fpylll Documentation

Release 0.2.4dev

Martin R. Albrecht

Jun 03, 2017

Contents

1 fpylll **1**
- 1.1 Requirements . . . 1
- 1.2 Online . . . 2
- 1.3 Getting Started . . . 2
- 1.4 Multicore Support . . . 3
- 1.5 Contributing . . . 4
- 1.6 Attribution & License . . . 4

2 Modules **7**
- 2.1 fplll Modules . . . 7
- 2.2 Python Algorithms . . . 36

3 Indices and Tables **41**

Python Module Index **43**

CHAPTER 1

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).

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

- IPython for interacting with Python
- Numpy for numerical computations (e.g. with Gram-Schmidt values)

Online

fpylll ships with Sage 7.4. Thus, it is available via SageMathCell and SageMathCloud (select a Jupyter notebook with a Sage 7.4 kernel, the default Sage worksheet still runs Sage 7.3 at the time of writing). You can also fire up a dply.co virtual server with the latest fpylll/fplll preinstalled (it takes perhaps 15 minutes until everything is compiled).

Getting Started

Note: fpylll 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.

1. Create a new virtualenv and activate it:

   ```
   $ virtualenv env
   $ source ./env/bin/activate
   ```

   We indicate active virtualenvs by the prefix `(fpylll)`.

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
   ```

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.

8. Start Python:

```
$ (fpylll) ipython
```

To reactivate the virtual environment later, simply run:

```
$ source ./env/bin/activate
```

Note that you can also patch `activate` to set `LD_LIBRRY_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.

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{0} done: {1:.2%}'.format(prefix, float(i)/len(A)))
        sys.stderr.flush()
    sys.stderr.write('\r{0} done {1:.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.

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.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.

Attribution & License

fpylll is maintained by Martin Albrecht.

The following people have contributed to **fpylll**

- Martin Albrecht
- Guillaume Bonnoron
- Jeroen Demeyer

- Leo Ducas
- Omer Katz

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

CHAPTER 2

Modules

fplll Modules

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

Integer Matrices

Integer matrices.

class `fpylll.fplll.integer_matrix.`**`IntegerMatrix`**(*arg0*, *arg1=None*)

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__

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
```

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*)
Construct new matrix from file.

Parameters **filename** – name of file to read from

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

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 ]
```

is_empty (*self*)

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

Apply moduluar 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*A'` where `A'` is `A` reduced to `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*, ***kwds*)
Construct new random matrix.

Seealso $IntegerMatrix.randomize$

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

Parameters **algorithm** – string, see below for choices.

Available algorithms:

- `"intrel"` - generate a knapsack like matrix of dimension `d x (d+1)` and `bits` 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"` - generate a `d x d` matrix of a form similar to that is involved when trying to find rational approximations to reals with the same small denominator. The first vector starts with a random integer of bit-length `<=bits2` and continues with `d-1` independent integers of bit-lengths `<=bits`; the i-th vector for `i>1` is the i-th canonical unit vector scaled by a factor `2^b`.
- `"uniform"` - generate a `d x d` matrix whose entries are independent integers of bit-lengths `<=bits`.
- `"ntrulike"` - generate an NTRU-like matrix. If `bits` is given, then it first samples an integer `q` of bit-length `<=bits`, 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 x 2 block matrix `[[I, Rot(h)], [0, q*I]]`, where each block is `d x d`, the first row of

`Rot(h)` is the coefficient vector of `h`, and the i-th row of `Rot(h)` is the shift of the (i-1)-th (with last entry put back in first position), for all i>1. Warning: this does not produce a genuine ntru lattice with h a genuine public key.

- `ntrulike2"` : as the previous option, except that the constructed matrix is `[[q*I, 0], [Rot(h), I]]`.
- `"qary"` : generate a q-ary matrix. If `bits` is given, then it first samples an integer `q` of bit-length <=`bits`; if `q` is provided, then set `q` to the provided value. It returns a `2 x 2` block matrix `[[q*I, 0], [H, I]]`, where `H` is `k x (d-k)` and uniformly random modulo q. These bases correspond to the SIS/LWE q-ary lattices. Goldstein-Mayer lattices correspond to `k=1` and `q` prime.
- `"trg"` - generate a `d x d` lower-triangular matrix `B` with `B_ii = 2^(d-i+1)^f` for all `i`, and `B_ij` is uniform between `-B_jj/2` and `B_jj/2` for all `j<i`.

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

> **Warning:** entries starting at `A[nrows * 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]

> **Warning:** entries starting at `A[nrows, 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, set_random_seed
>>> A = IntegerMatrix(10, 10)
>>> 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 ]
[ 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

transpose(*self*)
Transpose.

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

Parameters

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

```
>>> 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`

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

norm

Return _2 norm of this vector.

```
>>> 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.

