, ncols, it, **kwds`)

Construct a new integer matrix from matrix-like object A

Parameters

- **nrows** – number of rows
- **ncols** – number of columns
- **it** – an iterable of length at least `nrows * ncols`

```
>>> A = IntegerMatrix.from_iterable(2, 3, [1, 2, 3, 4, 5, 6])
>>> print(A)
[ 1 2 3 ]
[ 4 5 6 ]
```

classmethod from_matrix (`type cls, A, nrows=None, ncols=None, **kwds`)

Construct a new integer matrix from matrix-like object A

Parameters

- **A** – a matrix like object, with element access `A[i,j]` or `A[i][j]`
- **nrows** – number of rows (optional)
- **ncols** – number of columns (optional)

```
>>> A = IntegerMatrix.from_matrix([[1, 2, 3], [4, 5, 6]])
>>> print(A)
[ 1 2 3 ]
[ 4 5 6 ]
```

`gen_identity(self, int nrows=-1)`

Generate identity matrix.

Parameters `nrows` – number of rows

get_max_exp (*self*)

```
>>> A = IntegerMatrix.from_matrix([[0, 2], [3, 4]])
>>> A.get_max_exp()
3
```

```
>>> A = IntegerMatrix.from_matrix([[0, 2], [3, 9]])
>>> A.get_max_exp()
4
```

classmethod identity (*type cls, nrows, int_type='mpz'*)

Construct a new identity matrix of dimension *nrows* × *nrows*

Parameters **nrows** – number of rows.

```
>>> A = IntegerMatrix.identity(4)
>>> print(A)
[ 1 0 0 0 ]
[ 0 1 0 0 ]
[ 0 0 1 0 ]
[ 0 0 0 1 ]
```

int_type

is_empty (*self*)

mod (*self, q, int start_row=0, int start_col=0, int stop_row=-1, int stop_col=-1*)

Apply modular reduction modulo *q* to this matrix.

Parameters

- **q** – modulus
- **start_row** (*int*) – starting row
- **start_col** (*int*) – starting column
- **stop_row** (*int*) – last row (excluding)
- **stop_col** (*int*) – last column (excluding)

```
>>> A = IntegerMatrix(2, 2)
>>> A[0, 0] = 1001
>>> A[1, 0] = 13
>>> A[0, 1] = 102
>>> print(A)
[ 1001 102 ]
[ 13 0 ]
```

```
>>> A.mod(10, start_row=1, start_col=0)
>>> print(A)
[ 1001 102 ]
[ 3 0 ]
```

```
>>> A.mod(10)
>>> print(A)
[ 1 2 ]
[ 3 0 ]
```

```
>>> A = IntegerMatrix(2, 2)
>>> A[0,0] = 1001
>>> A[1,0] = 13
>>> A[0,1] = 102
>>> A.mod(10, stop_row=1)
>>> print(A)
[ 1 2 ]
[ 13 0 ]
```

`multiply_left` (*self, v, start=0*)

Return $v \star A'$ where A' is A reduced to $\text{len}(v)$ rows starting at `start`.

Parameters

- **v** – a tuple-like object
- **start** – start in row `start`

`ncols`

Number of Columns

Returns number of columns

```
>>> from fpylll import IntegerMatrix
>>> IntegerMatrix(10, 10).ncols
10
```

`nrows`

Number of Rows

Returns number of rows

```
>>> from fpylll import IntegerMatrix
>>> IntegerMatrix(10, 10).nrows
10
```

`classmethod random` (*type cls, d, algorithm, int_type='mpz', **kwds*)

Construct new random matrix.

Parameters

- **d** – dominant size parameter, see below for details
- **algorithm** – type of matrix create, see below for details
- **int_type** – underlying integer type

Returns a random lattice basis

Examples:

```
>>> from fpylll import FPLLL
>>> FPLLL.set_random_seed(1337)

>>> print(IntegerMatrix.random(10, "intrel", bits=30))
[ 285965362 1 0 0 0 0 0 0 0 0 0 ]
[ 714553900 0 1 0 0 0 0 0 0 0 0 ]
[ 1017994245 0 0 1 0 0 0 0 0 0 0 ]
[ 256743299 0 0 0 1 0 0 0 0 0 0 ]
[ 602398079 0 0 0 0 1 0 0 0 0 0 ]
[ 159503182 0 0 0 0 0 1 0 0 0 0 ]
```

(continues on next page)

(continued from previous page)

```
[ 450941699 0 0 0 0 0 0 1 0 0 0 ]
[ 125249023 0 0 0 0 0 0 0 1 0 0 ]
[ 158876382 0 0 0 0 0 0 0 0 1 0 ]
[ 514616289 0 0 0 0 0 0 0 0 0 1 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(10, "simdioph", bits=10, bits2=30))
[ 1073741824 50 556 5 899 383 846 771 511 734 ]
[ 0 1024 0 0 0 0 0 0 0 0 ]
[ 0 0 1024 0 0 0 0 0 0 0 ]
[ 0 0 0 1024 0 0 0 0 0 0 ]
[ 0 0 0 0 1024 0 0 0 0 0 ]
[ 0 0 0 0 0 1024 0 0 0 0 ]
[ 0 0 0 0 0 0 1024 0 0 0 ]
[ 0 0 0 0 0 0 0 1024 0 0 ]
[ 0 0 0 0 0 0 0 0 1024 0 ]
[ 0 0 0 0 0 0 0 0 0 1024 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(10, "uniform", bits=10))
[ 50 556 5 899 383 846 771 511 734 993 ]
[ 325 12 242 43 374 815 437 260 541 50 ]
[ 492 174 215 999 186 189 292 497 832 966 ]
[ 508 290 160 247 859 817 669 821 258 930 ]
[ 510 933 588 895 18 546 393 868 858 790 ]
[ 620 72 832 133 263 121 724 35 454 385 ]
[ 431 347 749 311 911 937 50 160 322 180 ]
[ 517 941 184 922 217 563 1008 960 37 85 ]
[ 5 855 643 824 43 525 37 988 886 118 ]
[ 27 944 560 993 662 589 20 694 696 205 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(5, "ntrulike", q=127))
[ 1 0 0 0 0 25 50 44 5 3 ]
[ 0 1 0 0 0 3 25 50 44 5 ]
[ 0 0 1 0 0 5 3 25 50 44 ]
[ 0 0 0 1 0 44 5 3 25 50 ]
[ 0 0 0 0 1 50 44 5 3 25 ]
[ 0 0 0 0 0 127 0 0 0 0 ]
[ 0 0 0 0 0 0 127 0 0 0 ]
[ 0 0 0 0 0 0 0 127 0 0 ]
[ 0 0 0 0 0 0 0 0 127 0 ]
[ 0 0 0 0 0 0 0 0 0 127 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(5, "ntrulike2", q=127))
[ 127 0 0 0 0 0 0 0 0 0 ]
[ 0 127 0 0 0 0 0 0 0 0 ]
[ 0 0 127 0 0 0 0 0 0 0 ]
[ 0 0 0 127 0 0 0 0 0 0 ]
[ 0 0 0 0 127 0 0 0 0 0 ]
[ 25 3 5 44 50 1 0 0 0 0 ]
[ 50 25 3 5 44 0 1 0 0 0 ]
[ 44 50 25 3 5 0 0 1 0 0 ]
[ 5 44 50 25 3 0 0 0 1 0 ]
```

(continues on next page)

