

# Lifting Factorization in Maple

Sreedevi V P, Arathi T, and Soman K P  
 Amrita Vishwa Vidyapeetham/CEN, Coimbatore, India  
[sreedevivp4u@gmail.com](mailto:sreedevivp4u@gmail.com)  
[arathiviswam09@yahoo.co.in](mailto:arathiviswam09@yahoo.co.in)  
[kp\\_soman@ettimadai.amrita.edu](mailto:kp_soman@ettimadai.amrita.edu)

**Abstract** - This paper gives an insight into the polyphase matrix factorization by using Euclidean algorithm and its implementation in maple. The maple implementation allows the program to be called from Matlab. Polynomial reduction using Groebner bases is also incorporated into the program. This reduces the number of wavelet filter coefficients appearing in a given expression through use of the relations they satisfy, thereby permitting exact symbolic factorizations for any polyphase matrix.

## I. INTRODUCTION

Wavelets are sets of basis functions used in the analysis of signals and images. Wavelets have the ability to examine signals simultaneously in both time and frequency domains, since they have both spatial and frequency localization, making them useful for the analysis of sharply-varying or non-periodic signals. Analyzing a signal with short duration finite energy functions and transforming the signal under consideration into a more useful form is called wavelet transform. Here, the lifting scheme is used to calculate wavelet transform using polyphase matrix factorization in an efficient way using the Euclidean algorithm extended to Laurent polynomials.

Lifting scheme [1] is a new approach in the construction of second generation wavelets. Second generation wavelets are those, which needn't be necessarily translations and dilations of one function, which is the characteristic feature of first generation wavelets (classical wavelets). Wavelet transform is built through lifting using three main steps: Split, Predict and Update.

The use of lifting scheme for obtaining the discrete wavelet transform of signals is well established. It is an efficient technique in image compression applications and has been incorporated into the JPEG2000 standards. Since the lifting scheme involves the symbolic factorization of a matrix called the polyphase matrix, its implementation calls for symbolic computation. The factorization of the polyphase matrix replaces the process of matrix multiplication with a sequence of trivial operations, involving simple multiplication, addition, and shifting of data vectors. This has the advantages of being faster and easily invertible, and allows in-place calculation. In [2] M.Maslen describes the polyphase matrix factorization by employing Euclidean algorithm and gives a mathematica program for it. The maple implementation of the same is presented here. Since students are mostly used to working in Matlab, the maple implementation gives them the flexibility to call the program from Matlab.

## II. UNDERLYING PRINCIPLE OF POLYPHASE FACTORIZATION

The key idea of polyphase factorization is the application of the Euclidean algorithm to Laurent polynomials. A Laurent polynomial [2] is an expression of the form:

$$p(z) = \sum_{k=a}^b c_k z^{-k} \quad (1)$$

The degree of a non-zero Laurent polynomial is defined as  $|p(z)| = b - a$  and the degree of the zero polynomial is defined to be  $-\infty$ . A finite filter can be described by a set of coefficients  $h = \{h_k: k \in \mathbb{Z}\}$ , which are zero outside the finite range  $k_b \leq k \leq k_e$ . The z-transform of a finite filter is the Laurent polynomial whose coefficients are the filter coefficients namely:

$$h(z) = \sum_{k=k_b}^{k_e} h_k z^{-k} \quad (2)$$

The Euclidean algorithm finds the greatest common divisor (gcd) of two integers. The algorithm states that for any two integers  $a$  and  $b$ , where  $abs(a) > abs(b)$  and  $b \neq 0$ , there exist integers  $q_1$  and  $r_1$  such that  $a = q_1 b + r_1$  with  $0 \leq r_1 \leq b$ . The algorithm proceeds as follows:

- 1) If  $r_1 = 0$ ,  $gcd(a, b) = b$ . If  $r_1 \neq 0$ , then the common divisor of  $b$  and  $r_1$  is a divisor of  $a$  and hence is a common divisor of  $a$  and  $b$ .
- 2) Since  $r_1 = a - q_1 b$ , any common divisor of  $a$  and  $b$  is also a common divisor of  $r_1$  and  $b$ . Hence,  $gcd(a, b) = gcd(r_1, b)$ .
- 3) This process is iterated, yielding a smaller remainder each time, finally resulting in  $r_i = 0$  for some  $i$ . Then,  $r_{i-2} = q_i r_{i-1}$  and hence,  $gcd(a, b) = gcd(r_{i-2}, r_{i-1}) - r_{i-1}$ .

