Ticket #1183 is fixed, as of Feb 2017. https://trac.sagemath.org/ticket/1183

1. (done) implement I.free_module() and O.free_module() as ZZ-submodules of K.vector_space()
2. Compute O/I we instead first compute (O mod p) and (Ibar subset O mod p). Thus we have two F_p vector spaces and compute the quotient of one by the other -- hey we just finished implementing that nicely.
3.
```
  def _p_quotient(self, p):
        """
        EXAMPLES:
            sage: K.<i> = NumberField(x^2 + 1); O = K.maximal_order()
            sage: I = K.factor_integer(3)[0][0]
            sage: I._p_quotient(3)
            Vector space quotient V/W of dimension 2 over Finite Field of size 3
where
            V: Vector space of dimension 2 over Finite Field of size 3
            W: Vector space of degree 2 and dimension 0 over Finite Field of
size 3
            Basis matrix:
            []
            sage: I = K.factor_integer(5)[0][0]
            sage: I._p_quotient(5)
            currently broken
        """
        return quotient_char_p(self, p)


def quotient_char_p(I, p):
    """
    Given an integral ideal I that contains a prime number p, compute
    a vector space V = (OK mod p) / (I mod p), along with a
    homomorphism OK --> V and a section V --> OK.
    """
    if not I.is_integral():
        raise ValueError, "I must be an integral ideal."

    K    = I.number_field()
    OK   = K.maximal_order(p)  # really only need a p-maximal order.
    M_OK = OK.free_module()
    M_I  = I.free_module()

    # Now we have to quite explicitly find a way to compute
```

```
    # with OK / I viewed as a quotient of two F_p vector space,
    # and itself viewed as an F_p vector space.

    # Step 1. Write each basis vector for I (as a ZZ-module)
    # in terms of the basis for OK.
    B_I = M_I.basis()
    M_OK_change = M_OK.basis_matrix()**(-1)
    B_I_in_terms_of_M = M_I.basis_matrix() * M_OK_change

    # Step 2. Define "M_OK mod p" to just be (F_p)^n and
    # "M_I mod p" to be the reduction mod p of the elements
    # compute in step 1.
    n = K.degree()
    k = FiniteField(p)
    M_OK_modp = k**n
    B_mod = B_I_in_terms_of_M.change_ring(k)
    M_I_modp = M_OK_modp.span(B_mod.row_space())

    # Step 3. Compute the quotient of these two F_p vector space.
    Q = M_OK_modp.quotient(M_I_modp)

    # Step 4. Now we get the maps we need from the above data.
    return Q
```