```
>>> 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*)

Deserialize an integer matrix.

Parameters

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

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: $B = RQ = (D)(D^{-1}B^*)$ 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'*)

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

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_addmul`. See its documentation.

- **float_type** – A floating point type, i.e. an element of `fpylll.fpylll.float_types`.

Note: 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.

---

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_tranform=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, set_precision
>>> 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, set_precision
>>> A = IntegerMatrix(10, 10)
>>> M = GSO.Mat(A)
>>> M.float_type
'double'
>>> 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 $\mathbb{Z}^n$ 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, b_j. If `enable_row_expo` is true, returns $b_i, b_j / 2^{(r_i + r_j)}$, 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 $< b_i, b_j^* > / ||b_j^*||^2$.

Parameters

- **i** –
- **j** –

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

Return $f =_{i,j}$ and exponent x such that $f2^x = b_i, b_j^*/b_j^{*2}$. If `enable_row_expo` is false, x is always zero. If `enable_row_expo` is true, $x = r_i - r_j$, 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 *
>>> A = IntegerMatrix.random(5, "uniform", bits=5)
>>> M = GSO.Mat(A)
>>> M.update_gso()
True
>>> M.get_r(1, 0)
833.0
```

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

Return $f = r_{i,j}$ and exponent x such that $b_i, b_j^* = f2^x$. If `enable_row_expo` is false, x is always 0. If `enable_row_expo` is true, $x = r_i + r_j$, 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, set_precision
>>> 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
```

inverse_transform_enabled
Computation of the inverse transform matrix (transposed).

```
>>> from fpylll import IntegerMatrix, GSO, set_precision
>>> 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 to $-b_i$.

Parameters **i** (*int*) – index of the row to negate

Example:

```
>>> from fpylll import *
>>> 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)
>>> 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   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^expo * long`) instead of (`2^expo * ZT`), which is faster if `ZT=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, set_precision
>>> 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, set_precision
>>> 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).

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, set_precision
>>> 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 $_{i,j}$ if needed for all j in $[0, last_j]$. All coefficients of r and 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

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

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 of an integer matrix or an integer matrix.

param delta LLL parameter $\delta < 1$

param eta LLL parameter $\eta > 0.5$

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_tpe` 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__
FIXME! briefly describe function

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 of an integer matrix or an integer matrix.
- **delta** – LLL parameter $\delta < 1$
- **eta** – LLL parameter $\eta > 0.5$

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`

BKZ

Block Korkine Zolotarev algorithm.

class `fpylll.fplll.bkz.`**BKZ**

AUTO_ABORT = 32

AutoAbort
alias of *BKZAutoAbort*

BOUNDED_LLL = 16

DEFAULT = 0

DEFAULT_AUTO_ABORT_MAX_NO_DEC = 5

DEFAULT_AUTO_ABORT_SCALE = 1.0

DEFAULT_GH_FACTOR = 1.1

DEFAULT_MIN_SUCCESS_PROBABILITY = 0.5

DEFAULT_RERANDOMIZATION_DENSITY = 3

DEFAULT_STRATEGY = '/home/docs/checkouts/readthedocs.org/user_builds/fpylll/conda/latest/share/fplll/strategies/defau

DEFAULT_STRATEGY_PATH = '/home/docs/checkouts/readthedocs.org/user_builds/fpylll/conda/latest/share/fplll/strategie

DUMP_GSO = 64

GH_BND = 128

MAX_LOOPS = 4

MAX_TIME = 8

NO_LLL = 2

Param
alias of `BKZParam`

Reduction
alias of *BKZReduction*

SD_VARIANT = 256

SLD_RED = 512

VERBOSE = 1

static reduction(*B*, *o*, *float_type=None*, *precision=0*)
bkz_reduction(IntegerMatrix B, BKZParam o, float_type=None, int precision=0)

Run BKZ reduction.

Parameters

- **B** (`IntegerMatrix`) – Integer matrix, modified in place.
- **o** (*BKZParam*) – BKZ parameters

- **float_type** – either `None`: for automatic choice or an entry of $fpylll.float_types$
- **precision** – bit precision to use if `float_tpe` is `'mpfr'`

Returns modified matrix `B`

class `fpylll.fplll.bkz.`**BKZAutoAbort**(*MatGSO M*, *int num_rows*, *int start_row=0*)
Utility class for aborting BKZ when slope does not improve any longer.

__init__
Create new auto abort object.

Parameters

- **M** – GSO matrix
- **num_rows** – number of rows
- **start_row** – start at this row

test_abort(*self*, *scale=1.0*, *int max_no_dec=5*)
Test if new slope fails to be smaller than $scale * old_slope$ for max_no_dec iterations.

