CATAM 16.1 Galois Groups

January 23, 2024

1 Producing Algorithms

1.1 Elementary Operations

A program called ***Q1.py*** was written with methods to perform polynomial division with remainders and computing the highest common factor over the field $\mathbb{F}_p$. The code is shown in Section 3, with results in Table 1.

Prime	f	g	quotient	remainder	hcf
2	X^6+X^4+1	X^4+X^3+1	X^2+X	X^2+X+1	1
43	$X^2+34X+21$	$X+32$	$X+2$	0	$X+32$
11	$X^4+6X^3+3X^2+5X+9$	X^2+9X+5	X^2+8X+3	$4X+5$	1
43	$X+1$	$X^5+13X^4+11X^3+12X^2+29X+28$	0	$X+1$	1
3	$X^6+2X^4+2X^3+X$	$X^5+X^4+X^3+2X^2+2X+2$	$X+2$	$2X^4+X^3+X+2$	X^4+2X^3+2X+1

Table 1: Example computations of quotient, remainder and highest common factor

One possible way of computing large powers of one polynomial modulo another, say $f^n \bmod g$, is to compute each of $f^{2^i} \bmod g$ by iteratively squaring and taking the remainder modulo g. Once sufficiently many f^{2^i} are computed, we can multiply these terms together such that the exponent finishes as n. This is as if $n = \sum_{i=0}^{k} \varepsilon_i 2^i$, with $\varepsilon_i \in \{0,1\}$ then $f^n = \prod_{i=0}^{k} f^{\varepsilon_i 2^i}$.

1.2 Computing Decomposition Groups

Another program called ***Q2.py*** was written to compute the decomposition of a polynomial mod p. It is listed in Section 3.

1.2.1 Analysis of Computational Efficiency

Let us think a bit about the time complexity of these tasks.

Finding the quotient (and remainder) of two polynomials f, g involves iterating over all powers of X in f greater than $\deg(g)$, followed by adding multiples of g each time to cancel out terms. This yields a complexity of $O(\deg g \cdot (\deg f - \deg g))$.

To find the highest common factor of two polynomials, we must run the quotient-remainder algorithm first on (f,g), then on (g,r_1), (r_1,r_2) and so on, with r_i being the i-th remainder. As all remainders have degree $< \deg g$ we can view these tasks as being of $O(\deg g \cdot (\deg r_i - \deg r_{i+1}))$; or the whole task of finding the highest common factor being $O(\deg g \cdot \deg f)$.

In the provided method of factorisation, in which we calculate the highest common factor of f and $\phi_r = X^{p^r} - X$, it is clear that the task must be $O\left(\deg f \cdot p^{\left\lfloor \frac{\deg f}{2} \right\rfloor}\right)$, because we only need to check up to $r = \left\lfloor \frac{\deg f}{2} \right\rfloor$.

However, ***Q2.py*** also utilises a brute force method by simply checking if any possible polynomial is a factor, iterating through linear, quadratic and onwards in increasing degree. By the same logic as before, the maximum degree of any non-trivial factor is $\left\lfloor \frac{\deg f}{2} \right\rfloor$ so the task of checking if one specific polynomial is a factor is at worst $O\left(\left\lfloor \frac{\deg f}{2} \right\rfloor \cdot \left(\deg f - \left\lfloor \frac{\deg f}{2} \right\rfloor\right)\right)$ so the overall task is $O\left(\left\lfloor \frac{\deg f}{2} \right\rfloor \cdot \left(\deg f - \left\lfloor \frac{\deg f}{2} \right\rfloor\right) \cdot p^{\left\lfloor \frac{\deg f}{2} \right\rfloor}\right)$.

The brute force method has worse time complexity than the highest common factor method. However, in practice the brute force method is faster, due to various optimisations such as only considering fairly low order polynomials, and often being able to factor out linear or quadratic terms very quickly. This, along with the fact that brute force is also much more memory efficient, means that it will be the default choice from now on.

As an example of the difference in speed, we ran the code to compute the cycle types of the elements of a selection of 8 polynomials over 17 prime numbers $2 - 59$. The HCF algorithm completed in 2.436 seconds, whereas the brute force algorithm completed in 0.302 seconds.

