Homework 1

September 25, 2024

[1]:
```
import sympy as sp
import numpy as np
import galois as gf
import pandoc

np.set_printoptions(legacy='1.25')
```

[2]:
```
# import os
# import shutil
# import sys

# print(shutil.which("c:
 ↪\\Users\\coopb\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\pandoc"))
# print(shutil.which('pandoc'))
# print(shutil.which("C:
 ↪\\Users\\coopb\\AppData\\Local\\Programs\\Python\\Python311\\python.EXE"))
# environpath = os.environ['PATH']
# print(f'os.environ[\'Path\']={environpath}')

# print(f'os.defpath={os.defpath}')
# whchsympy = shutil.which('sympy')
# print(f'shutil.which(\'sympy\')={whchsympy}')

# print('')
# print(shutil.which('git'))

# for i in sys.path :
#     print(i)
# print(os.environ.get("PATH", None))
# #C:\Users\coopb\AppData\Local\Programs\Python\Python311\Lib\site-packages
# #c:\Users\coopb\AppData\Local\Programs\Python\Python311\Lib\site-packages;
```

```
# #C:\Users\coopb\AppData\Local\Programs\Python\Python311\Lib;C:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311\DLLs;C:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311\lib-tk;C:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311\Scripts;e:
 ↪\OneDrive\Documents\VSCodeProjects\Cryptography-Homework;c:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311\python311.zip;c:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311;c:
 ↪\Users\coopb\AppData\Local\Programs\Python\Python311\Lib\site-packages;
```

1 Homework 1

1.1 Problem 1

In a finite field GF(p), we can divide a polynomial $a(x)$ by another polynomial $b(x)$ to obtain a quotient $q(x)$ and a reminder $r(x)$ that satisfy:

(i) $a(x) \overset{p}{\equiv} b(x).q(x) + r(x)$,

(ii) $deg(r(x)) < deg(b(x))$.

Here is an example of how it works. Let $a(x) = x^3 + x + 2$, $b(x) = 2x + 1$ and $p = 3$. Note that for a polynomial $m(x)$, we denote the coefficient of x^k by m_k. For instance, we have $(b_1, b_0) = (2, 1)$: - Set $q(x) = 0$; - Find the multiplicative inverse of b_1 in GF(3), i.e., $2^{-1} \overset{3}{\equiv} 2$. - Find the difference between the degree of $a(x)$ and $b(x)$, which is $d = 3 - 1 = 2$. - Set s(x) $= b_1^{-1} * a_3 * x^d = 2x^2$ and update $q(x) \leftarrow q(x) + s(x) = 2x^2$. - Update $a(x) \leftarrow a(x) - b(x) \cdot s(x) = x^3 + x + 2 - 4x^3 - 2x^2 \overset{3}{\equiv} x^2 + x + 2$. - Find the difference between the degree of $a(x)$ and $b(x)$, which is $d = 2 - 1 = 1$. - Set $s(x) = b_1^{-1} * a_2 * x^d = 2x$ and update $q(x) \leftarrow q(x) + s(x) = 2x^2 + 2x$. - Update $a(x) \leftarrow a(x) - b(x) \cdot s(x) = (x^2 + x + 2) - 4x^2 - 2x \overset{3}{\equiv} 2x + 2$. - Find the difference between the degree of $a(x)$ and $b(x)$, which is $d = 1 - 1 = 0$. - Set $s(x) = b_1^{-1} * a_1 * x^d = 1$ and update $q(x) \leftarrow q(x) + s(x) = 2x^2 + 2x + 1$. - Update $a(x) \leftarrow a(x) - b(x) \cdot s(x) = (2x + 2) - 2x - 1 \overset{3}{\equiv} 1$. - Set $r(x) = a(x) = 1$, since $deg(a(x)) < deg(b(x))$.

Therefore, at the end, we get $q(x) = 2x^2 + 2x + 1$ and $r(x) = 1$.