(continued from previous page)

```
[ 3 5 44 50 25 0 0 0 0 1 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(10, "qary", k=8, q=127))
[ 1 0 50 44 5 3 78 3 94 97 ]
[ 0 1 69 12 114 43 118 47 53 4 ]
[ 0 0 127 0 0 0 0 0 0 0 ]
[ 0 0 0 127 0 0 0 0 0 0 ]
[ 0 0 0 0 127 0 0 0 0 0 ]
[ 0 0 0 0 0 127 0 0 0 0 ]
[ 0 0 0 0 0 0 127 0 0 0 ]
[ 0 0 0 0 0 0 0 127 0 0 ]
[ 0 0 0 0 0 0 0 0 127 0 ]
[ 0 0 0 0 0 0 0 0 0 127 ]
```

```
>>> FPLLL.set_random_seed(1337)
>>> print(IntegerMatrix.random(10, "trg", alpha=0.99))
[ 228404 0 0 0 0 0 0 0 0 0 ]
[ -80428 34992 0 0 0 0 0 0 0 0 ]
[ -104323 -3287 24449 0 0 0 0 0 0 0 ]
[ -54019 -5306 9234 42371 0 0 0 0 0 0 ]
[ -17118 -13604 6537 -10587 4082 0 0 0 0 0 ]
[ 108869 8134 4954 -17719 -1984 15326 0 0 0 0 ]
[ -111858 -7328 5192 8105 -1109 1910 5818 0 0 0 ]
[ -97654 -16219 -2181 14658 -1879 7195 -100 2347 0 0 ]
[ -46340 13109 6265 12205 -1848 6113 1049 -170 1810 0 ]
[ 10290 16293 4131 -4313 -525 2068 -262 248 715 592 ]
```

Available Algorithms:

- "intrel" - (bits = b) generate a knapsack like matrix of dimension $d \times (d+1)$ and b bits: the i-th vector starts with a random integer of bit-length b and the rest is the i-th canonical unit vector.
- "simdioph" - (bits = b_1 , bits2 = b_2) generate addmatrix of a form similar to that is involved when trying to find length b_2 and continues with $d-1$ independent integers of bit-lengths b_1 ; the i-th vector for $i > 1$ is the i-th canonical unit vector scaled by a factor 2^{b_1} .
- "uniform" - (bits = b) - generate a $d \times d$ matrix whose entries are independent integers of bit-lengths b.
- "ntrulike" - (bits = b or q) generate a $2d \times 2d$ NTRU-like matrix. If bits is given, then it first samples an integer q of bit-length b, whereas if q, then it sets q to the provided value. Then it samples a uniform h in the ring $Z_q[x]/(x^n - 1)$. It finally returns the 2×2 block matrix $\begin{bmatrix} I, \text{rot}(h) \\ [0, qI] \end{bmatrix}$, where each block is $d \times d$, the first row of $\text{rot}(h)$ is the coefficient vector throw of $\text{rot}(h)$ is the shift of the $(i-1)$ -th (with last entry put back in first position), for all $i > 1$.
- ntrulike2" - (bits = b or q) as the previous option, except that the constructed matrix is $\begin{bmatrix} [qI, 0], [\text{rot}(h), I] \end{bmatrix}$.
- "qary" - (bits = b or q, k) generate a $d \times d$ q-ary matrix with determinant q^k . If bits is given, then it first samples an integer q of bit-length b; if q is provided, then set q to the provided value. It returns a 2×2 block matrix $\begin{bmatrix} I, \text{rot}(h) \\ [0, qI] \end{bmatrix}$, where each block is $d \times d$, the first row of $\text{rot}(h)$ is the shift of the $(i-1)$ -th (with last entry put back in first position), for all $i > 1$.
- "trg" - (alpha) generate a $d \times d$ lower-triangular matrix B with $B_{ii} = 2^{(d-i+1)\alpha}$ for all i, and B_{ij} is uniform between $-B_{jj}/2$ and $B_{jj}/2$ for all $j < i$.

Warning The NTRU options above do *not* produce genuine NTRU lattice with an unusually short dense sublattice.

Seealso [randomize\(\)](#)

randomize (*self*, *algorithm*, ***kwds*)
Randomize this matrix using *algorithm*.

Parameters **algorithm** – see [random\(\)](#)

Seealso [random\(\)](#)

resize (*self*, *int rows*, *int cols*)

Parameters

- **rows** (*int*) –
- **cols** (*int*) –

rotate (*self*, *int first*, *int middle*, *int last*)

Rotates the order of the elements in the range [first,last), in such a way that the element pointed by middle becomes the new first element.

$(M[first], \dots, M[middle-1], M[middle], M[last])$ becomes $(M[middle], \dots, M[last], M[first], \dots, M[middle-1])$

Parameters

- **first** (*int*) – first index
- **middle** (*int*) – new first index
- **last** (*int*) – last index (inclusive)

```
>>> A = IntegerMatrix.from_matrix([[0,1,2],[3,4,5],[6,7,8]])
>>> A.rotate(0,0,2)
>>> print(A)
[ 0 1 2 ]
[ 3 4 5 ]
[ 6 7 8 ]
```

```
>>> A = IntegerMatrix.from_matrix([[0,1,2],[3,4,5],[6,7,8]])
>>> A.rotate(0,2,2)
>>> print(A)
[ 6 7 8 ]
[ 0 1 2 ]
[ 3 4 5 ]
```

rotate_gram_left (*self*, *int first*, *int last*, *int n_valid_rows*)

Transformation needed to update the lower triangular Gram matrix when `rotateLeft(first, last)` is done on the basis of the lattice.

Parameters

- **first** (*int*) –
- **last** (*int*) –
- **n_valid_rows** (*int*) –

```
>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
```

rotate_gram_right (*self*, *int first*, *int last*, *int n_valid_rows*)

Transformation needed to update the lower triangular Gram matrix when `rotateRight(first, last)` is done on the basis of the lattice.

Parameters

- **first** (*int*) –
- **last** (*int*) –
- **n_valid_rows** (*int*) –

```
>>> A = IntegerMatrix.from_matrix([[0, 2], [3, 4]])
```

rotate_left (*self, int first, int last*)

Row permutation.

(M[first], ..., M[last]) becomes (M[first+1], ..., M[last], M[first])

Parameters

- **first** (*int*) –
- **last** (*int*) –

```
>>> A = IntegerMatrix.from_matrix([[0, 2], [3, 4]])
```

rotate_right (*self, int first, int last*)

Row permutation.

(M[first], ..., M[last]) becomes (M[last], M[first], ..., M[last-1])

Parameters

- **first** (*int*) –
- **last** (*int*) –

```
>>> A = IntegerMatrix.from_matrix([[0, 2], [3, 4]])
```

set_cols (*self, int cols*)

Parameters **cols** (*int*) –

set_iterable (*self, A*)

Set this matrix from iterable A

Parameters **A** – an iterable object such as a list or tuple

EXAMPLE:

```
>>> z = range(16)
>>> A = IntegerMatrix(4, 4)
>>> A.set_iterable(z)
>>> print(A)
[ 0  1  2  3 ]
[ 4  5  6  7 ]
[ 8  9 10 11 ]
[ 12 13 14 15 ]

>>> A = IntegerMatrix(3, 3)
>>> A.set_iterable(z)
>>> print(A)
[ 0 1 2 ]
[ 3 4 5 ]
[ 6 7 8 ]
```

Warning: entries starting at $A[\text{nrows} * \text{ncols}]$ are ignored.

set_matrix(*self, A*)

Set this matrix from matrix-like object *A*.

Parameters **A** – a matrix like object, with element access $A[i,j]$ or $A[i][j]$

Example:

```
>>> z = [[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]]
>>> A = IntegerMatrix(4, 4)
>>> A.set_matrix(z)
>>> print(A)
[ 1  2  3  4 ]
[ 5  6  7  8 ]
[ 9 10 11 12 ]
[ 13 14 15 16 ]

>>> A = IntegerMatrix(3, 3)
>>> A.set_matrix(z)
>>> print(A)
[ 1  2  3 ]
[ 5  6  7 ]
[ 9 10 11 ]
```

Warning: entries starting from $A[\text{nrows}, \text{ncols}]$ are ignored.

set_rows(*self, int rows*)

Parameters **rows**(*int*) –

submatrix(*self, a, b, c=None, d=None*)

Construct a new submatrix.