2 Analysing Specific Polynomials

Below is a list of the polynomials we will discuss:

1. $X^2 + X + 41$
2. $X^3 + 2X + 1$
3. $X^3 + X^2 - 2X - 1$
4. $X^4 - 2X^2 + 4$
5. $X^4 - X^3 - 4X + 16$
6. $X^4 - 2X^3 + 5X + 5$
7. $X^4 + 7X^2 + 6X + 7 = (X^2 - X + 7)(X^2 + X + 1)$
8. $X^4 + 3X^3 - 6X^2 - 9X + 7 = (X^2 + 2X - 7)(X^2 + X - 1)$
9. $X^5 + 36$
10. $X^5 - 5X + 3 = (X^2 + X - 1)(X^3 - X^2 + 2X - 3)$
11. $X^5 + X^3 - 3X^2 + 3$
12. $X^5 - 11X^3 + 22X^2 - 11$
13. $X^6 + X + 1$
14. $X^7 - 2X^6 + 2X + 2$
15. $X^7 + X^4 - 2X^2 + 8X + 4 = (X^3 + 2X + 1)(X^4 - 2X^2 + 4)$
16. $X^7 + X^5 - 4X^4 - X^3 + 5X + 1$

If a polynomial is reducible, we show the irreducible factorisation alongside it.

2.1 Performing Computations

The program ***Q3.py*** was written to utilise ***Q2.py***, introduced in Section 1.2, to compute the decomposition groups modulo p, where $p < 100$. The results are shown in Table 2 and Table 3. It is important to note that on the occasion a polynomial is inseperable $\bmod p$, or equivalently does not produce a decomposition group, we use a dash to denote the non-existence of the group. Moreover, the way the group is represented uses the cycle type of the generator, as we know all the groups (if they exist) must be cyclic.

Prime	Polynomial Group Cycle Type							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
2	(2)	(2)	(3)	-	-	(4)	-	-
3	(2)	(3)	(3)	-	-	-	-	(2, 2)
5	(2)	(3)	(3)	(2, 2)	(4)	-	(2, 2)	-
7	(2)	(3)	-	(2, 2)	(4)	(4)	e	(2)
11	(2)	(2)	(3)	(2, 2)	-	(4)	(2, 2)	(2)
13	(2)	(2)	e	(2, 2)	(2, 2)	(3)	-	(2, 2)
17	(2)	e	(3)	(2, 2)	(2)	(3)	(2, 2)	(2)
19	(2)	(3)	(3)	e	(4)	(3)	e	(2)
23	(2)	(2)	(3)	(2, 2)	(2, 2)	(4)	(2, 2)	(2)
29	(2)	(3)	e	(2, 2)	(2)	(3)	(2, 2)	(2)
31	(2)	(2)	(3)	(2, 2)	(2)	(3)	e	e
37	(2)	(2)	(3)	(2, 2)	(2, 2)	(4)	e	(2, 2)
41	e	(3)	e	(2, 2)	(2)	(3)	(2, 2)	-
43	e	(2)	e	e	(4)	(2)	e	(2, 2)
47	e	(2)	(3)	(2, 2)	(2, 2)	(3)	(2, 2)	(2)
53	e	(3)	(3)	(2, 2)	(4)	(2, 2)	(2, 2)	(2, 2)
59	(2)	-	(3)	(2, 2)	(2, 2)	(3)	(2, 2)	(2)
61	e	(2)	(3)	(2, 2)	(2, 2)	(4)	e	(2)
67	(2)	(2)	(3)	e	(2)	(3)	e	(2, 2)
71	e	e	e	(2, 2)	(2, 2)	(3)	(2, 2)	e
73	(2)	(2)	(3)	e	(2, 2)	(3)	e	(2)
79	(2)	(3)	(3)	(2, 2)	(4)	-	e	e
83	e	(2)	e	(2, 2)	(2, 2)	(4)	(2, 2)	(2, 2)
89	(2)	(2)	(3)	(2, 2)	(4)	(4)	(2, 2)	e
97	e	(2)	e	e	e	(2, 2)	e	(2)

