A Computational Library Using *P*-adic Arithmetic for Exact Computation with Rational Numbers in Quantum Computing

Final Report (Grant # FA9550-05-1-0363*)*

Chao Lu

Computer & Information Sciences

Towson University

Nov. 30, 2005

REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-06-0428

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
7/20/2006	Final Report	7 Jan 2005- 30 June 2006

4. TITLE AND SUBTITLE
A p-adic Computational Library for fast Computation in Quantum Computing

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-05-1-0363

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Chao Lu

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Towson University
8000 York Rd.
Towson, MD 21252

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
4015 Wilson Blvd
Mail Room 713
Arlington, VA 22203
Dr Fogen Kuo

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Many classes of physical problem can be models through the use of sets of linear equations. The solution of the sets of equations is equivalent to calculation of a matrix inverse or generalized inverse, or to the reduction of the matrix to some type of canonical form, including determination of characteristic equation. Conventional machine computation relies on p-ary (for a radix number p such as 2 or 10), or floating-point computation, poor conditioning in connection with round-off error can result in unreliable answers. For scientific computations related to quantum physics, a possible approach is to use techniques of exact linear computation

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Chao Lu
U	U	U	UU		19b. TELEPHONE NUMBER *(Include area code)* 410404-3701

Table of Contents

1. **Summary and Related Computational Issues**
   1.1 Project Summary
   1.2 Computational Issues
2. **Program Description**
   2.1 Program Interface
   2.2 Program Modules
      2.2.1 Fractional Number to *P*-adic Sequence Conversion
      2.2.2 *P*-adic Sequence to Fractional Number Conversion
      2.2.3 Single Number Addition
      2.2.4 Single Number Subtraction
      2.2.5 Single Number Multiplication
      2.2.6 Single Number Division
      2.2.7 Matrix Addition
      2.2.8 Matrix Subtraction
      2.2.9 Matrix Multiplication
      2.2.10 Square Matrix Inverse
      2.2.11 Matrix Addition (I/O with FILE)
      2.2.12 Matrix Subtraction (I/O with FILE)
      2.2.13 Matrix Multiplication (I/O with FILE)
      2.2.14 Square Matrix Inverse (I/O with FILE)
      2.2.15 Detail Description of some Functions
   2.3 Source Code Map
3. **Results Analyses and Comparisons**
   3.1 Some Test Examples and Results
   3.2 Comparison of our Results with Matlab Symbolic Toolbox
4. **Future Work**

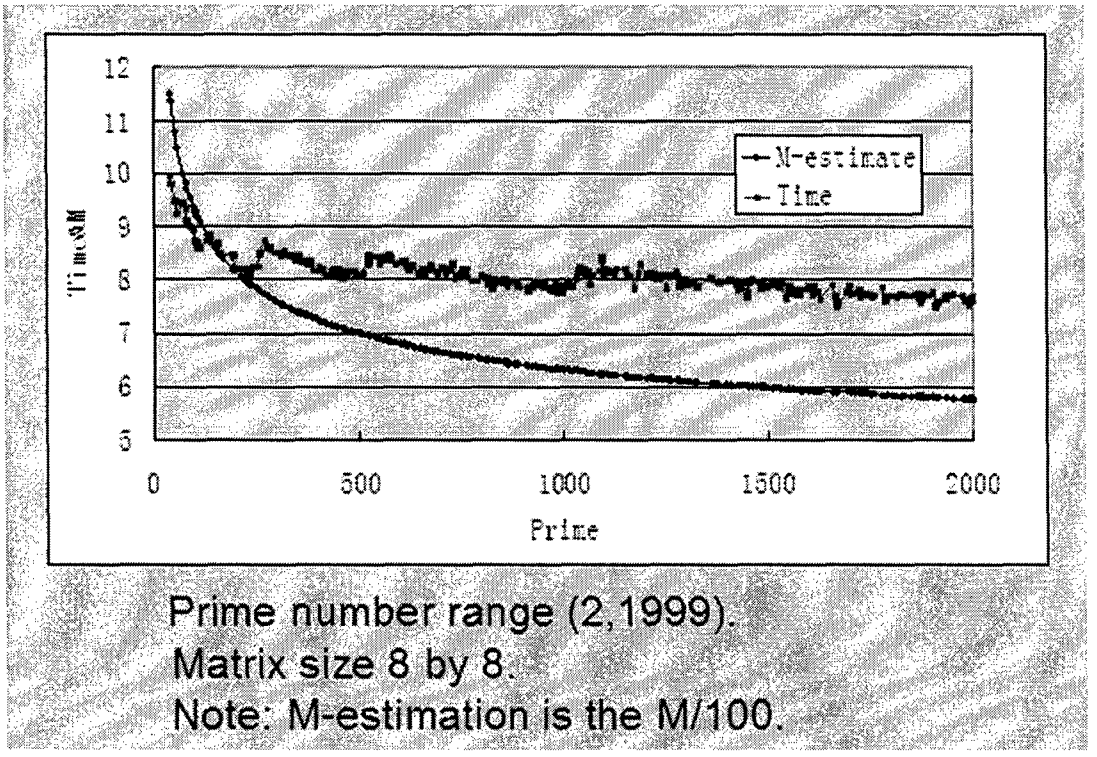
1. Summary and Related Computational Issues

1.1 Project Summary

In many scientific fields, physical problems can be modeled by a set of linear equations. Solving the linear equations is equivalent to finding the matrix inverse or generalized inverse, to reducing the matrix to a specified canonical form, or to determining the characteristic equation. If using conventional p-ary or floating-point arithmetic for such computations, the cumulative round off errors may make the results unreliable. For scientific computations related to quantum physics, one is interested in exact linear computation. The computational complexity of the rational arithmetic approach is very expensive and laborious, and it is a very challenging computer science problem to deal with dynamic memory allocation during the computing process. Many researchers have invested a lot of time to develop algorithms. Computational packages have been made available for the scientific research community. NTL written by Victor Shoup [4] is the one we have picked as a computational vehicle to develop a rational matrix computation library. The alternative approach represents all integers and rational numbers in terms of a set of residues with respect to a prime number and its powers, called a p-adic number system. The p-adic arithmetic has many attractive features [1].

We have developed algorithms of matrix operations with rational numbers by representing numerator and denominator with arbitrary length integers; and designed algorithms using modulo arithmetic of the p-adic number system, where all numbers are represented by their p-adic sequence, all arithmetic operations are carried out in the p-adic domain, then the results are converted back to rational numbers. We have built an *Exact* Scientific Computational Library (ESCL) using both approaches. The first approach serves as the basis for comparison and the main effort has been on the second approach. The algorithms are tested and compared with the MATLAB Symbolic Toolbox for small integers. The ESCL is implemented in C++.

