2015_12_12_Exemples

Annick Valibouze

16 avril 2016

Contents

Nous sommes dans une feuille worksheet sage de SageMathCloud

---

double click to edit this comment ; CMD+click pour créer un tel commentaire ; CMD plus return pour évaluer une cellule

(double click to edit)

```
1+1
```

```
2
```

Le langage est python ci-dessous l'instruction conditionnelle (voir aussi plus bas)

```
maxima.expand('(x+1)*x')
```

```
x^2+x
```

```
if i == 1:
    print 'i equals 1'
else:
    print 'i is not 1'
```

```
i is not 1
```

```
T=TransitiveGroup(3,2) ; T; T.degree() ; T.gens();
```

```
Transitive group number 2 of degree 3
3
[(1,2), (1,2,3)]
```

```
GF3=GF(3);GF3
```

```
GF3(8)
```

```
2
```

```
PGF3.<x>=PolynomialRing(GF3) ; PGF3
```

```
Univariate Polynomial Ring in x over Finite Field of size 3
Univariate Polynomial Ring in x over Rational Field
```

On accède aux autres logiciels R, Matlab, Maxima, etc :
pour R, voir cette vidéo de 37 sc : https://www.youtube.com/watch?v=JtVuX4yb70A
de même, voir ci-dessous pour utiliser par exemple la substitution du système de calcul formel maxima

```
maxima.subst(1,'x','[x^2+y,y]');
```

```
[y+1,y]
```

```
R.<x,y> = QQ[]
R.<xx, yy> = R.quo([y^2 - x^3 - x])
```

```
R = PolynomialRing(ZZ, ['x%s'%p for p in primes(100)])
R.inject_variables()
```

```
Defining x2, x3, x5, x7, x11, x13, x17, x19, x23, x29, x31, x37, x41, x43, x47, x53, x59,
x61, x67, x71, x73, x79, x83, x89, x97
```

```
Mod(5, 12)
```

```
5
```

```
EllipticCurve([1,2,3,4,5])
```

```
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5 over Rational Field
```

```
AA
```

```
Algebraic Real Field
```

```
GF(7)
```

```
Finite Field of size 7
```

```
g = graphs.PetersenGraph().chromatic_number()
show(g)
```

```
3
```

```
graphs.PetersenGraph().automorphism_group()
```

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

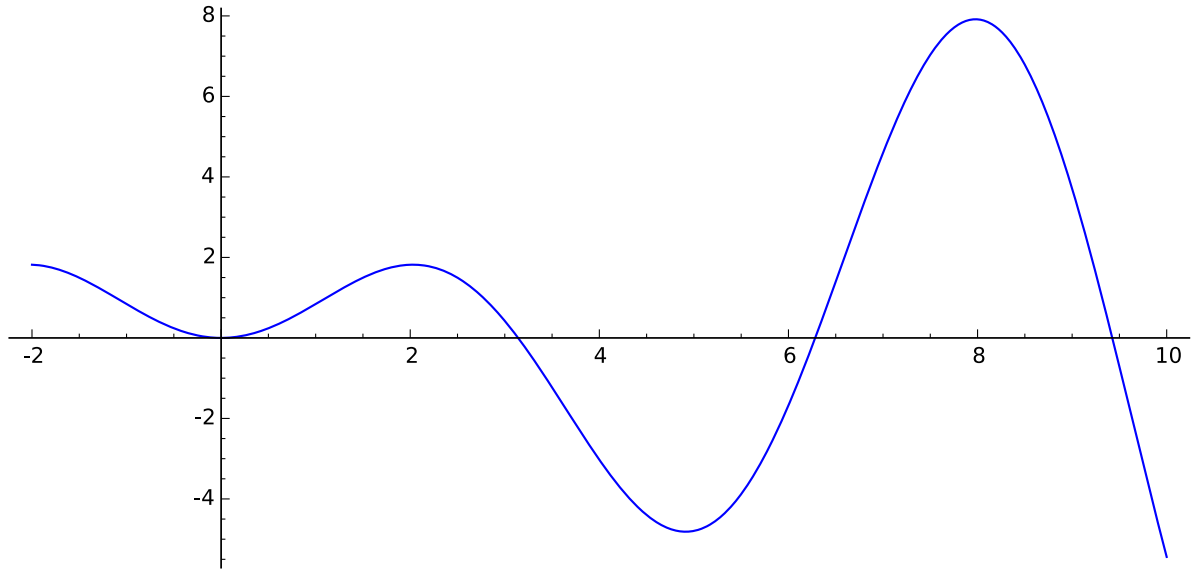
```
show(graphs.PetersenGraph())
```

```
f(x,y) = x * sin(y)
```

```
f(x,pi/4)
```

```
1/2*sqrt(2)*x
```

```
plot(x * sin(x), (x, -2, 10)
```




```
magics()
```

```
['%auto', '%axiom', '%capture', '%coffeescript', '%command', '%cython', '%default',
'%default_mode', '%exercise', '%file', '%fork', '%fortran', '%fricas', '%gap', '%gap3',
'%giac', '%go', '%gp', '%hide', '%hideall', '%html', '%javascript', '%julia', '%kash',
'%lie', '%lisp', '%load', '%macaulay2', '%magics', '%magma', '%maple', '%mathematica',
'%matlab', '%maxima', '%md', '%mupad', '%mwrank', '%octave', '%pandoc', '%perl', '%prun',
'%python', '%r', '%reset', '%ruby', '%runfile', '%sage0', '%scilab', '%script', '%sh',
'%singular', '%time', '%timeit', '%typeset_mode', '%var', '%wiki']
```

```
n = 0
while n < 5:
    print n
    n += 1
```

```
for k in [1, 2, 5, 10]:
    if k == 3:
        print "found k, returning"
        break
else:
    print "Haven't found k == 3"
```

```
def f(a, b=0):
    """
    This function returns the sum of a and b.
    """
    return a + b
```

```
f = lambda a, b: a + b
```

```
class MyClass(object):
    """
    This is a simple class.
    """
    def __init__(self, a):
        self.a = a
    def __repr__(self):
        return "Instance of MyClass with a = %s"%self.a

print(MyClass(5))
```

```
Instance of MyClass with a = 5
```

```
class A(object):
    def __repr__(self):
        return "instance of A"
    def foo(self):
        return "foo"

class B(object):
    def __repr__(self):
        return "instance of B"
    def bar(self):
        return "bar"

class C(A, B):
    """
    This is a class that inerits from classes A and B.