Table 2: Decomposition groups mod p

Prime	Polynomial Groups							
	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
2	-	$(2,3)$	-	(5)	(6)	-	-	(7)
3	-	$(2,2)$	-	(5)	$(2,3)$	-	-	-
5	-	-	(5)	(5)	$(3,3)$	(7)	$(2,2,3)$	$(2,2,2)$
7	(4)	-	(3)	(5)	(5)	(7)	$(2,2,3)$	(7)
11	(5)	(3)	(3)	-	$(2,3)$	-	$(2,2,2)$	$(2,2,2)$
13	(4)	$(2,2)$	(5)	(5)	(6)	$(2,2,3)$	$(2,2,2)$	$(2,2,2)$
17	(4)	$(2,2)$	(5)	(5)	$(2,3)$	$(2,4)$	$(2,2)$	(7)
19	$(2,2)$	(2)	(5)	(5)	$(2,4)$	$(2,4)$	(3)	(7)
23	(4)	$(2,3)$	(5)	e	$(3,3)$	(7)	$(2,2,2)$	$(2,2,2)$
29	$(2,2)$	(3)	$(2,2)$	(5)	$(2,3)$	(7)	$(2,2,3)$	(7)
31	e	(2)	$(2,2)$	(5)	$(2,3)$	(7)	$(2,2,2)$	(7)
37	(4)	$(2,3)$	(5)	(5)	(6)	(7)	$(2,2,2)$	(7)
41	(5)	(2)	-	(5)	(4)	(5)	$(2,2,3)$	(7)
43	(4)	$(2,3)$	(3)	-	(6)	(5)	(2)	(7)
47	(4)	$(2,2)$	(3)	(5)	(6)	(7)	$(2,2,2)$	(7)
53	(4)	(2)	(5)	(5)	(4)	$(2,4)$	$(2,2,3)$	(7)
59	$(2,2)$	(2)	(3)	(5)	(4)	(7)	-	$(2,2,2)$
61	(5)	(2)	(5)	(5)	(6)	$(2,4)$	$(2,2,2)$	$(2,2,2)$
67	(4)	$(2,3)$	(3)	e	(5)	(5)	(2)	$(2,2,2)$
71	(5)	(3)	$(2,2)$	(5)	(4)	(7)	$(2,2)$	$(2,2,2)$
73	(4)	$(2,2)$	$(2,2)$	(5)	$(2,4)$	$(3,3)$	(2)	(7)
79	$(2,2)$	(3)	(3)	(5)	$(2,3)$	$(2,4)$	$(2,2,3)$	$(2,2,2)$
83	(4)	$(2,2)$	(3)	(5)	(5)	(7)	$(2,2,2)$	(7)
89	$(2,2)$	(2)	(3)	e	$(2,4)$	(5)	$(2,2,2)$	(7)
97	(4)	$(2,2)$	(3)	(5)	(3)	$(2,4)$	(2)	$(2,2,2)$

Table 3: Decomposition groups mod p

2.2 Analysis

2.2.1 Irreducible Polynomials

We can perform a few elementary pieces of analysis given the data we have to determine the groups $\mathrm{Gal}(f|\mathbb{Q})$. The following two Lemmas will be useful in our analysis. They have have been either proven in example sheets or given in the project notes.

Lemma 1 If p is an odd prime and f is a degree p polynomial with $\mathrm{Gal}(f|\mathbb{Q}) = G$ having a 2 cycle and a p cycle, then $G = S_p$.

Lemma 2 If n is a number ≥ 3 and f is a degree n polynomial with $\mathrm{Gal}(f|\mathbb{Q}) = G$ having a 2 cycle, an $n-1$ cycle and an n cycle, then $G = S_n$.

We will now discuss the polynomials from the list at the beginning of Section 2.

Quadratics

1. The Galois group is $S_2 = C_2$ as it is the only transitive group of order 2.

Cubics

2. This satisfies the conditions for Lemma 1, and so has group S_3.
3. This has cycle type (3), and the only subgroups of S_3 of that type are C_3 and S_3.

Quartics First we remark that the transitive subgroups of S_4 are C_4, $V = K_4$, D_8, A_4 and S_4.

4. This polynomial's only nontrivial cycle type is $(2,2)$, so we can gather that $\{e, (12)(34)\} \leq \mathrm{Gal}(f|Q)$. This is true of all the transitive subgroups.
5. This polynomial has cycle types $(4), (2,2), (2)$, so must be one of D_8, A_4 or S_4.
6. The conditions of Lemma 2 are satisfied so is S_4.

Quintics The transitive subgroups of S_5: C_5, D_{10}, $GA(1,5) = \langle (1,2,3,4,5), (2,3,5,4) \rangle$, A_5 and S_5.