We investigated various ways to improve computation speed. First, selecting an optimal prime number for p-adic expansion. We did experiments on random rational matrices with different prime numbers (P) and record the execution time and tried to get the right prime number by analyzing the experimental results. Second, choose the right length M of the p-adic expansion for the rational numbers in the matrix. The following chart shows that when P gets bigger, M gets smaller (blue curve), while the runtime stays relatively unchanged (red curve).



Both the experiments and the theory showed that the runtime is related to the length of p-adic sequence (*M*). A large prime *P* will need a smaller *M*, which in turn will cut the computational complexity. The *M* and *P* are related as:

$$m = 2\lfloor \log(\delta\lambda^{-1}) / \log p \rfloor, \qquad (1)$$

where the $\delta = \prod\prod \lambda_i$ and λ_i is the Euclidean length of the *i*th column of the matrix.

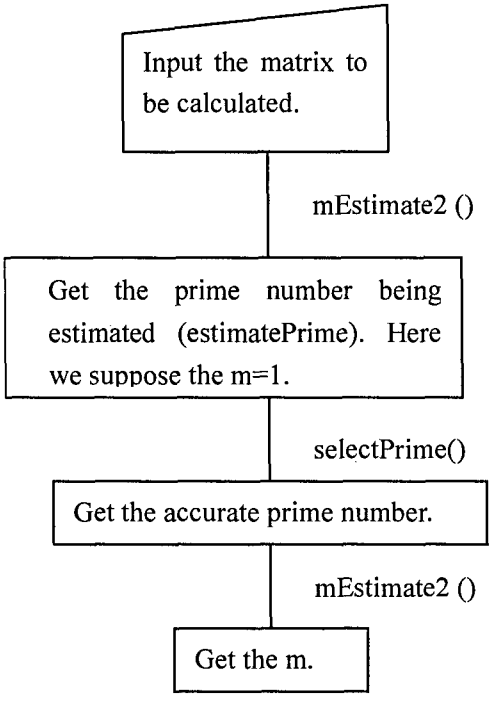
A few examples are given here for comparison of runtime.
Example 1:
1/19999999999999999999 + 2/ 19999999999999999999

Prime	M	Time
2	169	20 ms
8971	46	<1 ms

Example 2:
1/9876543211234567 * 9876543211234567/2

Prime	M	Time
3	100	316 ms
8707	60	120 ms

Here is the flow chart of getting the right "*M*" from an estimated "*P*", then use equation (1) to find *P* with the *M*.



1.2 Computational Issues

By November 2005, we have completed the following functions:

1) Fractional number to *p*-adic sequence, and its inverse.
2) Single number operations (addition, subtraction, multiplication and division) in the *p*-adic field.
3) Matrix operations(addition, subtraction, multiplication and inverse)in the *p*-adic field.
4) Matrix operations (addition, subtraction, multiplication and inverse) in the *p*-adic field with I/O file support.

For all operations in *p*-adic field, we need to deal with the offset first. Just like we add two numbers with different exponents, we need to change them to the same format first. (eg.$5*10^5 + 2*10^3 = 500000+2000=502000$) We make all the vectors with the offset=0,which is easier to calculate. If the offset is not equal to zero, we need to shift the *p*-adic sequence. Here is an example.

$[1/3]_{p=5}$=[1,2,3,1,3], the offset (first digit) =1, after the shift, we get [0,0,2,3,1]

Arithmetic Operations:

1) Addition/subtraction

The algorithm for addition aligns the *p*-adic point of the mantissa, retaining the lower exponent and finds the sum digit s_i and carry digit c_{i+1} from the knowledge a_i, b_i and

$c_{i..}$. Thus $s_i = (a_i + b_i + c_i) \bmod p$

for i =0 , 1, 2,…, (r-1)

c_{i+1}=1, if $a_i + b_i + c_i \geq p$

=0, otherwise

$c_0 = 0$, and ignore c_r.

Subtraction is realized as the complement of addition.

Let us see this example.

Prime=5;

1/3 + 2/9= [0 2 3 1 3 1 3 1 3 1 3 1 3 1 3]

+ [0 3 0 1 2 4 3 2 0 1 2 4 3 2 0]

=[0 0 4 2 0 1 2 4 3 2 0 1 2 4 3]

=5/9

Here is another example whose offset is not zero.

Prime=5;

25/3 + 1/7 = [2 2 3 1 3 1 3 1 3 1 3 1 3 1 3 1]

+ [0 3 3 0 2 1 4 2 3 0 2 1 4 2 3 0]

= [0 0 0 2 3 1 3 1 3 1 3 1 3 1 3 1]

+ [0 3 3 0 2 1 4 2 3 0 2 1 4 2 3 0]

= [0 3 3 2 0 3 2 4 1 2 0 3 2 4 1 2]

=178/21

2) Multiplication

This is similar to p-ary multiplication, except that the product is developed to only lower r digits (modulo p^r). The algorithm consists of first forming the cross-products of the mantissa,

$P_{ij} = b_i a_j \quad 0 \leq i \leq (r-1)$

where $j = 0,1,...r-1$, and the partial product P_i and product P,

$$P_i = \sum_{j=0}^{r-1} P_{ij} \Delta(j)$$

$$P = \sum_{i=0}^{r-1} P_i \Delta(i)$$

where $\Delta(X)$ denotes a right shift of X digits. The offset of the result is offset1+offset2.

Let us see this example. 2/3 * 1/6, prime=5.

04131313131313 · · ·

01404040404040 · · ·

00000000000000

$$
\begin{array}{r}
4131313131313\cdots \\
123131313131\cdots \\
00000000000\cdots \\
1231313131\cdots \\
000000000\cdots \\
12313131\cdots \\
0000000\cdots \\
123131\cdots \\
00000\cdots \\
1231\cdots \\
000\cdots \\
12\cdots \\
+0\cdots \\
\hline
04201243201243\cdots
\end{array}
$$

3) *Multiplicative Inverse* and *Division*

Given that $0 \le m_\beta \le P^r - 1$ and $GCD\,(p, m_\beta) = 1, m_\beta^{-1} \bmod p^r$ can be obtained very simply by a recursive solution of the congruence with respect to p.

