Content and output of `FrobeniusMinimalPolynomials.sage`

Nicolas Müller, Richard Pink

November 17, 2017

This is a formatted version of the Sage code in the file `FrobeniusMinimalPolynomials.sage`, which can be found at [1], together with its output. This worksheet belongs to [2].

```
# Date: November 17, 2017

# This function calculates the minimal polynomial, if it is irreducible,
# of the Frobenius endomorphism of the reduction at p of the jacobian
# of the hyperelliptic curve X with the affine equation y^2=f(x) for a
# separable monic polynomial f.
def getFrobeniusMinimalPolynomialIfIrreducible(f, p):
    K = f.base_ring()
    Kp = K.residue_field(p)
    Rp.<xp> = Kp[]

    # We define the reduction X_p of X at p.
    # If X does not have good reduction, there is an error.
    C = HyperellipticCurve(f.change_ring(Kp).substitute(x=xp))
    # We calculate the characteristic polynomial of the Frobenius
    # of Jac(X_p)
    fr = C.frobenius_polynomial()
    # It has more than one distinct irreducible factors if and only if
    # the minimal polynomial is not irreducible.
    fac = fr.factor()
    if len(fac) > 1:
        # If the minimal polynomial is not irreducible, we return None
        return None
    return fac[0][0]

# This function computes n very good primes for the jacobian of the
# hyperelliptic curve with affine equation y^2=f(x), where the minimal
# polynomial of the Frobenius is irreducible over Q.
# The very good primes are returned together with the respective
# minimal polynomials of Frobenius.
def getVeryGoodPrimesAndFrobeniusMinimalPolynomials(n, f):
    # We first calculate the primes at which X has bad reduction ..
    K = f.base_ring()
    R.<x,y> = K[]
    S = ZZ[x]
    f = R(f)
```

```
    curveBadRedPrimes = [p[0] for p in f.univariate_polynomial(). \
                         discriminant().norm().factor()]
    curveBadRedPrimes.append(2)

    # .. and then try to find very good primes until we have n of them.
    veryGoodPrimes = []
    P = Primes()
    p = 2
    while len(veryGoodPrimes) < n:
        p = P.next(p)

        # If p is a bad prime for X, we skip this prime
        if p in curveBadRedPrimes:
            continue

        # Since all primes at which X has good reduction are good primes
        # for Jac(X), p is a good prime for Jac(X).
        pp = K.primes_above(p)[0]
        fr = getFrobeniusMinimalPolynomialIfIrreducible(f, pp)

        # We skip this prime, if the minimal polynomial is not irreducible.
        if fr is None:
            continue
        # The roots of fbar are the ratios of the roots of f.
        fbar = (fr.subs(x=y)).resultant(fr.subs(x=x*y),y)
        assert fbar.denominator() == 1

        # fbar has a non-trivial root of unity as a root if and only if
        # it has a cyclotomic factor that is not x-1.
        # Since p is a good prime, it is a very good prime if and only if
        # fbar does not have a non-trivial root of unity as a root.
        factors = S(fbar.numerator()).factor()
        if any([S(g[0]).is_cyclotomic() for g in factors if g[0] != x-1]):
            continue
        print str(pp)+(' is very good. The residue field '
                       'has characteristic ')+str(p)
        veryGoodPrimes.append((pp, fr))
    return veryGoodPrimes

# Input: a list [f_1,...,f_n] of irreducible polynomials defined over
#        the rationals
# Output: None, if the fields Q[x]/(f_1),...,Q[x]/f(x_n) are not
#         linearly disjoint.
#         The degree of the field Q[x]/(f_1) tensor ... tensor Q[x]/(f_n)
#         over the rational field otherwise.
def testIrreduciblePolynomialsLinearlyDisjoint(polys):
    K = QQ;
    R.<x> = K[]
    for f,k in zip(polys, range(len(polys))):
        if not R(f).is_irreducible():
            print "Not linearly disjoint."
            return None
        K = K.extension(R(f), "a"+str(k))
        R = K["x"+str(k)]
```

```
    assert K.absolute_degree() == prod([f.degree() for f in polys])
    return K.absolute_degree()


K = CyclotomicField(1)
R.<x> = K[]

############################## X10:
f = x^5 - 19*x^4 - 494*x^3 - 494*x^2 - 19*x + 1
veryGoodPrimesX10 = getVeryGoodPrimesAndFrobeniusMinimalPolynomials(3, f)
print "Very good primes are: "+str([k[0] for k in veryGoodPrimesX10])
# Output: 37, 61, 157
print ("The fields generated by the Frobenius elements at those primes "
       "are linearly disjoint and the product of their degrees is: "+
       str(testIrreduciblePolynomialsLinearlyDisjoint([k[1] for k in
                                                       veryGoodPrimesX10])))
# Output: 8
```

