AUGMENTING & COMPRESSING THE FAREY TABLE WITH APPLICATIONS TO IMAGE PROCESSING

By

Kishaloy Halder

&

Soham Das

Under the kind supervision of

Dr. Partha Bhowmick

Assistant Professor

Computer Science and Engineering Department

IIT Kharagpur

Declaration

No part of this report has been previously submitted by the author for a degree at any other university and all the results contained within unless otherwise stated are claimed as original. The results obtained are all due to fresh work.

Kishaloy Halder

Soham Das

Abstract

Farey Sequence has been a point of interest for the mathematicians over the years. Here, we have discussed techniques of generating the Farey Sequence, creating the Farey Table from it and compressing it in an efficient manner. Searching the fraction, in Farey sequence, closest to a given fraction with an arbitrarily large denominator, in minimum time is another very important issue. Most importantly, all the techniques have been developed using only integer operations as floating point operations need extra space and as well as time and leads to truncation of the actual value of a fraction.

Keywords: Farey sequence, Farey table, compressing the Farey table.

Contents

List of Figures

Chapter 1

Introduction

1.1 A Brief Overview of the Farey Sequence

Farey sequence of order *n* is a sequence of completely reduced fractions between 0 and 1 which have denominators less than or equal to *n*, arranged in ascending order.

Each Farey sequence starts with the value 0, denoted by the fraction $^0/_1$, and ends with the value 1, denoted by the fraction $^1/_1$

A few Farey sequences:-

$F_1 = \{^0/_1, ^1/_1\}$

$F_2 = \{^0/_1, ^1/_2, ^1/_1\}$

$F_3 = \{^0/_1, ^1/_3, ^1/_2, ^2/_3, ^1/_1\}$

$F_4 = \{^0/_1, ^1/_4, ^1/_3, ^1/_2, ^2/_3, ^3/_4, ^1/_1\}$

Except for F_1 , each has an odd number of terms and the middle term is always 1/2.

1.2 The Farey table

A Farey table for Fn is a 2-D matrix in which each row represents a numerator and each column represents a denominator.

Thus a location {i, j} in this matrix corresponds to the fraction i/j in Fn and it contains the index of the fraction in Fn.

For ex:-F4: {0/1, 1/4/, 1/3, 1/2, 2/3, 3/4, 1/1}

Index: 1 2 3 4 5 6 7

The redundant fractions are filled up by the

same Farey index as that of the reduced fraction.

The Farey sequence generates proper fractions between 0 & 1. So the improper zone in the table (lower triange in the matrix) has been filled up with an invalid marker - 1 s.

Fig1.	1	2	3	4
0	1	1	1	1
1	7	4	3	2
2	-1	7	5	4
3	-1	-1	7	6
4	-1	-1	-1	7

The Farey index of a fraction has a direct correspondence with the value of the fraction. So if we need a comparative study among fractions then we can refer to their Farey index in a sequence rather than computing their values. We are given 2 fractions and asked to find if they are close to one another (the degree of proximity is known).We can avoid the floating point operations and can get an estimate by simply looking at the indices of the 2 fractions from the Farey table.

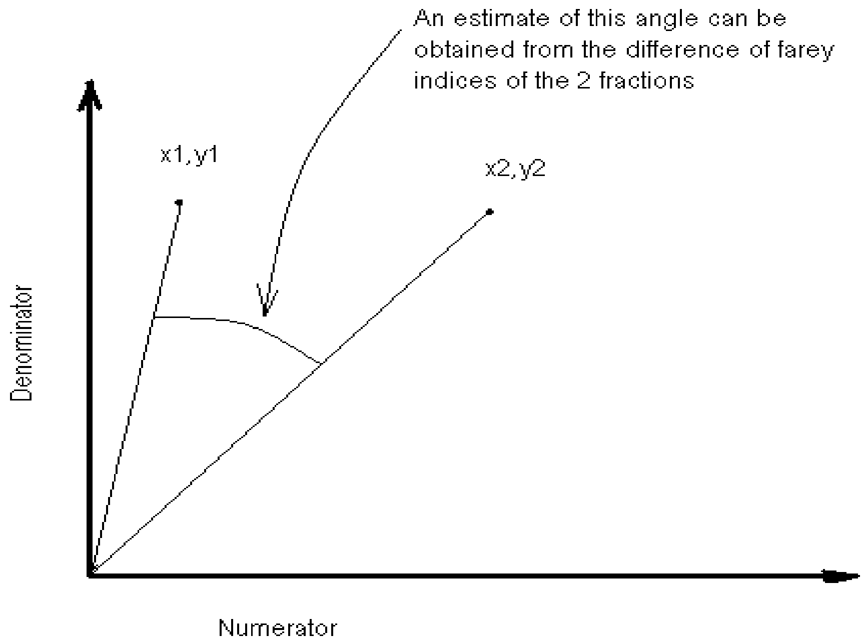

Fig. 2

For an example: Computation of difference between 51/99 & 50/100.

In modern compilers floating variables require 8 bytes of memory. So both the fractions (51/99 & 50/100) will need 8B each. Thereafter there will be a subtraction between the two, which is again a floating point operation. These operations are very expensive in terms of time. Instead we can find an estimate by looking at the Farey indices of the two and check out the difference which is an integer operation (requiring 4B for each fraction), faster than the previous one.

To search a fraction in the Farey sequence we need a binary search of O(log n) whereas from the table we can obtain the Farey index of a fraction in O(1) time.

Each fraction x/y can be thought as a pixel in the 2D plane where x and y are the coordinates. Thus this table has immense applications in Digital Image Processing as we can get a measure of the proximity of pixels.

Chapter 2

Previous Work

2.1 History of Farey sequence

British mathematician John Farey invented this amazing procedure to generate proper fractions between 0 & 1 in the year 1816. He proposed the generation of vulgar fractions in the Philosophical Magazine, but he didn't provide the formal proof. In the later years other mathematicians came up with the formal proof and some interesting properties of the sequence.

2.2 Properties of the Farey Sequence

- For any two successive Farey fractions a/b and c/d in F_n, b + d $\geq$ n + 1, and cb − ad = 1 for ab<cd.
- One of the outstanding properties of the Farey fractions is that given any real number x, there is always a `nearby' Farey fraction a/b belonging to n such that |x-a/b| <= 1 / (b*(n+1)).
- The no. of fractions in the Farey sequence of order n is $3n^2/\pi^2$ approximately $0.304n^2$.
- The mediant of two fractions a/b and e/f is defined by Mediant(a/b , e/f) = (a+e) / (b+f), which lies in the interval (a/b , e/f). Each term in a Farey series . . .a/b ,c/d ,e/f. . .is the mediant of its two neighbours: c/d= (a + e)/(b + f). In fact, the mediant of any two terms is contained in the Farey series, unless the sum of their (reduced) denominators exceeds the order *n* of the series.
- Another important practical application of Farey fractions is the solution of Diophantine equations. Suppose we are looking for a solution of
  $243b - 256a = 1$
  in integer *a* and *b*. By locating the Farey fraction just below 243/256, namely 785/827, we find *a* = 785 and *b* = 827. Check: 243 * 827 = 200961 and
  256 * 785 = 200960.

2.3 Generation of fractions using Farey tree

While Farey sequences have many useful applications, such as classifying the rational numbers according to the magnitudes of their denominators, they suffer from a great irregularity: the number of additional fractions in going from Farey sequences of order n-1 to those of order *n* equals the highly fluctuating Euler function $\varphi(n)$. A much more regular order is infused into the rational numbers by *Farey trees*, in which the number of fractions added with each generation is simply a power of 2.