Parameters

- **scale** – target decrease
- **max_no_dec** (*int*) – number of rounds allowed to be stuck

class `fpylll.fplll.bkz.`**BKZReduction**(*MatGSO M*, *LLLReduction lll_obj*, *BKZParam param*)

M

__call__
Call BKZ, SD-BKZ or slide reduction.

Note: To enable the latter, set flags `BKZ.SLD_RED` or `BKZ.SD_VARIANT` when calling the constructor of this class.

__init__
Construct new BKZ object.

Parameters

- **M** – GSO object
- **lll_obj** – LLL object called as a subroutine
- **param** – parameters

dsvp_postprocessing(*self*, *int kappa*, *int block_size*, *tuple solution*)
Insert solution into basis after Dual-SVP oracle call

Parameters

- **kappa** – index
- **block_size** – block size
- **solution** – solution to insert

hkz(*self*, *BKZParam param*, *int min_row*, *int max_row*)
HKZ reduction between `min_row` and `max_row`.

Parameters

- **param** – reduction parameters
- **min_row** – start row
- **max_row** – maximum row to consider (exclusive)

Returns `True` if no changes were made, `False` otherwise.

lll_obj

nodes
Total number of enumeration nodes visited during this reduction.

param

rerandomize_block(*self*, *int min_row*, *int max_row*, *int density*)
Rerandomize block between `min_row` and `max_row` with a transform of `density`

Parameters

- **min_row** –
- **max_row** –
- **density** –

sd_tour(*self*, *int loop*, *BKZParam param*, *int min_row*, *int max_row*)
One Dual-BKZ tour.

Parameters

- **loop** – loop index
- **param** – reduction parameters
- **min_row** – start row
- **max_row** – maximum row to consider (exclusive)

Returns `True` if no changes were made, `False` otherwise.

slide_tour(*self*, *int loop*, *BKZParam param*, *int min_row*, *int max_row*)
One slide reduction tour.

Parameters

- **loop** – loop index
- **param** – reduction parameters
- **min_row** – start row
- **max_row** – maximum row to consider (exclusive)

Returns `True` if no changes were made, `False` otherwise.

Note: You must run `lll_obj()` before calling this function, otherwise this function will produce an error.

---

status
Status of this reduction.

svp_postprocessing(*self*, *int kappa*, *int block_size*, *tuple solution*)
Insert solution into basis after SVP oracle call

Parameters

- **kappa** – index
- **block_size** – block size
- **solution** – solution to insert

svp_preprocessing(*self*, *int kappa*, *int block_size*, *BKZParam param*)
Preprocess before calling (Dual-)SVP oracle.

Parameters

- **kappa** – index
- **block_size** – block size
- **param** – reduction parameters

svp_reduction(*self*, *int kappa*, *int block_size*, *BKZParam param*, *dual=False*)
Run (Dual-)SVP reduction (incl. pre and postprocessing)

Parameters

- **kappa** – index
- **block_size** – block size
- **param** – reduction parameters
- **dual** – dual or primal reduction

tour(*self*, *int loop*, *BKZParam param*, *int min_row*, *int max_row*)
One BKZ tour.

Parameters

- **loop** – loop index
- **param** – reduction parameters
- **min_row** – start row
- **max_row** – maximum row to consider (exclusive)

Returns tuple `(clean, max_kappa)` where `clean == True` if no changes were made, and `max_kappa` is the maximum index for which no changes were made.

fpylll.fplll.bkz.**bkz_reduction**(*IntegerMatrix B*, *BKZParam o*, *float_type=None*, *int precision=0*)
Run BKZ reduction.

Parameters

- **B** (`IntegerMatrix`) – Integer matrix, modified in place.
- **o** (*BKZParam*) – BKZ parameters
- **float_type** – either `None`: for automatic choice or an entry of $fpylll.float_types$
- **precision** – bit precision to use if `float_tpe` is `'mpfr'`

Returns modified matrix `B`

SVP and CVP

Shortest and Closest Vectors.

class fpylll.fplll.svpcvp.**CVP**

DEFAULT = 0

VERBOSE = 1

static closest_vector(*IntegerMatrix B*, *target*, *int flags=CVP_DEFAULT*)
Return a closest vector.

Parameters

- **B** (IntegerMatrix) –
- **target** (*vector[Z_NR[mpz_t]]*) –
- **flags** (*int*) –

Returns coordinates of the solution vector

Rtype tuple

```
>>> from fpylll import *
>>> set_random_seed(42)
>>> A = IntegerMatrix.random(5, 'uniform', bits=10)
>>> lll = LLL.reduction(A)
>>> t = (94, -42, 123, 512, -1337)
>>> print (CVP.closest_vector(A, t))
(-34, 109, 204, 360, -1548)
```

class fpylll.fplll.svpcvp.**SVP**

DEFAULT = 0

OVERRIDE_BND = 2

VERBOSE = 1

static shortest_vector(*IntegerMatrix B*, *method=None*, *int flags=SVP_DEFAULT*, *pruning=None*, *run_lll=True*, *max_aux_sols=0*)
Return a shortest vector.