9. The (4) cycle informs us the group is not C_5, D_{10} nor A_5. We remark $GA(1,5)$ has elements of all cycle types that appear, so the possible groups are $GA(1,5)$ or S_5.
11. The (3) cycle removes the possibility of C_5, D_{10} and $GA(1,5)$ as they have no element of order 3. All cycle types are members of A_5 so the possible groups are A_5 or S_5.
12. This is clearly any group with C_5 as a subgroup. Indeed this is every possible transitive group.

Higher Order Polynomials

13. The existence of a cycle type $(2,3)$ gives the existence of a cycle type (2), by cubing. We also have cycles $(5), (6)$ so this satisfies conditions of Lemma 2, and hence has group S_6.
14. Not much can be said for this polynomial, however all the elements that appear are in A_7. Moreover the cycle types $(2,2)$ and e can be constructed from elements that exist. This means that we expect that the group is some subgroup of A_7, or even A_7 itself.
16. We observe that (7) and $(2,2,2)$ cycle types could correspond to D_{14}, being rotations and reflections respectively. However, this is not necessarily the case as $(1234567)(26)(37)(45) = (163527)$ is a 6 cycle in another valid subgroup of S_7.

2.2.2 Reducible Polynomials

The reducible polynomials were, $7, 8, 10$ and 15 or more explicitly:

7. $X^4 + 7X^2 + 6X + 7 = (X^2 - X + 7)(X^2 + X + 1)$
8. $X^4 + 3X^3 - 6X^2 - 9X + 7 = (X^2 + 2X - 7)(X^2 + X - 1)$
10. $X^5 - 5X + 3 = (X^2 + X - 1)(X^3 - X^2 + 2X - 3)$
15. $X^7 + X^4 - 2X^2 + 8X + 4 = (X^3 + 2X + 1)(X^4 - 2X^2 + 4)$

2.2.2.1 Polynomial 7

$$X^4 + 7X^2 + 6X + 7 = (X^2 - X + 7)(X^2 + X + 1)$$

Considering 7 first, we only get cycles of the form $(2,2)$ or e, leading us to suspect that the true Galois group is just a double transposition $\cong C_2$. We can confirm this by considering the roots, and hence the $\mathbb{Q}$-automorphisms.

$x^2 - x + 7$ gives $x = \frac{1 \pm \sqrt{1-4*7}}{2} = \frac{1 \pm 3\sqrt{-3}}{2}$. So the automorphism must be $\alpha\left(\sqrt{-3}\right) = -\sqrt{-3}$.

Similarly $x^2 + x + 1$, yields $x = \frac{-1 \pm \sqrt{-3}}{2}$ giving the same automorphism.

Thus we may conclude that there is only one nontrivial automorphism for polynomial 7 and thus its group is $\{e, (12)(34)\}$.

2.2.2.2 Polynomial 8

$$X^4 + 3X^3 - 6X^2 - 9X + 7 = (X^2 + 2X - 7)(X^2 + X - 1)$$

Polynomial 8 is easier as we already have $(2,2)$ and (2) as cycles, so we must have at least 2 distinct nontrivial automorphisms. And thus $\mathrm{Gal}(f|\mathbb{Q}) = \{e, (12), (34), (12)(34)\} \cong C_2 \times C_2 \cong V$.

2.2.2.3 Polynomial 10

$$X^5 - 5X + 3 = (X^2 + X - 1)(X^3 - X^2 + 2X - 3)$$

For 10 the existence of cycles of type $(3), (2,3)$ and $(2,2)$ give enough information to demonstrate that we can cover every possible permutation of roots in the two polynomials at the same time.

We can do this by listing the roots of polynomial 10 from 1 to 5 with $\{1,2\}$ being the quadratic roots and $\{3,4,5\}$ being the cubic roots. The first cycle type, (3), must now only be acting on $\{3,4,5\}$ wheras $(2,3)$ must act on both sets simultaneously. WLOG the $(2,3)$ cycle element is exactly $(12)(345)$. The (3) cycle must be either (345) or (354). Either way it is possible to construct the element (12) and so we can generate all automorphisms of the quadratic while fixing the roots of the cubic. To do the same for the cubic, all we need is the $(2,2)$ cycle element which is WLOG $(12)(xy)$ with x, y distinct in $\{3,4,5\}$. Composing appropriate automorphisms, we now have (xy) as its own cycle and thus we can generate S_3, the full Galois group of the cubic polynomial.