(a) Write a code that takes two polynomials $a(x)$ and $b(x)$, together with a prime number p and returns $q(x)$ and $r(x)$. Note that polynomial $m(x) = m_k x^k + m_{k-1} x^{k-1} + \cdots + m_1 x + m_0$ can be represented in the form of the vector of coefficients, i.e., $[m_k, m_{k-1}, ..., m_2, m_1, m_0]$.

[3]:
```
# Calculate a polynomial
def calc_polynomial(poly_a, input) :
    degree_a = len(poly_a)-1
    i=degree_a
    output = 0
    while i>=0 :
        output = poly_a[degree_a-i]*input**i + output
        i-=1
    return output
```

```
#Finds inverse in modulo. e.g. 2 inverse in mod 3 is 2 b/c 2*2 = 4(mod 3) = 1
#==Inputs==
#input: value to find the inverse of
#modulo: value of the modulus
def find_modulo_inverse(input, modulo) :
    i=modulo
    while i>0 :
        if np.mod(input*i,modulo) == 1 :
            return i
        i-=1
    return 0

#Finds the degree of a polynomial vector by ignoring finding the first non-zero␣
 ↪value.
def find_degree(polynomial) :
    shift = 0
    for i in polynomial :
        if i!=0 :
            return len(polynomial)-shift-1
        else :
            shift +=1
    return 0

#Removes all zeros in a polynomial vector preceeding the first non-zero element.
#e.g. [0, 0, 0, 2, 0, 3, 0] --> [2, 0, 3, 0]
def remove_zeros(polynomial) :
    shift = 0
    for i in polynomial :
        if i!=0 :
            while shift>0 :
                polynomial.pop(0)
                shift-=1
        else :
            shift +=1
    return polynomial


#Description:   follows the steps outlined in Problem 1 to divide a polynomial␣
 ↪a(x) by another polynomial, b(x)
#               to get the quotient q(x) and the remainder r(x)
#==Inputs==
#a: Vector of polynomial coefficients starting at the highest degree. e.g. a(x)␣
 ↪3x^2 + 1 --> poly_a=[3 0 1]
#b: Vector of polynomial coefficients to divide into a(x), same format as a.␣
 ↪Note: len(b)<len(a) must be true!
#GF: Galois Field order, e.g. GF(p) --> GF=p, where p must be prime.
#==Outputs
```

```
#divide_in_GF[0] = q(x)
#divide_in_GF[1] = r(x)
def divide_in_GF(poly_a, poly_b, GF):
    degree_a = find_degree(poly_a) # Get degree of each polynomial
    print(f"degree a(x): {degree_a}")
    degree_b = find_degree(poly_b)
    print(f"degree b(x): {degree_b}")
    poly_q = 0

    #Calculate degree difference between a(x) and b(x)
    if degree_a > degree_b :
        degree_diff = degree_a - degree_b
        print(f"degree difference: {degree_a} - {degree_b} = {degree_diff}")
    else :
        print("ERROR: degree of a < degree of b !")
        return 0

    bk_inverse = find_modulo_inverse(poly_b[0],GF) # find inverse of highest␣
 ↪power of b
    print(f"b{degree_b} = {bk_inverse}")

    a_index = 0 #index of poly_a, to be looped over
    poly_s = [0] * (degree_a) #Create an empty matrix of n=degree_diff elements
    poly_q = [0] * (degree_a)
    while a_index<degree_a :
        print('')
        print(f'STARTLOOP a_index={a_index}')
        print(f'degree_diff = {degree_diff}')


        sq_index = len(poly_s)-1-degree_diff #starts at the first non-zero␣
 ↪element and decrements with degree_diff
        print(f'sq_index={sq_index}')
        print(f'a_index={a_index}')

        poly_s[sq_index] = np.mod((bk_inverse*poly_a[a_index]),GF)
        print(f's(x) = {poly_s}')

        poly_q =  np.mod(np.polyadd(poly_s,poly_q),GF)
        print(f'q(x) = {poly_q}')
        print(f's(x)*b(x)={np.mod(np.polymul(poly_s,poly_b),GF)}')

        poly_a = np.mod(np.polysub(poly_a,np.mod(np.
 ↪polymul(poly_s,poly_b),GF)),GF)
        print(f'a(x) = {poly_a}')

        poly_s[sq_index]=0 #reset s(x) = 0 for next loop
```

```
        print('ENDLOOP')
        if degree_diff == 0 : #end loop condition.
            break

        #incrementing/decrementing values
        a_index+=1
        sq_index+=1
        degree_diff = find_degree(poly_a) - find_degree(poly_b)


    #remove all the preceeding zeros and output answers.
    remainder = remove_zeros(poly_a.tolist())
    poly_q = remove_zeros(poly_q.tolist())

    return (poly_q,remainder)

#pretty simple function to print polynomial coefficient vectors readably
def print_polynomial(polynomial) :
    degree = len(polynomial)-1
    for i in polynomial :
        print(f'{i}x^{degree} +')
        degree-=1
```