4.
5. Choose elements of O (not at random) until we find one whose minpoly generators Obar/Ibar. When computing the minpoly, I think we do that by computing the matrix whose rows are the powers of the reduction of our candidate.

```
def ResidueField(p, name = None, check = True):
    """
    A function that takes in a prime ideal and returns a number field.

    INPUT:
       p -- a prime integer or prime ideal of an order in a number field.
       name -- the variable name for the finite field created.  Defaults to the name
of the number field variable.
       check -- whether or not to check if p is prime.

    OUTPUT:
      The residue field at the prime p.

    EXAMPLES:
```

```
        sage: from sage.rings.residue_field import ResidueField
        sage: K.<a> = NumberField(x^3-7)
        sage: P = K.ideal(29).factor()[0][0]
        sage: k = K.residue_field(P)
        sage: k
        Residue field of Fractional ideal (2*a^2 + 3*a - 10)
        sage: k.order()
        841
    """
    key = (p, name)
    if residue_field_cache.has_key(key):
        ans = residue_field_cache[key]()
        if ans is not None:
            return ans
    if PY_TYPE_CHECK(p, Integer):
        if check and not p.is_prime():
            raise ValueError, "p must be prime"
        if name is None:
            name = 'x'
        ans = ResidueFiniteField_prime_modn(p, name)
    elif is_NumberFieldIdeal(p):
        if name is None:
            name = p.number_field().variable_name()
        if check and not p.is_prime():
            raise ValueError, "p must be prime"
        # Should generalize to allowing residue fields of relative extensions to be
extensions of finite fields.
        characteristic = p.smallest_integer()
        K = p.number_field()
        OK = K.maximal_order() # should change to p.order once this works.
        U, to_vs, to_order = p._p_quotient(characteristic)
        k = U.base_ring()
        R = PolynomialRing(k, name)
        n = p.residue_class_degree()
        gen_ok = False
        try:
            x = K.gen()
            M = matrix(k, n+1, n, [to_vs(x**i).list() for i in range(n+1)]).transpose()
            if M.rank() == n:
                gen_ok = True
                f = K.polynomial().change_ring(k)
```

```
        except TypeError:
            pass
        if not gen_ok:
            for u in U: # using this iterator may not be optimal, we may get a long
string of non-generators
                x = to_order(u)
                M = matrix(k, n+1, n, [to_vs(x**i).list() for i in
range(n+1)]).transpose()
                M.echelonize()
                if M.rank() == n:
                    f = R((-M.column(n)).list() + [1])
                    break
        if n == 1:
            ans = ResidueFiniteField_prime_modn(p, name, im_gen = -f[0], intp =
p.smallest_integer())
        else:
            q = characteristic**(f.degree())
            if q < Integer(2)**Integer(16):
                ans = ResidueFiniteField_givaro(p, q, name, g, characteristic)
            else:
                ans = ResidueFiniteField_ext_pari(p, q, name, g, characteristic)
    else: # Add support for primes in other rings later.
        raise TypeError, "p must be a prime in the integers or a number field"
    residue_field_cache[key] = weakref.ref(ans)
    return ans
```

OR, my clean up of it:

```
def ResidueField(p, names = None, check = True):
    """
    A function that returns the residue class field of a prime ideal p
    of the ring of integers of a number field.

    INPUT:
        p -- a prime integer or prime ideal of an order in a number
             field.
        names -- the variable name for the finite field created.
                 Defaults to the name of the number field variable but
                 with bar placed after it.
        check -- whether or not to check if p is prime.

    OUTPUT:
         -- The residue field at the prime p.
```

```
    EXAMPLES:
        sage: K.<a> = NumberField(x^3-7)
        sage: P = K.ideal(29).factor()[0][0]
        sage: ResidueField(P)
        Residue field in abar of Fractional ideal (2*a^2 + 3*a - 10)
        sage: k = K.residue_field(P); k
        Residue field in abar of Fractional ideal (2*a^2 + 3*a - 10)
        sage: k.order()
        841
    """
    if isinstance(names, tuple):
        if len(names) > 0:
            names = str(names[0])
        else:
            names = None
    key = (p, names)
    if residue_field_cache.has_key(key):
        k = residue_field_cache[key]()
        if k is not None:
            return k
    if PY_TYPE_CHECK(p, Integer):
        if check and not p.is_prime():
            raise ValueError, "p must be prime"
        if names is None:
            names = 'x'
        k = ResidueFiniteField_prime_modn(p, names)
    elif is_NumberFieldIdeal(p):
        if names is None:
            names = '%sbar'%(p.number_field().variable_name())
        if check and not p.is_prime():
            raise ValueError, "p must be prime"
        # Should generalize to allowing residue fields of relative extensions to be
extensions of finite fields.
        characteristic = p.smallest_integer()

        K = p.number_field()
        OK = K.maximal_order() # should change to p.order once this works.

        U, to_vs, to_order = p._p_quotient(characteristic)
        k = U.base_ring()
```

```
        R = PolynomialRing(k, names)
        n = p.residue_class_degree()
        gen_ok = False
        try:
            x = K.gen()
            from sage.matrix.constructor import matrix
            M = matrix(k, n+1, n, [to_vs(x**i).list() for i in range(n+1)]).transpose()
            if M.rank() == n:
                gen_ok = True
                f = K.polynomial().change_ring(k)
        except TypeError:
            pass
        if not gen_ok:
            for u in U: # using this iterator may not be optimal, we may get a long
string of non-generators
                x = to_order(u)
                M = matrix(k, n+1, n, [to_vs(x**i).list() for i in
range(n+1)]).transpose()
                M.echelonize()
                if M.rank() == n:
                    f = R((-M.column(n)).list() + [1])
                    break
        if n == 1:
            k = ResidueFiniteField_prime_modn(p, names, im_gen = -f[0], intp =
p.smallest_integer())
        else:
            q = characteristic**(f.degree())
            if q < Integer(2)**Integer(16):
                k = ResidueFiniteField_givaro(p, q, names, f, characteristic)
            else:
                k = ResidueFiniteField_ext_pari(p, q, names, f, characteristic)
    else: # Add support for primes in other rings later.
        raise TypeError, "p must be a prime in the integers or a number field"
    residue_field_cache[key] = weakref.ref(k)
    return k
```

