Module 11: Hill Cipher

JHU EP 606.206 - Introduction to Programming Using Python

Introduction

The Hill Cipher is a polygraphic encryption cipher that is based on some basic linear algebra and modular arithmetic. It can be broken via chosen plaintext attacks (CPA) where an attacker has access to a particular plaintext as well as its corresponding ciphertext. This week we'll implement the Hill Cipher for a 2x2 key matrix using NumPy arrays and various concepts we've learned so far throughout the course. First, we'll introduce some of the foundational mathematical concepts that will be of use in the assignment, followed by more information on the Hill Cipher, and ending with details for the assignment.

Skills: NumPy matrices, functions, exceptions

Foundational Concepts

Linear Algebra:

1. Matrix multiplication is done by taking the dot product of the i^{th} row of the first matrix with the i^{th} column of the second matrix. Note that if the first matrix has dimension mxn (m rows and n columns) then, for the product to exist, the second matrix must have n rows and can have an arbitrary number columns k. (In other words, the number of columns on the first matrix **must** match the number of rows on the second matrix.)

$$(m \times n) \times (n \times k)$$

2. A matrix A is invertible if it is a square matrix and its determinant is non-zero. Let's take a look at this sample matrix A:

$$\mathbf{A} \equiv \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

3. The inverse of a matrix A, denoted A^{-1}, can be found using its determinant:

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Modular Arithmetic:

1. The modular multiplicative inverse of a number modulus m is an integer b such that when a is multiplied by b and then reduced modulo m the result is 1.

$$a^{-1} = ab \equiv 1 \mod m$$

Example: The modular multiplicative inverse of 3 mod 11 = 4 because when 3 (a) is multiplied by 4 (b) and then reduced modulo 11, 12 mod 11 = 1. As another example, the modular multiplicative inverse of 4 mod 23 is 6 because (4x6) mod 23 = 24 mod 23 = 1.

The modular multiplicative inverse of a number can be found recursively using the extended Euclidean algorithm. It can also be found fairly simply using an iterative method that checks to each number from 1 to m in succession to find when the result of the multiplication is 1 mod m.

Hill Cipher Example:

Let's dive straight into an example for how these principles can be used in practice: the Hill Cipher. First we'll take a look at encrypting using the Hill Cipher, followed by decryption on the next page.

Encryption:

We'll start with our key matrix K and our plaintext matrix P:

$$K = \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}$$

$$P = \begin{bmatrix} H & L \\ E & P \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}, \begin{bmatrix} 11 \\ 15 \end{bmatrix}$$

Next, we multiply our key matrix K times our two plaintext vectors and take the result mod 26:

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 7 \\ 4 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 8 \end{pmatrix} \pmod{26}, \text{ and}$$

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 11 \\ 15 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 19 \end{pmatrix} \pmod{26}$$

We then convert the results back to their letter equivalents to get our encrypted ciphertext:

$$\begin{pmatrix} 7 \\ 8 \end{pmatrix}, \begin{pmatrix} 0 \\ 19 \end{pmatrix} \rightarrow \begin{pmatrix} H \\ I \end{pmatrix}, \begin{pmatrix} A \\ T \end{pmatrix}$$

Encrypted ciphertext: **HIAT**

Decryption:

To decrypt we calculate the modular multiplicative inverse of our key matrix K and converting our text-based ciphertext back into its numeric equivalent. A matrix is invertible if its determinant is non-zero.

$$K^{-1} \equiv 9^{-1}\begin{pmatrix} 5 & 23 \\ 24 & 3 \end{pmatrix} \equiv 3\begin{pmatrix} 5 & 23 \\ 24 & 3 \end{pmatrix} \equiv \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \pmod{26}$$

$$HIAT \rightarrow \begin{pmatrix} H \\ I \end{pmatrix}, \begin{pmatrix} A \\ T \end{pmatrix} \rightarrow \begin{pmatrix} 7 \\ 8 \end{pmatrix}, \begin{pmatrix} 0 \\ 19 \end{pmatrix}$$

Now we multiply the inverse of our key matrix K times our encrypted ciphertext:

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix}\begin{pmatrix} 7 \\ 8 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 4 \end{pmatrix} \pmod{26}, \text{and}$$

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix}\begin{pmatrix} 0 \\ 19 \end{pmatrix} \equiv \begin{pmatrix} 11 \\ 15 \end{pmatrix} \pmod{26}$$

Finally, we take the result and convert it back into its text equivalent to obtain the original plaintext message.

$$\begin{pmatrix} 7 \\ 4 \end{pmatrix}, \begin{pmatrix} 11 \\ 15 \end{pmatrix} \rightarrow \begin{pmatrix} H \\ E \end{pmatrix}, \begin{pmatrix} L \\ P \end{pmatrix} \rightarrow HELP$$

Programming Assignment

For this assignment you will implement the Hill Cipher. When performing encryption and decryption, you will break the plaintext/ciphertext into 2-element vectors exactly as shown in the example above.