Parameters

- **B** (IntegerMatrix) –
- **method** –
- **flags** (*int*) –
- **pruning** –
- **run_lll** –
- **max_aux_sols** –

Returns

Return type

fpylll.fplll.svpcvp.**closest_vector**(*IntegerMatrix B*, *target*, *int flags=CVP_DEFAULT*)
Return a closest vector.

Parameters

- **B** (IntegerMatrix) –

- **target** (*vector[Z_NR[mpz_t]]*) –
- **flags** (*int*) –

Returns coordinates of the solution vector

Rtype tuple

```
>>> from fpylll import *
>>> set_random_seed(42)
>>> A = IntegerMatrix.random(5, 'uniform', bits=10)
>>> lll = LLL.reduction(A)
>>> t = (94, -42, 123, 512, -1337)
>>> print (CVP.closest_vector(A, t))
(-34, 109, 204, 360, -1548)
```

fpylll.fplll.svpcvp.**shortest_vector**(*IntegerMatrix B*, *method=None*, *int flags=SVP_DEFAULT*, *pruning=None*, *run_lll=True*, *max_aux_sols=0*)

Return a shortest vector.

Parameters

- **B** (IntegerMatrix) –
- **method** –
- **flags** (*int*) –
- **pruning** –
- **run_lll** –
- **max_aux_sols** –

Returns

Return type

Enumeration

class fpylll.fplll.enumeration.**Enumeration**(*MatGSO M*, *nr_solutions=1*, *strategy=EvaluatorStrategy.BEST_N_SOLUTIONS*)

M

__init__

Create new enumeration object

Parameters **M** (MatGSO) – GSO matrix

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 and their lengths

get_nodes(*self*)
Return number of visited nodes in last enumeration call.

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

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

BEST_N_SOLUTIONS = 0

FIRST_N_SOLUTIONS = 2

OPPORTUNISTIC_N_SOLUTIONS = 1

Utilities

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*)

fpylll.util.**gaussian_heuristic**(*r*)

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:

```
>>> from fpylll import get_precision, set_precision
>>> get_precision('double')
53
>>> get_precision('long double')
64
>>> get_precision('dpe')
53
```

For the MPFR type different precisions are supported:

```
>>> _ = set_precision(212)
>>> get_precision('mpfr')
212
>>> get_precision()
212
>>> _ = set_precision(53)
```

fpylll.util.**precision**(*prec*)
Create new precision context.

Parameters **prec** – internal precision

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.

Python Algorithms

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

Simple BKZ

A minimal implementation of the Block Korkine Zolotarev algorithm in Python.

class fpylll.algorithms.simple_bkz.**BKZReduction**(*A*)

__call__(*block_size*)
Perform BKZ reduction with given‘‘block_size‘‘.

Nothing is returned, the matrix A given during construction is modified in-place.

Parameters **block_size** – an integer > 2

__init__(*A*)
Construct a new BKZ reduction instance.

Parameters **A** – Integer matrix to reduce.

bkz_loop(*block_size*, *min_row*, *max_row*)
Perform one BKZ loop, often also called a "BKZ tour".

Parameters

- **block_size** – an integer > 2
- **min_row** – algorithm starts in this row (inclusive)
- **max_row** – algorithm stops at this row (exclusive)

svp_reduction(*kappa*, *block_size*)
Call the SVP oracle and insert found vector into basis.

Parameters

- **kappa** – row index
- **block_size** – an integer > 2

Simple Dual BKZ

class fpylll.algorithms.simple_dbkz.**DBKZReduction**(*A*)

bkz_loop(*block_size*, *min_row*, *max_row*)
FIXME! briefly describe function

Parameters

- **block_size** –
- **min_row** –
- **max_row** –

Returns

Return type

dsvp_reduction(*kappa*, *block_size*)
FIXME! briefly describe function

Parameters

- **kappa** –
- **block_size** –

Returns

Return type

euclid(*pair1*, *pair2*)
FIXME! briefly describe function

Parameters

- **pair1** –
- **pair2** –

Returns

Return type

BKZ

Block Korkine Zolotarev algorithm in Python.

This module reimplements fplll's BKZ algorithm in Python. It has feature parity with the C++ implementation in fplll's core. Additionally, this implementation collects some additional statistics. Hence, it should provide a good basis for implementing variants of this algorithm.

class fpylll.algorithms.bkz.**BKZReduction**(*A*)
An implementation of the BKZ algorithm in Python.

This class has feature parity with the C++ implementation in fplll's core. Additionally, this implementation collects some additional statistics. Hence, it should provide a good basis for implementing variants of this algorithm.