Now, as we have now independently constructed the Galois groups of the quadratic and the cubic, the resultant group of the quintic must be $C_2 \times S_3$.

2.2.2.4 Polynomial 15

$$X^7 + X^4 - 2X^2 + 8X + 4 = (X^3 + 2X + 1)(X^4 - 2X^2 + 4)$$

Finally we consider 15. It is important to note that the cubic term is polynomial 2 and the quartic term is polynomial 4. Let these polynomials have groups S_3 and $K \leq S_4$ respectively. K is currently unknown.

We first note that the same paradigm may be established as before by catagorising the roots as $\{1,2,3,4\}$ for the quartic and $\{5,6,7\}$ for the cubic. Now considering $p = 19, 43$ we note that the cycle types are (3) and (2) and, by looking at the data for the factors, we see that this must generate the cubic while fixing the roots of the quartic.

This means that for any p and the corresponding automorphism, we may apply another suitable automorphism in $\{e\} \times S_3$ to yield some new automorphism in $K \times \{e\}$. This new automorphism will have the same cycle type as simply considering the quartic with prime p.

In all this informs us that polynomial 15 must have Galois group $K \times S_3$.

2.3 The Conjecture

2.3.1 Motivation

For certain polynomials, we know the group and can observe a pattern.

- Polynomial 7 has been established to have group $\{e, (12)(34)\}$ and we note that cycle types e and $(2,2)$ occur 10 and 12 times respectively.
- Polynomial 8 has group $\{e, (12), (34), (12)(34)\}$ with the cycle types $e, (2), (2,2)$ appearing $4, 11$ and 6 times respectively.
- Polynomial 5 has a minimal group of D_8 with cycle types $e, (2), (2,2)$ and (4) appearing $1, 5, 9$ and 7 times each.

We can argue that there is a pattern: that the probability of $f \bmod p$ yielding a cycle type C is close to the probability that a randomly chosen element of $G = \mathrm{Gal}(f|\mathbb{Q})$ has the same cycle type. In cases where a 'minimal possible group' is obvious, this pattern seems to hold true assuming we take that group as the Galois group.

2.3.2 Statement

Conjecture Let f be a monic polynomial with coefficients in $\mathbb{Z}$, and G be its Galois group over $\mathbb{Q}$. Then for p an arbitrary prime number, and C an arbitrary cycle type:

$$P(f \bmod p \text{ gives cycle type C}) = P(g \text{ has cycle type C}|\ g \in G)$$

2.3.3 Ramifications

Supposing the conjecture were true, we can use it as a tool to compute Galois groups.

Example 1 Consider polynomial 4. We know its Galois group can be any transitive subgroup of S_4. However, counting the appearances of cycles in Table 2, we have e and $(2,2)$ appearing 5 and 18 times, and nothing else appearing at all. The conjecture suggests that it is highly probable that the true Galois group is V.

Example 2 Consider polynomial 16. We have only cycle types (7) and $(2,2,2)$ occuring 14 and 10 times each. Given they are approximately equal, this lends itself to the idea that the true Galois group is D_{14}.

Example 3 By similar logic on polynomial 14, we have cycle types $(7), (5), (2,4), (3,3)$, and $(2,2,3)$ appearing $10, 4, 6, 1$ and 1 times respectively, which can be considered approximately in line with the group being A_7.

We must note that these are not proofs, but can certainly be used to inform us of what to look out for with more sophisticated analyses. In Table 4 we list out the polynomials and their associated Galois groups, proven or conjectured.

	Polynomial	Galois Group	Proven
1	$X^2 + X + 41$	C_2	Yes
2	$X^3 + 2X + 1$	S_3	Yes
3	$X^3 + X^2 - 2X - 1$	C_3	No
4	$X^4 - 2X^2 + 4$	V	No
5	$X^4 - X^3 - 4X + 16$	D_8	No
6	$X^4 - 2X^3 + 5X + 5$	S_4	Yes
7	$X^4 + 7X^2 + 6X + 7$	$\{e, (12)(34)\} \cong C_2$	Yes
8	$X^4 + 3X^3 - 6X^2 - 9X + 7$	$C_2 \times C_2$	Yes
9	$X^5 + 36$	$GA(1, 5)$	No
10	$X^5 - 5X + 3$	$C_2 \times S_3$	Yes
11	$X^5 + X^3 - 3X^2 + 3$	A_5	No
12	$X^5 - 11X^3 + 22X^2 - 11$	C_5	No
13	$X^6 + X + 1$	S_6	Yes
14	$X^7 - 2X^6 + 2X + 2$	A_7	No
15	$X^7 + X^4 - 2X^2 + 8X + 4$	$V \times S_3$	No
16	$X^7 + X^5 - 4X^4 - X^3 + 5X + 1$	D_{14}	No