    """
    def __repr__(self):
        return "instance of C"

# Both foo and bar are defined on instances of C.
c = C()
print(c.foo(), c.bar())
```

```
('foo', 'bar')
```

```
var('z')
```

```
z
```

```
z+1
```

```
z + 1
```

```
import csv
import sys

f = open('example.csv', 'rt')
try:
    reader = csv.reader(f)
    for row in reader:
        print row
finally:
    f.close()
```

```
{7, 3, 2,2}
```

```
set([2, 3, 7])
```

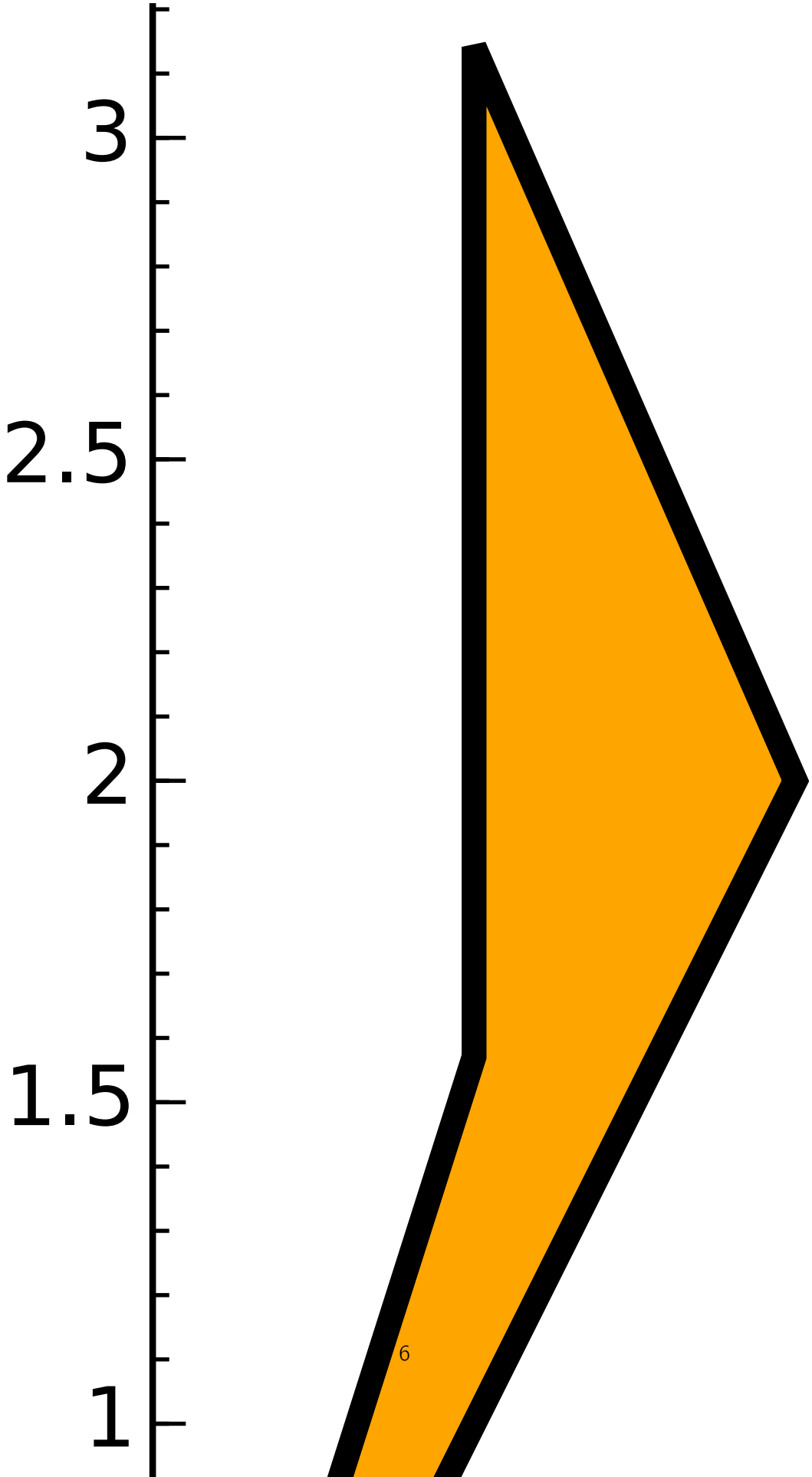
```
range(1,10)
```

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

```
[n+1 for n in range(10) if n%2==0]
```

```
[1, 3, 5, 7, 9]
```

```
a = polygon2d([(0,0), (1,2), (1/2,pi), (1/2,pi/2)], color='orange')
b = polygon2d([(0,0), (1,2), (1/2,pi), (1/2,pi/2)], color='black', \
   fill=False, thickness=3)
show(a + b)
```


3
2.5
2
1.5
1

```
mm??
```

```
WARNING: Output truncated.  Type 'smc?' to learn how to raise the output limit.
```

```
   Source:
   cdef class Matrix_integer_dense(matrix_dense.Matrix_dense):   # \
   dense or sparse
    r"""
    Matrix over the integers, implemented using FLINT.

    On a 32-bit machine, they can have at most `2^{32}-1` rows or
    columns.  On a 64-bit machine, matrices can have at most
    `2^{64}-1` rows or columns.

    EXAMPLES::

        sage: a = MatrixSpace(ZZ,3)(2); a
        [2 0 0]
        [0 2 0]
        [0 0 2]
        sage: a = matrix(ZZ,1,3, [1,2,-3]); a
        [ 1  2 -3]
        sage: a = MatrixSpace(ZZ,2,4)(2); a
        Traceback (most recent call last):
        ...
        TypeError: nonzero scalar matrix must be square
    """
    #\
   ######################################################################\

    # LEVEL 1 functionality
    # x * __cinit__
    # x * __dealloc__
    # x * __init__
    # x * set_unsafe
    # x * get_unsafe
    # x * def _pickle
    # x * def _unpickle
    #\
   ######################################################################\


    def __cinit__(self, parent, entries, coerce, copy):
        """
        Create and allocate memory for the matrix. Does not actually
        initialize any of the memory.

        INPUT:
```

```
    -  ``parent, entries, coerce, copy`` - as for
       __init__.


    EXAMPLES::

        sage: from sage.matrix.matrix_integer_dense import \
Matrix_integer_dense
        sage: a = Matrix_integer_dense.__new__(\
Matrix_integer_dense, Mat(ZZ,3), 0,0,0)
        sage: type(a)
        <type 'sage.matrix.matrix_integer_dense.\
Matrix_integer_dense'>

    TESTS::

        sage: Matrix(ZZ, sys.maxsize, sys.maxsize)
        Traceback (most recent call last):
        ...