Parameters

- **a** – either the index of the first row or an iterable of row indices
- **b** – either the index of the first column or an iterable of column indices
- **c** – the index of first excluded row (or *None*)
- **d** – the index of first excluded column (or *None*)

Returns**Return type**

We illustrate the calling conventions of this function using a 10 x 10 matrix:

```
>>> from fpylll import IntegerMatrix, FPLLL
>>> A = IntegerMatrix(10, 10)
>>> FPLLL.set_random_seed(1337)
>>> A.randomize("ntrulike", bits=22, q=4194319)
>>> print(A)
[ 1 0 0 0 0 3021421 752690 1522220 2972677 119630 ]
[ 0 1 0 0 0 119630 3021421 752690 1522220 2972677 ]
```

(continues on next page)

(continued from previous page)

```
[ 0 0 1 0 0 2972677 119630 3021421 752690 1522220 ]
[ 0 0 0 1 0 1522220 2972677 119630 3021421 752690 ]
[ 0 0 0 0 1 752690 1522220 2972677 119630 3021421 ]
[ 0 0 0 0 0 4194319 0 0 0 0 ]
[ 0 0 0 0 0 0 4194319 0 0 0 ]
[ 0 0 0 0 0 0 0 4194319 0 0 ]
[ 0 0 0 0 0 0 0 0 4194319 0 ]
[ 0 0 0 0 0 0 0 0 0 4194319 ]
```

We can either specify start/stop rows and columns:

```
>>> print(A.submatrix(0,0,2,8))
[ 1 0 0 0 0 3021421 752690 1522220 ]
[ 0 1 0 0 0 119630 3021421 752690 ]
```

Or we can give lists of rows, columns explicitly:

```
>>> print(A.submatrix([0,1,2],range(3,9)))
[ 0 0 3021421 752690 1522220 2972677 ]
[ 0 0 119630 3021421 752690 1522220 ]
[ 0 0 2972677 119630 3021421 752690 ]
```

`swap_rows (self, int r1, int r2)`

Parameters

- `r1 (int)` –
- `r2 (int)` –

```
>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
>>> A.swap_rows(0, 1)
>>> print(A)
[ 3 4 ]
[ 0 2 ]
```

`to_matrix (self, A)`

Write this matrix to matrix-like object A

Parameters `A` – a matrix like object, with element access `A[i,j]` or `A[i][j]`

Returns A

Example:

```
>>> from fpylll import FPLLL
>>> z = [[0 for _ in range(10)] for _ in range(10)]
>>> A = IntegerMatrix.random(10, "qary", q=127, k=5)
>>> _ = A.to_matrix(z)
>>> z[0] == list(A[0])
True
```

`transpose (self)`

Inline transpose.

```
>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
>>> _ = A.transpose()
>>> print(A)
```

(continues on next page)

(continued from previous page)

[0 3]
[2 4]

class fpylll.fplll.integer_matrix.IntegerMatrixRow (*IntegerMatrix M, int row*)

A reference to a row in an integer matrix.

__getitem__()

Return entry at column

Parameters **column** (*int*) – integer offset**__init__()**

Create a row reference.

Parameters

- **M** (*IntegerMatrix*) – Integer matrix
- **row** (*int*) – row index

Row references are immutable:

```
>>> from fpylll import IntegerMatrix
>>> A = IntegerMatrix(2, 3)
>>> A[0,0] = 1; A[0,1] = 2; A[0,2] = 3
>>> r = A[0]
>>> r[0]
1
>>> r[0] = 1
Traceback (most recent call last):
...
TypeError: 'fpylll.fplll.integer_matrix.IntegerMatrixRow' object does not support item assignment
```

addmul (*self, IntegerMatrixRow v, x=1, int expo=0*)In-place add row vector $2^{\text{expo}} \times v$ **Parameters**

- **v** (*IntegerMatrixRow*) – a row vector
- **x** – multiplier
- **expo** (*int*) – scaling exponent.

Example:

```
>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
>>> A[0].addmul(A[1])
>>> print(A[0])
(3, 6)

>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
>>> A[0].addmul(A[1],x=0)
>>> print(A[0])
(0, 2)

>>> A = IntegerMatrix.from_matrix([[0,2],[3,4]])
>>> A[0].addmul(A[1],x=1,expo=2)
>>> print(A[0])
(12, 18)
```

is_zero(self, int frm=0)

Return True if this vector consists of only zeros starting at index frm

Example:

```
>>> A = IntegerMatrix.from_matrix([[1, 0, 0]])
>>> A[0].is_zero()
False
>>> A[0].is_zero(1)
True
```

norm()

Return $\sqrt{2}$ norm of this vector.

Example:

```
>>> A = IntegerMatrix.from_iterable(1, 3, [1, 2, 3])
>>> A[0].norm()
3.74165...
>>> 1*1 + 2*2 + 3*3
14
>>> from math import sqrt
>>> sqrt(14)
3.74165...
```

size_nz(self)

Index at which an all zero vector starts.

Example:

```
>>> A = IntegerMatrix.from_matrix([[0, 2, 3], [0, 2, 0], [0, 0, 0]])
>>> A[0].size_nz()
3
>>> A[1].size_nz()
2
>>> A[2].size_nz()
0
```

`fpylll.fplll.integer_matrix.unpickle_IntegerMatrix(nrows, ncols, l, int_type='mpz')`
Deserialize an integer matrix.

Parameters

- **nrows** – number of rows
- **ncols** – number of columns
- **l** – list of entries

2.1.2 Gram Schmidt Orthogonalization

Elementary basis operations, Gram matrix and Gram-Schmidt orthogonalization.

A MatGSO object stores the following information:

- The integral basis B,
- the Gram-Schmidt coefficients $i, j = b_i, b_j^*/||b_j^*||^2$ for $i > j$, and
- the coefficients $r_{i,j} = b_i, b_j^*$ for $i > j$

It holds that: $B = R \times Q = (\times D) \times (D^{-1}B^*)$ where Q is orthonormal and R is lower triangular.

```
class fpylll.fplll.gso.GSO
```

```
DEFAULT = 0
```

```
INT_GRAM = 1
```

```
Mat
```

alias of `MatGSO`

```
OP_FORCE_LONG = 4
```

```
ROW_EXPO = 2
```

```
class fpylll.fplll.gso.MatGSO(IntegerMatrix B, U=None, UinvT=None, int flags=GSO_DEFAULT, float_type='double', gram=False)
```

`MatGSO` provides an interface for performing elementary operations on a basis and computing its Gram matrix and its Gram-Schmidt orthogonalization. The Gram-Schmidt coefficients are computed on demand. The object keeps track of which coefficients are valid after each row operation.

```
B
```

```
G
```

Return the Gram matrix.

- If this GSO object operates on a Gram matrix, return that.
- If this GSO object operates on a basis with `GSO.INT_GRAM` set, construct the Gram matrix and return it
- Otherwise, a `NotImplementedError` is raised

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> FPLLL.set_random_seed(1337)
>>> A = IntegerMatrix.random(10, "qary", k=5, bits=10)
>>> M = GSO.Mat(A, flags=GSO.INT_GRAM); _ = M.update_gso()
>>> G = M.G
>>> print(G)
[ 2176   0   0   0   0   0   0   0   0   0 ]
[ 1818 4659   0   0   0   0   0   0   0   0 ]
[ 2695 5709 7416   0   0   0   0   0   0   0 ]
[ 2889 5221 7077 7399   0   0   0   0   0   0 ]
[ 2746 3508 4717 4772 4618   0   0   0   0   0 ]
[ 2332 1590 2279 2332 2597 2809   0   0   0   0 ]
[ 265 1749 2491 2438   0   0 2809   0   0   0 ]
[ 159 265 212 1219 318   0   0 2809   0   0 ]
[ 742 636 1537 2067 1802   0   0   0 2809   0 ]
[ 159 2650 2650 1908 1696   0   0   0   0 2809 ]
```