[4]:
```
#The Problem 1 Example
a = [1, 0, 1, 2]
b = [2, 1]
GF = 3

example1 = divide_in_GF(a,b,GF)
```

```
degree a(x): 3
degree b(x): 1
degree difference: 3 - 1 = 2
b1 = 2

STARTLOOP a_index=0
degree_diff = 2
sq_index=0
a_index=0
s(x) = [2, 0, 0]
q(x) = [2 0 0]
s(x)*b(x)=[1 2 0 0]
a(x) = [0 1 1 2]
ENDLOOP

STARTLOOP a_index=1
degree_diff = 1
```

```
sq_index=1
a_index=1
s(x) = [0, 2, 0]
q(x) = [2 2 0]
s(x)*b(x)=[1 2 0]
a(x) = [0 0 2 2]
ENDLOOP

STARTLOOP a_index=2
degree_diff = 0
sq_index=2
a_index=2
s(x) = [0, 0, 1]
q(x) = [2 2 1]
s(x)*b(x)=[2 1]
a(x) = [0 0 0 1]
ENDLOOP
```

Let's print out the answer to see if it's correct.

[5]:
```
print('q(x) =')
print_polynomial(example1[0])
print('')
print('r(x) =')
print_polynomial(example1[1])
```

```
q(x) =
2x^2 +
2x^1 +
1x^0 +

r(x) =
1x^0 +
```

Yep! That looks like $q(x) = 2x^2 + 2x + 1$ and $r(x) = 1$. to me!

(b) Let $a(x) = 5x^8 + 3x^3 + 2x^2 + 4$ and $b(x) = 4x^3 + x^2 + 6$. Compute $q(x)$ and $r(x)$ in GF(7) using your code.

[6]:
```
a = [5, 0, 0, 0, 0, 3, 2, 0, 4]
b = [4, 1, 0, 6]
GF = 7

answer = divide_in_GF(a,b,GF)
```

```
degree a(x): 8
degree b(x): 3
degree difference: 8 - 3 = 5
b3 = 2
```

```
STARTLOOP a_index=0
degree_diff = 5
sq_index=2
a_index=0
s(x) = [0, 0, 3, 0, 0, 0, 0, 0]
q(x) = [0 0 3 0 0 0 0 0]
s(x)*b(x)=[5 3 0 4 0 0 0 0 0]
a(x) = [0 4 0 3 0 3 2 0 4]
ENDLOOP

STARTLOOP a_index=1
degree_diff = 4
sq_index=3
a_index=1
s(x) = [0, 0, 0, 1, 0, 0, 0, 0]
q(x) = [0 0 3 1 0 0 0 0]
s(x)*b(x)=[4 1 0 6 0 0 0 0]
a(x) = [0 0 6 3 1 3 2 0 4]
ENDLOOP

STARTLOOP a_index=2
degree_diff = 3
sq_index=4
a_index=2
s(x) = [0, 0, 0, 0, 5, 0, 0, 0]
q(x) = [0 0 3 1 5 0 0 0]
s(x)*b(x)=[6 5 0 2 0 0 0]
a(x) = [0 0 0 5 1 1 2 0 4]
ENDLOOP

STARTLOOP a_index=3
degree_diff = 2
sq_index=5
a_index=3
s(x) = [0, 0, 0, 0, 0, 3, 0, 0]
q(x) = [0 0 3 1 5 3 0 0]
s(x)*b(x)=[5 3 0 4 0 0]
a(x) = [0 0 0 0 5 1 5 0 4]
ENDLOOP

STARTLOOP a_index=4
degree_diff = 1
sq_index=6
a_index=4
s(x) = [0, 0, 0, 0, 0, 0, 3, 0]
q(x) = [0 0 3 1 5 3 3 0]
s(x)*b(x)=[5 3 0 4 0]
a(x) = [0 0 0 0 0 5 5 3 4]
```

```
ENDLOOP

STARTLOOP a_index=5
degree_diff = 0
sq_index=7
a_index=5
s(x) = [0, 0, 0, 0, 0, 0, 0, 3]
q(x) = [0 0 3 1 5 3 3 3]
s(x)*b(x)=[5 3 0 4]
a(x) = [0 0 0 0 0 0 2 3 0]
ENDLOOP
```