Let $m_\beta = b_0, b_1, \ldots b_{r-1} (b_0 \neq 0)$ and $m_\beta^{-1} = q_0, q_1, \ldots, q_{r-1}$. The q_i can be obtained by solving for q_i in

$$m_\beta \sum_{i=0}^{r-1} q_i p^i \equiv 1 \bmod p^r.$$

Thus, starting with $q_0 \equiv b_0^{-1} \bmod p, q_k (k \ge 1)$ is computed by solving for

$$(q_k p^k + \sum_{i=0}^{k-1} q_i p^i) b \equiv 1 \bmod p^{k+1}.$$

This leads to the following deterministic trial-error-free division algorithm, the quotient, digit by digit, proceeding from the lower index to the higher index position.

The following is the algorithm for finding $m_\alpha \cdot m_\beta^{-1}$.

Let

R_0 zero-th partial remainder or initial numerator (=1 for finding b^{-1});
R_i ith partial remainder;
R_{ii} ith positional digit of R_i.

Then

$$q_i = R_{ii} b_0^{-1} \bmod p$$

for i=0, 1, 2, …, (r-1) and $R_{i+1} = R_i - q_i b\Delta(i)$, where $\Delta(i)$ is the right shift by i digits.

Note that this algorithm can be applied for any numerator by setting $R_0 = 1$ and all other digits of R_j to zero, one can obtain the multiplicative inverse of m_β. The offset of the result is offset1-offset2.

As an example, we divide 2/3 by 1/12. In this example, we do not include the offset.

We have

2/3= .4131313

1/12= .3424242

The first digit of the divisor is $b0 = 3$ and its multiplicative inverse modulo 5 is

$b_0^{-1} \pmod p = 3^{-1} \pmod 5 = 2$

The first digit of the partial remainder (which, in the first step, is the dividend) is $d0 = 4$, which gives

$a_0 = b_0^{-1} d_0 \pmod p = 2 \cdot 4 \pmod 5 = 2$.

Thus, we obtain the first digit of the quotient. We then update the partial remainder by subtracting 3 times the divisor from it.

```
                .3
                ----------------------
.3424242 · · · .4131313 · · ·
.4333333 · · · .1111111 · · ·
                  ----------------------
                .0342424 · · ·
```

To obtain the second digit, we multiply $b^{-1}{}_0 \pmod p$ by the first digit of the partial remainder and reduce the result modulo p.

$a1 = 2 \cdot 3 \pmod 5 = 1$.

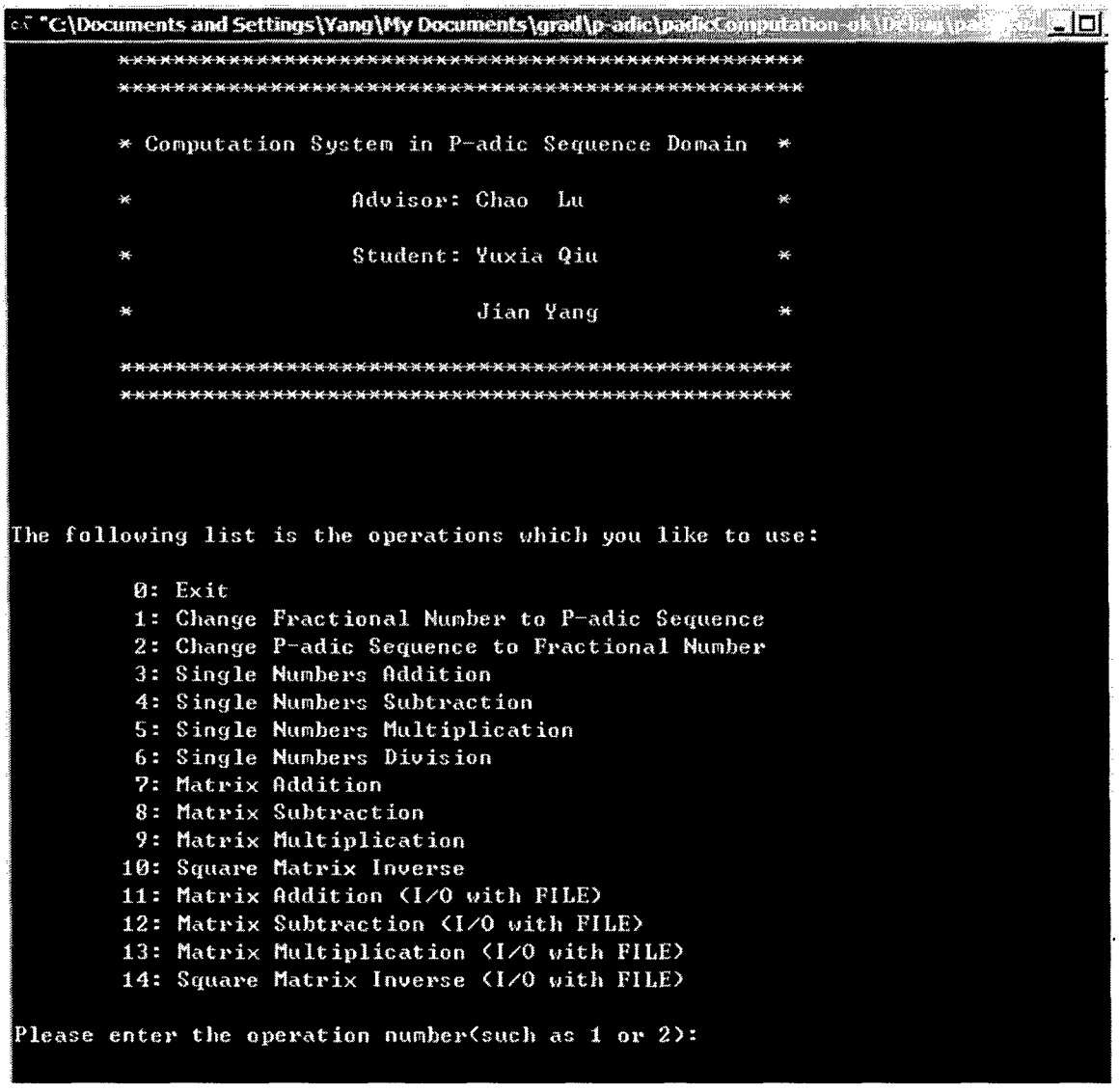
Thus, the second step of the division procedure gives us

```
                .31
                ----------------------
.3424242 · · · .0342424 · · ·
.0342424 · · · .0202020 · · ·
                  ----------------------
                .0000000 · · ·
```

This procedure produces the partial remainder, which is zero, hence we terminate the expansion. In general, this will not happen and we will have to continue until the period is exhibited.