```
>>> A[0].norm() **2
2176.0
```

```
>>> M = GSO.Mat(G, gram=True); _ = M.update_gso()
>>> G == M.G
True
```

```
>>> M = GSO.Mat(A)
>>> M.G
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
NotImplementedError: Computing the Gram Matrix currently requires GSO.INT_GRAM
```

U**UinvT****__init__()****Parameters**

- **B** (`IntegerMatrix`) – The matrix on which row operations are performed. It must not be empty.
- **U** (`IntegerMatrix`) – If U is not empty, operations on B are also done on u (in this case both must have the same number of rows). If u is initially the identity matrix, multiplying transform by the initial basis gives the current basis.
- **UinvT** (`IntegerMatrix`) – Inverse transform (should be empty, which disables the computation, or initialized with identity matrix). It works only if U is not empty.
- **flags** (`int`) – Flags
 - GSO.INT_GRAM - If true, coefficients of the Gram matrix are computed with exact integer arithmetic. Otherwise, they are computed in floating-point. Note that when exact arithmetic is used, all coefficients of the first `n_known_rows` are continuously updated, whereas in floating-point, they are computed only on-demand. This option cannot be enabled when GSO.ROW_EXPO is set.
 - GSO.ROW_EXPO - If true, each row of B is normalized by a power of 2 before doing conversion to floating-point, which hopefully avoids some overflows. This option cannot be enabled if GSO.INT_GRAM is set and works only with `float_type="double"` and `float_type="long double"`. It is useless and **must not** be used for `float_type="dpe"`, `float_type="dd"`, `float_type="qd"` or `float_type=mpfr_t`.
 - GSO.OP_FORCE_LONG - Affects the behaviour of `row_admmul`. See its documentation.
- **float_type** – A floating point type, i.e. an element of `fpylll.fpylll.float_types`. If `float_type="mpfr"` set precision with `set_precision()` before constructing this object and do not change the precision during the lifetime of this object.
- **gram** – The input B is a Gram matrix of the lattice, rather than a basis.

Note that matching integer types for B, U and UinvT are enforced:

```
>>> from fpylll import IntegerMatrix, LLL, GSO
>>> B = IntegerMatrix.random(5, 'uniform', bits = 8, int_type = "long")
>>> M = GSO.Mat(B, U = IntegerMatrix.identity(B.nrows))
Traceback (most recent call last):
...
TypeError: U.int_type != B.int_type

>>> from fpylll import IntegerMatrix, LLL, GSO
>>> B = IntegerMatrix.random(5, 'uniform', bits=8, int_type="long")
>>> M = GSO.Mat(B, U = IntegerMatrix.identity(B.nrows, int_type="long"))
```

babai (*self*, *v*, *int start=0*, *int dimension=-1*, *gso=False*)
Return lattice vector close to *v* using Babai's nearest plane algorithm.

Parameters

- **v** – a tuple-like object
- **start** – only consider subbasis starting at *start`*
- **dimension** – only consider *dimension* vectors or all if *-1`*
- **gso** – if *True* vector is represented wrt to the Gram-Schmidt basis, otherwise canonical basis is assumed.

Returns a tuple of dimension *M.B.nrows*

create_row (*self*)

Adds a zero row to *B* (and to *U* if *enable_transform=true*). One or several operations can be performed on this row with *row_addmul*, then *row_op_end* must be called. Do not use if *inverse_transform_enabled=true*.

d

Number of rows of *B* (dimension of the lattice).

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.d
11
```

discover_all_rows (*self*)

Allows *row_addmul* for all rows even if the GSO has never been computed.

float_type

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(10, 10)
>>> M = GSO.Mat(A)
>>> M.float_type
'double'
>>> FPLLL.set_precision(100)
53
>>> M = GSO.Mat(A, float_type='mpfr')
>>> M.float_type
'mpfr'
```

from_canonical (*self*, *v*, *int start=0*, *int dimension=-1*)

Given a vector *v* wrt the canonical basis Z^n return a vector wrt the Gram – Schmidt basis B^*

Parameters

- **v** – a tuple-like object of dimension *M.B.ncols*
- **start** – only consider subbasis starting at *start`*
- **dimension** – only consider dimension vectors or all if *-1`*

Returns a tuple of dimension *dimension`* or *M.d`* when *dimension* is *None*

This operation is the inverse of *to_canonical*:

```
>>> import random
>>> A = IntegerMatrix.random(5, "uniform", bits=6)
>>> M = GSO.Mat(A)
>>> _ = M.update_gso()
>>> v = tuple(IntegerMatrix.random(5, "uniform", bits=6)[0]); v
(35, 24, 55, 40, 23)
>>> w = M.from_canonical(v); w
(0.98294..., 0.5636..., -3.4594479..., 0.9768..., 0.261316...)
>>> v_ = tuple([int(round(wi)) for wi in M.to_canonical(w)]); v_
(35, 24, 55, 40, 23)
>>> v == v_
True
```

get_current_slope(*self*, *int start_row*, *int stop_row*)

Finds the slope of the curve fitted to the lengths of the vectors from *start_row* to *stop_row*. The slope gives an indication of the quality of the LLL-reduced basis.

Parameters

- **start_row** (*int*) – start row index
- **stop_row** (*int*) – stop row index (exclusive)

Note: we call `get_current_slope` which is declared in `bkz.h`

get_gram(*self*, *int i*, *int j*)

Return Gram matrix coefficients (0 *i* *n_known_rows* and 0 *j* *i*). If `enable_row_expo` is false, returns the dot product $b_i \cdot b_j$. If `enable_row_expo` is true, returns $b_i \cdot b_j / 2^{(r_i + r_j)}$, where r_i and r_j are the row exponents of rows *i* and *j* respectively.

Parameters

- **i** (*int*) –
- **j** (*int*) –

get_log_det(*self*, *int start_row*, *int stop_row*)

Return log of the (squared) determinant of the basis.

Parameters

- **start_row** (*int*) – start row (inclusive)
- **stop_row** (*int*) – stop row (exclusive)

get_mu(*self*, *int i*, *int j*)

Return $\langle b_i, b_j^* \rangle / \|b_j^*\|^2$.

Parameters

- **i** –
- **j** –

get_mu_exp(*self*, *int i*, *int j*)

Return $f = {}_{i,j} \text{and exponent } x \text{ such that } f 2^x = b_i \cdot b_j^* / \|b_j^*\|^2$. If `enable_row_expo` is false, *x* is always zero. If `enable_row_expo` is true, $r_i - r_j$, where r_i and r_j are the row exponents of rows *i* and *j* respectively.

Note: It is assumed that i, j is valid.

Parameters

- **i** –
- **j** –

get_r (*self*, *int i, int j*)Return $b_i, b *_j$.**Parameters**

- **i** –
- **j** –

```
>>> from fpylll import *
>>> FPLLL.set_random_seed(0)
>>> A = IntegerMatrix.random(5, "uniform", bits=5)
>>> M = GSO.Mat(A)
>>> M.update_gso()
True
>>> M.get_r(1, 0)
1396.0
```

get_r_exp (*self*, *int i, int j*)Return $f = r_{i,j}$ and exponent x such that $b_i, b_j^* = f 2^x$. If `enable_row_expo` is `false`, x is always 0. If `enable_row_expo` is `true`, $r_i + r_j$, where r_i and r_j are the row exponents of rows i and j respectively.

Note: It is assumed that $r(i, j)$ is valid.

Parameters

- **i** –
- **j** –

get_root_det (*self*, *int start_row, int stop_row*)

Return (squared) root determinant of the basis.

Parameters

- **start_row** (*int*) – start row (inclusive)
- **stop_row** (*int*) – stop row (exclusive)

get_slide_potential (*self*, *int start_row, int stop_row, int block_size*)

Return slide potential of the basis

Parameters

- **start_row** (*int*) – start row (inclusive)
- **stop_row** (*int*) – stop row (exclusive)
- **block_size** (*int*) – block size

int_gram_enabled

Exact computation of dot products.

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.int_gram_enabled
False
```

```
>>> M = GSO.Mat(A, flags=GSO.INT_GRAM)
>>> M.int_gram_enabled
True
```

int_type**inverse_transform_enabled**

Computation of the inverse transform matrix (transposed).