        RuntimeError: FLINT exception
    """
    self._parent = parent
    self._base_ring = ZZ
    self._nrows = parent.nrows()
    self._ncols = parent.ncols()
    self._pivots = None
    self._initialized_mpz = False
    self._entries = NULL
    self._rows = NULL
    sig_str("FLINT exception")
    fmpz_mat_init(self._matrix, self._nrows, self._ncols)
    sig_off()

cdef inline int _init_mpz(self) except -1:
    if self._initialized_mpz:
        return 0
    else:
        return self._init_mpz_impl()

cdef inline int _init_linbox(self) except -1:
    if not self._initialized_mpz:
        self._init_mpz_impl()
    linbox.set(self._rows, self._nrows, self._ncols)
    return 0
```

```
    cdef int _init_mpz_impl(self) except -1:
        cdef Py_ssize_t i, j, k

        sig_on()
        self._rows = <mpz_t **> sage_malloc(sizeof(mpz_t*) * self.\
_nrows)
        if not self._rows:
            raise MemoryError
        self._entries = <mpz_t *> sage_malloc(sizeof(mpz_t) * self.\
_nrows * self._ncols)
        if not self._entries:
            sage_free(self._rows)
            raise MemoryError
        k = 0
        for i in range(self._nrows):
            self._rows[i] = self._entries + k
            for j in range(self._ncols):
                mpz_init(self._entries[k])
                fmpz_get_mpz(self._entries[k],fmpz_mat_entry(self.\
_matrix,i,j))
                k += 1
        sig_off()
        self._initialized_mpz = True
        return 1

    cdef void _dealloc_mpz(self):
        if not self._initialized_mpz:
            return
        cdef Py_ssize_t k
        for k in range(self._nrows * self._ncols):
            mpz_clear(self._entries[k])
        sage_free(self._rows)
        sage_free(self._entries)
        self._initialized_mpz = False

    def __hash__(self):
        r"""
        Returns hash of self.

        self must be immutable.

        EXAMPLES::

            sage: a = Matrix(ZZ,2,[1,2,3,4])
            sage: hash(a)
            Traceback (most recent call last):
            ...
```

```
            TypeError: mutable matrices are unhashable

        ::

            sage: a.set_immutable()
            sage: hash(a)
            8
        """
        return self._hash()

    def __dealloc__(self):
        """
        Frees all the memory allocated for this matrix.

        EXAMPLE::

            sage: a = Matrix(ZZ,2,[1,2,3,4])
            sage: del a
        """
        fmpz_mat_clear(self._matrix)
        self._dealloc_mpz()

    def __init__(self, parent, entries, copy, coerce):
        r"""
        Initialize a dense matrix over the integers.

        INPUT:


        -  ``parent`` - a matrix space

        -  ``entries`` - list - create the matrix with those
           entries along the rows.

        -  ``other`` - a scalar; entries is coerced to an
           integer and the diagonal entries of this matrix are set \
to that
           integer.

        -  ``coerce`` - whether need to coerce entries to the
           integers (program may crash if you get this wrong)

        -  ``copy`` - ignored (since integers are immutable)


        EXAMPLES:
```

```
     The __init__ function is called implicitly in each of the
     examples below to actually fill in the values of the matrix.

     We create a `2 \times 2` and a `1\times 4` matrix::

         sage: matrix(ZZ,2,2,range(4))
         [0 1]
         [2 3]
         sage: Matrix(ZZ,1,4,range(4))
         [0 1 2 3]

     If the number of columns isn't given, it is determined from \
the
     number of elements in the list.

     ::

         sage: matrix(ZZ,2,range(4))
         [0 1]
         [2 3]
         sage: matrix(ZZ,2,range(6))
         [0 1 2]
         [3 4 5]

     Another way to make a matrix is to create the space of \
matrices and
     coerce lists into it.

     ::

         sage: A = Mat(ZZ,2); A
         Full MatrixSpace of 2 by 2 dense matrices over Integer \
Ring
         sage: A(range(4))
         [0 1]
         [2 3]

     Actually it is only necessary that the input can be coerced \
to a
     list, so the following also works::

         sage: v = reversed(range(4)); type(v)
         <type 'listreverseiterator'>
         sage: A(v)
         [3 2]
         [1 0]
```

```
    Matrices can have many rows or columns (in fact, on a 64-bit
    machine they could have up to `2^64-1` rows or columns)::

        sage: v = matrix(ZZ,1,10^5, range(10^5))
        sage: v.parent()
        Full MatrixSpace of 1 by 100000 dense matrices over \
Integer Ring
    """
    cdef Py_ssize_t i, j, k
    cdef bint is_list
    cdef Integer x
    cdef list entries_list

    if entries is None:
        x = ZZ.zero()
        is_list = False
    elif isinstance(entries, (int,long,Element)):
        try:
            x = ZZ(entries)
        except TypeError:
            raise TypeError("unable to coerce entry to an \
integer")
        is_list = False
    elif type(entries) is list:
        entries_list = entries
        is_list = True
    else:
        entries_list = list(entries)
        is_list = True
    if is_list:
        # Create the matrix whose entries are in the given entry\
 list.
        if len(entries_list) != self._nrows * self._ncols:
            raise TypeError("entries has the wrong length")
        if coerce:
            k = 0
            for i from 0 <= i < self._nrows:
                for j from 0 <= j < self._ncols:
                    x = ZZ(entries_list[k])
                    k += 1
                    # todo -- see integer.pyx and the TODO there\
; perhaps this could be
                    # sped up by creating a mpz_init_set_sage \
function.
                    fmpz_set_mpz(fmpz_mat_entry(self._matrix, i,\
 j),(<Integer>x).value)
        else:
```

```
                k = 0
                for i from 0 <= i < self._nrows:
                    for j from 0 <= j < self._ncols:
                        fmpz_set_mpz(fmpz_mat_entry(self._matrix, i,\
j),(<Integer> entries_list[k]).value)
                        k += 1
        else:
            # If x is zero, make the zero matrix and be done.
            if mpz_sgn(x.value) == 0:
                fmpz_mat_zero(self._matrix)
                return

            # the matrix must be square:
            if self._nrows != self._ncols:
                raise TypeError("nonzero scalar matrix must be \
square")

            # Now we set all the diagonal entries to x and all other\
 entries to 0.
            fmpz_mat_zero(self._matrix)
            for i from 0 <= i < self._nrows:
                fmpz_set_mpz(fmpz_mat_entry(self._matrix,i,i), x.\
value)

    cdef set_unsafe(self, Py_ssize_t i, Py_ssize_t j, object x):
        """
        Set position i,j of this matrix to ``x``.

        The object ``x`` must be of type ``Integer``.

        INPUT:

        - ``i`` -- row

        - ``j`` -- column

        - ``x`` -- must be Integer! The value to set self[i,j] to.

        EXAMPLES::

            sage: a = matrix(ZZ,2,3, range(6)); a
            [0 1 2]
            [3 4 5]
            sage: a[0,0] = 10
            sage: a
            [10  1  2]
            [ 3  4  5]
```

```
        """
        self.set_unsafe_mpz(i, j, (<Integer>x).value)

    cdef void set_unsafe_mpz(self, Py_ssize_t i, Py_ssize_t j, const\
    mpz_t value):
        """
        Set position i,j of this matrix to ``value``.

        INPUT:

        - ``i`` -- row

        - ``j`` -- column

        - ``value`` -- The value to set self[i,j] to. This will make\
    a
          copy of ``value``.