The algorithm is implemented using a simple recursive procedure, given by the two equations below:

$$a_i \leftarrow b_{i-1} \quad (3)$$

$$b_i \leftarrow a_{i-1} - a_i q_i \quad (4)$$

When generalizing the Euclidean algorithm to Laurent polynomials, the procedure remains same, with the requirement that  $|r| < |b|$ , where  $| \cdot |$  is the Laurent degree.

Although the algorithm yields a unique gcd, the quotients can be chosen in several different ways (Illustrated with example in [2]). When using the algorithm for factorizing the polyphase matrix, whose entries will be Laurent polynomials, the matrices obtained after factorizing will be determined by the quotients obtained from the algorithm. Hence, we can

obtain several distinct matrix factorizations, by using different sets of quotients.

### III. PROGRAM DESCRIPTION

This section describes the various modules used for the Maple implementation of the factorization of the polyphase matrix using the lifting scheme.

#### A. Overview

The program developed, finds all possible factorizations of two Laurent polynomials using the Euclidean algorithm with degree one quotients. Initially, the program checks if the number of steps required is odd, and if so, prepends a zero to the quotient list found in the Euclidean algorithm. This will effectively produce an identity matrix as the first factor, and hence the only effect this step has, is to cause the following matrix factors to be transposed. The lifting term  $s(z)$  is found by equating the factorized and unfactorized forms of the polyphase matrix and solving appropriately.

The polyphase matrix factorization is implemented here as several modules.

#### B. SingleIteration Module

The input to this module are the initial  $a$  and  $b$  on which the first iteration of the Euclidean algorithm should be executed, together with an elimination option. The output of this module is the quotient, the remainder and the Laurent polynomial, which should be used in the next step. This module is iterated (while loop), till the remainder becomes equal to zero. The general expression for a quotient of degree one is defined as  $z^n(c + \frac{d}{z})$ .  $n$  is determined by matching the maximum powers.  $c$  and  $d$  are determined by the elimination option [2]. The four possible options are:

*EliminateEndsMatchHigh*: This option means that the highest and lowest powers of the expression are to be eliminated, but the highest powers are to be eliminated by the suitable choice of  $n$ .

*EliminateEndsMatchLow*: The extreme power terms will be eliminated, but the lowest two powers will be matched.

*EliminateLow*: In this case, the lowest power terms will be matched and eliminated.

*EliminateHigh*: The highest two power terms will be matched and eliminated.

#### Steps Involved

- 1) Read the input polynomials  $a$  and  $b$ .
- 2) Degree of generic quotient  $n := 1\text{degree}(a) - 1\text{degree}(b) + 1$ ;
- 3) Generic quotient  $:= z^n(c + d/z)$ ;
- 4) Find remainder by  $a - b * q$
- 5) Collect coefficients of  $z$  in descending order of degree.
- 6) Remove integer coefficients and make list of coefficients involving 'c' and 'd'.
- 7) Apply elimination options, select two equations in each step, solve and find values of  $c$  and  $d$ .
- 8) Substitute values of 'c' and 'd' and find the quotient.
- 9) Find the remainder and form the list as  
Output list [Quotient, Polynomial 'b', Remainder]

#### C. FindAll Module

This module generates all possible elimination branches and applies the SingleIteration module on each choice, until a zero remainder is obtained. The module runs so that every equation possible is solved once, stored and recombined appropriately.

#### Steps Involved

- 1) Read input and set the iteration level to 1.
- 2) Make the list of elimination options as the PossibleOption.
- 3) Call SingleIteration module to apply Euclidean Algorithm on  $a$  and  $b$  with all possible options and collect the quotients, second polynomial and remainder from each option into a list IterationLevel. Check whether remainder obtained in any one option is zero, if so, it can be taken as gcd and the quotients and gcds can be collected.
- 4) If not, check whether a monomial is obtained in that iteration. If so, only one elimination option is required in the next iteration, since all options yield the same result. SingleIteration module is iterated, until gcd is obtained.
- 5) If not a monomial in the current iteration, all options are applied and the iteration is continued. If a monomial is obtained, go to step 5.
- 6) Collect quotients from each IterationLevel as separate lists.
- 7) Collect gcds from each IterationLevel as separate lists.
- 8) Collect from different levels, quotients and gcds of same option. Make a list of quotients and gcds from same option and return as result.