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.inverse_transform_enabled
False
```

```
>>> U = IntegerMatrix.identity(11)
>>> UinvT = IntegerMatrix.identity(11)
>>> M = GSO.Mat(A, U=U, UinvT=UinvT)
>>> M.inverse_transform_enabled
True
```

move_row(self, int old_r, int new_r)

Row `old_r` becomes row `new_r` and intermediate rows are shifted. If `new_r < old_r`, then `old_r` must be < `n_known_rows`.

Parameters

- `old_r (int)` – row index
- `new_r (int)` – row index

negate_row(self, int i)

Set $b_i \rightarrow -b_i$.

Parameters

`i (int)` – index of the row to negate

Example:

```
>>> from fpylll import *
>>> FPLLL.set_random_seed(42)
>>> A = IntegerMatrix(6, 6)
>>> A.randomize("ntrulike", bits=6, q=31)
>>> print(A)
[ 1  0  0  12  25  25 ]
[ 0  1  0  25  12  25 ]
[ 0  0  1  25  25  12 ]
[ 0  0  0  31  0   0  ]
[ 0  0  0   0  31  0  ]
[ 0  0  0   0   0  31 ]
```



```
>>> M = GSO.Mat(A)
```

(continues on next page)

(continued from previous page)

```
>>> M.update_gso()
True
>>> with M.row_ops(2, 2):
...     M.negate_row(2)
...
>>> print(A)
[ 1  0   0   12  25  25 ]
[ 0  1   0   25  12  25 ]
[ 0  0  -1  -25 -25 -12 ]
[ 0  0   0   31   0   0 ]
[ 0  0   0   31   0   0 ]
[ 0  0   0   0   31   ]
```

r(*self*, *start*=0, *end*=-1)

Return r vector from start to end

remove_last_row(*self*)

Remove the last row of B (and of U if enable_transform=true). Do not use if inverse_transform_enabled=true.

row_addmul(*self*, *int i*, *int j*, *x*)Set $b_i = b_i + xb_j$.After one or several calls to `row_addmul`, `row_op_end` must be called.If `row_op_force_long`=true, *x* is always converted to $(2^{\text{expo}} * \text{long})$ instead of $(2^{\text{expo}} * \text{ZT})$, which is faster if $\text{ZT}=\text{mpz_t}$ but might lead to a loss of precision in LLL, more Babai iterations are needed.**Parameters**

- **i** (*int*) – target row
- **j** (*int*) – source row
- **x** – multiplier

row_expo_enabled

Normalization of each row of b by a power of 2.

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.row_expo_enabled
False
```

```
>>> M = GSO.Mat(A, flags=GSO.ROW_EXPO)
>>> M.row_expo_enabled
True
```

row_op_begin(*self*, *int first*, *int last*)Must be called before a sequence of `row_addmul`.**Parameters**

- **first** (*int*) – start index for `row_addmul` operations.
- **last** (*int*) – final index (exclusive).

Note: It is preferable to use `MatGSORowOpContext` via `row_ops`.

row_op_end(*self, int first, int last*)

Must be called after a sequence of `row_addmul`. This invalidates the *i*-th line of the GSO.

Parameters

- **first** (*int*) – start index to invalidate.
- **last** (*int*) – final index to invalidate (exclusive).

Note: It is preferable to use `MatGSORowOpContext` via `row_ops`.

row_op_force_long

Changes the behaviour of `row_addmul`, see its documentation.

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.row_op_force_long
False
```

```
>>> M = GSO.Mat(A, flags=GSO.OP_FORCE_LONG)
>>> M.row_op_force_long
True
```

row_ops(*self, int first, int last*)

Return context in which `row_addmul` operations are safe.

Parameters

- **first** (*int*) – start index.
- **last** (*int*) – final index (exclusive).

swap_rows(*self, int i, int j*)

Swap rows *i* and *j*.

Parameters

- **i** (*int*) – row index
- **j** (*int*) – row index

to_canonical(*self, v, int start=0*)

Given a vector *v* wrt the Gram-Schmidt basis *B** return a vector wrt the canonical basis \mathbb{Z}^n

Parameters

- **v** – a tuple-like object of dimension *M.d*
- **start** – only consider subbasis starting at *start`*

Returns a tuple of dimension *M.B.ncols*

transform_enabled

Computation of the transform matrix.

```
>>> from fpylll import IntegerMatrix, GSO, FPLLL
>>> A = IntegerMatrix(11, 11)
>>> M = GSO.Mat(A)
>>> M.transform_enabled
False
```

```
>>> U = IntegerMatrix.identity(11)
>>> M = GSO.Mat(A, U=U)
```

```
>>> M.transform_enabled
True
```

update_gso (self)

Updates all GSO coefficients (and r).

update_gso_row (self, int i, int last_j)

Updates $r_{i,j}$ and $d_{i,j}$ if needed for all $j \in [0, last_j]$. All coefficients of r and d above the i -th row in columns $[0, \min(last_j, i - 1)]$ must be valid.

Parameters

- **i** (*int*) –
- **last_j** (*int*) –

class fpylll.fplll.gso.MatGSORowOpContext (M, i, j)

A context in which performing row operations is safe. When the context is left, the appropriate updates are performed by calling `row_op_end()`.

__init__ (self, M, i, j)

Construct new context for $M[i:j]$.

Parameters

- **M** – MatGSO object
- **i** – start row
- **j** – stop row

2.1.3 LLL Wrapper

class fpylll.fplll.wrapper.Wrapper (IntegerMatrix B, double delta=LLL_DEF_DELTA, double eta=LLL_DEF_ETA, int flags=LLL_DEFAULT)**B****U****UinvT****__call__ ()**

Run LLL.

Returns**Return type**

```
>>> from fpylll import LLL, IntegerMatrix, GSO
>>> A = IntegerMatrix(40, 40)
>>> A.randomize("ntrulike", bits=10, q=1023)
>>> W = LLL.Wrapper(A)
>>> W()
```

__init__ ()

FIXME! briefly describe function

Parameters

- **B** (`IntegerMatrix`) –
- **delta** (`double`) –
- **eta** (`double`) –
- **flags** (`int`) –

```
>>> from fpylll import LLL, IntegerMatrix
>>> A = IntegerMatrix(50, 50)
>>> A.randomize("ntrulike", bits=100, q=1023)
>>> W = LLL.Wrapper(A)
```

status

2.1.4 LLL

class fpylll.fplll.lll.LLL**DEFAULT** = 0**DEFAULT_DELTA** = 0.99**DEFAULT_ETA** = 0.51**EARLY_RED** = 2**Reduction**alias of `LLLReduction`**SIEGEL** = 4**VERBOSE** = 1**class** **Wrapper** (*IntegerMatrix B*, *double delta=LLL_DEF_DELTA*, *double eta=LLL_DEF_ETA*, *int flags=LLL_DEFAULT*)**B****U****UinvT****__call__()**

Run LLL.

Returns**Return type**

```
>>> from fpylll import LLL, IntegerMatrix, GSO
>>> A = IntegerMatrix(40, 40)
>>> A.randomize("ntrulike", bits=10, q=1023)
>>> W = LLL.Wrapper(A)
>>> W()
```

__init__()

FIXME! briefly describe function

Parameters

- **B** (`IntegerMatrix`) –
- **delta** (`double`) –

- **eta** (*double*) –
- **flags** (*int*) –

```
>>> from fpylll import LLL, IntegerMatrix
>>> A = IntegerMatrix(50, 50)
>>> A.randomize("ntrulike", bits=100, q=1023)
>>> W = LLL.Wrapper(A)
```

status**static is_reduced**(*M*, *delta*=0.99, *eta*=0.51)is_LLL_reduced(*M*, *delta*=LLL_DEF_DELTA, *eta*=LLL_DEF_ETA) Test if *M* is LLL reduced.**param M** either an GSO object or an integer matrix.**param delta** LLL parameter < 1**param eta** LLL parameter > 0.5**returns** Return `True` if *M* is definitely LLL reduced, `False` otherwise.

Random matrices are typically not LLL reduced:

```
>>> from fpylll import IntegerMatrix, LLL
>>> A = IntegerMatrix(40, 40)
>>> A.randomize('uniform', bits=32)
>>> LLL.is_reduced(A)
False
```

LLL reduction should produce matrices which are LLL reduced:

```
>>> LLL.reduction(A)
<IntegerMatrix(40, 40) at 0x...>
>>> LLL.is_reduced(A)
True
```

Note: This function may return `False` for LLL reduced matrices if the precision used to compute the GSO is too small.

static reduction(*B*, *U*=None, *delta*=0.99, *eta*=0.51, *method*=None, *float_type*=None, *precision*=0, *flags*=0)
LLL_reduction(IntegerMatrix *B*, *U*=None, double *delta*=LLL_DEF_DELTA, double *eta*=LLL_DEF_ETA, method=None, float_type=None, int precision=0, int flags=LLL_DEFAULT) Run LLL reduction.**param IntegerMatrix B** Integer matrix, modified in place.**param U** Transformation matrix or None**param double delta** LLL parameter $0.25 < 1$ **param double eta** LLL parameter $0 <$ **param method** one of 'wrapper', 'proved', 'heuristic', 'fast' or None.**param float_type** an element of `fpylll.float_types` or `None`**param precision** bit precision to use if *float_type* is 'mpfr'**param int flags** LLL flags.**returns** modified matrix *B*

```
class fpylll.fplll.lll.LLLReduction(MatGSO M, double delta=LLL_DEF_DELTA, double
                                     eta=LLL_DEF_ETA, int flags=LLL_DEFAULT)
```

M

__call__()

LLL reduction.

Parameters

- **kappa_min** (*int*) – minimal index to go back to
- **kappa_start** (*int*) – index to start processing at
- **kappa_end** (*int*) – end index (exclusive)
- **size_reduction_start** (*int*) – only perform size reductions using vectors starting at this index

__init__()

Construct new LLL object.

Parameters

- **M** ([MatGSO](#)) –
- **delta** (*double*) –
- **eta** (*double*) –
- **flags** (*int*) –
 - DEFAULT:
 - VERBOSE:
 - EARLY_RED:
 - SIEGEL:

delta

eta

final_kappa

FIXME! briefly describe function

Returns

Return type

last_early_red

FIXME! briefly describe function

Returns

Return type

nswaps

FIXME! briefly describe function

Returns

Return type

size_reduction (*self, int kappa_min=0, int kappa_end=-1, int size_reduction_start=0*)

Size reduction.

Parameters

- **kappa_min** (*int*) – start index
- **kappa_end** (*int*) – end index (exclusive)
- **size_reduction_start** (*int*) – only perform size reductions using vectors starting at this index

zeros

FIXME! briefly describe function

Returns**Return type**

`fpylll.fplll.lll.is_LLL_reduced(M, delta=LLL_DEF_DELTA, eta=LLL_DEF_ETA)`

Test if M is LLL reduced.

Parameters

- **M** – either an GSO object or an integer matrix or an integer matrix.
- **delta** – LLL parameter < 1
- **eta** – LLL parameter > 0.5

Returns Return `True` if M is definitely LLL reduced, `False` otherwise.

Random matrices are typically not LLL reduced:

```
>>> from fpylll import IntegerMatrix, LLL
>>> A = IntegerMatrix(40, 40)
>>> A.randomize('uniform', bits=32)
>>> LLL.is_reduced(A)
False
```

LLL reduction should produce matrices which are LLL reduced:

```
>>> LLL.reduction(A)
<IntegerMatrix(40, 40) at 0x...>
>>> LLL.is_reduced(A)
True
```

Note: This function may return `False` for LLL reduced matrices if the precision used to compute the GSO is too small.

`fpylll.fplll.lll.lll_reduction(IntegerMatrix B, U=None, double delta=LLL_DEF_DELTA, double eta=LLL_DEF_ETA, method=None, float_type=None, int precision=0, int flags=LLL_DEFAULT)`

Run LLL reduction.

Parameters

- **B** (`IntegerMatrix`) – Integer matrix, modified in place.
- **U** – Transformation matrix or `None`
- **delta** (`double`) – LLL parameter $0.25 < 1$
- **eta** (`double`) – LLL parameter $0 <$
- **method** – one of '`wrapper`', '`proved`', '`heuristic`', '`fast`' or `None`.
- **float_type** – an element of `fpylll.float_types` or `None`

- **precision** – bit precision to use if `float_tpe` is 'mpfr'
- **flags** (`int`) – LLL flags.

Returns modified matrix `B`

2.1.5 BKZ

2.1.6 SVP and CVP

2.1.7 Pruning

2.1.8 Enumeration

```
class fpylll.fplll.enumeration.Enumeration
```

M

```
enumerate(self, int first, int last, max_dist, max_dist_expo, target=None, subtree=None, pruning=None, dual=False, subtree_reset=False)
```

Run enumeration on `M`

Parameters

- **first** (`int`) – first row
- **last** (`int`) – last row (exclusive)
- **max_dist** – length bound
- **max_dist_expo** – exponent of length bound
- **target** – target coordinates for CVP/BDD or `None` for SVP
- **subtree** –
- **pruning** – pruning parameters
- **dual** – run enumeration in the primal or dual lattice.
- **subtree_reset** –

Returns list of pairs containing the solutions' coefficient vectors and their lengths

```
get_nodes(self)
```

Return number of visited nodes in last enumeration call.

```
sub_solutions
```

Return sub-solutions computed in last enumeration call.

```
>>> from fpylll import *
>>> FPLLL.set_random_seed(1337)
>>> _ = FPLLL.set_threads(1)
>>> A = IntegerMatrix.random(80, "qary", bits=30, k=40)
>>> _ = LLL.reduction(A)
>>> M = GSO.Mat(A)
>>> _ = M.update_gso()
>>> pruning = Pruning.run(M.get_r(0, 0), 2**40, M.r()[:30], 0.8)
>>> enum = Enumeration(M, strategy=EvaluatorStrategy.BEST_N_SOLUTIONS, sub_
->solutions=True)
```

(continues on next page)

(continued from previous page)

```
>>> _ = enum.enumerate(0, 30, 0.999*M.get_r(0, 0), 0, pruning=pruning.
>>> coefficients)
>>> [int(round(a)) for a,b in enum.sub_solutions[:5]]
[5569754193, 5556022462, 5083806188, 5022873440, 4260865083]
```

exception fpylll.fplll.enumeration.**EnumerationError**

class fpylll.fplll.enumeration.**EvaluatorStrategy**

Strategies to update the enumeration radius and deal with multiple solutions. Possible values are:

- **BEST_N_SOLUTIONS** Starting with the nr_solutions-th solution, every time a new solution is found the enumeration bound is updated to the length of the longest solution. If more than nr_solutions were found, the longest is dropped.
- **OPPORTUNISTIC_N_SOLUTIONS** Every time a solution is found, update the enumeration distance to the length of the solution. If more than nr_solutions were found, the longest is dropped.
- **FIRST_N_SOLUTIONS** The enumeration bound is not updated. As soon as nr_solutions are found, enumeration stops.

BEST_N_SOLUTIONS = 0

FIRST_N_SOLUTIONS = 2

OPPORTUNISTIC_N_SOLUTIONS = 1

2.1.9 Utilities

class fpylll.util.**FPLLL**

static get_precision(*float_type='mpfr'*)

Get currently set precision

Parameters float_type – one of ‘double’, ‘long double’, ‘dpe’, ‘dd’, ‘qd’ or ‘mpfr’

Returns precision in bits

This function returns the precision per type:

```
>>> import fpylll
>>> from fpylll import FPLLL
>>> FPLLL.get_precision('double')
53
>>> if fpylll.config.have_long_double:
...     FPLLL.get_precision('long double') > 53
... else:
...     True
True
>>> FPLLL.get_precision('dpe')
53
```

For the MPFR type different precisions are supported:

```
>>> _ = FPLLL.set_precision(212)
>>> FPLLL.get_precision('mpfr')
212
>>> FPLLL.get_precision()
```

(continues on next page)

(continued from previous page)

```
212
>>> _ = FPLLL.set_precision(53)
```

static get_threads()
Get the number of threads.

static randint(a, b)
Return random integer in range [a, b], including both end points.

static set_external_enumerator(enumerator)
Set an external enumeration library.

For example, assume you compiled a fpyll-extenum

First, we load the required Python modules: fpyll and ctypes

```
>>> from fpyll import *
>>> import ctypes
```

Then, using ctypes we dlopen enumlib.so

```
>>> enumlib = ctypes.cdll.LoadLibrary("enumlib.so")
```

For demonstration purposes we increase the loglevel. Note that functions names are result of C++ compiler name mangling and may differ depending on platform/compiler/linker.