        EXAMPLES::

            sage: a = matrix(ZZ,2,3, range(6)); a
            [0 1 2]
            [3 4 5]
            sage: a[0,0] = 10
            sage: a
            [10  1  2]
            [ 3  4  5]
        """
        fmpz_set_mpz(fmpz_mat_entry(self._matrix,i,j), value)
        if self._initialized_mpz:
            mpz_set(self._entries[i*self._ncols + j], value)

    cdef void set_unsafe_si(self, Py_ssize_t i, Py_ssize_t j, long \
    value):
        """
        Set position i,j of this matrix to ``value``.
        """
        fmpz_set_si(fmpz_mat_entry(self._matrix,i,j), value)
        if self._initialized_mpz:
            mpz_set_si(self._entries[i*self._ncols + j], value)

    cdef void set_unsafe_double(self, Py_ssize_t i, Py_ssize_t j, \
    double value):
        """
        Set position i,j of this matrix to ``value``.
        """
        fmpz_set_d(fmpz_mat_entry(self._matrix,i,j), value)
```

```
        if self._initialized_mpz:
            mpz_set_d(self._entries[i*self._ncols + j], value)

    cdef get_unsafe(self, Py_ssize_t i, Py_ssize_t j):
        """
        Returns (i, j) entry of self as a new Integer.

        .. warning::

           This is very unsafe; it assumes i and j are in the right
           range.

        EXAMPLES::

            sage: a = MatrixSpace(ZZ,3)(range(9)); a
            [0 1 2]
            [3 4 5]
            [6 7 8]
            sage: a[1,2]
            5
            sage: a[4,7]
            Traceback (most recent call last):
            ...
            IndexError: matrix index out of range
            sage: a[-1,0]
            6
        """
        cdef Integer z = PY_NEW(Integer)
        self.get_unsafe_mpz(i, j, z.value)
        return z

    cdef inline void get_unsafe_mpz(self, Py_ssize_t i, Py_ssize_t j\
, mpz_t value):
        """
        Copy entry i,j of the matrix ``self`` to ``value``.

        .. warning::

           This is very unsafe; it assumes i and j are in the right
           range.

        EXAMPLES::

            sage: a = MatrixSpace(ZZ,3)(range(9)); a
            [0 1 2]
            [3 4 5]
            [6 7 8]
```

```
            sage: a[1,2]
            5
            sage: a[4,7]
            Traceback (most recent call last):
            ...
            IndexError: matrix index out of range
            sage: a[-1,0]
            6
        """
        fmpz_get_mpz(value,fmpz_mat_entry(self._matrix, i, j))

    cdef inline double get_unsafe_double(self, Py_ssize_t i, \
Py_ssize_t j):
        """
        Returns (j, i) entry of self as a new Integer.

        .. warning::

           This is very unsafe; it assumes i and j are in the right
           range.

        EXAMPLES::

            sage: a = MatrixSpace(ZZ,3)(range(9)); a
            [0 1 2]
            [3 4 5]
            [6 7 8]
            sage: a[1,2]
            5
            sage: a[4,7]
            Traceback (most recent call last):
            ...
            IndexError: matrix index out of range
            sage: a[-1,0]
            6
        """
        return fmpz_get_d(fmpz_mat_entry(self._matrix, i, j))

    def _pickle(self):
        """
        EXAMPLES::

            sage: a = matrix(ZZ,2,3,[1,193,15,-2,3,0])
            sage: a._pickle()
            ('1 61 f -2 3 0', 0)

            sage: S = ModularSymbols(250,4,sign=1).\
```

```
cuspidal_submodule().new_subspace().decomposition() # long time
        sage: S == loads(dumps(S)) # long time
        True
    """
    return self._pickle_version0(), 0

cdef _pickle_version0(self):
    """
    EXAMPLES::

        sage: matrix(ZZ,1,3,[1,193,15])._pickle()   # indirect \
doctest
        ('1 61 f', 0)

    """
    return self._export_as_string(32)

cpdef _export_as_string(self, int base=10):
    """
    Return space separated string of the entries in this matrix,\
 in the
    given base. This is optimized for speed.

    INPUT: base -an integer = 36; (default: 10)

    EXAMPLES::

        sage: m = matrix(ZZ,2,3,[1,2,-3,1,-2,-45])
        sage: m._export_as_string(10)
        '1 2 -3 1 -2 -45'
        sage: m._export_as_string(16)
        '1 2 -3 1 -2 -2d'
    """
    # TODO: *maybe* redo this to use mpz_import and mpz_export
    # from sec 5.14 of the GMP manual. ??
    cdef int i, j, len_so_far, m, n
    cdef char *a
    cdef char *s
    cdef char *t
    cdef char *tmp

    if self._nrows == 0 or self._ncols == 0:
        data = ''
    else:
        n = self._nrows*self._ncols*10
        s = <char*> sage_malloc(n * sizeof(char))
        t = s
```

```
            len_so_far = 0

            sig_on()
            for i from 0 <= i < self._nrows:
                for j from 0 <= j < self._ncols:
                    # mat_entry = fmpz_mat_entry(self._matrix,i,j)
                    m = fmpz_sizeinbase(fmpz_mat_entry(self._matrix,\
i,j), base)
                    if len_so_far + m + 2 >= n:
                        # copy to new string with double the size
                        n = 2*n + m + 1
                        tmp = <char*> sage_malloc(n * sizeof(char))
                        strcpy(tmp, s)
                        sage_free(s)
                        s = tmp
                        t = s + len_so_far
                    #endif
                    fmpz_get_str(t, base, fmpz_mat_entry(self.\
_matrix,i,j))
                    m = strlen(t)
                    len_so_far = len_so_far + m + 1
                    t = t + m
                    t[0] = <char>32
                    t[1] = <char>0
                    t = t + 1
            sig_off()
            data = str(s)[:-1]
            sage_free(s)
        return data

    def _unpickle(self, data, int version):
        if version == 0:
            if isinstance(data, str):
                self._unpickle_version0(data)
            elif isinstance(data, list):
                self._unpickle_matrix_2x2_version0(data)
            else:
                raise RuntimeError("invalid pickle data")
        else:
            raise RuntimeError("unknown matrix version (=%s)"%\
version)

    cdef _unpickle_version0(self, data):
        cdef Py_ssize_t i, j, n, k
        data = data.split()
        n = self._nrows * self._ncols
        if len(data) != n:
```

```
            raise RuntimeError("invalid pickle data")
        k = 0
        for i from 0 <= i < self._nrows:
            for j from 0 <= j < self._ncols:
                s = data[k]
                k += 1
                if fmpz_set_str(fmpz_mat_entry(self._matrix,i,j), s,\
 32):
                    raise RuntimeError("invalid pickle data")

    def _unpickle_matrix_2x2_version0(self, data):
        if len(data) != 4 or self._nrows != 2 or self._ncols != 2:
            raise RuntimeError("invalid pickle data")
        self.set_unsafe(0, 0, data[0])
        self.set_unsafe(0, 1, data[1])
        self.set_unsafe(1, 0, data[2])
        self.set_unsafe(1, 1, data[3])

    #\
######################################################################\

    # LEVEL 1 helpers:
    #   These function support the implementation of the level 1 \
functionality.