2. Program Description

2.1 Program Interface



The user inputs the operation number first and then follows the instructions to do the operation.

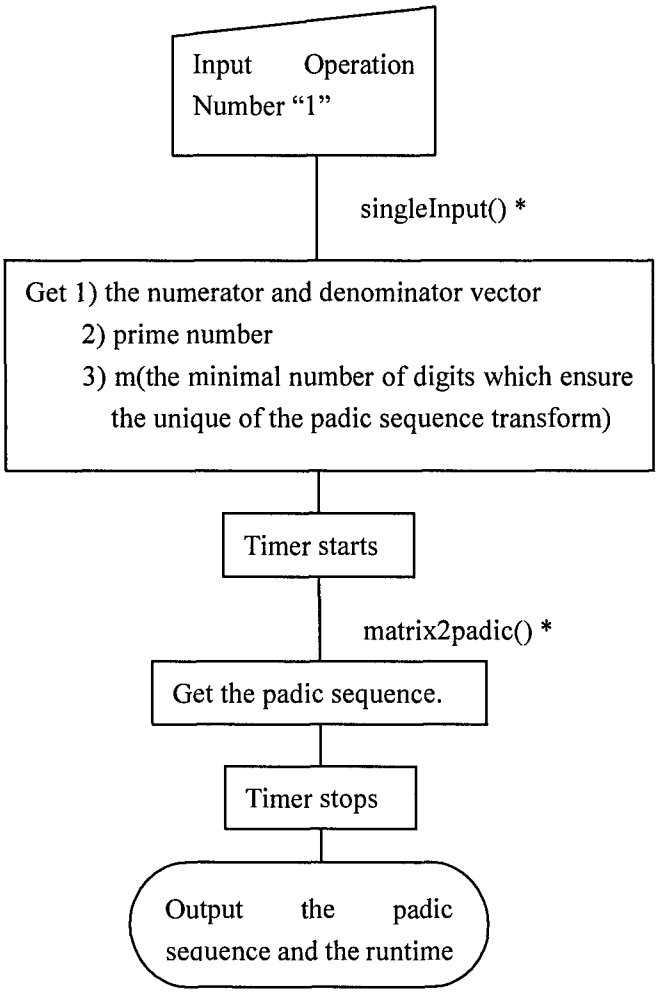
2.2 Program Modules

In this section, we will give the detail description of each operation module. Note that the function with the star mark "*" means that you can find the detail description in that part of the program.

2.2.1 Fractional Number to P-adic Sequence Conversion

Input: fractional number;

Output: The smallest prime number selected by the program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence and the runtime.



Here is an example of this operation.

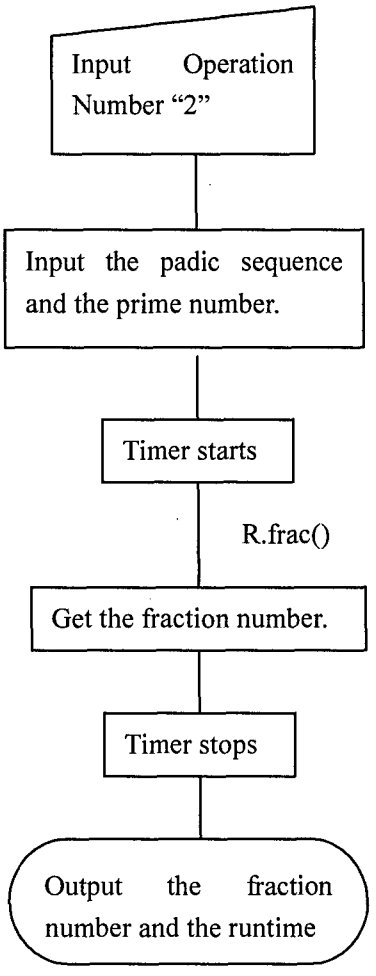
```
Please enter the operation number(such as 1 or 2): 1
Please enter the fractional number: 1/333333
The prime number is: 5
The minimal digits number of padic sequence is: 27
The P-adic sequence for this fractional number is:
[[0 2 4 4 4 4 4 2 1 2 3 4 4 1 2 3 1 0 1 2 3 2 3 1 3 1 0]
]

The computation time is 3 Second(s).
```

2.2.2 P-adic Sequence to Fractional Number Conversion

Input: *P*-adic sequence, prime number;

Output: The fractional number and the runtime.



Here is an example of this operation.

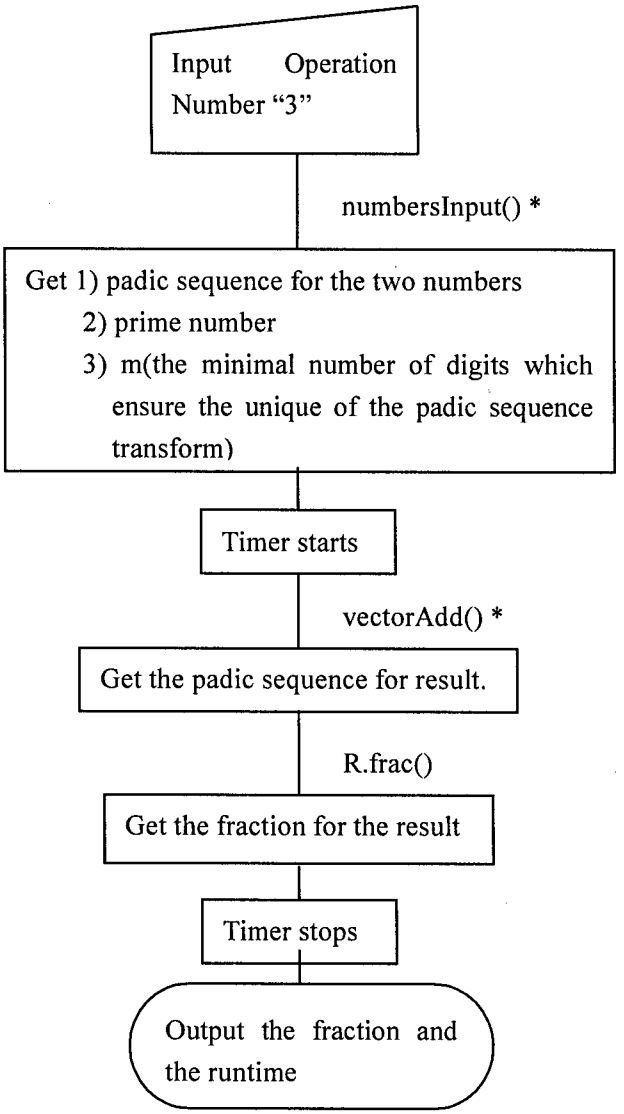
```
Please enter the operation number(such as 1 or 2): 2
Please enter the P-adic sequence(such as [0 1 2 3 4]):
[0 2 3 1 5]
Please enter the prime number for this p-adic sequence: 7
The result fractional number is: 9/43

The computation time is 0 Second(s).
```