#### D. Polyphase Factorization Module

This module takes in as arguments, the elements of the polyphase matrix and gives the factorization of the polyphase matrix. The given polyphase matrix must have determinant 1. This module calls FindAll module with 2 polynomials Heven and Hodd as arguments. FindAll module iterates SingleIteration module with all possible elimination options, until a zero remainder is obtained and collect all the quotients and gcds appropriately.

The module first checks if the number of quotients obtained is odd, and if so, it prepends a zero to the quotient list found in the Euclidean algorithm. This produces a valid factorization of the polyphase matrix. The factorization is then given by constructing the appropriate matrices with the quotients, the lifting term, and the scaling matrix which is determined by the gcd.

#### Steps Involved

- 1) Generate quodd matrices which have the form:
 
$$\begin{bmatrix} 1 & q_{2i-1}(z) \\ 0 & 1 \end{bmatrix}$$
 and qeven matrices of the form:
 
$$\begin{bmatrix} 1 & 0 \\ q_{2i} & 1 \end{bmatrix}.$$
- 2) Generate Lifting matrix of the form:
 
$$\begin{bmatrix} 1 & -(\text{gcd new})^2 * T(z) \\ 0 & 1 \end{bmatrix},$$
 where gcdnew is the gcd obtained by Euclidean factorization.

3) Generate scaling matrix of the form:

$$\begin{bmatrix} \text{gcd}_{new} & 0 \\ 0 & \frac{1}{\text{gcd}_{new}} \end{bmatrix}$$

4) The lifting term  $T(z)$  is found by equating the factorized and not factorized forms of the polyphase matrix and solving appropriately.

5) The representation obtained will be of the form:

$$\begin{bmatrix} H_{\text{even}}(z) & G_{\text{even}}(z) \\ H_{\text{odd}}(z) & G_{\text{odd}}(z) \end{bmatrix} = \prod_{i=1}^{(n/2)-1} \begin{bmatrix} 1 & q_{2i-1}(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ q_{2i}(z) & 1 \end{bmatrix} \begin{bmatrix} 1 & -c^2 s(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c & 0 \\ 0 & \frac{1}{c} \end{bmatrix}$$

6) Multiply the factor matrices on the RHS and equate with the unfactorized form on the LHS. Two equations are obtained, involving the lifting term  $T(z)$ . The set can be reduced by using Groebner reduction and the system can be easily solved.

7) Since there are 4 different elimination options, 4 different sets of quotients and gcds are obtained from FindAll module. Using these, 4 different factorizations for the polyphase matrix can be obtained.

#### IV. MAPLE CODE

*A. Program to perform polyphase factorization by lifting scheme in Daubechies 4 tap filter.*

Aim: To factorize polyphase matrix.

```
>restart;
>with(Groebner);
>with(linalg);
with(ListTools);
```

1) Single iteration procedure to perform first step of polyphase factorization

Aim: To perform Euclidean algorithm on 2 input polynomials, find quotient, remainder and second polynomial.

```
>SingleIteration := proc(a, b, choice)
local q, n, GenericQuotient, Remainder, RemainderCoefficient,
QuotientCoefficient, c, d, i, k, LC, T, r, j, newlist, newcoeff;
c := 'c';
d := 'd';
```

Degree of quotient

```
n := 1degree(a) - 1degree(b) + 1;
```

Form of quotient

```
GenericQuotient := z^n*(c + d/z);
```

Finding remainder by  $a-b*q$

```
Remainder := collect(expand(a-b*GenericQuotient,z),z);
```

Finding number of terms in Remainder

```
k := numboccur(Remainder, z) + 1;
```

Collecting coefficients of z in descending order of degree for i from 1 to k do

```
LC := lcoeff(Remainder,z)*z^degree(Remainder,z);
```

```
RemainderCoefficient[i] := lcoeff(Remainder,z);
```

```
Remainder := Remainder-LC;
```

```
end do;
```

```
j := 1;
```

Removing integer coefficients

```
for i from 1 to k do
```

```
if(type(RemainderCoefficient[i],numeric))then i;
```

```
else newcoeff[j] := RemainderCoefficient[i];
```

```
j := j + 1;
```

```
end if;
```

```
end do; j := j - 1;
```