Starting with two fractions, we can construct a Farey tree by repeatedly taking the mediants of all numerically adjacent fractions. For the interval [0, 1], we start with 0/1 and 1/1 as the initial fractions, or "seeds". The first five generations of the Farey tree then appear as follows:

$$\frac{0}{1} \qquad \frac{1}{1}$$

$$\frac{1}{2}$$

$$\frac{1}{3} \qquad \frac{2}{3}$$

$$\frac{1}{4} \qquad \frac{2}{5} \qquad \frac{3}{5} \qquad \frac{3}{4}$$

$$\frac{1}{5} \qquad \frac{2}{7} \qquad \frac{3}{8} \qquad \frac{3}{7} \qquad \frac{4}{7} \qquad \frac{5}{8} \qquad \frac{5}{7} \qquad \frac{4}{5}$$

Fig. 3

Each rational number between 0 and 1 occurs exactly once somewhere in the infinite Farey tree.

The location of each fraction within the tree can be specified by a binary address, in which 0 stands for moving to the left in going from level n to level n+1 and 1 stands for moving to the right. Thus, starting at 1/2, the rational number 3/7 has the binary address 011. The complement of 3/7 with respect to 1 (i. e., 4/7) has the complementary binary address: 100.

Chapter 3

The Farey table

3.1 A closer look to Farey Table

Though Farey sequence has been a field of interest for many of the mathematicians all over the world it was not that well renowned in the computer application. But as more & more image processing oriented works are being explored, the need for faster algorithms regarding floating point arithmetic has grown beyond expectations. In many systems it is very critical to respond within a finite & short time span while operating on a huge amount of data.

In these circumstances the concept of Farey Table may help us a bit. Truly speaking the concept of Farey Table is not that old. The possibilities of utilization of Farey table are yet to be explored. We have worked on the basic properties of the Farey table and have tried to implement them in practical applications.

Some of its properties directly come from the properties of Farey Sequence. We will state the correspondence whenever it is necessary. The properties are discussed below.

3.2 Properties of Farey Table

- The Farey index gives us a very good measure of relative values of different fractions. To establish this correspondence we have plotted value of the fractions Vs Farey indices from sequences of different orders.

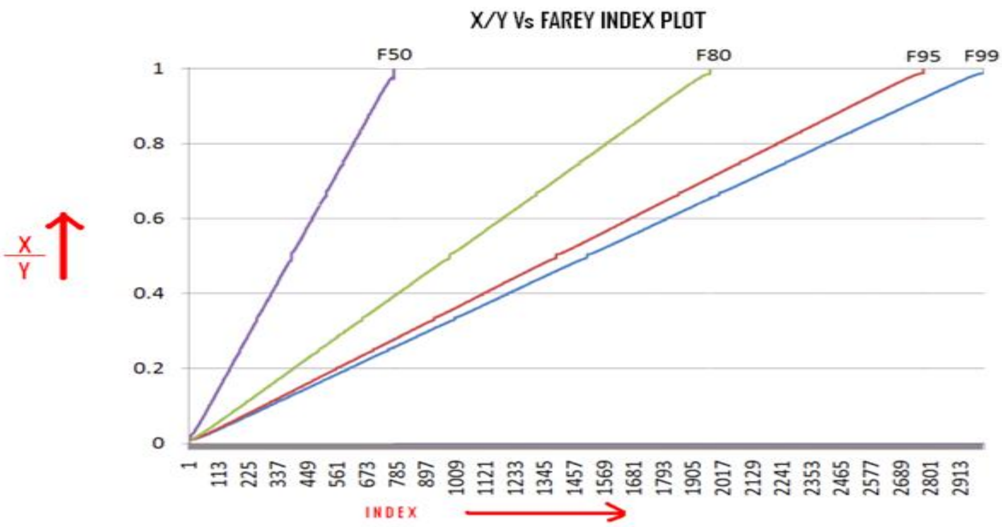

Fig 4.

In the above plot apart from a very few undulations we can see that the nature of the curve is almost linear. As the order increases the undulations are reduced.

Hence we can say that the Farey indices are related linearly to the fractions. So to find the proximity of 2 given fractions, we can simply refer to a Farey table that has both the fractions & get an estimate in constant time by studying how far they are in the sequence as described earlier.

- The maximum index value present in a Farey table of order n is approximately $3n^2/\pi^2$ i.e. $0.304n^2$.
  We have stated that there are $3n^2/\pi^2$ fractions in the sequence of order n. And index is just the positional value of a fraction in the sequence. Now, 1/1 is the last fraction of the sequence. So its index is approximately $3n^2/\pi^2$.

- Indices are decreasing row-wise from left to right.
  Now along a row the numerators of all the fractions are same say x & the denominators are x, x+1, x+2, x+3, … , n-1, n. It is trivial that,
  (x/x) > (x/(x+1)) > (x/(x+2)) > (x/(x+3))>….> (x/(n-1))> (x/n).
  We know if f(fraction) is Farey index of a fraction then for 2 fractions frac1 & frac2,

  frac1>frac2
  =>f(frac1) > f(frac2).

  So in the above case,

  f(x/x)> f(x/x+1)> f(x/x+2)> f(x/x+3)> …. > f(x/n-1)> f(x/n).

  Hence follows the property.

- Indices are increasing along a column downwards.
  This property can be proved using similar logic as of the above.

- Differences between consecutive indices along a column are symmetric.
  If we consider a particular column no. c it looks like
  f(0/c), f(1/c), f(2/c),…., f((c/2)/c),…., f((c-2)/c), f((c-1)/c), f(c/c).
  Now the differences between consecutive indices will look like,
  D_1, D_2, D_3, ... , $D_{c/2}$, ... ,D_3 , D_2 , D_1.
  This property comes from the fact that the fractions in the Farey sequence are symmetric.
  If we consider the distance of a fraction (a/b) (<1/2) from the beginning of the sequence and the distance of the complement of (a/b) i.e. (1-(a/b)) from the end then the two will be the same.

- In column no. c the sum of the jth index & (c-j)th index is equal to (max +1) where max is the maximum index in the table.

Basically this property comes from the previous one. As the distance of a fraction & its complement fraction from both the ends are same, when we take the position of these two from the beginning the sum is the length of the sequence i.e. the maximum index.

A general Farey Sequence

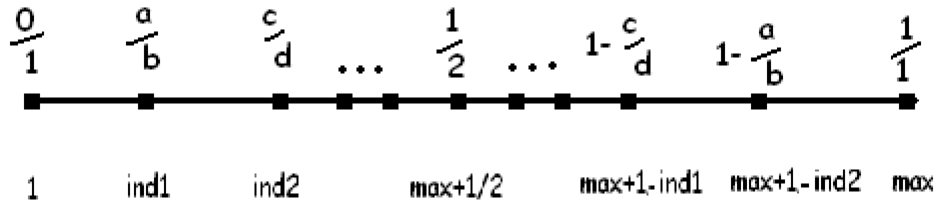

Fig 5

Here we can easily see that for each fraction the sum of the two distances is max+1 which proves the property.

There are also some useful observations that we have found while studying a no. of tables of different orders. These will be discussed later.

3.3 Necessity of Farey Table

- To search a fraction in the Farey sequence we need a binary search of O(log n) whereas from the table we can obtain the Farey index of a fraction in O(1) time.
- We are given 2 fractions and asked to find if they are close to one another (the range is given).We can avoid the floating point operations and can get an estimate by simply looking at the indices of the 2 fractions from the Farey table.
- Each fraction x/y can be thought as a pixel in the 2D plane where x and y are the coordinates. Thus this table has immense applications in Digital Image Processing as we can get a measure of the proximity of pixels.