2.2.3 *Single Number Addition*

Input: Two fractional numbers that need to be added;

Output: The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence of the addition result, the result converted back in fraction number and the runtime.




```
Please enter the operation number(such as 1 or 2): 3
Please enter the first fractional number: 10000/3
Please enter the second fractional number: -20
The proper prime number for them is: 5
The minimal number digit of padic sequence is: 24
After addition the P-adic sequence is:
[0 0 1 4 4 1 4 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 1]
The result fractional number is: 9940/3

The computation time is 0 Second(s).
```

2.2.4 Single Number Subtraction

Input: Minuend and subtrahend.

Output: The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence for the subtraction result, the result converted back in fraction number

and the runtime.

The only difference with Single Number Addition (2.2.3) is that we use vectorSub()* instead of vectorAdd() to do the subtraction.

2.2.5 *Single Number Multiplication*

Input: Two numbers.

Output: The smallest prime number selected by program, the minimal number of digits for p-adic sequence, the p-adic sequence of the multiplication result, the result converted back in fraction number and the runtime.

The only difference with Single Number Addition (2.2.3) is we use vectorMul()* instead of vectorAdd() to do the multiplication.

2.2.6 Single Number Division

Input: Two numbers.

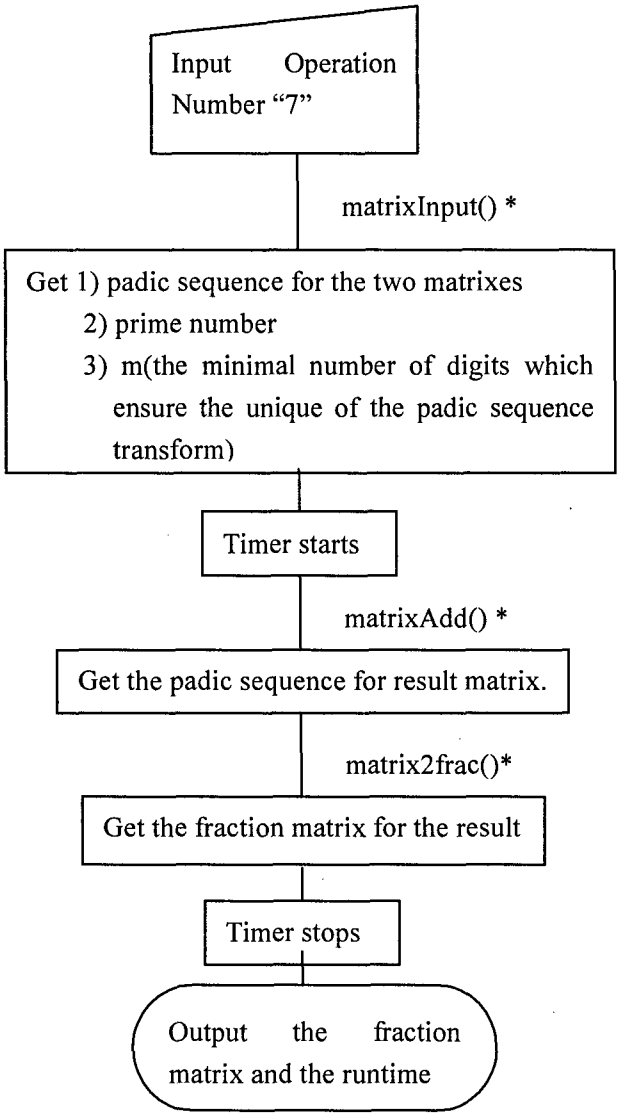
Output: The smallest prime number selected by program, the minimal number of digits for p-adic sequence, the p-adic sequence of the division result, the result converted back in fraction number and the runtime.

The only difference with Single Number Addition (2.2.3) is we use padicDivision()* instead of vectorAdd() to do the division.

2.2.7 Matrix Addition

Input: Two matrixes.

Output: The smallest prime number selected by program, the minimal number of digits for p-adic sequence, the p-adic sequence of the matrix addition result, the result converted back in fractional matrix and the runtime.


Input Operation Number "7"
matrixInput() *
Get 1) padic sequence for the two matrixes
2) prime number
3) m(the minimal number of digits which ensure the unique of the padic sequence transform)
Timer starts
matrixAdd() *
Get the padic sequence for result matrix.
matrix2frac()*
Get the fraction matrix for the result
Timer stops
Output the fraction matrix and the runtime

Here is an example of this operation.

```
Please enter the operation number(such as 1 or 2): 7

Please input the NUMBER OF ROWS of the FIRST MATRIX: 2
Please input the NUMBER OF COLUMNS of the FIRST MATRIX: 2

Please follow the instructions to input the elements of the matrix.


Please input ROW 1 elements:
1,1/333


Please input ROW 2 elements:
123,222/1

Please input the NUMBER OF ROWS of the SECOND MATRIX: 2
Please input the NUMBER OF COLUMNS of the SECOND MATRIX: 2



Please input ROW 1 elements:
34,12/2


Please input ROW 2 elements:
13,33
The proper prime number for them is: 5
The minimal number digit of padic sequence is: 45

The Padic Matrix for the Result is :
[[0 0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 3 0 0 1 0 4 2 1 2 2 2 3 4 4 2 2 4 3 0 1 1 1 4 4 4 3 3 4 1 0 3 0 3 4 4 4 1 4 4 0 0 4 2 1]
[0 1 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
]

The Fractional Matrix is:
[35/1,1999/333
136/1,255/1]

The computation time is 0 Second(s).
```

2.2.8 Matrix Subtraction

Input: Minuend matrix and subtrahend matrix.

Output: The smallest prime number selected by program, the minimal number of digits for p-adic sequence, the p-adic sequence matrix for subtraction result, the result in fraction matrix and the runtime.

The only difference with Matrix Addition (2.2.7) is we use matrixSub()* instead of matrixAdd() to do the subtraction.

2.2.9 Matrix Multiplication

Input: Two numbers.