__call__(*params*, *min_row=0*, *max_row=-1*)
Run the BKZ algorithm with parameters $param$.

Parameters

- **params** – BKZ parameters
- **min_row** – start processing in this row
- **max_row** – stop processing in this row (exclusive)

__init__(*A*)
Construct a new instance of the BKZ algorithm.

Parameters **A** – an integer matrix, a GSO object or an LLL object

svp_call(*kappa*, *block_size*, *params*, *tracer=None*)
Call SVP oracle

Parameters

- **kappa** – current index
- **params** – BKZ parameters
- **block_size** – block size
- **tracer** – object for maintaining statistics

Returns Coordinates of SVP solution or `None` if none was found.

Note: `block_size` may be smaller than `params.block_size` for the last blocks.

---

svp_postprocessing(*kappa*, *block_size*, *solution*, *tracer*)
Insert SVP solution into basis and LLL reduce.

Parameters

- **solution** – coordinates of an SVP solution
- **kappa** – current index
- **block_size** – block size
- **tracer** – object for maintaining statistics

Returns `True` if no change was made and `False` otherwise

svp_preprocessing(*kappa*, *block_size*, *params*, *tracer*)
Perform preprocessing for calling the SVP oracle

Parameters

- **kappa** – current index
- **params** – BKZ parameters
- **block_size** – block size
- **tracer** – object for maintaining statistics

Returns `True` if no change was made and `False` otherwise

Note: `block_size` may be smaller than `params.block_size` for the last blocks.

---

svp_reduction(*kappa*, *block_size*, *params*, *tracer=<fpylll.algorithms.bkz_stats.Tracer object>*)
Find shortest vector in projected lattice of dimension `block_size` and insert into current basis.

Parameters

- **kappa** – current index
- **params** – BKZ parameters
- **block_size** – block size
- **tracer** – object for maintaining statistics

Returns `True` if no change was made and `False` otherwise

tour(*params*, *min_row=0*, *max_row=-1*, *tracer=<fpylll.algorithms.bkz_stats.Tracer object>*)
One BKZ loop over all indices.

Parameters

- **params** – BKZ parameters
- **min_row** – start index 0
- **max_row** – last index n

Returns `True` if no change was made and `False` otherwise

CHAPTER 3

Indices and Tables

- genindex
- modindex
- search

Python Module Index

f

fpylll.algorithms.bkz, 38
fpylll.algorithms.simple_bkz, 36
fpylll.algorithms.simple_dbkz, 37
fpylll.fplll.bkz, 29
fpylll.fplll.enumeration, 34
fpylll.fplll.gso, 17
fpylll.fplll.integer_matrix, 7
fpylll.fplll.lll, 25
fpylll.fplll.svpcvp, 32
fpylll.fplll.wrapper, 24
fpylll.util, 35

Index

Symbols

__call__ (fpylll.fplll.bkz.BKZReduction attribute), 30
__call__ (fpylll.fplll.lll.LLL.Wrapper attribute), 25
__call__ (fpylll.fplll.lll.LLLReduction attribute), 27
__call__ (fpylll.fplll.wrapper.Wrapper attribute), 24
__call__() (fpylll.algorithms.bkz.BKZReduction method), 38
__call__() (fpylll.algorithms.simple_bkz.BKZReduction method), 36
__copy__() (fpylll.fplll.integer_matrix.IntegerMatrix method), 7
__getitem__ (fpylll.fplll.integer_matrix.IntegerMatrix attribute), 7
__getitem__ (fpylll.fplll.integer_matrix.IntegerMatrixRow attribute), 15
__init__ (fpylll.fplll.bkz.BKZAutoAbort attribute), 30
__init__ (fpylll.fplll.bkz.BKZReduction attribute), 30
__init__ (fpylll.fplll.enumeration.Enumeration attribute), 34
__init__ (fpylll.fplll.gso.MatGSO attribute), 17
__init__ (fpylll.fplll.integer_matrix.IntegerMatrix attribute), 8
__init__ (fpylll.fplll.integer_matrix.IntegerMatrixRow attribute), 15
__init__ (fpylll.fplll.lll.LLL.Wrapper attribute), 25
__init__ (fpylll.fplll.lll.LLLReduction attribute), 27
__init__ (fpylll.fplll.wrapper.Wrapper attribute), 24
__init__() (fpylll.algorithms.bkz.BKZReduction method), 38
__init__() (fpylll.algorithms.simple_bkz.BKZReduction method), 36
__init__() (fpylll.fplll.gso.MatGSORowOpContext method), 24
__init__() (fpylll.util.PrecisionContext method), 35
__setitem__ (fpylll.fplll.integer_matrix.IntegerMatrix attribute), 8