3.4 Technique of creating a Farey Table

To generate the Farey Table we need to generate the Farey Sequence first. Then using that sequence we can go for the Farey Table.

3.4.1 Creation of the Farey Sequence

The generation of the Farey sequence is quite straightforward. The generation of fractions in the Farey sequence can be better understood from the following figure.

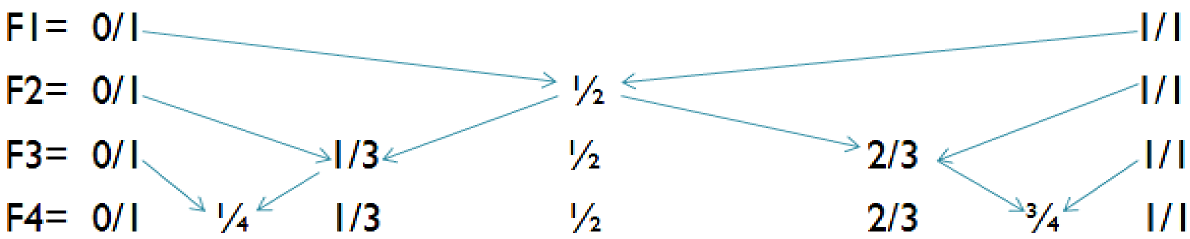

Fig 6

Here we can see that except F1 in any other sequence say Fn the ith fraction has the numerator as the sum of the numerators of ith & (i-1)th fraction in Fn-1 and similarly the denominator as the sum of the denominators of the two. And in case the newly formed denominator is found to be greater than the order of the sequence then that fraction is discarded & the ith fraction in Fn-1 directly comes down as the ith fraction in Fn.

Algorithm:

We know that a Farey Sequence is sorted in ascending order. Hence, instead of following the conventional recurrence we can simply generate the sequence as follows:

FareySeq (order : n)

1. All the proper fractions with denominators ≤ order are generated and stored in an array of structures **a** in the following way:-

```
a[0].x←0;   a[0].y←1;   j←1;   k←0;              //k is used as the index of array a

repeat until (j≤n)

        i←1;

        repeat until (i<j)

                if ( gcd(i,j)==1 )

                        a[k].x←i;

                        a[k].y←j;

                        k←k+1;

                else

                        continue;

        End loop

End loop.
```

a[k].x←-1;

2. The formed array is sorted using quicksort.

3. End.

N.B: From the definition, generation of Farey Sequence takes exponential time, but here, we need only $O(n^2 \log n)$ time.

3.4.2 Farey Table Generation

Now we just fill up the Farey Table with the positions of the fractions under consideration. Whenever we get a duplicate fraction (e.g. 2/4 duplicate to ½) we fill the cell with the index of the completely reduced fraction. The following algorithm generates the Farey Table from a given Farey sequence that is produced by the previous algorithm.

Algorithm

FareyTable(a[ ],order:n)

We scan the array **a** obtained from 3.4.1 algorithm. For each fraction encountered we update the corresponding Farey table (tab [ ][ ]) cell and all the cells that can be obtained by multiplying the numerator & denominator of the fraction with the same integer.

```
k←0;                          //k is used as the index of array a
repeat until (a[k].x!=-1)
        l←2;                  //l is the integer used for multiplying the fraction
        tab[a[k].x][a[k].y]←k+1;
        repeat until ((a[k].y*l)≤(2*i+1))
                tab[a[k].x*l][a[k].y*l]←k+1;
                l←l+1;
        End loop;
End loop;
```

Chapter 4

Compression of the Farey table

4.1 Need for Compression

For higher order Farey tables say F_{1000} we need a 1000X1000 matrix to represent it. This eats up a lot of space. So if we can go for an efficient algorithm to compress the Farey table, minimizing the errors and store the compressed one in memory we can save some space and carry out accessing the fractions easily.

Now the rows in a Farey table show some irregularities. So compression can be done by dropping columns maintaining the tabular structure using an appropriate algorithm.

4.2 The Technique

We have taken the respective Farey table, stored in a file, and the degree of compression from the user as inputs. For compression, we have used the following observations.

Useful Observations

- The maximum differences between two consecutive columns (difference between elements at the same level in 2 consecutive columns) and the levels at which they occur are computed and stored in the arrays- the diff and the zonelevel arrays respectively. Based on the compression degree we move to a certain range of values in the diff array to get to a threshold that meets up the desired compression percentage. The differences are non-increasing from left to right.

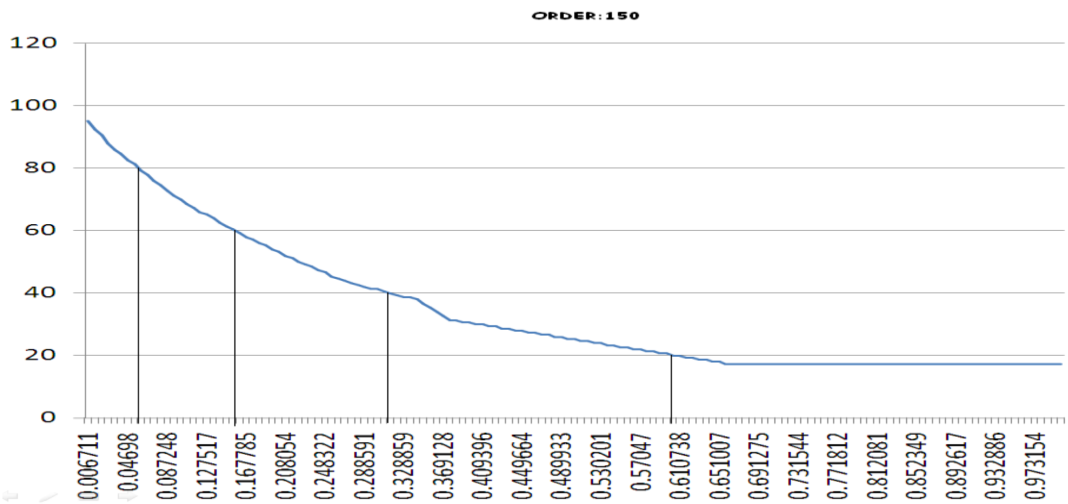

Fig 7: Compression % vs. relative position of threshold in F_{150}.

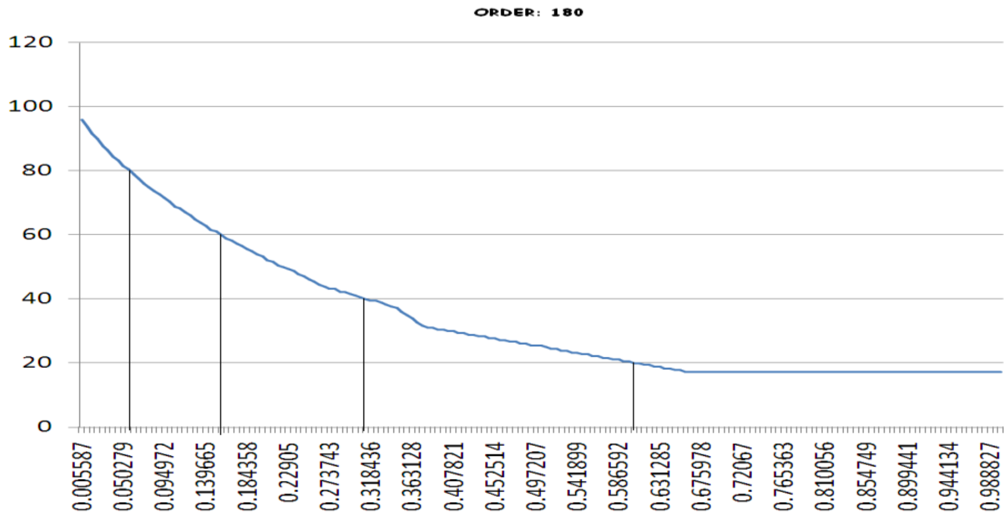

Fig. 8: Compression % vs. relative position of threshold in F_{180}.