Table 4: Theorised and known Galois groups

2.3.4 Testing

A program called ***Q4.py*** was written to test higher values of primes for certain polynomials and compute approximate probability distributions of cycle types. It is found in Section 3.

This is useful as we can provide a few examples of polynomials which do not necessarily agree with the established pattern in Table 2.

1. This polynomial has group C_2 and has 8 and 17 appearances of e and (2) respectively.
2. This polynomial has group S_3 and with cycle types $e, (2)$ and (3) occuring 2, 14 and 8 times respectively.
6. This polynomial has group S_4, with cycle types $(4), (3), (2, 2)$ and (2) appearing $8, 11, 2$ and 1 time respectively.

After running ***Q4.py*** we ended up getting the following probabilities:

Polynomial	Probabilities			
	Initial	**Theoretical**	**n**	**Updated**
1. $X^2 + X + 41$	$(2) : 0.68$ $e : 0.32$	$(2) : 0.5$ $e : 0.5$	2000	$(2) : 0.5397351$ $e : 0.4602649$
2. $X^3 + 2X + 1$	$(3) : 0.3333333$ $(2) : 0.5833333$ $e : 0.0833333$	$(3) : 0.3333333$ $(2) : 0.5$ $e : 0.1666667$	2000	$(3) : 0.3377483$ $(2) : 0.5132450$ $e : 0.1490066$
6. $X^4 - 2X^3 + 5X + 5$	$(4) : 0.3636363$ $(3) : 0.5$ $(2) : 0.0454545$ $(2, 2) : 0.0909091$ $e : 0$	$(4) : 0.25$ $(3) : 0.3333333$ $(2) : 0.3333333$ $(2, 2) : 0.1666667$ $e : 0.0416667$	500	$(4) : 0.25$ $(3) : 0.3695652$ $(2) : 0.2065217$ $(2, 2) : 0.1304348$ $e : 0.0434783$

Table 5: Probabilities of getting cycle types for all primes $< n$

We note that all these polynomials now agree more closely with the theoretical results. Also we remark that n is lower for the quartic polynomial as the program became incredibly slow, so a lower bound was selected in a trade-off between precision and performance.

3 Programs

3.1 Q1.py

```
import numpy as np

def GetInverses (p):
    inv = [0 for i in range(p)]
    for i in range(1, p):
        if (inv[i] != 0): continue
        for j in range(1, p):
            if ((i*j) % p != 1): continue
            inv[i] = j
            inv[j] = i
            break

    return inv

# Work under assumption that deg f >= deg g
def NegatePoly(p, inv, f, degf):
    h = [-f[i] % p for i in range(degf + 1)]
    return h

def AddPolys(p, inv, f, degf, g, degg):
    h = [0 for i in range(degf + 1)]
    for i in range(degg + 1):
        h[i] = (f[i] + g[i]) % p

    for i in range(degg + 1, degf + 1):
        h[i] = f[i] % p

    return h

def SubProcedure(p, inv, f, degf, g, degg):
    q = f[degf] * inv[g[degg]] % p

    r = [f[i] for i in range(degf + 1)]
    for i in range(degg + 1):
        r[i + degf - degg] = (r[i + degf - degg] - q * g[i]) % p

    return q, r

def Reduce(f):
    d = 0
    for i in range(len(f)):
        if (f[i] != 0):
            d = i

    return f[:d+1], d

def GetQuotRem (p, inv, f, degf, g, degg):
    r = [f[i] for i in range(degf + 1)]
    q = [0 for i in range(degf - degg + 1)]

    for i in range(degf - degg + 1):
        q[degf - degg - i], r = SubProcedure(p, inv, r, degf - i, g, degg)
```

```
    r, degr = Reduce(r)

    return q, r, degr

def PolysHCF (p, inv, f, degf, g, degg):
    degh = degf
    h = [f[i] for i in range(degf+1)]
    degk = degg
    k = [g[i] for i in range(degg+1)]

    while (degk > 0 or k[0] != 0):
        q, r, degr = GetQuotRem(p, inv, h, degh, k, degk)
        h = k
        degh = degk
        k = r
        degk = degr

    for i in range(degh+1):
        h[i] = h[i] * inv[h[degh]] % p

    return h, degh


def GenRandPoly(p):
    deg = np.random.randint(1, 7)
    f = [1 for i in range(deg+1)]
    for i in range(0, deg):
        f[i] = np.random.randint(1, 100) % p
    return f, deg

def Test():
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]

    for i in range(5):
        p = primes[np.random.randint(0, len(primes))]
        inv = GetInverses(p)
        f, degf = GenRandPoly(p)
        g, degg = GenRandPoly(p)
        print(p, f, g, end=" ")
        print(GetQuotRem(p, inv, f, degf, g, degg), end=" ")
        print(PolysHCF(p, inv, f, degf, g, degg), end=" ")
        print()

if __name__ == "__main__":
    Test()
```