Output: The smallest prime number selected by program, the minimal number of digits for p-adic sequence, the p-adic sequence matrix of the multiplication result, the result converted back in fractional matrix and the runtime.

The only difference with Matrix Addition (2.2.7) is we use matrixM()* instead of matrixAdd() to do the multiplication.

2.2.10 Matrix Inverse

Input: One square matrix.

Output: The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence matrix of the inverse result, the result converted back in fraction matrix and the runtime.

The only difference with Matrix Addition (2.2.7) is we use Determinant(),Transpose (), padicDivision() and cofactor() instead of matrixAdd() to do the inverse calculation. The logic in this part is the same as the normal matrix inverse.

2.2.11 Matrix Addition (I/O with FILE)

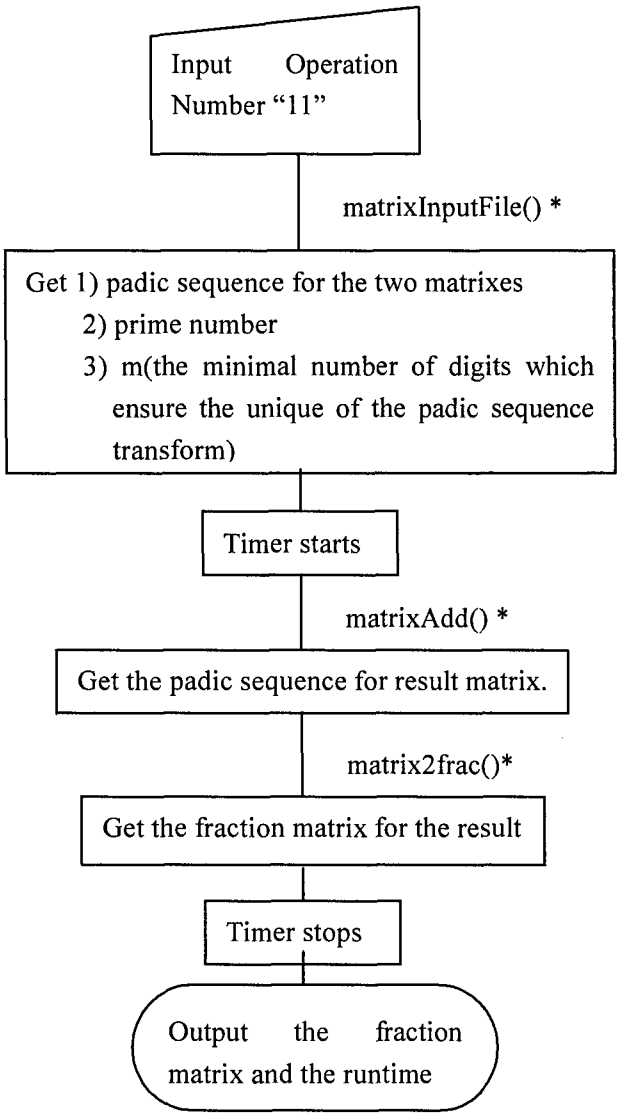
Input: No need to input the specific number. The input can be founded in file "Input.txt".

Output:

1) On the screen:

The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence matrix of the addition result, and the result converted back in fraction matrix with the runtime.

2) In the file: You can find the final result--fraction matrix in the file



"Out.txt".

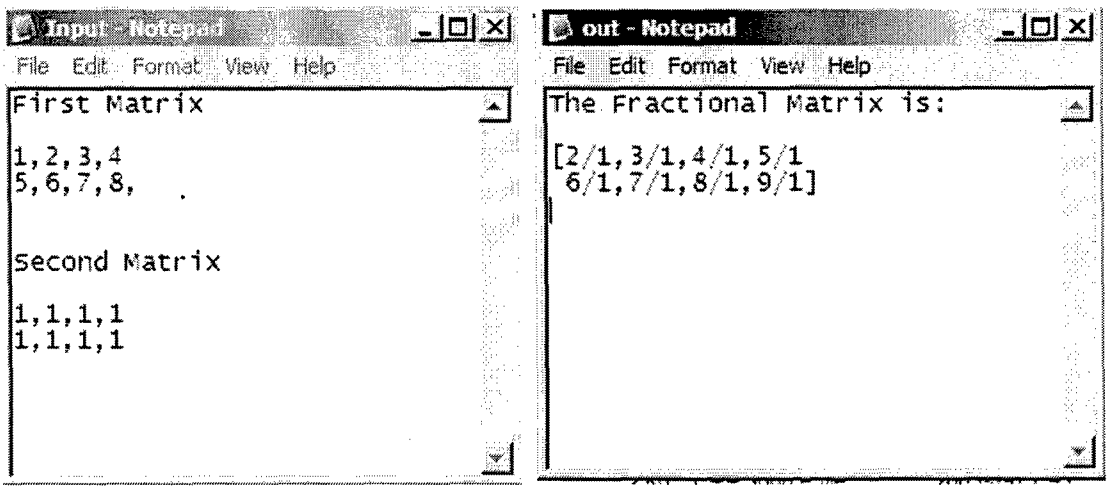
Here is the example of this operation.

```
Please enter the operation number(such as 1 or 2): 11
The proper prime number for them is: 3
The minimal number digit of padic sequence is: 27

The Padic Matrix for the Result is :
[[0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
]

The Fractional Matrix is:
[2/1,3/1,4/1,5/1
6/1,7/1,8/1,9/1]

The computation time is 0 Second(s).
```



2.2.12 Matrix Subtraction (I/O with FILE)

Input: Input.txt

Output: The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence matrix of the subtraction result, the result converted back in fraction matrix and the runtime.

The only difference with Matrix Addition (2.2.11) is we use matrixSub()* instead of matrixAdd() to do the subtraction.

2.2.13 *Matrix Multiplication (I/O with FILE)*

Input: Input.txt

Output: The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence matrix for multiplication result, the result in fraction matrix and the

runtime.

The only difference with Matrix Addition (2.2.11) is we use matrixM()* instead of matrixAdd() to do the multiplication.

2.2.14 *Matrix Inverse (I/O with FILE)*

Input: Input.txt
Output:
1) On the screen:
The smallest prime number selected by program, the minimal number of digits for *p*-adic sequence, the *p*-adic sequence matrix for addition result, the result in fraction matrix and the runtime.
2) In the file: Out.txt.