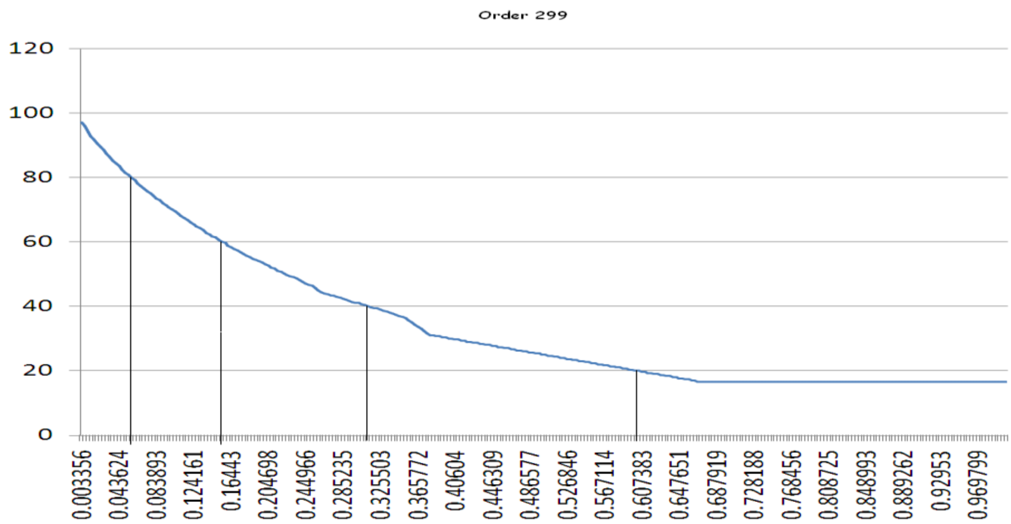

Fig. 9: Compression % vs. relative position of threshold in F_{299}.

N.B. When the max diff between 2 columns of the Farey table is less than this threshold we compress the 2 into a single column.

In the above figures we have seen that for any Farey table, given a degree of compression the relative position of the threshold in the diff array is approximately a constant. Thus depending on the degree we can limit our threshold search in the diff array within certain ranges. In that range we now apply the bisection method i.e. we start from the middle value in the range, go for compression, check the degree. If it is close to the desired value (+/- 5%) we ask for a prompt from the user if he is satisfied else we continue towards a closer value by the same method.

After selecting the threshold we scan the diff array to get the 1st value that is less than the threshold. Say the index found is i. So for the leftmost (i-1) columns there will be no compression. We simply copy the (i-1) columns to the compressed table.

From the ith column we go for the compression. We compute the maximum differences between the columns i & i+1, i & i+2 and so on till the max diff obtained in some pair is greater than the threshold. That means excluding the last column all the previous columns can be compressed and merged to keep a single column. So the columns compressed form a zone of compression. Now if say the jth column to the (j+k)th column has formed a zone, then for optimality we keep the (j+k/2)th column in the compressed table.

- The zone-size never decreases.
  We have used an array-map array to obtain the mapping between the original table & the compressed table. Here, map[i]=j implies that the ith column is represented by the jth column in the compressed table.
  For example at 40% compression of F_{30} we get the map array as

  1 2 3 4 5 6 7 8 9 9 10 10 11 11 12 12 13 13 14 14 15 15 15 16 16 16 17 17 17 18

  Here no. of repetitions denotes the no. of columns in a zone. Thus we see that the no. of columns compressed in a zone is non decreasing. So when the 1st zone in the table has been formed with zone-size say z for the next zone (starting from jth column say) we start by checking the diff between the jth and the (j+z-1)th column.

- Calculation of the max diff is also done in an interesting way.

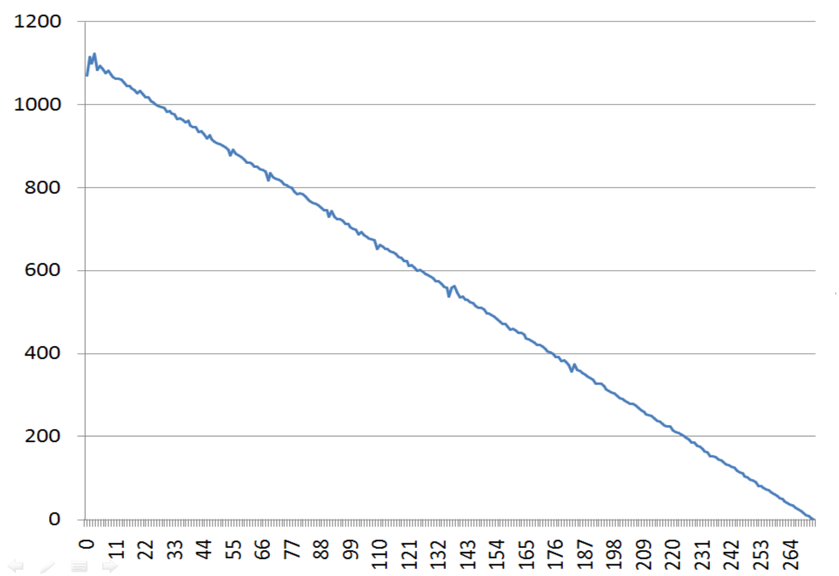

Fig. 10 Differences between 273rd and 274th column in F_{1000}

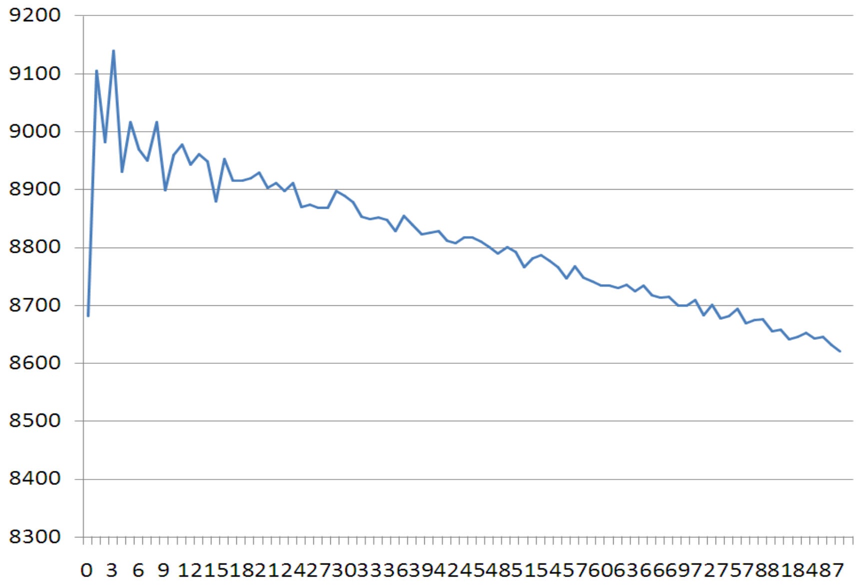

Fig. 11 Differences between 2162nd and 2263rd column in F_{8000}

In the above figures X axis:-Level from bottom, Y axis:-Differences

As we see in the above figures the maximum difference between any two columns, if falls below the difference found in the bottom row, it never exceeds the maximum obtained so far. So if anywhere it falls below that we take the max obtained so far as the max diff between the two columns.

- In a zone the level at which the maximum difference occurs between the 1st two columns (obtained from zonelevel[]) is the maximum level of the zone.