    #\
######################################################################\

    cdef Matrix_integer_dense _new(self, Py_ssize_t nrows, \
Py_ssize_t ncols):
        """
        Return a new matrix over the integers from given parent
        All memory is allocated for this matrix, but its
        entries have not yet been filled in.
        """
        if nrows == self._nrows and ncols == self._ncols:
            P = self._parent
        else:
            P = matrix_space.MatrixSpace(ZZ, nrows, ncols, sparse=\
False)
        cdef Matrix_integer_dense ans = Matrix_integer_dense.__new__\
(Matrix_integer_dense, P, None, None, None)
        return ans

    #\
######################################################################\

    # LEVEL 2 functionality
```

```
 # x * cdef _add_
 # x * cdef _sub_
 # x * cdef _mul_
 # x * cpdef _cmp_
 # x * __neg__
 # x * __invert__  -> SEE LEVEL 3 FUNCTIONALITIES
 # x * __copy__
 # x * _multiply_classical
 #   * _list -- list of underlying elements (need not be a copy)
 #   * _dict -- sparse dictionary of underlying elements (need \
not be a copy)
 #\
#######################################################################\


 # cdef _mul_(self, Matrix right):
 # def _multiply_classical(left, matrix.Matrix _right):
 # def _list(self):
 # def _dict(self):

 def __copy__(self):
     r"""
     Returns a new copy of this matrix.

     EXAMPLES::

         sage: a = matrix(ZZ,1,3, [1,2,-3]); a
         [ 1  2 -3]
         sage: b = a.__copy__(); b
         [ 1  2 -3]
         sage: b is a
         False
         sage: b == a
         True

         sage: M = MatrixSpace(ZZ,2,3)
         sage: m = M([1,2,3,3,2,1])
         sage: mc = m.__copy__()
         sage: mc == m and mc is not m
         True
     """
     cdef Matrix_integer_dense A
     A = self._new(self._nrows,self._ncols)

     sig_on()
     fmpz_mat_set(A._matrix,self._matrix)
     sig_off()
```

```
        if self._subdivisions is not None:
            A.subdivide(*self.subdivisions())
        return A

    def __nonzero__(self):
        r"""
        Tests whether self is the zero matrix.

        EXAMPLES::

            sage: a = MatrixSpace(ZZ, 2, 3)(range(6)); a
            [0 1 2]
            [3 4 5]
            sage: a.__nonzero__()
            True
            sage: (a - a).__nonzero__()
            False

        ::

            sage: a = MatrixSpace(ZZ, 0, 3)()
            sage: a.__nonzero__()
            False
            sage: a = MatrixSpace(ZZ, 3, 0)()
            sage: a.__nonzero__()
            False
            sage: a = MatrixSpace(ZZ, 0, 0)()
            sage: a.__nonzero__()
            False
        """
        return not fmpz_mat_is_zero(self._matrix)

    def _multiply_linbox(self, Matrix_integer_dense right):
        """
        Multiply matrices over ZZ using linbox.

        .. warning::

           This is very slow right now, i.e., linbox is very slow.

        EXAMPLES::

            sage: A = matrix(ZZ,2,3,range(6))
            sage: A*A.transpose()
            [ 5 14]
            [14 50]
            sage: A._multiply_linbox(A.transpose())
```

```
            [ 5 14]
            [14 50]

        TESTS:

        This fixes a bug found in :trac:`17094`::

            sage: A = identity_matrix(ZZ,3)
            sage: A._multiply_linbox(A)
            [1 0 0]
            [0 1 0]
            [0 0 1]
        """
        cdef int e
        cdef long int i,j
        cdef Matrix_integer_dense ans
        cdef Matrix_integer_dense left = <Matrix_integer_dense>self

        if self._nrows == right._nrows:
            # self acts on the space of right
            parent = right.parent()
        if self._ncols == right._ncols:
            # right acts on the space of self
            parent = self.parent()
        else:
            parent = self.matrix_space(left._nrows, right._ncols)

        ans = self._new(parent.nrows(),parent.ncols())

        left._init_linbox()
        right._init_mpz()
        ans._init_mpz()

        sig_on()
        linbox.matrix_matrix_multiply(ans._rows, right._rows, right.\
_nrows, right._ncols)
        for i from 0 <= i < ans._nrows:
            for j from 0 <= j < ans._ncols:
                fmpz_set_mpz(fmpz_mat_entry(ans._matrix,i,j),ans.\
_rows[i][j])
        sig_off()
        return ans

    def _multiply_classical(self, Matrix_integer_dense right):
        """
        EXAMPLE::
```

```
            sage: n = 3
            sage: a = MatrixSpace(ZZ,n,n)(range(n^2))
            sage: b = MatrixSpace(ZZ,n,n)(range(1, n^2 + 1))
            sage: a._multiply_classical(b)
            [ 18  21  24]
            [ 54  66  78]
            [ 90 111 132]
        """
        if self._ncols != right._nrows:
            raise IndexError("Number of columns of self must equal \
number of rows of right.")

        cdef Py_ssize_t i, j, k, nr, nc, snc
        cdef object parent

        nr = self._nrows
        nc = right._ncols
        snc = self._ncols

        if self._nrows == right._nrows:
            # self acts on the space of right
            parent = right.parent()
        if self._ncols == right._ncols:
            # right acts on the space of self
            parent = self.parent()
        else:
            parent = self.matrix_space(nr, nc)

        cdef Matrix_integer_dense M, _right
        _right = right

        M = self._new(parent.nrows(),parent.ncols())

        cdef fmpz_t s
        fmpz_init(s)
        sig_on()
        for i from 0 <= i < nr:
            for j from 0 <= j < nc:
                fmpz_set_si(s,0)   # set s = 0
                for k from 0 <= k < snc:
                    fmpz_addmul(s, fmpz_mat_entry(self._matrix,i,k),\
fmpz_mat_entry(_right._matrix,k,j))
                fmpz_set(fmpz_mat_entry(M._matrix,i,j),s)
        sig_off()
        fmpz_clear(s)
        return M
```

```
 cdef sage.structure.element.Matrix _matrix_times_matrix_(self, \
sage.structure.element.Matrix right):
     cdef Matrix_integer_dense M

     if self._ncols != right._nrows:
         raise IndexError("Number of columns of self must equal \
number of rows of right.")