1. Get isomorphism to a GF(q). Do this by invert that matrix A to write down an explicit isomorphism to a GF(q).

New stuff in for William.

In ResidueField:

```
      if n == 1:
            ans = ResidueFiniteField_prime_modn(p, names, x, im_gen = -f[0], intp
```

```
= p.smallest_integer())
        else:
            q = characteristic**(f.degree())
            if q < Integer(2)**Integer(16):
                ans = ResidueFiniteField_givaro(p, q, names, x, f, characteristic)
            else:
                ans = ResidueFiniteField_ext_pari(p, q, names, x, f, characteristic)
```

NFResidueFieldHomomorphism's init:

```
   def __init__(self, k, p, x, im_gen):
        """
        INPUT:
           k -- The residue field that is the codomain of this morphism.
           p -- The prime ideal defining this residue field
           x -- The element of the order that p belongs to that defined the minimal
poly of k
           im_gen -- The image of x in k.


        EXAMPLES:
        We create a residue field homomorphism:
        sage: K.<theta> = CyclotomicField(5)
        sage: P = K.factor_integer(7)[0][0]
        sage: P.residue_class_degree()
        4
        sage: kk.<a> = P.residue_field(); kk
        Residue field in a of Fractional ideal (7)
        sage: phi = kk.coerce_map_from(K.maximal_order()); phi
        Ring morphism:
          From: Maximal Order in Cyclotomic Field of order 5 and degree 4
          To:   Residue field in a of Fractional ideal (7)
        sage: type(phi)
        <type 'sage.rings.residue_field.NFResidueFieldHomomorphism'>
        """
        self.im_gen = im_gen
        if not is_FiniteFieldElement(im_gen):
            raise TypeError, "im_gen must be a finite field element"
        (<Element>self.im_gen)._set_parent_c(k)
        self.p = p
        self.x = x
        self.R = PolynomialRing(k, 'x')
        self.to_list = x.coordinates_in_terms_of_powers()
```

```
ResidueFieldHomomorphism.__init__(self,Hom(p.number_field().maximal_order(),
k, Rings())) # should eventually change to p.order()
```
call_c_impl:
```
        return self.R(self.to_list(x))(self.im_gen)
```
lift:
```
        return
self.domain()(x.polynomial().change_ring(self.domain().base_ring())(self.x))
#polynomial should change to absolute_polynomial?
```
In ResidueFiniteField_prime_modn:
```
   def __init__(self, p, name, x = None, im_gen = None, intp = None):
...
              self.f = NFResidueFieldHomomorphism(self, p, x, im_gen)
```
In ResidueFiniteField_ext_pari:
```
   def __init__(self, p, q, name, x, g, intp):
...
         self.f = NFResidueFieldHomomorphism(self, p, x, GF(q, name = name,
modulus = g).gen(0))
```
In ResidueFiniteField_givaro:
```
   def __init__(self, p, q, name, x, g, intp):
...
         self.f = NFResidueFieldHomomorphism(self, p, x, GF(q, name = name,
modulus = g).gen(0))
```