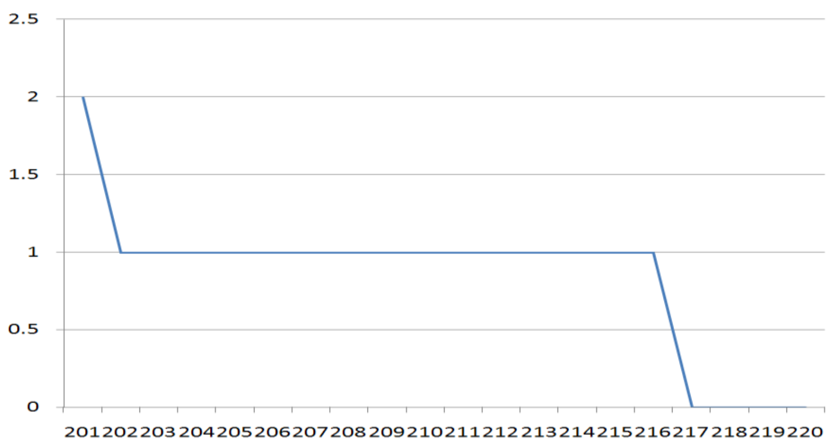

Fig. 12 Y axis:- Level from bottom, X axis:- Columns compared with col 200 in F_{500}.

From the above fig. we see that row at which the maximum differences are computed in a zone never goes below the row in which the 1st max diff for the zone is computed. So we can obtain the max diff between 2 columns in a zone just by checking up to the row from which the 1st max diff for the zone is obtained.

Following this technique we can compress the table using a feedback and reaching close to the desired compression degree. When the user is satisfied or when no more close value is possible (bisection has checked all possible values in the range) we save the compressed table in a file.

4.3 The Algorithm

1. The Farey Table is read from a file.
2. The degree of compression is taken as an input and assigned into prcnt.
3. //Based on prcnt the search for threshold r is bounded to a range (r1-r2) in the diff //array.

```
If(prcnt>80)
{
    r1←1;
    r2←0.056*ord;
}
If((prcnt<=80.0)&&(prcnt>60))
{
    r1←(.056*ord);
    r2←(0.155*ord);
}
If((prcnt<=60.0)&&(prcnt>40.0))
{
    r1←0.155*ord;
    r2←0.315*ord;
}
If((prcnt<=40.0)&&(prcnt>20.0))
{
    r1←0.315*ord;
    r2←0.606*ord;
}
Else If (prcnt<=20)
{
    r1←0.606*ord;
    r2←ord-1;
}
```

```
4.    //diff array and zonelevel array(contains level where the max diff is obtained for jth
5.    //and (j+1)th column) creation
       j←1;
       while(j<ord)
       {
            max←0;
            i←0;
            lowmax←tab[j][j]-tab[j][j+1]; //lowmax holds the difference between the
             //adjacent fractions at the lowest level.
            while((tab[j-i][j]-tab[j-i][j+1])>=lowmax)//differences at other levels are
             //considered until it exceeds the value of lowmax.
            {
                    if((tab[j-i][j]-tab[j-i][j+1])>max)
                    {
                            max←tab[j-i][j]-tab[j-i][j+1];
                            pos←i;
                    }
                    i←i+1;
             }
             diff[j]←max; //diff[j] holds the maximum difference between j^th & (j+1)^th
             //columns.
             zonelevel[j]←pos; //zonelevel[j] holds the position where the maximum
             //difference is occurring between j^th & (j+1)^th columns.
              j←j+1;
       } //completion of diff array & zonelevel array creation.
6.     r←((r1+r2)/2);      //r is the diff array index from where the bisection will start
7.     Repeat              //loop for approaching the desired degree of compression
     {
              temp←1;
              lmt←diff[r];                    //lmt i.e. the threshold is set to diff[r]
              j←1;
               while(diff[j]>lmt)         //we skip columns in which diif with adjacent is > lmt
                  j←j+1;
              k←j;
              while(j ≤ ord)             //compression till all columns are checked
               //ZONE CREATION
               {                          //j is the left col & temp is the zone size of compression.
                    flag←0;
                    If((j+temp)>ord)     //if zone goes out of the table, zonesize is reduced
                    {
                            while((j+temp)>ord)
```

```
                        temp←temp-1;
                }
                //since zonesize never reduces
                l←j+temp;                    //l is the left col
                max←diff[j];
                while(max<=lmt)      //till maxdiff<=lmt we increase the zone size
                {
                    l←l+1;
                    If(l>ord) //when l exceeds ord we break & check the
                  //percentage
                    {
                        l←l-1;
                        Break;
                    }
                    max←0;
                    for(i←j; j-i<=zonelevel[j]; i←i-1)//checked up to zonelevel only
                    {
                        If((tab[i][j]-tab[i][l])>max)
                        {
                            max←tab[i][j]-tab[i][l];
                            If(max>lmt)
                            {
                                flag←1; //set flag if loop break due to
                                //max>lmt
                                Break;
                            }
                        }
                    }
                    If(flag=1)                          //if flag set decrease l
                        l←l-1;
                }
            temp←l-j;                                   //zonesize calculation
            j←l+1;                                      //j set to next zone
            k←k+1;
        }                   //loop exit when compression degree is estimated
        k←k-1;
8.      prcnt1←((ord-k)/ord)*100;          //calculate percentage
        If((r2-r1)==2||((r2-r1)==1)||((r2-r1)==0)||((prcnt1-prcnt)==0))
            Break;
        If(((prcnt-prcnt1)*(prcnt-prcnt1))<=25.0)  //if compression is within 5 of the
        //desired range
```

```
{
    If(user is satisfied)
        Break;
    Else If((r2-r1)==2||((r2-r1)==1)||((r2-r1)==0)||((prcnt1-prcnt)==0))
    //If no further compression is possible.
    {
        Display(No further compression is possible closer to desired degree);
        break;
    }
    Else                        //negative feedback with user choice
    {
        If(prcnt1<=prcnt)   //If obtained % of compression <=desired
        {
            r2←r;
            r←(r2+r1)/2;
        }
        Else                //If obtained % of compression >desired
        {
            r1←r;
            r←(r2+r1)/2;
        }
    }
}
```

9. Else //if compression > compression desired +- 5

```
{                               //next bisection step
    If(prcnt1<=prcnt)
    {
        r2←r;
        r←(r2+r1)/2;
    }
    Else
    {
        r1←r;
        r←(r2+r1)/2;
    }
}
} //End of loop for approaching the desired degree of compression.
//compressed table generation
//the above infinite loop is repeated once with the obtained threshold value
//to generate the compressed table.
End.
```

30% Compression of Farey Table of order 10

Farey Table of order 10

Numerator ↓ \ Denominator →	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1
1	33	17	12	9	7	6	5	4	3	2
2	-1	33	22	17	14	12	10	9	8	7
3	-1	-1	33	25	20	17	15	13	12	11
4	-1	-1	-1	33	27	22	19	17	16	14
5	-1	-1	-1	-1	33	28	24	21	18	17
6	-1	-1	-1	-1	-1	33	29	25	22	20
7	-1	-1	-1	-1	-1	-1	33	30	26	23
8	-1	-1	-1	-1	-1	-1	-1	33	31	27
9	-1	-1	-1	-1	-1	-1	-1	-1	33	32
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	33

map array : { 1 2 3 4 4 5 5 6 6 7 }

The 30% Compressed Table

N ↓ \ D →	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1
1	33	17	12	9	6	4	2
2	-1	33	22	17	12	9	7
3	-1	-1	33	25	17	13	11
4	-1	-1	-1	33	22	17	14
5	-1	-1	-1	-1	28	21	17
6	-1	-1	-1	-1	33	25	20
7	-1	-1	-1	-1	-1	30	23
8	-1	-1	-1	-1	-1	33	27
9	-1	-1	-1	-1	-1	-1	32
10	-1	-1	-1	-1	-1	-1	33

N.B: Adjacent columns with their maximum difference lesser than a threshold value (here i.e. 8) are merged into a single column in the compressed table.