     M = self._new(self._nrows, right._ncols)

     sig_on()
     fmpz_mat_mul(M._matrix, self._matrix, (<Matrix_integer_dense\
>right)._matrix)
     sig_off()
     return M


 cpdef ModuleElement _lmul_(self, RingElement right):
     """
     EXAMPLES::

         sage: a = matrix(ZZ, 2, range(6))
         sage: 5 * a
         [ 0  5 10]
         [15 20 25]
     """
     cdef Integer x = Integer(right)
     cdef fmpz_t z
     cdef Matrix_integer_dense M = self._new(self._nrows, self.\
_ncols)

     sig_on()
     fmpz_init_set_readonly(z, x.value)
     fmpz_mat_scalar_mul_fmpz(M._matrix, self._matrix, z)
     fmpz_clear_readonly(z)
     sig_off()
     return M

 cpdef ModuleElement _add_(self, ModuleElement right):
     """
     Add two dense matrices over ZZ.

     EXAMPLES::

         sage: a = MatrixSpace(ZZ,3)(range(9))
         sage: a+a
         [ 0  2  4]
```

```
            [ 6  8 10]
            [12 14 16]
            sage: b = MatrixSpace(ZZ,3)(range(9))
            sage: b.swap_rows(1,2)
            sage: a+b
            [ 0  2  4]
            [ 9 11 13]
            [ 9 11 13]
        """
        cdef Matrix_integer_dense M = self._new(self._nrows,self.\
_ncols)

        sig_on()
        fmpz_mat_add(M._matrix,self._matrix,(<Matrix_integer_dense> \
right)._matrix)
        sig_off()
        return M

    cpdef ModuleElement _sub_(self, ModuleElement right):
        """
        Subtract two dense matrices over ZZ.

        EXAMPLES::

            sage: M = Mat(ZZ,3)
            sage: a = M(range(9)); b = M(reversed(range(9)))
            sage: a - b
            [-8 -6 -4]
            [-2  0  2]
            [ 4  6  8]
        """
        cdef Matrix_integer_dense M = self._new(self._nrows,self.\
_ncols)

        sig_on()
        fmpz_mat_sub(M._matrix,self._matrix,(<Matrix_integer_dense> \
right)._matrix)
        sig_off()
        return M

    def __pow__(sself, n, dummy):
        r"""
        Return the ``n``-th power of this matrix.

        EXAMPLES::

            sage: M = MatrixSpace(ZZ,3)
```

```
        sage: m = M([1, 1, 1, 2, 1, 1, -3, -2, -1])
        sage: m ** 3
        [-3 -2 -1]
        [-3 -2  0]
        [ 2  1 -3]
        sage: m ** -2
        [ 2 -3 -1]
        [-4  4  1]
        [ 1  0  0]
        sage: M(range(9)) ** -1
        Traceback (most recent call last):
        ...
        ZeroDivisionError: Matrix is singular

    TESTS::

        sage: m ** 3 == m ** 3r == (~m) ** (-3) == (~m) ** (-3r)
        True

    The following exponents do not fit in an unsigned long and \
the
    multiplication method fall back to the generic power \
implementation in
    :mod:`sage.structure.element`::

        sage: m = M.identity_matrix()
        sage: m ** (2**256)
        [1 0 0]
        [0 1 0]
        [0 0 1]
        sage: m ** (2r**256r)
        [1 0 0]
        [0 1 0]
        [0 0 1]

    In this case, the second argument to ``__pow__`` is a matrix\
,
    which should raise the correct error::

        sage: M = Matrix(2, 2, range(4))
        sage: None^M
        Traceback (most recent call last):
        ...
        TypeError: Cannot convert NoneType to sage.matrix.\
matrix_integer_dense.Matrix_integer_dense
        sage: M^M
        Traceback (most recent call last):
```

```
        ...
        NotImplementedError: non-integral exponents not \
supported
    """
    cdef Matrix_integer_dense self = <Matrix_integer_dense?>\
sself

    if dummy is not None:
        raise ValueError
    if self._nrows != self._ncols:
        raise ArithmeticError("self must be a square matrix")

    cdef unsigned long e

    if isinstance(n, int):
        if n < 0:
            return (~self) ** (-n)
        e = n
    else:
        if not isinstance(n, Integer):
            try:
                n = Integer(n)
            except TypeError:
                raise NotImplementedError("non-integral \
exponents not supported")
        if mpz_sgn((<Integer>n).value) < 0:
            return (~self) ** (-n)

        if mpz_fits_ulong_p((<Integer>n).value):
            e = mpz_get_ui((<Integer>n).value)
        else:
            # it is very likely that the following will never \
finish except
            # if self is nilpotent
            return generic_power_c(self, n, self._parent.one())

    if e == 0:
        return self._parent.identity_matrix()
    if e == 1:
        return self

    cdef Matrix_integer_dense M = self._new(self._nrows, self.\
_ncols)
    sig_on()
    fmpz_mat_pow(M._matrix, self._matrix, e)
    sig_off()
    return M
```

```
 def __neg__(self):
     r"""
     Return the negative of this matrix.

     TESTS::

         sage: a = matrix(ZZ,2,range(4))
         sage: a.__neg__()
         [ 0 -1]
         [-2 -3]
         sage: -a
         [ 0 -1]
         [-2 -3]
     """
     cdef Matrix_integer_dense M = self._new(self._nrows, self.\
_ncols)
     sig_on()
     fmpz_mat_neg(M._matrix, self._matrix)
     sig_off()
     return M


 cpdef int _cmp_(self, Element right) except -2:
     r"""
     Compares self with right, examining entries in lexicographic\
 (row
     major) ordering.

     EXAMPLES::

         sage: Matrix(ZZ, [[0, 10], [20, 30]]).__cmp__(Matrix(ZZ,\
 [[0, 10], [20, 30]]))
         0
         sage: Matrix(ZZ, [[0, 10], [20, 30]]).__cmp__(Matrix(ZZ,\
 [[0, 15], [20, 30]]))
         -1
         sage: Matrix(ZZ, [[5, 10], [20, 30]]).__cmp__(Matrix(ZZ,\
 [[0, 15], [20, 30]]))
         1
         sage: Matrix(ZZ, [[5, 10], [20, 30]]).__cmp__(Matrix(ZZ,\
 [[0, 10], [25, 30]]))
         1
     """
     cdef Py_ssize_t i, j
     cdef int k
```

```
        sig_on()
        for i from 0 <= i < self._nrows:
            for j from 0 <= j < self._ncols:
                k = fmpz_cmp(fmpz_mat_entry(self._matrix,i,j),\
fmpz_mat_entry((<Matrix_integer_dense>right)._matrix,i,j))
                if k:
                    sig_off()
                    if k < 0:
                        return -1
                    else:
                        return 1
        sig_off()
        return 0

    # TODO: Implement better
    cdef Vector _vector_times_matrix_(self, Vector v):
        """
        Returns the vector times matrix product.

        INPUT:


        -  ``v`` - a free module element.


        OUTPUT: The vector times matrix product v\*A.