2.2.15 Detail description of some functions

matrix2padic: Convert to the *p*-adic sequence matrix.
matrix2frac: Convert to the fraction matrix.

singleInput :Deal with the input process for the translation from single fraction to *p*-adic sequence.
numbersInput : Deal with the input process for the single numbers' operation.
matrixInput : Deal with the input process for the matrix's operation.
matrixInputFile : Deal with the input process for the matrix's operation with file.
matrixInputFileSingle : Deal with the input process for the matrix's inverse operation with file.
matrixOutputFile :Deal with the output process for the matrix's operation with file.

matrixAdd : Input two padic matrix, give out the addition result also in the format of *p*-adic matrix.
matrixSub: refer to ***MatrixAdd***
matrixM: refer to ***MatrixAdd***
matrixEleM : Input one padic matrix and one fraction, give out the addition result in the format of padic matrix.

vectorAdd: Input two padic sequence, give out the addition result also in the format of *p*adic sequence.
vectorSub: refer to ***vectorAdd***
vectorMul : refer to ***vectorAdd***
vectorWoutOffAdd: The only difference with ***vectorAdd*** is that the two padic sequence has no offset.

vectorShift : Shift the given vector to right n digits, n is also a given number. Use zero to fill in the blank digits.
vectorShiftwOff : Shift the given vector(exclude the first digit) to right n digits, n is the offset of the vector which is the first digit in the vector. Use zero to fill in the blank digits.

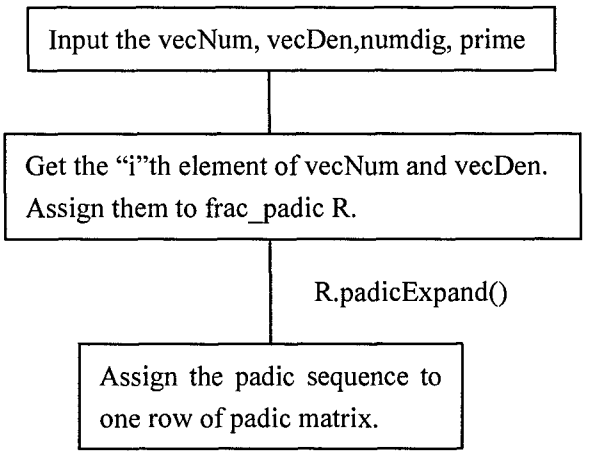
padicDivision : Single p-adic vector division.

Determinant : Calculate the determinant of the matrix. If the determinant matrix is zero, we exit the program. Otherwise, continue the calculation.

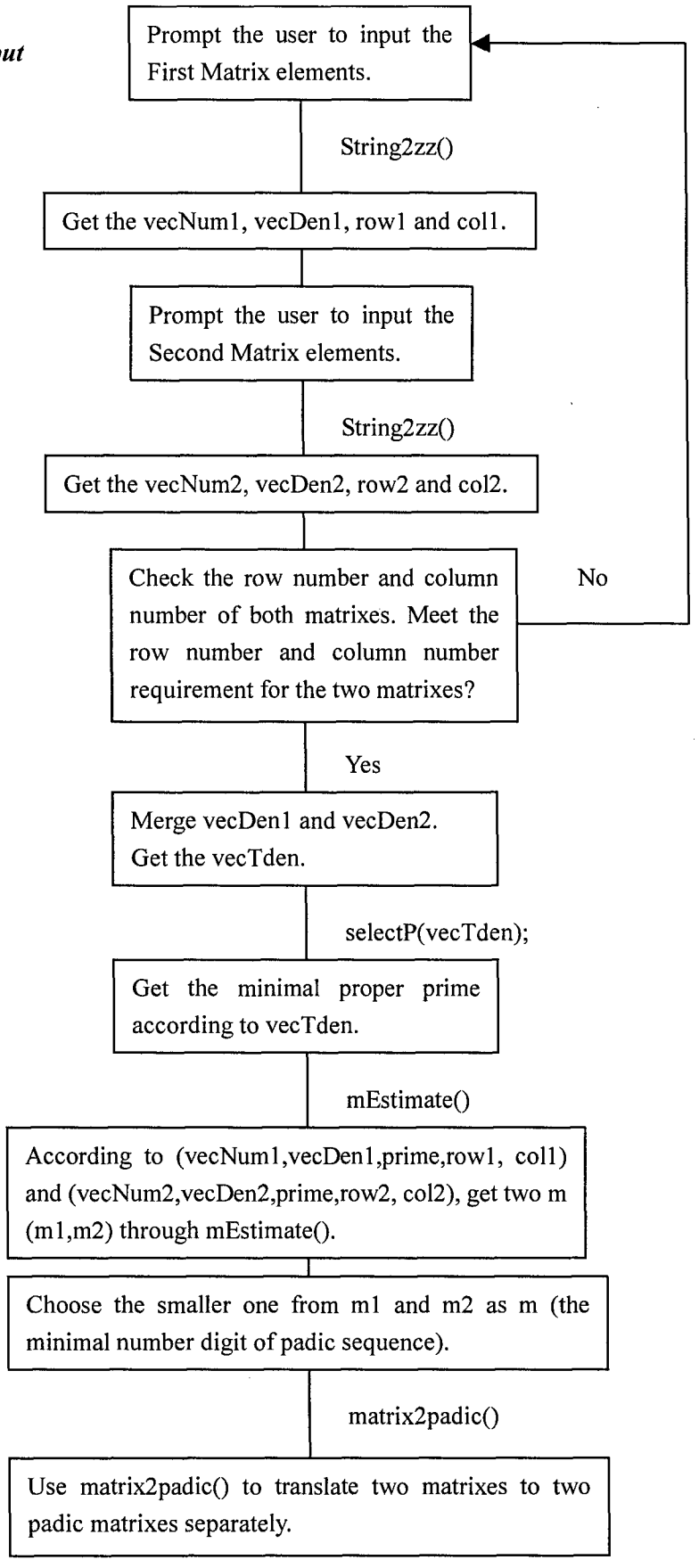
Cofactor: Calculate the matrix with elements that are the cofactors, term-by-term, of a given square matrix.

Transpose: Transpose the cofactor matrix to get the inverse of the original matrix.

