Project 1: Systems of linear disequations

Computer Algebra

Zirui Yan r0916941

April 2023

Task 1

(b) I wrote a MAGMA function **Heuristic(A::ModMatRngElt, B::SeqEnum, s:: SeqEnum, maxit::RngIntElt) -> BoolElt** which returns true if **s** is the unique solution. I use **RandomDiseq()** in task1 (a) first to generate disequation system and then use **Heuristic()** helps me to check if my guess for $C_N n$ is correct or not.

At first, I thought I could replace b with b':

$$Ax \not\equiv b => Ax = b'$$

Since I have p-1 choices for b', I have to raise the number of equations to p-1 times to ensure that enough possible solutions are eliminated. I use my **Heuristic()** to check my idea (because what I said is not a formal proof, so I'm sure about it). When n is large enough, $C_N = p - 1$ can guarantee unique solution. But when n is small (perhaps less than 6), $C_N = p-1$ is not enough and $C_N = O(N^2)$ might be a better estimation. My explanation for this is that when n is small, **RandomDiseq()** probably generate 2 disequations which are linearly dependent, so we need more disequations when n is small.

(c) True. $\phi(N)$ is the Euler's totient function, which counts the positive integers up to a given integer N that are relatively prime to N. $\phi(N)$ also represents the number of invertible elements in $\mathbb{Z}_N$. If we have a solution s for the system of disequations $Ax \not\equiv 0$. We can use one of the invertible elements in $\mathbb{Z}_N$ to multiply s. The product will also be a solution for the system of disequations $Ax \not\equiv 0$. Since these elements are invertible, it is not hard to prove that the products are different from each other. So The number of solutions to a homogeneous system of linear disequations over $\mathbb{Z}_N$ is always a multiple of $\phi(N)$.

Task 2

(a) $\mathbb{F}_2$ only has 2 elements. If the disequation is $\neq 0$, we can replace $\neq 0$ by $= 1$. Similarly, we can replace $\neq 1$ by $= 0$.

(b) If e is an non-zero element in $\mathbb{Z}_p$, then $e^{p-1} - 1 \equiv 0 \ (mod\, p)$. But if $e = 0$, $e^{p-1} - 1 \equiv -1 \ (mod\, p)$. Use this property, we can simply write down **FromDiseqToEq**.

(c) It always runs out of memory when n is large and p=5 or 7. So I choose to give the results with n=5:

p	r	runtimes
2	1	0
3	1	0.003
5	1	0.016
7	1	0.219

You can see that the runtimes are not very large, but it requires a lot of memory to compute. And the runtime increase quickly when p grows.

The following table is the runtimes with p=3, n=20 and different r (p=3 do not need so many memory):

p	n	r	runtimes
3	10	1	1.362
3	10	2	1.378
3	10	3	1.391
3	10	4	1.340

Changing r does not spend more time significantly.

For the last question, adding field equations can largely reduce the runtimes, but it cannot solve the memory problem.

(d) If we use **EulerPhi()** instead of p-1. it will work when N=p is prime. But it will give wrong result when N is not prime, because $a^{\phi} \equiv 1 \ mod \ N$ requires a and N coprime.

Task 3

(a) $((x-c)^{p-1} - 1)((y-d)^{p-1} - 1)$ can map (c,d) to a and all other points to 0.

(b) Digit-extraction map $\delta : \mathbb{Z}_{p^2} -> \mathbb{F}_p^2 : c_0 + c_1 p \mapsto (c0, c1)$. It is obvious that we can consider the elements in $\mathbb{Z}_{p^2}$ as p-base 2-digit numbers. So at p^0 place, it is just addition of two number. but when $Rep(c) + Rep(d) \geq p$, there is a carry. So at p^1 place, we should also add $W(c_0, d_0)$.

(c) We can compute f_0 and f_1 by repeatly using the result from (b). Since $a_1 x_1 + ... + a_n x_n - b = 0$ iff $f_0 = 0$ and $f_1 = 0$, we should find a polynomial f suh that $f = 0$ iff one of f_0 and f_1 not equal to 0. And I choose f to be $(f_0^{(p-1)} - 1) * (f_1^{(p-1)} - 1)$.