```
>>> enumlib._Z20enumlib_set_logleveli(1)
```

We grab the external enumeration function

```
>>> fn = enumlib._Z17enumlib_enumerateidSt8functionIFvPdmbS0_S0_EES_IFdds0_
    ↵EES_IFvdS0_iEEbb
```

and pass it to Fplll

```
>>> FPLLL.set_external_enumerator(fn)
```

To disable the external enumeration library, call

```
>>> FPLLL.set_external_enumerator(None)
```

static set_precision(unsigned int prec)

Set precision globally for MPFR

Parameters **prec** – an integer ≥ 53

Returns current precision

static set_random_seed(unsigned long seed)

Set random seed.

Parameters **seed** – a new seed.

static set_threads(int th=1)

Set the number of threads.

Parameters **th** – number of threads

This is currently only used for enumeration.

```
class fpylll.util.PrecisionContext(prec)

    __init__(self, prec)
        Create new precision context.

        Parameters prec – internal precision

exception fpylll.util.ReductionError

fpylll.util.adjust_radius_to_gh_bound(double dist, int dist_expo, int block_size, double
                                         root_det, double gh_factor)
    Use Gaussian Heuristic to reduce bound on the length of the shortest vector.

    Parameters

        • dist (double) – norm of shortest vector
        • dist_expo (int) – exponent of norm (for dpe representation)
        • block_size (int) – block size
        • root_det (double) – root determinant
        • gh_factor (double) – factor to multiply with

    Returns (dist, expo)

fpylll.util.ball_log_vol(n)
    Return volume of n-dimensional unit ball

    Parameters n – dimension

fpylll.util.gaussian_heuristic(r)
    Return squared norm of shortest vector as predicted by the Gaussian heuristic.

    Parameters r – vector of squared Gram-Schmidt norms

fpylll.util.get_precision(float_type='mpfr')
    Get currently set precision

    Parameters float_type – one of ‘double’, ‘long double’, ‘dpe’, ‘dd’, ‘qd’ or ‘mpfr’

    Returns precision in bits
```

This function returns the precision per type:

```
>>> import fpylll
>>> from fpylll import FPLLL
>>> FPLLL.get_precision('double')
53
>>> if fpylll.config.have_long_double:
...     FPLLL.get_precision('long double') > 53
... else:
...     True
True
>>> FPLLL.get_precision('dpe')
53
```

For the MPFR type different precisions are supported:

```
>>> _ = FPLLL.set_precision(212)
>>> FPLLL.get_precision('mpfr')
212
```

(continues on next page)

(continued from previous page)

```
>>> FPLLL.get_precision()
212
>>> _ = FPLLL.set_precision(53)
```

`fpylll.util.get_threads()`

Get the number of threads.

`fpylll.util.precision(prec)`

Create new precision context.

Parameters `prec` – internal precision

`fpylll.util.randint(a, b)`

Return random integer in range [a, b], including both end points.

`fpylll.util.set_external_enumerator(enumerator)`

Set an external enumeration library.

For example, assume you compiled a `fplll-extenum`

First, we load the required Python modules: `fpylll` and `ctypes`

```
>>> from fpylll import *
>>> import ctypes
```

Then, using `ctypes` we `dlopen enumlib.so`

```
>>> enumlib = ctypes.cdll.LoadLibrary("enumlib.so")
```

For demonstration purposes we increase the loglevel. Note that functions names are result of C++ compiler name mangling and may differ depending on platform/compiler/linker.

```
>>> enumlib._Z20enumlib_set_logleveli(1)
```

We grab the external enumeration function

```
>>> fn = enumlib._Z17enumlib_enumerateidSt8functionIFvPdmbS0_S0_EES_IFddS0_EES_
    ↵_IFvdS0_iEEbb
```

and pass it to `Fplll`

```
>>> FPLLL.set_external_enumerator(fn)
```

To disable the external enumeration library, call

```
>>> FPLLL.set_external_enumerator(None)
```

`fpylll.util.set_precision(unsigned int prec)`

Set precision globally for MPFR

Parameters `prec` – an integer ≥ 53

Returns current precision

`fpylll.util.set_random_seed(unsigned long seed)`

Set random seed.

Parameters `seed` – a new seed.

`fpylll.util.set_threads (int th=1)`

Set the number of threads.

Parameters `th` – number of threads

This is currently only used for enumeration.

2.2 Python Algorithms

The modules in this category extend the functionality of fplll in some way by implementing algorithms in Python.

2.2.1 Simple BKZ

2.2.2 Simple Dual BKZ

2.2.3 BKZ

CHAPTER
THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

`fpylll.fplll.enumeration`, 33
`fpylll.fplll.gso`, 19
`fpylll.fplll.integer_matrix`, 7
`fpylll.fplll.lll`, 29
`fpylll.fplll.wrapper`, 28
`fpylll.util`, 34

INDEX

Symbols

`__call__()` (*fpylll.fplll.lll.LLL.Wrapper method*), 29
`__call__()` (*fpylll.fplll.lll.LLLReduction method*), 31
`__call__()` (*fpylll.fplll.wrapper.Wrapper method*), 28
`__copy__()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 7
`__getitem__()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 7
`__getitem__()` (*fpylll.fplll.integer_matrix.IntegerMatrixRow method*), 18
`__init__()` (*fpylll.fplll.gso.MatGSO method*), 21
`__init__()` (*fpylll.fplll.gso.MatGSORowOpContext method*), 28
`__init__()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 7
`__init__()` (*fpylll.fplll.integer_matrix.IntegerMatrixRow method*), 18
`__init__()` (*fpylll.fplll.lll.LLL.Wrapper method*), 29
`__init__()` (*fpylll.fplll.lll.LLLReduction method*), 31
`__init__()` (*fpylll.fplll.wrapper.Wrapper method*), 28
`__init__()` (*fpylll.util.PrecisionContext method*), 36
`__setitem__()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 8

A

`addmul()` (*fpylll.fplll.integer_matrix.IntegerMatrixRow method*), 18
`adjust_radius_to_gh_bound()` (*in module fpylll.util*), 36
`apply_transform()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 8

B

`B` (*fpylll.fplll.gso.MatGSO attribute*), 20
`B` (*fpylll.fplll.lll.LLL.Wrapper attribute*), 29
`B` (*fpylll.fplll.wrapper.Wrapper attribute*), 28
`babai()` (*fpylll.fplll.gso.MatGSO method*), 21
`ball_log_vol()` (*in module fpylll.util*), 36
`BEST_N_SOLUTIONS` (*fpylll.fplll.enumeration.EvaluatorStrategy attribute*), 34

C

`clear()` (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 9
`create_row()` (*fpylll.fplll.gso.MatGSO method*), 22

D

`alpha` (*fpylll.fplll.gso.MatGSO attribute*), 22
`DEFAULT` (*fpylll.fplll.gso.GSO attribute*), 20
`DEFAULT` (*fpylll.fplll.lll.LLL attribute*), 29
`DEFAULT_DELTA` (*fpylll.fplll.lll.LLL attribute*), 29
`DEFAULT_ETA` (*fpylll.fplll.lll.LLL attribute*), 29
`delta` (*fpylll.fplll.lll.LLLReduction attribute*), 31
`discover_all_rows()` (*fpylll.fplll.gso.MatGSO method*), 22

E

`EARLY_RED` (*fpylll.fplll.lll.LLL attribute*), 29
`enumerate()` (*fpylll.fplll.enumeration.Enumeration method*), 33
`Enumeration` (*class in fpylll.fplll.enumeration*), 33
`EnumerationError`, 34
`eta` (*fpylll.fplll.lll.LLLReduction attribute*), 31
`EvaluatorStrategy` (*class in fpylll.fplll.enumeration*), 34

F

`final_kappa` (*fpylll.fplll.lll.LLLReduction attribute*), 31
`FIRST_N_SOLUTIONS` (*fpylll.fplll.enumeration.EvaluatorStrategy attribute*), 34
`float_type` (*fpylll.fplll.gso.MatGSO attribute*), 22
`FPLLL` (*class in fpylll.util*), 34
`fpylll.fplll.enumeration`
`module`, 33
`fpylll.fplll.gso`
`module`, 19
`fpylll.fplll.integer_matrix`
`module`, 7
`fpylll.fplll.lll`
`module`, 29
`fpylll.fplll.wrapper`