A

addmul() (fpylll.fplll.integer_matrix.IntegerMatrixRow method), 15
adjust_radius_to_gh_bound() (in module fpylll.util), 35
apply_transform() (fpylll.fplll.integer_matrix.IntegerMatrix method), 9
AUTO_ABORT (fpylll.fplll.bkz.BKZ attribute), 29
AutoAbort (fpylll.fplll.bkz.BKZ attribute), 29

B

B (fpylll.fplll.gso.MatGSO attribute), 17
B (fpylll.fplll.lll.LLL.Wrapper attribute), 25
B (fpylll.fplll.wrapper.Wrapper attribute), 24
babai() (fpylll.fplll.gso.MatGSO method), 18
ball_log_vol() (in module fpylll.util), 35
BEST_N_SOLUTIONS (fpylll.fplll.enumeration.EvaluatorStrategy attribute), 35
BKZ (class in fpylll.fplll.bkz), 29
bkz_loop() (fpylll.algorithms.simple_bkz.BKZReduction method), 36
bkz_loop() (fpylll.algorithms.simple_dbkz.DBKZReduction method), 37
bkz_reduction() (in module fpylll.fplll.bkz), 32
BKZAutoAbort (class in fpylll.fplll.bkz), 30
BKZReduction (class in fpylll.algorithms.bkz), 38
BKZReduction (class in fpylll.algorithms.simple_bkz), 36
BKZReduction (class in fpylll.fplll.bkz), 30
BOUNDED_LLL (fpylll.fplll.bkz.BKZ attribute), 29

C

clear() (fpylll.fplll.integer_matrix.IntegerMatrix method), 9
closest_vector() (fpylll.fplll.svpcvp.CVP static method), 33
closest_vector() (in module fpylll.fplll.svpcvp), 33
create_row() (fpylll.fplll.gso.MatGSO method), 18
CVP (class in fpylll.fplll.svpcvp), 32

D

d (fpylll.fplll.gso.MatGSO attribute), 18

DBKZReduction (class in fpylll.algorithms.simple_dbkz), 37
DEFAULT (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT (fpylll.fplll.gso.GSO attribute), 17
DEFAULT (fpylll.fplll.lll.LLL attribute), 25
DEFAULT (fpylll.fplll.svpcvp.CVP attribute), 33
DEFAULT (fpylll.fplll.svpcvp.SVP attribute), 33
DEFAULT_AUTO_ABORT_MAX_NO_DEC (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_AUTO_ABORT_SCALE (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_DELTA (fpylll.fplll.lll.LLL attribute), 25
DEFAULT_ETA (fpylll.fplll.lll.LLL attribute), 25
DEFAULT_GH_FACTOR (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_MIN_SUCCESS_PROBABILITY (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_RERANDOMIZATION_DENSITY (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_STRATEGY (fpylll.fplll.bkz.BKZ attribute), 29
DEFAULT_STRATEGY_PATH (fpylll.fplll.bkz.BKZ attribute), 29
delta (fpylll.fplll.lll.LLLReduction attribute), 27
discover_all_rows() (fpylll.fplll.gso.MatGSO method), 18
dsvp_postprocessing() (fpylll.fplll.bkz.BKZReduction method), 30
dsvp_reduction() (fpylll.algorithms.simple_dbkz.DBKZReduction method), 37
DUMP_GSO (fpylll.fplll.bkz.BKZ attribute), 29

E

EARLY_RED (fpylll.fplll.lll.LLL attribute), 25
enumerate() (fpylll.fplll.enumeration.Enumeration method), 34
Enumeration (class in fpylll.fplll.enumeration), 34
EnumerationError, 35
eta (fpylll.fplll.lll.LLLReduction attribute), 27
euclid() (fpylll.algorithms.simple_dbkz.DBKZReduction method), 37
EvaluatorStrategy (class in fpylll.fplll.enumeration), 35

F

final_kappa (fpylll.fplll.lll.LLLReduction attribute), 27
FIRST_N_SOLUTIONS (fpylll.fplll.enumeration.EvaluatorStrategy attribute), 35
float_type (fpylll.fplll.gso.MatGSO attribute), 18
fpylll.algorithms.bkz (module), 38
fpylll.algorithms.simple_bkz (module), 36
fpylll.algorithms.simple_dbkz (module), 37
fpylll.fplll.bkz (module), 29
fpylll.fplll.enumeration (module), 34
fpylll.fplll.gso (module), 17
fpylll.fplll.integer_matrix (module), 7
fpylll.fplll.lll (module), 25
fpylll.fplll.svpcvp (module), 32
fpylll.fplll.wrapper (module), 24
fpylll.util (module), 35
from_canonical() (fpylll.fplll.gso.MatGSO method), 18
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