(d) Since it easily runs out of memory when $p^2 = 9$, I choose n=3 when p^2=9 (the runtimes are the average value from 8 experiments):

p	n	r	runtimes
4	3	1	0.002
9	3	1	0.433
4	3	2	0.002
4	3	3	0.007

The conclusion is the same as in task2 c. And I can find my solution is in the variety, but there are also some other points in the variety.

(e)$\mathbb{F}_p^2$ is easier. p is a prime number, so we can use linearization in task 5 to solve the system over $\mathbb{F}_p^2$.

Task 5

(a) When we have n unknowns, we have $n^{N-1} + n^{N-2} ... + n^0 = (1 - n^N)/(1 - n)$ monomials according to task2 (b). So to make the linearized system have a unique solution, we need at least $(1 - n^N)/(1 - n)$ linear disequations to use Gaussian elimination.

Actually these new indeterminates already have some relationships before we get the linearized equations. For example: when $p = 3$,

$$x_1x_2 * x_1x_2 = x_1^2 * x_2^2$$

But these relations are not linear, so they cannot be used when using Gaussian elimination. However, we can use these relationship to check which solution is the real solution when the solution is not unique. (when the number of linear disequations is less than n^{N-1}, there come some solution which do not satisfy such relationships.)

Task 7

Follow the instruction to write the functions and we can use **RandomDiseq()** to help us generate C. (Compared to using **RandomMatrix()**, using **RandomDiseq()** to generate C row by row is more effective.)

The table for the signing and verification times and the sizes of the private key s, the public key A with different N and n is as followed:

N	n	signing time t_1	verification time t_2	size of s	size of A
6	30	4.656	1.860	30*1	150*30
6	60	17.656	7.469	60*1	300*60
6	15	1.109	0.469	15*1	75*15
3	30	1.114	0.735	30*1	60*30
12	30	12.703	4.406	30*1	330*30

The signing and verification times above are the average value of several experiments. We can see that both signing and verification times are proportional to N^2 and n^2. n and N are proportional to the number of rows of s, the number of rows of A and the number of columns of A.

For λ, we should find prime numbers of the form $2^\lambda - 1$ because we need $2^\lambda - 1$ to be prime in our **FakeHash()**. The prime numbers of this form are called Mersenne prime. λ could be 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607... But $\lambda > 607$ will require too much time. So I choose $\lambda = 31$ and $\lambda = 521$ with $N = 6$ and $n = 30$ in my table:

λ	signing time t_1	verification time t_2
31	0.922	0.313
127	4.656	1.860
521	24.500	12.813

So both signing and verification times are almost proportional to N and n.

Task 8

According to the result given in task5 (linearization technique), if we have enough linear disequations, we can solve the system of linear disequation by Gaussian elimination. Since $P(c = 0) = 50\%$, repeated runs of the protocal eventually give us many different **C** such that **C*t** $=$ **C*A*s** $\not\equiv 0$, which gives enough number of disequations.

Task 9

The difficulty for constructing **DiseqSignForge()** is that we need to know what c is before generating C or m, But C and m will affect the possible value of M. To solve this problem, I randomly generate a fake private key s. Although the real private key needs to satisfy $A \cdot s \not\equiv 0$, our fake private key s does not need to. And even some entries in $A \cdot s$ are equal to 0, we still can satisfy $M \cdot m = C \cdot A \cdot s \not\equiv 0$. My implementation procedure of **DiseqSignForge()** is as followed:

First, generate 127 C_i's based on fake private key s such that $C_i^{trunc(n)}$ invertible and $C \cdot A \cdot s \not\equiv 0$. (We also get M_i in this step.) Then, compute $c := H((A, mess, M_1, ..., M_\lambda))$. Implement the impersonator's method given in task 6 to forge the response based on $c = c_1 c_2 ... c_\lambda$. The signature tuple is the same as the one in task 6.