[7]:
```
print('q(x) =')
print_polynomial(answer[0])
print('')
print('r(x) =')
print_polynomial(answer[1])
```

```
q(x) =
3x^5 +
1x^4 +
5x^3 +
3x^2 +
3x^1 +
3x^0 +

r(x) =
2x^2 +
3x^1 +
0x^0 +
```

1.2 Problem 2

A square matrix is invertible (or full-rank) if and only if its determinant is non-zero or, equivalently, its rows are linearly independent. Let

$$

$$A = \begin{bmatrix} - & a_1 & - \\ - & a_2 & - \\ - & a_3 & - \\ & \vdots & \\ - & a_n & - \end{bmatrix}, \tag{1}$$

$$

be a **full-rank matrix** with its elements from GF(q), where a_i is the ith row of matrix A.

(a) How many choices for a_1 exists?

(b) Once you fix a_1, how many choices for a_2 we have, such that a_1 and a_2 are linearly independent?

(c) If a_1 and a_2 are linearly independent, how linear combination of a_1 and a_2 exixts?

(d) Once you fix a_1, a_2, how many choices for a_3 do we have, such that a_3 is linearly independent from (a_1, a_2)?

(e) Given rows $a_1, ..., a_{m-1}$, how many valid choices exist for a general row a_m to have a full-rank matrix?

(f) In general, how many full-rank matrices of size n in GF(q) exists?

(g) How many matrices (regardless of their rank) are in GF(q)?

(h) If you generate an $n \times n$ matrix uniformly at random, what is the probability that it is full-rank?

[ ]:

1.3 Problem 3

Write a computer program that takes a prime number p and an irreducible polynomial $m(x)$ of degree k in GF(p) to generate GF(p^k) and its operations. Let α be a root of $m(x)$. Note that each element of GF(p^k) is of the form of $b = b_{k-1}\alpha^{k-1} + b_{k-2}\alpha^{k-2} + \cdots + b_{1\alpha} + b_0$, which you can represent by $B = [b_{k-1}, b_{k-2}, ..., b_1, b_0]$. Then, you need to implement two commands

- MyGFAdd(A, B, M, p) that returns the summation of a and b in GF(p^k);
- MyGFMult(A, B, M, p) that returns the multiplication of a and b in GF(p^k).

[ ]:

1.4 Problem 4

In this problem, you are supposed to decode a ciphertext that is encrypted using the Vig'enere cipher. You should return the key, the plaintext, and your codes.

You may write all the codes from scratch or use the provided MATLAB codes to simplify your task. The provided files are.

- ciphertext.txt in which the ciphertext is provided.
- EngAlphabetFrequency.mat in which the empirical frequency of the English alphabet is provided. Note that the first character is space, which is followed by 26 letters.
- find rep.m that finds all repeats of a given length in a given text. For instance: > find rep('this is the best and simplest scheme.',4)

  {[14 27]}

as 'est ' appears in positions 14 and 27 of the sentence. > find rep('this is the best and simplest scheme.',5)

  0×0 empty cell array

as there is no repeat of length 5 in the sentence.

• EmpiricalFrequency.m that returns the empirical frequency of letters in a text. Also, plot the histograms of the ciphertext and the plaintext.