Output:

```
Fractional ideal (37) is very good. The residue field has characteristic 37
Fractional ideal (61) is very good. The residue field has characteristic 61
Fractional ideal (157) is very good. The residue field has characteristic ↙
   157
Very good primes are: [Fractional ideal (37), Fractional ideal (61), ↙
   Fractional ideal (157)]
The fields generated by the Frobenius elements at those primes are linearly↙
    disjoint and the product of their degrees is: 8
```

```
############################## X11:
f = (x - 1) * x * (x + 8) * (x^2 + 8) * (x^2 + 4*x - 8) * (x^2 + 8*x - 8)
veryGoodPrimesX11 = getVeryGoodPrimesAndFrobeniusMinimalPolynomials(2, f)
print "Very good primes are: "+str([k[0] for k in veryGoodPrimesX11])
# Output: 7, 73
print ("The fields generated by the Frobenius elements at those primes "
       "are linearly disjoint and the product of their degrees is: "+
       str(testIrreduciblePolynomialsLinearlyDisjoint([k[1] for k in
                                                       veryGoodPrimesX11])))
# Output: 16
```

Output:

```
Fractional ideal (7) is very good. The residue field has characteristic 7
Fractional ideal (73) is very good. The residue field has characteristic 73
Very good primes are: [Fractional ideal (7), Fractional ideal (73)]
The fields generated by the Frobenius elements at those primes are linearly↙
    disjoint and the product of their degrees is: 16
```

```
############################## X16:
f = x*(x^2+11*x-1)*(x^6+522*x^5-10005*x^4-10005*x^2-522*x+1)
veryGoodPrimesX16 = getVeryGoodPrimesAndFrobeniusMinimalPolynomials(2, f)
print "Very good primes are: "+str([k[0] for k in veryGoodPrimesX16])
# Output: 31, 151
print ("The fields generated by the Frobenius elements at those primes "
       "are linearly disjoint and the product of their degrees is: "+
```

```
      str(testIrreduciblePolynomialsLinearlyDisjoint([k[1] for k in
                                                      veryGoodPrimesX16])))
# Output: 16
```

Output:

```
Fractional ideal (31) is very good. The residue field has characteristic 31
Fractional ideal (151) is very good. The residue field has characteristic ↙
    151
Very good primes are: [Fractional ideal (31), Fractional ideal (151)]
The fields generated by the Frobenius elements at those primes are linearly↙
     disjoint and the product of their degrees is: 16
```

```
############################## X17:
f = (x^4-228*x^3+494*x^2+228*x+1)*(x^6+522*x^5-10005*x^4-10005*x^2-522*x+1)
veryGoodPrimesX17 = getVeryGoodPrimesAndFrobeniusMinimalPolynomials(2, f)
print "Very good primes are: "+str([k[0] for k in veryGoodPrimesX17])
# Output: 31, 41
print ("The fields generated by the Frobenius elements at those primes "
       "are linearly disjoint and the product of their degrees is: "+
       str(testIrreduciblePolynomialsLinearlyDisjoint([k[1] for k in
                                                       veryGoodPrimesX17])))
# Output: 16
```

Output:

```
Fractional ideal (31) is very good. The residue field has characteristic 31
Fractional ideal (41) is very good. The residue field has characteristic 41
Very good primes are: [Fractional ideal (31), Fractional ideal (41)]
The fields generated by the Frobenius elements at those primes are linearly↙
     disjoint and the product of their degrees is: 16
```

```
############################## X18:
f = x*(x^2+11*x-1)*(x^4-228*x^3+494*x^2+228*x+1)*\
    (x^6+522*x^5-10005*x^4-10005*x^2-522*x+1)
veryGoodPrimesX18 = getVeryGoodPrimesAndFrobeniusMinimalPolynomials(2, f)
print "Very good primes are: "+str([k[0] for k in veryGoodPrimesX18])
# Output: 131, 211
print ("The fields generated by the Frobenius elements at those primes "
       "are linearly disjoint and the product of their degrees is: "+
       str(testIrreduciblePolynomialsLinearlyDisjoint([k[1] for k in
                                                       veryGoodPrimesX18])))
# Output: 36
```

Output:

```
Fractional ideal (131) is very good. The residue field has characteristic ↙
    131
Fractional ideal (211) is very good. The residue field has characteristic ↙
    211
Very good primes are: [Fractional ideal (131), Fractional ideal (211)]
The fields generated by the Frobenius elements at those primes are linearly↙
     disjoint and the product of their degrees is: 36
```

References

[1] Worksheets for this paper: URL: `https://people.math.ethz.ch/~pink/ftp/MuellerPink2017/`

[2] Müller, N., Pink, R.: *Hyperelliptic Curves with Many Automorphisms*. Preprint 2017.