Making a list of coefficients

```
newlist := seq(newcoeff[p], p = 1..j);
```

Solving equations on the basis of elimination option

```
if(choice=EliminateHigh)then
```

```
QuotientCoefficient := solvefor[c,d](newlist[1],newlist[2]);
```

```
elif(choice = EliminateLow)
```

```
then
```

```
QuotientCoefficient := solvefor[c,d](newlist[-1],newlist[-2]);
```

```
elif(choice = EliminateEndsMatchHigh)
```

```
then
```

```
QuotientCoefficient := solvefor[c,d](newlist[1],newlist[-1]);
```

```
elif(choice = EliminateEndsMatchLow)
```

```
then
```

```
QuotientCoefficient := solvefor[c,d](newlist[1],newlist[-1]);
```

```
else
```

```
sorry;
```

```
end if;
```

```
assign(QuotientCoefficient);
```

Quotient

```
q := expand(z^n*(c + d/z), z);
```

Remainder

```
r := expand(a - b*q, z);
```

```
[q, b, r]; Output list [Quotient, Polynomial 'b', Remainder]
```

```
end proc;
```

2) FindAll procedure to find all factors of two polynomials using Euclidean Factorization

Aim: Iterate SingleIteration module with all possible elimination options, until a zero remainder is obtained and collect all the quotients and gcds appropriately.

```
>FindAll := proc(a, b)
```

```
local j, IterationLevel, RemainderZero, NChoices,
```

```
PossibleOption,k,q,qsec,gcds,i,result;
```

```
j := 1; Variable to denote number of levels of branch
```

```
PossibleOption := {EliminateLow,
```

```
EliminateHigh,EliminateEndsMatchHigh,Eliminate
```

```
EndsMatchLow};
```

```
IterationLevel[1] := seq(SingleIteration(a,b,
```

```
PossibleOption[i],i=1..4);
```

Performing all options in one step

```
RemainderZero := false; Variable to check whether gcd is obtained
```

```
if(max(abs(IterationLevel[1][1][3])) = 0)then
```

```
RemainderZero := true;
```

```
else RemainderZero := false;
```

```
endif;
```

```
j := j+1; Check whether monomial is obtained in an iteration
```

```
if(type(IterationLevel[j-1][1][3],monomial))then
```

```
NChoices := 1; Variable to denote number of options
```

```
else NChoices := 4;
```

```
end if;
```

```
if(NChoices=1)then
```

while RemainderZero = false do Only one option is required when monomial is obtained in an iteration

```
IterationLevel[j] := seq(SingleIteration(simplify(IterationLevel[j-1][i][2]),simplify(IterationLevel[j-1][i][3]),
```

```
PossibleOption[1]1),i=1..4);
```

```
if(max(abs(IterationLevel[j][1][3])) = 0)then
```

```
RemainderZero := true;
```

```
else RemainderZero := false;
```

```
j := j + 1;
```

```
end if;
```

```

end do;
else while RemainderZero = false do If not a monomial all
options are to be applied
IterationLevel[j] := [seq(seq(SingleIteration(simplify(IterationLe
vel[j-1][k][2]),simplify(IterationLevel[j-1][k][3]),
PossibleOption[i],k=1..4^(j-1)),i=1..4)];
if(max(abs(IterationLevel[j][1][3])) = 0)then
RemainderZero := true;
else RemainderZero := false;
j := j + 1;
end if;
end do;
end if;
end if;
Collecting quotients from all IterationLevel as separate
lists
q[1] := [seq(IterationLevel[1][i][1],i=1..4)];
for k from 2 to j do
if(NChoices=1)then
q[k] := [seq(IterationLevel[k][i][1],i=1..4)];
else
q[k] := [seq(IterationLevel[k][i][1],i=1..4^(k-1))];
end if;
end do;
Collecting gcds from all iteration levels
gcds := [seq(IterationLevel[j][i][2],i=1..4^(j-1))];
for i from 1 to 4 do
qsec[i] := [seq(q[k][i],k=1..j),gcds[i]]; Collecting from
different levels quotients and gcds of same option
end do;
result := seq(qsec[k],k=1..4); Making a list of result
end proc;