Encoding:

Please use the following scheme to map letters to their integer equivalents:

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Inputs:

1. Plaintext (P): ATTACKATDAWN
   Key matrix (K):
   $$K = \begin{bmatrix} 19 & 8 & 4 \\ 3 & 12 & 7 \end{bmatrix}$$
   Modulus (m): 26

2. Plaintext (P): ATTACKATDAWN
   Key matrix (K):
   $$K = \begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix}$$
   Modulus (m): 26

3. Plaintext (P): ATTACKATDAWN
   Key matrix (K):
   $$K = \begin{bmatrix} 5 & 15 \\ 4 & 12 \end{bmatrix}$$
   Modulus (m): 26

For each of the preceding inputs your code should:

1. Test if the given key matrix is invertible using the **`invertible()`** function (described below).
    a. If the matrix is invertible, please print: "The matrix is invertible." And proceed to #2.
    b. If the matrix is not invertible, raise a `MatrixNotInvertible` exception (described below) that displays the reason:
        i. "The matrix is not square."
        ii. "The determinant = 0."
2. Encrypt the given plaintext using the matrix key.
    a. Implement a function called **`encrypt()`** that takes a key matrix (NumPy array) and a plaintext string and returns a single matrix (NumPy array) containing the ciphertext.
    b. Print the resulting ciphertext (string) to the screen.
    c. Print the resulting ciphertext (NumPy array) to the screen.
3. Decrypt the resulting ciphertext from #2.
    a. Implement a function called **`decrypt()`** that accepts the ciphertext array (NumPy array) and returns the decrypted message in its original string form.
    b. Print the resulting plaintext (string) to the screen.
    c. Print the resulting plaintext (numeric NumPy array) to the screen.

Sample Output (plaintext before encrypting shown for demonstration purposes only). For reference, our solution contains ~90 lines of code for implementing the various functions and ~40 lines of code for implementing the "main" logic that makes use of those functions.

```
The matrix is not square.

Plaintext: ATTACKATDAWN
Plaintext column vectors: [array([[ 0],
       [19]]), array([[19],
       [ 0]]), array([[ 2],
       [10]]), array([[ 0],
       [19]]), array([[3],
       [0]]), array([[22],
       [13]])]

Ciphertext: WBDBQCWBVHYV
Ciphertext column vectors: [array([[22],
       [ 1]]), array([[3],
       [1]]), array([[16],
       [ 2]]), array([[22],
       [ 1]]), array([[21],
       [ 7]]), array([[24],
       [21]])]

Plaintext: ATTACKATDAWN
Plaintext column vectors: [array([[ 0],
       [19]]), array([[19],
       [ 0]]), array([[ 2],
       [10]]), array([[ 0],
       [19]]), array([[3],
       [0]]), array([[22],
       [13]])]

The determinant = 0.
```

Suggested Functions

For this assignment, we are not providing psuedocode for you to follow, but rather a text description of the functions that you may want to implement. Remember that you must use Numpy to satisfy this assignment – and reviewing Numpy's capabilities before starting is recommended. Here are some functions we believe you might find useful:

`determinant()`:

1. Accepts a matrix, calculates and returns its determinant

`invertible()`:

1. Calls **`determinant()`** and returns `True` if the matrix is invertible and `False` otherwise

`mod_inverse()`:

1. Accepts a number n (determinant) and a modulus m and returns the modular multiplicative inverse of n modulo m

`encrypt()`:

1. Accepts a matrix key K and a plaintext string P
2. Breaks the plaintext string into a matrix of appropriate size and dimensions ()
3. Computes the encrypted ciphertext using the Hill Cipher approach above. Remember, the ciphertext matrix C is given by multiplying the key matrix K by the plaintext matrix P and taking the result modulo m:

$$C = [K] \times [P] \mod m$$

4. Prints the ciphertext (numeric NumPy array) to the screen.
5. Prints the ciphertext (string) to the screen.
6. Returns the ciphertext matrix (NumPy array) C

`decrypt()`:

1. Accepts a ciphertext matrix C (the NumPy array returned by **`encrypt()`**)
2. Calculates the modular multiplicative inverse of the determinant of K (using **`mod_inverse()`**) and multiplies the scalar result by the key matrix
3. Uses the result of #2 and the ciphertext matrix to obtain the (numeric) plaintext matrix. Remember, the plaintext can be retrieved by multiplying the inverse of the key matrix K by the ciphertext matrix C:

$$P = [K]^{-1} \times [C] \mod m$$

4. Prints the plaintext (numeric NumPy array) to the screen
5. Prints the plaintext (string) to the screen (should match the initial plaintext input)
6. Returns the plaintext matrix (NumPy array) P

Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your ***hill.py*** source code file and a screenshot of the fully working output generated by **hill.py** in a PDF called JHEDID_mod11.pdf (ex: jkovba1_mod11.pdf). Please do not ZIP your files together.

Recap:

1. readme.txt
2. Your source code in a file named **hill.py**
3. A PDF containing screenshots of your successful outputs/exceptions described in the "Programming Assignment".

Please let us know if you have any questions via Teams or email!