Fig. 13: 30% Compression in F_{10}.

This is how this above algorithm works. We have shown it in the above figure to display the output of the algorithm for F_{10}.

Chapter 5

Performance Evaluation

5.1 Error due to Compression

It is obvious that due to mapping of one column into another some errors are bound to occur. We are mapping a fraction to another nearby fraction and hence introducing some error.

Error= |Fraction index in original table - Fraction index in compressed table |÷ Fraction index in original table

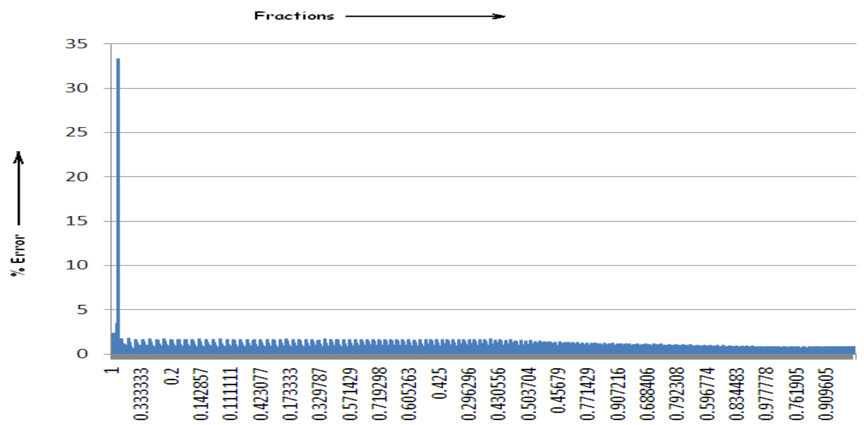

Fig 14: Percentage error in F_{200} at 40 % compression

Now, clearly the highest percentage error is 33.33% which is intolerable. This is giving an irreverent notion about the accuracy of the mapping accomplished during compression. But indeed the case is not so. Because the fraction at which this error is occurring is 1/199 with an index value of 3, is mapped to the fraction 1/198 with an index value of 4, producing the error as 33.33%. In fact 1/198 is the closest fraction to 1/199 differing by a negligible 0.00002. Except this, all the other fractions show error$\leq$2% to 3%.

5.2 Percentage Loss in Precision

Now we put forward another parameter called "Percentage loss in precision" which is evaluated as follows:

Loss in precision =((Farey index in the compressed table) – (the actual Farey index of the fraction))/max index.

- ⇨ Percentage loss in precision = loss in precision × 100.

We studied a number of tables of different orders and found some interesting measures which are depicted in the following figures.

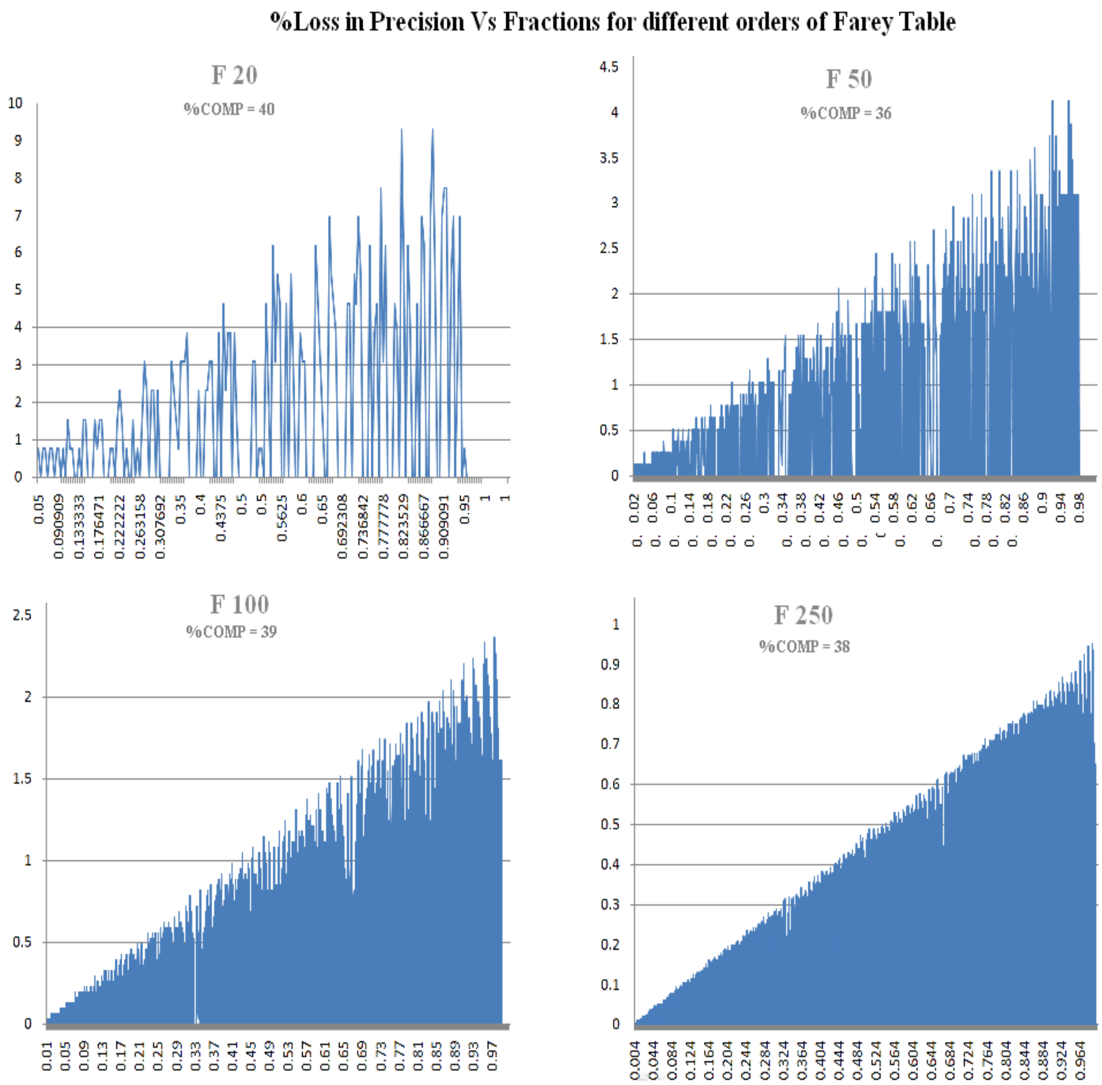

Fig. 15: X axis: The fractions Y axis: Percentage of loss in precision

In the above figures we can see that the maximum loss in precision is reducing as the order of the table increases e.g. it is 9.5% in F_{20} & 0.98% in F_{250}. For tables of even higher orders the maximum loss in precision becomes too negligible to lead to any considerable error.

Another point to be noted that the graphs are becoming denser as the order increases. This is due to the fact that the no. of fractions between 0 & 1 in the sequence increases with the order.

Chapter 6

Searching a Proper Fraction in The Farey Sequence

6.1 Introduction

Given a Farey Sequence of order n and an arbitrary proper fraction p/q, our objective is to find a fraction in the sequence which is closest to p/q.

