**HElib-basic**

**Baseline.cpp:**

Lines 69-104: Reads in data from dec.txt and stores a column of 1’s at the first column + the remaining columns of data in dec.txt (except last column) as mat1 (i.e. Matrix X). The last column of the data from dec.txt will be stored as mat2 (i.e. Matrix Y).

Lines 106-180: Normalize matrices X and Y (still stored as mat1 and mat2 respectively).

Lines 182-185: Finding the transpose of mat1 (i.e. transpose of matrix X), stored as mat\_trans.

Lines 187-189: Multiplying transpose of mat1 and matrix X together (mat\_trans \* mat1).

Lines 191-193: Finding the inverse of mat\_trans \* mat1, stored as XtransX\_Inv.

Lines 195-196: Finding XtransX\_Inv \* mat\_trans, stored as XtransXInv\_Xtrans.

Lines 200-202: Finding the final product – (XTX)-1XTY, stored as matrix.

Line 205: Obtains the value of phi m.

1) Splitting into fractional parts and integer parts of the respective decimals and

2) Encoding it 🡪 ZZX form.

- Lines 208-223: For (XTX)-1. Encoded form is stored in inv\_matrix.

- Lines 225-240: For XT. Encoded form is stored in x\_matrix.

- Lines 242-257: For mat2 (i.e. matrix Y). Encoded form is stored in y\_matrix.

Lines 259: Encryption begins.

Lines 262-271: Encrypting (XTX)-1 (inv\_matrix) and stored in inv\_enc.

Lines 273-282: Encrypting XT (x\_matrix) and stored in xtrans\_enc.

Lines 284-293: Encrypting Y (y\_matrix) and stored in y\_enc.

Line 295: End of Encryption.

Line 302: Multiplying the ctxt of inv\_enc and xtrans\_enc, storing it in ctxt\_mat\_temp.

Line 305: Multiplying the ctxt\_mat\_temp and y\_enc, storing it in ctxt\_mat.

Lines 309-315: Decrypting and Decoding ctxt\_mat. Should get back same result as what’s found in lines 200-202.

Lines 317-323: Test if first multiplication (i.e. ctxt\_mat\_temp) is correct (if needed).

**Functions in encoding.h:**

1. Frac\_Part

2. Int\_Part

3. Encode

4. frac\_encoder

5. frac\_to\_ZZX

6. Decrypt

7. Decode

**encoding.cpp:**

1. Frac\_Part – Returns the fractional parts from the decimal values in the matrix.

2. Int\_Part – Returns the integer parts from the decimal values in the matrix.

3. Encode – Returns binary of a single integer value.

Method used: eg. 6

6 / 2 = 3 + 0

3 / 2 = 1 + 1

1 / 2 = 0 + 1

=> Binary of 6: [1 1 0].

If integer is negative, eg. -6, => binary of -6: [-1 -1 0].

4. frac\_encoder – Returns binary of a single fractional value.

Method used: eg. 0.875

0.875 \* 2 = 1 + 0.75

0.75 \* 2 = 1 + 0.5

0.5 \* 2 = 1 + 0

=> Binary of 0.875: [1 1 1], i.e. x-1 + x-2 + x-3.

=> Add n to each exponent and flip the sign of each terms, i.e. -xn-1 - xn-2 - xn-3.

=> Lines 118-120: If the fractional part overflows into integer part, equate it to 0. \*\*Range of k has to be reset for each data set here (line **119**).

5. frac\_to\_ZZX – Combining 3. and 4. together to return a binary for a decimal value (integer + fractional value).

6. Decrypt – Decrypt and convert each ZZ into integer.

7. Decode – If the element at any index/position is greater than the floor of p/2 (i.e. floor(17/2) = 8 for now), take that element and subtract it by p. In a polynomial [ . . . . ], the first section of it is for the integer part while the remaining section is for the fractional part. \*\*Range of j here has to be reset for each data set to determine the proportion of the polynomial to be set for integer and fractional parts (line **206**).

=> For the fractional part: take the negative of each element and multiply it by 2j – phim.

result1 sums up the total value of the fractional parts.

=> For the integer part: take each element and multiply it by 2j  (j is its index).

result2 sums up the total value of the integer parts.

**Functions in MatrixUtility.h:**

1. matrix\_transpose

2. dotprod

3. Inv

4. mat\_mat\_mult

**MatrixUtility.cpp:**

1. matrix\_transpose – Returns the transpose of the matrix.

2. dotprod – Multiplication of matrices.

3. Inv – Returns the inverse of a matrix.

4. mat\_mat\_mult – Multiplication function for multiplying of ctxt. (more specifically, vec<vec<ctxt>>).

**dec.txt**: data set file.

**Some data sets tried:**

1. 4 2

0.341 0.98

-2.5 0.76

0.12 0.4

5 2

range: 1/7 (for 1st multiplication only).

2. 3 2

0.341 0.98

-2.5 0.76

0.12 0.4

range: 1/8

3. 15 3

3 0 4

2 0 1.25

1 0 4

3.5 1 8

2 1 21

1 1 20

3 2 26

2 2 22

1 2 23

3 3 15

2 3 12

1 3 9

3 4 7

2 4 4

1 4 4

range: 1/10 or 1/11 (for 1st multiplication only)

4. 6 3

2011 6.2 26.3

2012 6.5 26.65

2013 5.48 25.03

2014 6.54 26.01

2015 7.18 27.9

2016 7.93 30.47

range: 1/7 (for 1st multiplication only).

**Problems:**

1. 1st multiplication able to get answer but not for 2nd multiplication (can’t seem to find the right range for k and j).

2. Sometimes, 1st multiplication is not exactly correct, some values at few places are off.

3. After decrypting, answer still differs with each run.

4. Random interrupted error at times.