        EXAMPLES::

            sage: B = matrix(ZZ,2, [1,2,3,4])
            sage: V = ZZ^2
            sage: w = V([-1,5])
            sage: w*B
            (14, 18)
        """
        cdef Vector_integer_dense w, ans
        cdef Py_ssize_t i, j
        cdef fmpz_t x
        cdef fmpz_t z

        M = self._row_ambient_module()
        w = <Vector_integer_dense> v
        ans = M.zero_vector()

        sig_on()
        fmpz_init(x)
        fmpz_init(z)
```

```
        for i from 0 <= i < self._ncols:
            fmpz_set_si(x, 0)
            for j from 0 <= j < self._nrows:
                fmpz_set_mpz(z,w._entries[j])
                fmpz_addmul(x, z, fmpz_mat_entry(self._matrix,j,i))
            fmpz_get_mpz(ans._entries[i], x)
        fmpz_clear(x)
        fmpz_clear(z)
        sig_off()
        return ans


    #\
    ######################################################################\

    # LEVEL 3 functionality (Optional)
    #    * __deepcopy__
    #  x * __invert__
    #    * Matrix windows -- only if you need strassen for that base
    #    * Other functions (list them here):
    #    * Specialized echelon form
    #\
    ######################################################################\


    def _clear_denom(self):
        """
        INPUT:

        -  ``self`` - a matrix

        OUTPUT:  self, 1

        EXAMPLES::

            sage: a = matrix(ZZ,2,[1,2,3,4])
            sage: a._clear_denom()
            (
            [1 2]
            [3 4], 1
            )
        """
        return self, ZZ(1)

    def charpoly(self, var='x', algorithm='generic'):
        """
        INPUT:
```

```
     -  ``var`` - a variable name

     -  ``algorithm`` - 'generic' (default), 'flint' or 'linbox'


     .. note::

        Linbox charpoly disabled on 64-bit machines, since it \
hangs
        in many cases.

     EXAMPLES::

         sage: A = matrix(ZZ,6, range(36))
         sage: f = A.charpoly(); f
         x^6 - 105*x^5 - 630*x^4
         sage: f(A) == 0
         True
         sage: n=20; A = Mat(ZZ,n)(range(n^2))
         sage: A.charpoly()
         x^20 - 3990*x^19 - 266000*x^18
         sage: A.minpoly()
         x^3 - 3990*x^2 - 266000*x

     TESTS:

     The cached polynomial should be independent of the ``var``
     argument (:trac:`12292`). We check (indirectly) that the
     second call uses the cached value by noting that its result \
is
     not cached::

         sage: M = MatrixSpace(ZZ, 2)
         sage: A = M(range(0, 2^2))
         sage: type(A)
         <type 'sage.matrix.matrix_integer_dense.\
Matrix_integer_dense'>
         sage: A.charpoly('x')
         x^2 - 3*x - 2
         sage: A.charpoly('y')
         y^2 - 3*y - 2
         sage: A._cache['charpoly_linbox']
         x^2 - 3*x - 2

     """
```

```
        cdef long i,n
        cdef Integer z
        cdef Polynomial_integer_dense_flint g
        if algorithm == 'generic':
            algorithm = 'linbox'
        cache_key = 'charpoly_%s' % algorithm
        g = self.fetch(cache_key)
        if g is not None:
            return g.change_variable_name(var)

        if algorithm == 'flint' or (algorithm == 'linbox' and not \
USE_LINBOX_POLY):
            g = PolynomialRing(ZZ,names = var).gen()
            sig_on()
            fmpz_mat_charpoly(g.__poly,self._matrix)
            sig_off()
        elif algorithm == 'linbox':
            g = self._charpoly_linbox(var)
        else:
            raise ValueError("no algorithm '%s'"%algorithm)
        self.cache(cache_key, g)
        return g

    def minpoly(self, var='x', algorithm = 'linbox'):
        """
        INPUT:


        -  ``var`` - a variable name

        -  ``algorithm`` - 'linbox' (default) 'generic'


        .. note::

           Linbox charpoly disabled on 64-bit machines, since it \
hangs
           in many cases.

        EXAMPLES::

            sage: A = matrix(ZZ,6, range(36))
            sage: A.minpoly()
            x^3 - 105*x^2 - 630*x
            sage: n=6; A = Mat(ZZ,n)([k^2 for k in range(n^2)])
            sage: A.minpoly()
            x^4 - 2695*x^3 - 257964*x^2 + 1693440*x
```

```
        """
        key = 'minpoly_%s_%s'%(algorithm, var)
        x = self.fetch(key)
        if x: return x


        if algorithm == 'linbox' and not USE_LINBOX_POLY:
            algorithm = 'generic'
        if algorithm == 'linbox':
            g = self._minpoly_linbox(var)
        elif algorithm == 'generic':
            g = matrix_dense.Matrix_dense.minpoly(self, var)
        else:
            raise ValueError("no algorithm '%s'"%algorithm)
        self.cache(key, g)
        return g

    def _minpoly_linbox(self, var='x'):
        return self._poly_linbox(var=var, typ='minpoly')

    def _charpoly_linbox(self, var='x'):
        if self.is_zero():  # program around a bug in linbox on 32-\
bit linux
            x = self.base_ring()[var].gen()
            return x ** self._nrows
        return self._poly_linbox(var=var, typ='charpoly')

    def _poly_linbox(self, var='x', typ='minpoly'):
        """
        INPUT:


        -  ``var`` - 'x'

        -  ``typ`` - 'minpoly' or 'charpoly'

        """
        time = verbose('computing %s of %s x %s matrix using linbox'\
%(typ, self._nrows, self._ncols))
        if self._nrows != self._ncols:
            raise ArithmeticError("self must be a square matrix")
        if self._nrows <= 1:
            return matrix_dense.Matrix_dense.charpoly(self, var)
        self._init_linbox()
        if typ == 'minpoly':
            sig_on()
            v = linbox.minpoly()
```

```
            sig_off()
        else:
            sig_on()
            v = linbox.charpoly()
            sig_off()
        R = self._base_ring[var]
        verbose('finished computing %s'%typ, time)
        return R(v)


    def height(self):
        """
        Return the height of this matrix, i.e., the max absolute \
value of
        the entries of the matrix.

        OUTPUT: A nonnegative integer.

        EXAMPLE::

            sage: a = Mat(ZZ,3)(range(9))
            sage: a.height()
            8
            sage: a = Mat(ZZ,2,3)([-17,3,-389,15,-1,0]); a
            [ -17    3 -389]
            [  15   -1    0]
            sage: a.height()
            389
        """
        cdef Integer x = PY_NEW(Integer)
        self.mpz_height(x.value)
        return x

    cdef int mpz_height(self, mpz_t height) except -1:
        """
        Used to compute the height of this matrix.

        INPUT:

        - ``height`` -- a GMP mpz_t which has been initialized

        OUTPUT: sets the value of height to the height of this \
matrix,
        i.e., the max absolute value of the entries of the matrix.