```

### B. Procedure to factorize input polyphase matrix

```

>Polyphasefactorization := proc(a,b,c,d)
local Polym,Result,gcdnew,qarr,num,right1,right2,right3,subm,
P,T,G,sol,Gevennew,Goddnew,Polymlift,checkm,i,inp,z;
global qodd,qeven,gcdm,Tmat;
z := 'z';
Input polyphase matrix
Polym := matrix(2,2,[a,b,c,d]);
Applying Euclidean factorization on Heven and Hodd
filters. Finding all factors and gcds
Result := FindAll(a,c);
List of quotients and gcds are obtained as Result.
for inp from 1 to 4 do
gcdnew|inp := Result[inp][-1];
qarr|inp := Array(Result[inp][1]);
Check the number of quotients obtained, if odd, prepend
a 0 value to the qarr array to make num even. Make the
list of quotients.
num|inp := ArrayNumElems(qarr|inp);
if(type(num|inp, odd))then
qarr|inp := [0, Result[inp][1]];
qarr|inp := Flatten(qarr|inp);
num|inp := num|inp + 1;
else
qarr|inp := Result[inp][1];
qarr|inp := Flatten(qarr|inp);
end if;
Creating factor matrices
qodd|inp := matrix(2,2,[1,0,0,1]);
qeven|inp := matrix(2,2,[1,0,0,1]);
Generate qodd matrices and qeven matrices
for i from 1 to num|inp do
if (type(i, odd))then

```

```

qodd|inp := multiply(qodd|inp,matrix(2,2,[1,quarr|
inp[i],0,1]));
else
qeven|inp := multiply(qeven|inp,matrix(2,2,
[1,0,quarr|inp[i],1]));
end if;
end do;
Lifting matrix
Tmat|inp := matrix(2,2,[1,-(gcdnew|inp)^2*T|inp,0,1]);
Gcd matrix
gcdm|inp := matrix(2,2,[gcdnew|inp,0,0,1/gcdnew|inp]);
Finding the lifting term T(z) by equating the factorized
and unfactorized polyphase matrices.
Multiplying the factors on the RHS
right1 := evalm(multiply(qodd|inp,qeven|inp));
right2 := evalm(multiply(right1,Tmat|inp));
right3 := evalm(multiply(right2,gcdm|inp));
Equating the LHS and RHS and solving the equations
using Groebner basis
subm := evalm(Polym - right3);
P := [subm[1,1],subm[2,2]];
sol := solvefor[T|inp](P);assign(sol);
Substituting T(z) and updating Geven and Godd
Gevennew := evala(right3[1,2]);
Goddnew := evala(right3[2,2]);
Tmat|inp := matrix(2,2,[1,-
simplify(evala((gcdnew|inp)^2*T|inp)),0,1]);

```

## V. PROGRAM DEMONSTRATION

This section demonstrates the use of the program with examples.

### A. SingleIteration module

Input to the module

$$a := \frac{1}{z} + 3 - 2z; b := -\frac{6}{z^2} + \frac{3}{z};$$

Output of the module

SingleIteration(a, b, EliminateLow);

$$\left\{ -\frac{7z^2}{12} - \frac{z}{6}, -\frac{6}{z^2} + \frac{3}{z}, -\frac{z}{4} \right\}$$

SingleIteration(a, b, EliminateHigh);

$$\left\{ -\frac{2z^2}{3} - \frac{z}{3}, -\frac{6}{z^2} + \frac{3}{z}, -\frac{1}{z} \right\}$$

SingleIteration(a,b,EliminateEndsMatchHigh);

$$\left\{ -\frac{2z^2}{3} - \frac{z}{6}, -\frac{6}{z^2} + \frac{3}{z}, -\frac{1}{2} \right\}$$

SingleIteration(a,b,EliminateEndsMatchLow);

$$\left\{ -\frac{2z^2}{3} - \frac{z}{6}, -\frac{6}{z^2} + \frac{3}{z}, -\frac{1}{2} \right\}$$

### B. FindAll module

Input to the module

$$a := \frac{1}{z} + 3 - 2zb := -\frac{6}{z^2} + \frac{3}{z}$$

Output of the module

FindAll (a, b);

$$\left\{ \left\{ -\frac{7z^2}{12} - \frac{z}{6}, -\frac{12}{z^2} + \frac{24}{z^3}, -\frac{z}{4} \right\}, \left\{ -\frac{2z^2}{3} - \frac{z}{6}, -3 + \frac{6}{z}, -\frac{1}{z} \right\} \right\}$$

$$\left\{ \left\{ -\frac{2z^2}{3} - \frac{z}{6}, \frac{12}{z^2} + \frac{6}{z} \right\}, -\frac{1}{2} \right\}, \left\{ \left\{ -\frac{2z^2}{3} - \frac{z}{6}, \frac{12}{z^2} + \frac{6}{z} \right\}, -\frac{1}{2} \right\}$$

*C.Polyphase Factorization module*

Performing polyphase factorization by lifting scheme in Daubechies 4-tap filter.