gaussian_heuristic() (in module fpylll.util), 35
gen_identity() (fpylll.fplll.integer_matrix.IntegerMatrix method), 9
get_current_slope() (fpylll.fplll.gso.MatGSO method), 19
get_gram() (fpylll.fplll.gso.MatGSO method), 19
get_log_det() (fpylll.fplll.gso.MatGSO method), 19
get_max_exp() (fpylll.fplll.integer_matrix.IntegerMatrix method), 9
get_mu() (fpylll.fplll.gso.MatGSO method), 19
get_mu_exp() (fpylll.fplll.gso.MatGSO method), 20
get_nodes() (fpylll.fplll.enumeration.Enumeration method), 35
get_precision() (in module fpylll.util), 35
get_r() (fpylll.fplll.gso.MatGSO method), 20
get_r_exp() (fpylll.fplll.gso.MatGSO method), 20
get_root_det() (fpylll.fplll.gso.MatGSO method), 20
get_slide_potential() (fpylll.fplll.gso.MatGSO method), 20
GH_BND (fpylll.fplll.bkz.BKZ attribute), 29
GSO (class in fpylll.fplll.gso), 17

H

hkz() (fpylll.fplll.bkz.BKZReduction method), 30

I

identity() (fpylll.fplll.integer_matrix.IntegerMatrix class method), 10
INT_GRAM (fpylll.fplll.gso.GSO attribute), 17
int_gram_enabled (fpylll.fplll.gso.MatGSO attribute), 21
IntegerMatrix (class in fpylll.fplll.integer_matrix), 7
IntegerMatrixRow (class in fpylll.fplll.integer_matrix), 15
inverse_transform_enabled (fpylll.fplll.gso.MatGSO attribute), 21
is_empty() (fpylll.fplll.integer_matrix.IntegerMatrix method), 10
is_LLL_reduced() (in module fpylll.fplll.lll), 28
is_reduced() (fpylll.fplll.lll.LLL static method), 26

is_zero() (fpylll.fplll.integer_matrix.IntegerMatrixRow method), 16

L

last_early_red (fpylll.fplll.lll.LLLReduction attribute), 27
LLL (class in fpylll.fplll.lll), 25
LLL.Wrapper (class in fpylll.fplll.lll), 25
lll_obj (fpylll.fplll.bkz.BKZReduction attribute), 31
lll_reduction() (in module fpylll.fplll.lll), 28
LLLReduction (class in fpylll.fplll.lll), 26

M

M (fpylll.fplll.bkz.BKZReduction attribute), 30
M (fpylll.fplll.enumeration.Enumeration attribute), 34
M (fpylll.fplll.lll.LLLReduction attribute), 27
Mat (fpylll.fplll.gso.GSO attribute), 17
MatGSO (class in fpylll.fplll.gso), 17
MatGSORowOpContext (class in fpylll.fplll.gso), 24
MAX_LOOPS (fpylll.fplll.bkz.BKZ attribute), 29
MAX_TIME (fpylll.fplll.bkz.BKZ attribute), 29
mod() (fpylll.fplll.integer_matrix.IntegerMatrix method), 10
move_row() (fpylll.fplll.gso.MatGSO method), 21
multiply_left() (fpylll.fplll.integer_matrix.IntegerMatrix method), 11

N

ncols (fpylll.fplll.integer_matrix.IntegerMatrix attribute), 11
negate_row() (fpylll.fplll.gso.MatGSO method), 21
NO_LLL (fpylll.fplll.bkz.BKZ attribute), 29
nodes (fpylll.fplll.bkz.BKZReduction attribute), 31
norm (fpylll.fplll.integer_matrix.IntegerMatrixRow attribute), 16
nrows (fpylll.fplll.integer_matrix.IntegerMatrix attribute), 11
nswaps (fpylll.fplll.lll.LLLReduction attribute), 27

O

OP_FORCE_LONG (fpylll.fplll.gso.GSO attribute), 17
OPPORTUNISTIC_N_SOLUTIONS (fpylll.fplll.enumeration.EvaluatorStrategy attribute), 35
OVERRIDE_BND (fpylll.fplll.svpcvp.SVP attribute), 33

P

Param (fpylll.fplll.bkz.BKZ attribute), 29
param (fpylll.fplll.bkz.BKZReduction attribute), 31
precision() (in module fpylll.util), 36
PrecisionContext (class in fpylll.util), 35