        """
        cdef fmpz_t x,h
        cdef Py_ssize_t i,j
```

```
        sig_on()
        fmpz_init(h)
        fmpz_init(x)
        for i from 0 <= i < self._nrows:
            for j from 0 <= j < self._ncols:
                fmpz_abs(x, fmpz_mat_entry(self._matrix,i,j))
                if fmpz_cmp(h, x) < 0:
                    fmpz_set(h, x)
        fmpz_get_mpz(height,h)
        fmpz_clear(h)
        fmpz_clear(x)
        sig_off()
        return 0   # no error occurred.

    def _multiply_multi_modular(self, Matrix_integer_dense right):
        """
        Multiply this matrix by ``left`` using a multi modular \
algorithm.

        EXAMPLES::

            sage: M = Matrix(ZZ, 2, 3, range(5,11))
            sage: N = Matrix(ZZ, 3, 2, range(15,21))
            sage: M._multiply_multi_modular(N)
            [310 328]
            [463 490]
            sage: M._multiply_multi_modular(-N)
            [-310 -328]
            [-463 -490]
        """
        cdef Integer h
        cdef Matrix_integer_dense left = <Matrix_integer_dense>self
        cdef mod_int *moduli
        cdef int i, n, k
        cdef object parent

        nr = left._nrows
        nc = right._ncols
        snc = left._ncols


        cdef Matrix_integer_dense result


        h = left.height() * right.height() * left.ncols()
        verbose('multiplying matrices of height %s and %s'%(left.\
```

```
height(),right.height()))
     mm = MultiModularBasis(h)
     res = left._reduce(mm)
     res_right = right._reduce(mm)
     k = len(mm)
     for i in range(k):  # yes, I could do this with zip, but to \
conserve memory...
         t = cputime()
         res[i] *= res_right[i]
         verbose('multiplied matrices modulo a prime (%s/%s)'%(i\
+1,k), t)
     result = left.new_matrix(nr,nc)
     _lift_crt(result, res, mm)  # changes result
     return result

 def _mod_int(self, modulus):
     """
     Reduce the integer matrix modulo a positive integer.

     EXAMPLES::

         sage: M = Matrix(ZZ, 2, [1,2,-2,3])
         sage: M._mod_int(2)
         [1 0]
         [0 1]
         sage: M._mod_int(1000000)
         [     1      2]
         [999998      3]
     """
     cdef mod_int c = modulus
     if int(c) != modulus:
         raise OverflowError
     else:
         return self._mod_int_c(modulus)

 cdef _mod_two(self):
     cdef Matrix_mod2_dense res
     res = Matrix_mod2_dense.__new__(Matrix_mod2_dense, \
matrix_space.MatrixSpace(IntegerModRing(2), self._nrows, self.\
_ncols, sparse=False), None, None, None)
     res.__init__(matrix_space.MatrixSpace(IntegerModRing(2), \
self._nrows, self._ncols, sparse=False), self.list(), None, None)
     return res

 cdef _mod_int_c(self, mod_int p):
     from matrix_modn_dense_float import MAX_MODULUS as \
MAX_MODULUS_FLOAT
```

```
        from matrix_modn_dense_double import MAX_MODULUS as \
MAX_MODULUS_DOUBLE

        cdef Py_ssize_t i, j
        cdef mpz_t* self_row

        cdef float* res_row_f
        cdef Matrix_modn_dense_float res_f

        cdef double* res_row_d
        cdef Matrix_modn_dense_double res_d

        if p == 2:
            return self._mod_two()
        elif p < MAX_MODULUS_FLOAT:
            res_f = Matrix_modn_dense_float.__new__(\
Matrix_modn_dense_float,
                                                    matrix_space.\
MatrixSpace(IntegerModRing(p), self._nrows, self._ncols, sparse=\
False), None, None, None)
            for i from 0 <= i < self._nrows:
                res_row_f = res_f._matrix[i]
                for j from 0 <= j < self._ncols:
                    res_row_f[j] = <float>fmpz_fdiv_ui(\
fmpz_mat_entry(self._matrix,i,j), p)
            return res_f

        elif p < MAX_MODULUS_DOUBLE:
            res_d = Matrix_modn_de[...]
```

```
matrix(3,[1,2,3, 4,5,6, 7,8,9]).right_eigenvectors()
```

```
[(0, [
(1, -2, 1)
], 1), (-1.116843969807043?, [(1, 0.11039450377411963?, -0.7792109924517608?)], 1),
(16.11684396980705?, [(1, 2.264605496225881?, 3.529210992451761?)], 1)]
```

```
numerical_integral(1 + x + x^2, 0, 3)[0]  # [1] gives error bound
f(x,y) = x * sin(y)
```

```
integrate(1 + x + x^2, x)
```

```
matrix([[1,2], [3,8]]).Eigenvectors()
```

```
Error in lines 1-1
Traceback (most recent call last):
```

```
  File ‘‘/projects/sage/sage-6.9/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/element.pyx’’, line 418, in
sage.structure.element.Element.__getattr__
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/element.c:4670)
    return getattr_from_other_class(self, P._abstract_element_class, name)
  File ‘‘sage/structure/misc.pyx’’, line 259, in
sage.structure.misc.getattr_from_other_class
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/misc.c:1771)
    raise dummy_attribute_error
AttributeError: 'sage.matrix.matrix_integer_dense.Matrix_integer_dense' object has no
attribute 'Eigenvectors'
```

```
      m.[tab key]
```

```
Error in lines 0-1
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.9/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘<string>’’, line 1
    m.[tab key]
      ^
SyntaxError: invalid syntax
```

```
matrix([[1,2], [3,8]]).Eigenvectors();
```

```
Error in lines 1-1
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.9/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/element.pyx’’, line 418, in
sage.structure.element.Element.__getattr__
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/element.c:4670)
    return getattr_from_other_class(self, P._abstract_element_class, name)
  File ‘‘sage/structure/misc.pyx’’, line 259, in
sage.structure.misc.getattr_from_other_class
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/misc.c:1771)
    raise dummy_attribute_error
AttributeError: 'sage.matrix.matrix_integer_dense.Matrix_integer_dense' object has no
attribute 'Eigenvectors'
```

```
m.g
```

```
m.[tab key]
```

```
Error in lines 1-1
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.9/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘<string>’’, line 1
    m.[tab key]
      ^
SyntaxError: invalid syntax
```

```
mm.Eigenvectors()
```

```
Error in lines 1-1
Traceback (most recent call last):
  File ‘‘/projects/sage/sage-6.9/local/lib/python2.7/site-
packages/smc_sagews/sage_server.py’’, line 905, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File ‘‘’’, line 1, in <module>
  File ‘‘sage/structure/element.pyx’’, line 418, in
sage.structure.element.Element.__getattr__
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/element.c:4670)
    return getattr_from_other_class(self, P._abstract_element_class, name)
  File ‘‘sage/structure/misc.pyx’’, line 259, in
sage.structure.misc.getattr_from_other_class
(/projects/sage/sage-6.9/src/build/cythonized/sage/structure/misc.c:1771)
    raise dummy_attribute_error
AttributeError: 'sage.matrix.matrix_integer_dense.Matrix_integer_dense' object has no
attribute 'Eigenvectors'
```