```

    module, 28
fpylll.util
    module, 34
from_canonical()          (fpylll.fplll.gso.MatGSO
    method), 22
from_file() (fpylll.fplll.integer_matrix.IntegerMatrix
    class method), 9
from_iterable() (fpylll.fplll.integer_matrix.IntegerMatrix
    class method), 9
from_matrix() (fpylll.fplll.integer_matrix.IntegerMatrix
    class method), 9

```

G

- G (fpylll.fplll.gso.MatGSO attribute), 20
- gaussian_heuristic () (in module fpylll.util), 36
- gen_identity () (fpylll.fplll.integer_matrix.IntegerMatrix
 method), 9
- get_current_slope () (fpylll.fplll.gso.MatGSO
 method), 23
- get_gram () (fpylll.fplll.gso.MatGSO method), 23
- get_log_det () (fpylll.fplll.gso.MatGSO method), 23
- get_max_exp () (fpylll.fplll.integer_matrix.IntegerMatrix
 method), 9
- get_mu () (fpylll.fplll.gso.MatGSO method), 23
- get_mu_exp () (fpylll.fplll.gso.MatGSO method), 23
- get_nodes () (fpylll.fplll.enumeration.Enumeration
 method), 33
- get_precision () (fpylll.util.FPLLL static method),
 34
- get_precision () (in module fpylll.util), 36
- get_r () (fpylll.fplll.gso.MatGSO method), 24
- get_r_exp () (fpylll.fplll.gso.MatGSO method), 24
- get_root_det () (fpylll.fplll.gso.MatGSO method),
 24
- get_slide_potential () (fpylll.fplll.gso.MatGSO
 method), 24
- get_threads () (fpylll.util.FPLLL static method), 35
- get_threads () (in module fpylll.util), 37
- GSO (class in fpylll.fplll.gso), 20

I

- identity () (fpylll.fplll.integer_matrix.IntegerMatrix
 class method), 10
- INT_GRAM (fpylll.fplll.gso.GSO attribute), 20
- int_gram_enabled (fpylll.fplll.gso.MatGSO
 attribute), 24
- int_type (fpylll.fplll.gso.MatGSO attribute), 25
- int_type (fpylll.fplll.integer_matrix.IntegerMatrix
 attribute), 10
- IntegerMatrix (class in fpylll.fplll.integer_matrix), 7
- IntegerMatrixRow (class
 in fpylll.fplll.integer_matrix), 18
- inverse_transform_enabled
 (fpylll.fplll.gso.MatGSO attribute), 25

L

- is_empty () (fpylll.fplll.integer_matrix.IntegerMatrix
 method), 10
- is_LLL_reduced () (in module fpylll.fplll.lll), 32
- is_reduced () (fpylll.fplll.lll.LLL static method), 30
- is_zero () (fpylll.fplll.integer_matrix.IntegerMatrixRow
 method), 18

M

- last_early_red (fpylll.fplll.lll.LLLReduction
 attribute), 31
- LLL (class in fpylll.fplll.lll), 29
- LLL.Wrapper (class in fpylll.fplll.lll), 29
- lll_reduction () (in module fpylll.fplll.lll), 32
- LLLReduction (class in fpylll.fplll.lll), 30

N

- ncols (fpylll.fplll.integer_matrix.IntegerMatrix
 attribute), 11
- negate_row () (fpylll.fplll.gso.MatGSO method), 25
- norm () (fpylll.fplll.integer_matrix.IntegerMatrixRow
 method), 19
- nrows (fpylll.fplll.integer_matrix.IntegerMatrix
 attribute), 11
- nswaps (fpylll.fplll.lll.LLLReduction attribute), 31

O

- OP_FORCE_LONG (fpylll.fplll.gso.GSO attribute), 20
- OPPORTUNISTIC_N_SOLUTIONS
 (fpylll.fplll.enumeration.EvaluatorStrategy
 attribute), 34

P

- precision () (in module fpylll.util), 37
- PrecisionContext (class in fpylll.util), 35

R

r () (*fpylll.fplll.gso.MatGSO method*), 26
randint () (*fpylll.util.FPLLL static method*), 35
randint () (*in module fpylll.util*), 37
random() (*fpylll.fplll.integer_matrix.IntegerMatrix class method*), 11
randomize() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 14
Reduction (*fpylll.fplll.lll.LLL attribute*), 29
reduction() (*fpylll.fplll.lll.LLL static method*), 30
ReductionError, 36
remove_last_row() (*fpylll.fplll.gso.MatGSO method*), 26
resize() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 14
rotate() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 14
rotate_gram_left() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 14
rotate_gram_right() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 14
rotate_left() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 15
rotate_right() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 15
row_addmull() (*fpylll.fplll.gso.MatGSO method*), 26
ROW_EXPO (*fpylll.fplll.gso.GSO attribute*), 20
row_expo_enabled (*fpylll.fplll.gso.MatGSO attribute*), 26
row_op_begin() (*fpylll.fplll.gso.MatGSO method*), 26
row_op_end() (*fpylll.fplll.gso.MatGSO method*), 26
row_op_force_long (*fpylll.fplll.gso.MatGSO attribute*), 27
row_ops() (*fpylll.fplll.gso.MatGSO method*), 27

S

set_cols() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 15
set_external_enumerator() (*fpylll.util.FPLLL static method*), 35
set_external_enumerator() (*in module fpylll.util*), 37
set_iterable() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 15
set_matrix() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 16
set_precision() (*fpylll.util.FPLLL static method*), 35
set_precision() (*in module fpylll.util*), 37
set_random_seed() (*fpylll.util.FPLLL static method*), 35
set_random_seed() (*in module fpylll.util*), 37
set_rows() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 16
set_threads() (*fpylll.util.FPLLL static method*), 35
set_threads() (*in module fpylll.util*), 37
SIEGEL (*fpylll.fplll.lll.LLL attribute*), 29
size_nz() (*fpylll.fplll.integer_matrix.IntegerMatrixRow method*), 19
size_reduction() (*fpylll.fplll.lll.LLLReduction method*), 31
status (*fpylll.fplll.lll.LLL.Wrapper attribute*), 30
status (*fpylll.fplll.wrapper.Wrapper attribute*), 29
sub_solutions (*fpylll.fplll.enumeration.Enumeration attribute*), 33
submatrix() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 16
swap_rows() (*fpylll.fplll.gso.MatGSO method*), 27
swap_rows() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 17

T

to_canonical() (*fpylll.fplll.gso.MatGSO method*), 27
to_matrix() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 17
transform_enabled (*fpylll.fplll.gso.MatGSO attribute*), 27
transpose() (*fpylll.fplll.integer_matrix.IntegerMatrix method*), 17

U

U (*fpylll.fplll.gso.MatGSO attribute*), 21
U (*fpylll.fplll.lll.LLL.Wrapper attribute*), 29
U (*fpylll.fplll.wrapper.Wrapper attribute*), 28
UinvT (*fpylll.fplll.gso.MatGSO attribute*), 21
UinvT (*fpylll.fplll.lll.LLL.Wrapper attribute*), 29
UinvT (*fpylll.fplll.wrapper.Wrapper attribute*), 28
unpickle_IntegerMatrix() (*in module fpylll.fplll.integer_matrix*), 19
update_gso() (*fpylll.fplll.gso.MatGSO method*), 28
update_gso_row() (*fpylll.fplll.gso.MatGSO method*), 28

V

VERBOSE (*fpylll.fplll.lll.LLL attribute*), 29

W

Wrapper (*class in fpylll.fplll.wrapper*), 28

Z

zeros (*fpylll.fplll.lll.LLLReduction attribute*), 32