R

r() (fpylll.fplll.gso.MatGSO method), 22
random() (fpylll.fplll.integer_matrix.IntegerMatrix class method), 11
randomize() (fpylll.fplll.integer_matrix.IntegerMatrix method), 11
Reduction (fpylll.fplll.bkz.BKZ attribute), 29
Reduction (fpylll.fplll.lll.LLL attribute), 25
reduction() (fpylll.fplll.bkz.BKZ static method), 29
reduction() (fpylll.fplll.lll.LLL static method), 26
ReductionError, 35
remove_last_row() (fpylll.fplll.gso.MatGSO method), 22
rerandomize_block() (fpylll.fplll.bkz.BKZReduction method), 31
resize() (fpylll.fplll.integer_matrix.IntegerMatrix method), 12
rotate() (fpylll.fplll.integer_matrix.IntegerMatrix method), 12
rotate_gram_left() (fpylll.fplll.integer_matrix.IntegerMatrix method), 12
rotate_gram_right() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
rotate_left() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
rotate_right() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
row_addmul() (fpylll.fplll.gso.MatGSO method), 22
ROW_EXPO (fpylll.fplll.gso.GSO attribute), 17
row_expo_enabled (fpylll.fplll.gso.MatGSO attribute), 22
row_op_begin() (fpylll.fplll.gso.MatGSO method), 22
row_op_end() (fpylll.fplll.gso.MatGSO method), 23
row_op_force_long (fpylll.fplll.gso.MatGSO attribute), 23
row_ops() (fpylll.fplll.gso.MatGSO method), 23

S

sd_tour() (fpylll.fplll.bkz.BKZReduction method), 31
SD_VARIANT (fpylll.fplll.bkz.BKZ attribute), 29
set_cols() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
set_iterable() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
set_matrix() (fpylll.fplll.integer_matrix.IntegerMatrix method), 13
set_precision() (in module fpylll.util), 36
set_random_seed() (in module fpylll.util), 36
set_rows() (fpylll.fplll.integer_matrix.IntegerMatrix method), 14
shortest_vector() (fpylll.fplll.svpcvp.SVP static method), 33
shortest_vector() (in module fpylll.fplll.svpcvp), 34
SIEGEL (fpylll.fplll.lll.LLL attribute), 25
size_nz() (fpylll.fplll.integer_matrix.IntegerMatrixRow method), 16
size_reduction() (fpylll.fplll.lll.LLLReduction method), 27

SLD_RED (fpylll.fplll.bkz.BKZ attribute), 29
slide_tour() (fpylll.fplll.bkz.BKZReduction method), 31
status (fpylll.fplll.bkz.BKZReduction attribute), 31
status (fpylll.fplll.lll.LLL.Wrapper attribute), 26
status (fpylll.fplll.wrapper.Wrapper attribute), 25
submatrix() (fpylll.fplll.integer_matrix.IntegerMatrix method), 14
SVP (class in fpylll.fplll.svpcvp), 33
svp_call() (fpylll.algorithms.bkz.BKZReduction method), 38
svp_postprocessing() (fpylll.algorithms.bkz.BKZReduction method), 38
svp_postprocessing() (fpylll.fplll.bkz.BKZReduction method), 31
svp_preprocessing() (fpylll.algorithms.bkz.BKZReduction method), 38
svp_preprocessing() (fpylll.fplll.bkz.BKZReduction method), 32
svp_reduction() (fpylll.algorithms.bkz.BKZReduction method), 39
svp_reduction() (fpylll.algorithms.simple_bkz.BKZReduction method), 37
svp_reduction() (fpylll.fplll.bkz.BKZReduction method), 32
swap_rows() (fpylll.fplll.integer_matrix.IntegerMatrix method), 14

T

test_abort() (fpylll.fplll.bkz.BKZAutoAbort method), 30
to_canonical() (fpylll.fplll.gso.MatGSO method), 23
to_matrix() (fpylll.fplll.integer_matrix.IntegerMatrix method), 15
tour() (fpylll.algorithms.bkz.BKZReduction method), 39
tour() (fpylll.fplll.bkz.BKZReduction method), 32
transform_enabled (fpylll.fplll.gso.MatGSO attribute), 23
transpose() (fpylll.fplll.integer_matrix.IntegerMatrix method), 15

U

U (fpylll.fplll.gso.MatGSO attribute), 17
U (fpylll.fplll.lll.LLL.Wrapper attribute), 25
U (fpylll.fplll.wrapper.Wrapper attribute), 24
UinvT (fpylll.fplll.gso.MatGSO attribute), 17
UinvT (fpylll.fplll.lll.LLL.Wrapper attribute), 25
UinvT (fpylll.fplll.wrapper.Wrapper attribute), 24
unpickle_IntegerMatrix() (in module fpylll.fplll.integer_matrix), 16
update_gso() (fpylll.fplll.gso.MatGSO method), 24
update_gso_row() (fpylll.fplll.gso.MatGSO method), 24

V

VERBOSE (fpylll.fplll.bkz.BKZ attribute), 29
VERBOSE (fpylll.fplll.lll.LLL attribute), 25
VERBOSE (fpylll.fplll.svpcvp.CVP attribute), 33
VERBOSE (fpylll.fplll.svpcvp.SVP attribute), 33

W

Wrapper (class in fpylll.fplll.wrapper), 24

Z

zeros (fpylll.fplll.lll.LLLReduction attribute), 28