1) ***matrix2padic***



2) ***matrixInput***



2.3 Source code map

File Name	Main Function	Include
PadicComputation.cpp	1) Main() of the program 2) I/O process	SingleInput() NumbersInput() MatrixInput() MatrixInputFile() MatrixOutputFile() MatrixInputFileSingle()
RationalNumber.cpp	The data structure and operation for RationalNumber.	
Frac_padic.cpp	Frac2padic, padic2frac.	
MatrixASM.cpp	Calculate the matrix addition, subtraction and multiplication.	
MatrixInv.cpp	Calculate the matrix inverse.	
MatrixCal.cpp	Transform the matrix between fraction and Padic sequence.	
Division.cpp	Calculate the padic sequence division.	
SelectP.cpp	Select the proper prime. (Old version)	
PredictM.cpp	Select the minimal number digit of padic sequence. Estimate the prime number based on the assumption of m=1.	mEstimate() mEstimate2()
Zztools.cpp	Transform the input into numerator/denominator vectors.	

3. **Results Analyses and Comparisons**

3.1 Some Testing examples and Results

Note: In the following examples, we use "*p*" as the prime number selected by the program and "*m*" as the digit number of *p*-adic sequence.

1) frac2padic

Fraction	p	m	Result
0/1	3	11	[0 0 0 0 0 0 0 0 0 0 0]
1/0			The denominator CAN NOT be zero. Please try again.
-23/15	7	17	[0 5 0 5 3 6 0 5 3 6 0 5 3 6 0 5 3]
1/1000000	3	37	[0 1 0 0 1 2 0 0 0 0 1 1 1 0 1 0 1 1 2 0 1 2 1 1 0 1 1 1 2 0 0 0 0 0 1 2 1]

2) padic2frac

Padic sequence	p	Result
[7 2 3 1]	5	78125/3
[0 1 2 1 3 15 6 9 21]	23	-6910333/3444
[0 199 233 123]	241	55647/35
[0 2 3 1 3]	5	1/3

3) Single number addition

100000+2000000000= 2000100000/1

1/233+4/67=999/15611

-1799/3+200/7= -11993/21

4) Single number subtraction

10000000000-1=9999999999

1/777777777-2/3= -518518517/777777777

5) Single number multiplication

0* 1/3 =0/1

-1/333*900=-100/37

10000000000*56=560000000000

6) Single number division

0/233=0/1

6/2=3/1

100000000/ (1/56)=5600000000

7) Matrix addition

[0,100 [199999999999,1/33 [199999999999/1,3301/33
2,99999] + -222222222, -1/2] = -222222220/1, 199997/2]

8) Matrix subtraction

[0,100 [199999999999,1/33 [-199999999999/1,3299/33
2,99999] - -222222222, -1/2] = 222222224/1,199999/2]

9) Matrix multiplication

[10000,22/3 [900,33/7 [8999846/1, 330462/7
-100000/3,3] * -21, 9] = -30000063/1, -1099811/7]

10) Matrix inverse

x=[1,2,4,6;
-2,3,7,9,
1,2,3,6,
2,3,5,-9];

x^{-1}= [5/7,-2/7,-2/7,0
-2/1,1/6,2/1,1/6
1/1,0,-1/1,0
1/21,-1/126,1/21,-1/18]

3.2 Comparison of our program's results and Matlab symbolic results

In this part, we will compare the two programs in the following three ways.

1) When the data sizes are small.

When the data sizes are small, the results are the same.

```
x = [5/7 9/3 0;                    y = [5/7 2/3 0 2/3;
   -9/8 0 3/8;                        7/8 0 5/3 6/5;
  -2/3 2/3 3/4;                            2/3 2/5 3/4 7/8];
   5/6 -7/8 0];
```

Our Program Result for x*y =

```
[  1229/392,      10/21,          5,    428/105]
[    -31/56,      -3/5,        9/32,     -27/64]
[     17/28,    -13/90,   241/144, 1457/1440]
[-229/1344,       5/9,     -35/24,    -89/180]
```

Matlab symbolic x*y=

```
[  1229/392,      10/21,          5,    428/105]
[    -31/56,      -3/5,        9/32,     -27/64]
[     17/28,    -13/90,   241/144, 1457/1440]
[-229/1344,       5/9,     -35/24,    -89/180]
```

2) When the data sizes are large

When the data sizes are large, the results are different. The special data set chosen can easily show that our result is correct, while the Symbolic Toolbox is not.

Here is an example.

```
x=[123456789987654321/77777777777777777777,    88888888888888888888/33 ;
  -123456789987654321,     88888888888888888888/33];

y=[77777777777777777777/123456789987654321,    1   ;
  33/88888888888888888888,          0];
```

Our Program Result for x*y =

```
[2                          111111111/70000000007]
[-77777777777777777776   -123456789987654321   ]
```

Matlab symbolic x*y=

```
[81129638414606679562133097328419/40564819207303340847894502572032,
```

1830034132283545/1152921504606846976]
[-3155041493901370661340022104799488036864391447196301/405648192073033408
47894502572032, -123456789987654320]

3) Runtime comparison

We have tested many examples, the results show that our programs run faster than Matlab codes for large matrix sizes, while Matlab still give the correct results, but slower than that of Matlab symbolic result when the matrix size is small. When the rational numbers have more than 20 digits, Matlab Symbolic toolbox does not give the right results, the timing is meaningless. The ESCL library is using ARBITRARY length integer type zz, and all the calculation is carried out in the p-adic field. The runtime in our program highly depends on several factors. First of all, it requires an appropriate number "M", "the minimal number of digits for p-adic sequence". This number can be given as the smallest number only when deal with single number.

4. Future Work

Improve the EstimateM(). Currently, the m estimation cannot always work well. In some situation, they cannot give the sufficient number of digits for p-adic sequence for exact computation. To avoid this kind of error, we have increased its size to plus 30 and more. We will continue to work on this problem.

We will investigate specific applications of ESCL on P-adic cyclic coding theory and quantum computational Weyl-Heisenberg representations.

References

[1] Krishnamurthy, F. V. "Matrix Processors Using P-adic Arithmetic for Exact Linear Computations", IEEE Transactions on Computers, vol. C-26, No. 7, July 1977.

[2] Vladimirov, V.S., Volovich, I.V. and Zelenov, E.I. *P-adic Analysis and Mathematical Physics,* Series on Soviet & East European Mathematics – Vol. 1.World Scientific 1993.

[3] Lu, C. and An, M. "Final Report of Simulation of Quantum Time-Frequency Transform Algorithms", *FA9550-04-1-0406*", 2005.

[4] Shoup, V *NTL library* at: http://shoup.net/ntl/

[5] Kornerup, P. and Gregory, R.T. "Mapping Integers and Hensel Codes onto Farey Fractions", BIT 23 (1983), 9-20.

[6] Dixon, J. "Exact Solution of Linear Equations Using P-adic Expansions", Numerische Mathematik 40, 137-141 (1982) Springer-Verlag.