3.2 Q2.py

```
from Q1 import *
import numpy as np
import time

def CheckSeperable(p, inv, f, degf):
    g, degg = Reduce([f[i+1]*(i+1) % p for i in range(degf)])
    hcf, d = PolysHCF(p, inv, f, degf, g, degg)
    return (d == 0)

def ReduceModP(p, f, degf):
    g, degg = Reduce([f[i] % p for i in range(degf+1)])
    return g, degg

def DecomposePolyModP(p, f, degf):
    inv = GetInverses(p)
    g, degg = ReduceModP(p, f, degf)
    isSep = CheckSeperable(p, inv, g, degg)
    if (not isSep): return False

    # Note we only need to check degrees up to floor(degf/2)
    # moreover if we are systematic once we find a factor we only need
    # to consider the poly divided out by said factor
    # to be even more efficient we only need to check irriducible polys
    # but that seems more difficult :/

    factors = []

    maxFactorDeg = int(degf/2)
    for j in range(1, maxFactorDeg + 1):

        maxNum = np.power(p, j)
        i = 0
        while(i < maxNum):
            # Use each of these numbers as a polynomial
            prelimh = [int(i/np.power(p, k)) % p for k in range(j)]
            prelimh.append(1)
            h, degh = Reduce(prelimh)

            q, r, degr = GetQuotRem(p, inv, g, degg, h, degh)

            i += 1

            if (degg <= degh):
                break

            if (r != [0]):
                continue

            g = q
            degg = degg - degh
            factors.append(h)

    factors.append(g)

    return factors
```

```
def PrintFactorisation(factors):
    for factor in factors:
        flag = False
        print("(", end="")
        for j in range(len(factor)):
            i = len(factor) - j - 1
            if (factor[i] == 0): continue
            if (flag): print(" + ", end="")

            if (i == 0):
                print(str(factor[i]), end="")
            elif (i == 1):
                if (factor[i] == 1):
                    print("X", end="")
                else:
                    print(str(factor[i]) + "X", end="")
            else:
                if (factor[i] == 1):
                    print("X^{" + str(i) + "}", end="")
                else:
                    print(str(factor[i]) + "X^{" + str(i) + "}", end="")

            flag = True
        print(")", end="")
    print("$ & ", end="")

def GetCycleTypeModP(p, f, degf):
    factors = DecomposePolyModP(p, f, degf)
    if (factors == False):
        return factors

    cycleType = []
    for i in range(len(factors)):
        ord = len(factors[i]) - 1
        if (ord != 1):
            cycleType.append(ord)

    return cycleType


def GetCycleTypeHCF(p, f, degf):

    inv = GetInverses(p)
    isSep = CheckSeperable(p, inv, f, degf)
    if (not isSep): return False

    cycleType = []
    sameOrderFactors = []
    i = 1
    while (i < int(np.floor(degf/2) + 1)):
        # Initialise phi
        ptoi = int(np.power(p, i))
        phi = [0 for i in range(1 + ptoi)]
        phi[1] = p - 1
        phi[ptoi] = 1
```

```
        hcf, degh = PolysHCF(p, inv, f, degf, phi, len(phi) - 1)
        if (degh > 0):
            sameOrderFactors.append(hcf)
            t = degh
            while (t > 0 and i != 1):
                cycleType.append(i)
                t -= i
            f, r, degr = GetQuotRem(p, inv, f, degf, hcf, degh)
            degf -= degh

        i += 1

    if (degf > 1):
        cycleType.append(degf)
    sameOrderFactors.append(f)

    return cycleType


if __name__ == "__main__":
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59]#  61, 67,
71, 73, 79, 83, 89, 97]

    polys = [
            [41, 1, 1],
            [1, 2, 0, 1],
            [-1, -2, 1, 1],
            [4, 0, -2, 0, 1],
            [16, -4, 0, -1, 1],
            [5, 5, 0, -2, 1],
            [7, 6, 7, 0, 1],
            [7, -9, -6, 3, 1],
    ]
    degs = [2, 3, 3, 4, 4, 4, 4, 4]

    t = time.time()

    for p in primes:
        for i in range(0, 8):
            GetCycleTypeHCF(p, polys[i], degs[i])


    print(time.time() - t)
    t = time.time()

    for p in primes:
        for i in range(0, 8):
            GetCycleTypeModP(p, polys[i], degs[i])


    print(time.time() - t)
    t = time.time()
```