6.2 Using Binary Search

The range in the sequence, to be searched, can be obtained in the following manner:-

left boundary(lb): floor(p×n/q)/n

right boundary(rb): ceiling(p×n/q)/n

It is obvious that lb ≤ p/q ≤ rb.

Since the fractions in a Farey sequence are sorted in ascending order we can now use binary search in this range to obtain the closest fraction.

Suppose, the maximum index in the Farey sequence of order n be maxind. We know that maxind=O(n^2). Again the numerators in lb & rb differ by 1, so (rb-lb)×maxind will approximately give the length of the range(l_r). If lb is m/n, rb will be (m+1)/n, then l_r = maxind×(m+1-m)/n = maxind/n = O(n^2/n) = O(n).

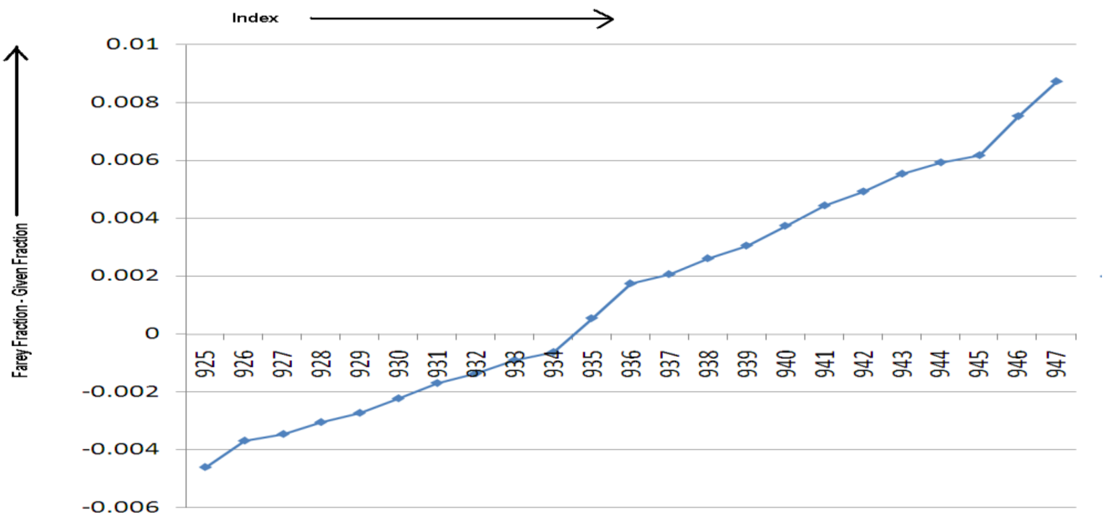

Fig16: Farey fraction – given fraction(78/145) v/s index plot in F_{75} .

In this range we calculate the differences between each Farey fraction and the given fraction. These differences are strictly increasing along the range(left to right). They are negative initially and become positive towards right with a transition somewhere in between. Clearly the closest fraction is at the index where this transition is taking place. So we search this index in O(log n) time using binary search.

6.3 Improving the algorithm

There are instances where the transition index is located near the boundaries. In that case the previous algorithm will take considerable time to find the result. Let us take an example:

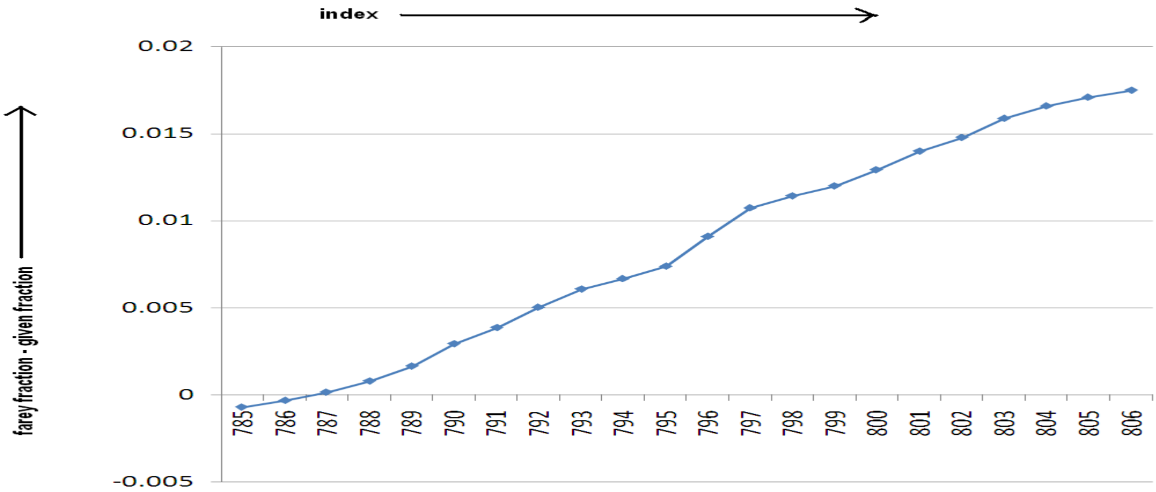

Fig 17: Farey fraction – given fraction(335/448) v/s index plot in F_{55} .

From the curves we see that the nature is nearly linear. So, the Regula Falsi method can be a good alternative. Regula Falsi approximates the curve by joining the end points with a straight line and then considers the intersecting point with the x-axis. The iteration continues depending on the value of the function at the intersecting point.

Due to the linearity of the curve, this intersecting point in all cases is found to be very close to the root of the function. As a result, Regula Falsi finds the root faster.

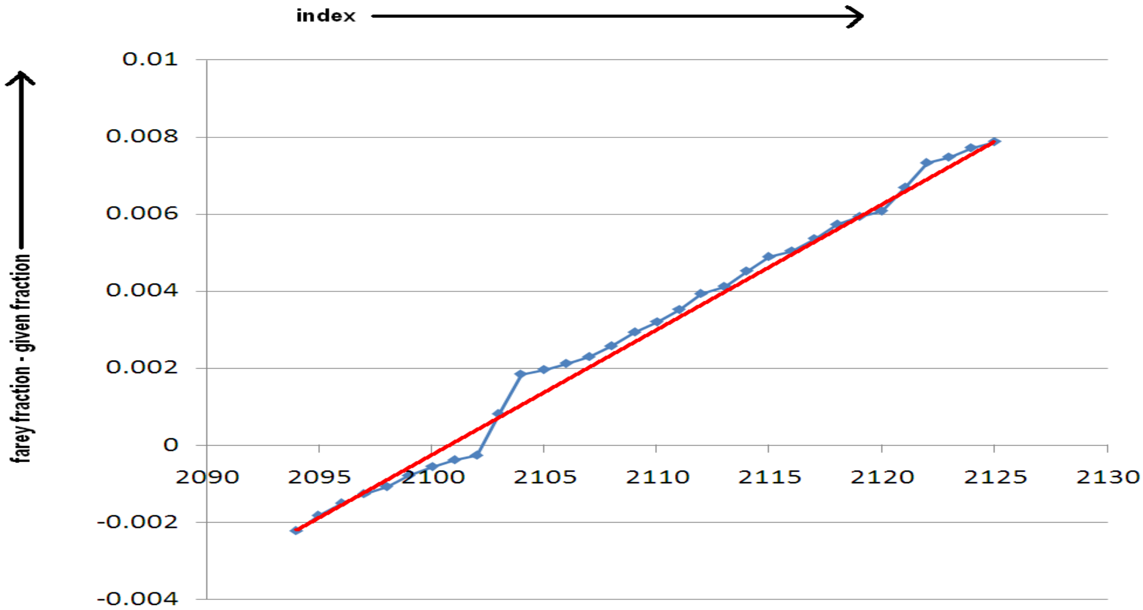

Fig 18: Farey fraction – given fraction (1957/2788) v/s index plot in F_{99} .

6.4 Algorithm

FindRoot (Seq, Table, lb, rb, p/q)

1. Suppose the range starts from Farey index x_1 and ends at x_2(obtained from the Farey table). Let the fractions at x_1 and x_2 be a/b & c/d resp.

2. The differences are calculated as follows:-

   Y_1 = a/b-p/q = (aq-bp)/bq.

   Y_2 = c/d-p/q = (cq-dp)/dq.

   Thus we obtain 2 points on the (Farey fraction – given fraction) v/s Index graph (x_1,y_1), (x_2,y_2).

3. We are required to find the point on the x axis where the straight line joining these two points, intersects the x axis. From coordinate geometry basics we have the equation of the straight line as:-

   $(y-y_1)/(x-x_1) = (y_1-y_2)/(x_1-x_2)$.

4. Putting y=0, we get
   $x = (x_2y_1-x_1y_2)/(y_1-y_2)$.

5. In the next step we calculate the differences in numerator/denominator form at x & (x+1) of Fn.

   If we find one of them +ve & the other –ve we have got the solution and we compare the 2 fractions at x & (x+1) to obtain the closest.
   Else if both are +ve we take the lower index i.e. x and the left most index point to continue Regula Falsi.
   Else (both are –ve) we take the higher index i.e. (x+1) & the rightmost index to continue.

Output:
Enter the order of the Farey table you want: 55
Enter the fraction to be searched(x/y): 341/556
Closest fraction is 27/44 at 577^{th}

6.5 Avoiding Floating Point operations

Now in order to avoid floating point operations we are keeping all the fractions in numerator/denominator form. Hence, the above mentioned equations are computed as:

5.4.2 y_1 = a/b-p/q = (aq-bp)/bq = y_{1n}/y_{1d}.

y_2 = c/d-p/q = (cq-dp)/dq = y_{2n}/y_{2d}.

5.4.3 $(y-y_1)/(x-x_1) = (y_1-y_2)/(x_1-x_2)$.

5.4.4 Putting y=0, we get

$x = (x_2y_1-x_1y_2)/(y_1-y_2) = (x_2*y_{1n}/y_{1d} - x_1*y_{2n}/y_{2d})/(y_{1n}/y_{1d}-y_{2n}/y_{2d}) = (x_2y_{1n}y_{2d}-x_1y_{2n}y_{1d})/(y_{1n}y_{2d}-y_{2n}y_{1d})$.

6.6 Regula Falsi Algorithm v/s Binary Search

We have plotted the graph, below to make a comparative study of the above mentioned algorithms. A set of 10 randomly generated proper fractions is taken and the no. of iterations required by these 2 methods for each of these fractions is studied. We have plotted the average no. of iterations required by the entire set of fractions for different orders of Farey sequences.

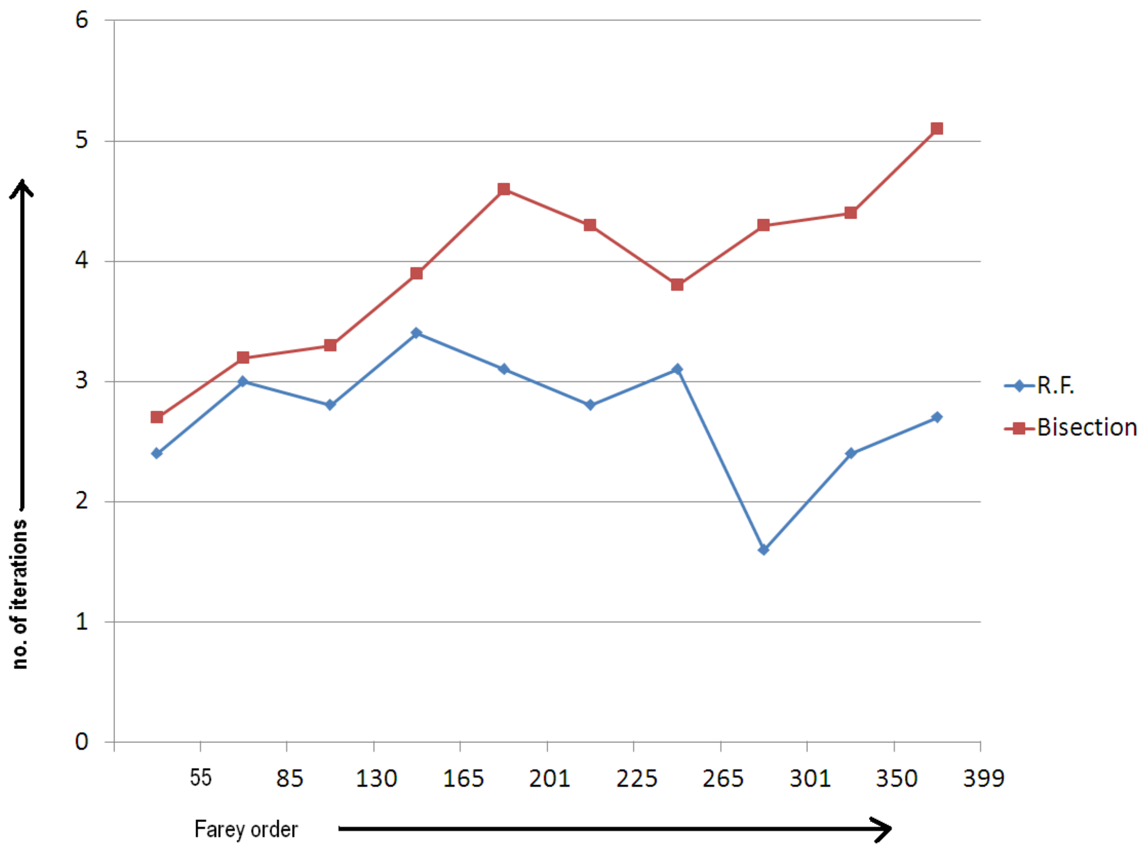

Fig 19: no. of iterations v/s order of Farey sequence.

Thus we see from the graph that the Regula Falsi method is producing quite better results when compared to the previous method.

Conclusion

We have seen that the Farey indices are giving a good measure of relative values of two or more fractions. Searching a fraction in the Farey sequence has been improved by using the Farey table. The space required to store a table is reduced by compressing the table. We have used many amazing properties of the Farey fractions in generating the sequences and compressing the table. We have also developed algorithms to find a fraction closest to any arbitrary fraction in a given Farey sequence. All the algorithms have avoided expensive floating point operations and have saved time to perform their respective functions. The compressed Farey table can have wide applications in Digital Image Processing and it puts forward new dimensions to be explored in future.

Bibliography

1. Concrete Mathematics by Graham-Knuth-Patashnik.
2. Introduction to Algorithms by Thomas H. Cormen-Charles E. Leiserson-Ronald L. Rivest-Clifford Stein.
3. Farey Sequences and Discrete Radon Transform Projection Angles by Imants Svalbe , Andrew Kingston.
4. A Property of Farey Tree by Ljubiˇsa Koci´c and Liljana Stefanovska.
5. Computing Order Statistics in the Farey Sequence by Corina E. Pˇatra¸scu and Mihai Pˇatra¸scu.
6. http://en.wikipedia.org/wiki/Farey_sequence
7. http://mathworld.wolfram.com/FareySequence.html
8. Programming with C by Gottfried.
9. Fundamentals of Computer Algorithms by Ellis Horowitz-Sartaj Sahni-Sanguthevar Rajasekaran.
10. Data Structures in C by Lipschutz.