3.3 Q3.py

```
from Q2 import GetCycleTypeModP


def FindDecompGroupModP(p, f, degf):
    cycleType = GetCycleTypeModP(p, f, degf)
    if (cycleType == False):
        print("[-], ", end="")
        return

    print("[", end="")
    if (cycleType == []):
        print("$e$", end="")
    else:
        print("$(", end="")
        for i in range(len(cycleType)):
            if (i != 0): print(", ", end="")
            print(cycleType[i], end="")
        print(")$", end="")
    print("], ", end="")


def ComputeDecompsOfPolys():

    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]

    polys = [
            [41, 1, 1],
            [1, 2, 0, 1],
            [-1, -2, 1, 1],
            [4, 0, -2, 0, 1],
            [16, -4, 0, -1, 1],
            [5, 5, 0, -2, 1],
            [7, 6, 7, 0, 1],
            [7, -9, -6, 3, 1],
            [36, 0, 0, 0, 0, 1],
            [3, -5, 0, 0, 0, 1],
            [3, 0, -3, 1, 0, 1],
            [-11, 22, 0, -11, 0, 1],
            [1, 1, 0, 0, 0, 0, 1],
            [2, 2, 0, 0, 0, 0, -2, 1],
            [4, 8, -2, 0, 1, 0, 0, 1],
            [1, 5, 0, -1, -4, 1, 0, 1]
    ]
    degs = [2, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 7, 7, 7]

    for p in primes:

        print("\n[" + str(p) + "], ", end="")
        for i in range(0, 8):
            FindDecompGroupModP(p, polys[i], degs[i])

    print()

    for p in primes:
```

```
        print("\n[" + str(p) + "], ", end="")
        for i in range(8, 16):
            FindDecompGroupModP(p, polys[i], degs[i])


if __name__ == "__main__":
    ComputeDecompsOfPolys()
```

3.4 Q4.py

```
from Q2 import GetCycleTypeModP
import numpy as np


def ComputeProbabilities():

    polys = [
            [41, 1, 1],
            [1, 2, 0, 1],
            [5, 5, 0, -2, 1],
    ]
    degs = [2, 3, 4]

    # This is an inefficient test but will do for now.
    def TrialDivisionTest(n):
        if (n <= 1): return False
        for i in range(2, int(np.sqrt(n)) + 1):
            if (n % i == 0):
                return False
        return True

    maxNum = [2000, 2000, 500]

    for i in range(0, 3):
        dictionary = {}
        keys = []
        for p in range(2, maxNum[i]):
            # Test Primality
            if (TrialDivisionTest(p) == False): continue

            cycleType = GetCycleTypeModP(p, polys[i], degs[i])
            if (cycleType == False):
                continue
            string = "("
            for cyc in cycleType:
                string += str(cyc) + ","
            string += ")"

            if (string in keys):
                dictionary[string] = dictionary[string] + 1
            else:
                dictionary[string] = 1
                keys.append(string)

        total = 0
        for key in keys:
            total += dictionary[key]
        for key in keys:
            dictionary[key] = dictionary[key] / total

        print(dictionary)

if __name__ == "__main__":
    ComputeProbabilities()
```