Designing the orthogonal wavelet system

Using the function DaubProc(N), we obtain:

$$> h0; \frac{\sqrt{2}}{8} + \frac{\sqrt{6}}{8} > h1; \frac{3\sqrt{2}}{8} + \frac{\sqrt{6}}{8} > h2; \frac{3\sqrt{2}}{8} - \frac{\sqrt{6}}{8} > h3; \frac{\sqrt{2}}{8} - \frac{\sqrt{6}}{8}$$

$$> a := \text{Heven}; a := \frac{\sqrt{2}}{2} + \frac{\sqrt{6}}{8} + \frac{3\sqrt{2} - \sqrt{6}}{8z}$$

$$> b := \text{Geven}; b := -\left( \frac{\sqrt{2}}{8} - \frac{\sqrt{6}}{6} \right) z - \frac{3\sqrt{2}}{8} - \frac{\sqrt{6}}{8}$$

$$> c := \text{Hodd}; c; + \frac{3\sqrt{2}z + \sqrt{6}z + \sqrt{2} - \sqrt{6}}{8z}$$

$$> d := \text{Godd}; d := \left( \frac{3\sqrt{2}}{8} - \frac{\sqrt{6}}{8} \right) z + \frac{\sqrt{2}}{8} + \frac{\sqrt{6}}{8}$$

>Polyphasefactorization (a, b, c, d);

Factorization 1

$$\begin{bmatrix} 1 & -(2+\sqrt{3})(4Z+2\sqrt{3}-3) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \frac{-(-2+\sqrt{3})(3Z-2\sqrt{3}+3)}{12Z^2} & 1 \end{bmatrix} X$$

$$\begin{bmatrix} 1 & (7+4\sqrt{3})Z^2(3Z+4\sqrt{3}) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{Z\sqrt{3}\sqrt{2}}{(-2+\sqrt{3})(\sqrt{3}-1)} & 0 \\ 0 & \frac{\sqrt{3}(-2+\sqrt{3})\sqrt{2}(\sqrt{3}-1)}{6Z} \end{bmatrix}$$

Factorization 2

$$\begin{bmatrix} 1 & -\sqrt{3} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{3}(3z-2\sqrt{3}+3)}{12z} & 1 \end{bmatrix} \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}-1} & 0 \\ 0 & \frac{\sqrt{2}(\sqrt{3}-1)}{2} \end{bmatrix}$$

Factorization 3

$$\begin{bmatrix} 1 & -\sqrt{3} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{3}(3z-2\sqrt{3}+3)}{12z} & 1 \end{bmatrix} \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}-1} & 0 \\ 0 & \frac{\sqrt{2}(\sqrt{3}-1)}{2} \end{bmatrix}$$

Factorization 4

$$\begin{bmatrix} 1 & \frac{\sqrt{3}}{3} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \frac{(2+\sqrt{3})(3z-2\sqrt{3}+3)}{4} & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{3z} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{z(3+\sqrt{3})} & 0 \\ 0 & \frac{z\sqrt{2}(3+\sqrt{3})}{2} \end{bmatrix}$$

Factorization of the polyphase matrix using lifting scheme has also been implemented for the following orthogonal and biorthogonal filters:

- i) Daubechies 6-tap filter
- ii) Daubechies 8-tap filter
- iii) Daubechies 10-tap filter
- iv) Cohen-Daubechies-Feauveau Biorthogonal (5, 3) filter [6]
- v) Cohen-Daubechies-Feauveau Biorthogonal (7, 5) filter [6]

VI. CONCLUSION

The maple program developed here, finds all the possible factorizations of the polyphase matrix. Conversion of this factorization into a sequence of lifting steps has been automated.

The idea illustrated in this paper, using lifting scheme, leads to a simpler implementation of wavelet transforms, resulting in simple inversion, and lossless compression. Lifting scheme allows a parallel, in-place implementation, which is almost two-fold faster for large datasets. It reduces the implementation to a sequence of simple steps involving elementary algebraic operations, making it trivial to find the inverse transform. The simplicity in finding the wavelet transform as a result of using the lifting scheme, is further enhanced by using the symbolic computing package maple for its implementation, which lends it the flexibility to be called externally from Matlab.

REFERENCES

[1] K.P.Soman, K.I.Ramachandran, 'Insight into wavelets – From theory to practice', 14: 215-260, Second Edition, Amrita Vishwa Vidyapeetham, Ettimadai, Coimbatore.

[2] M. Maslen, P. Abbott, 'Automation of the lifting factorisation of wavelet transforms', Department of Physics, University of Western Australia, Nedlands, WA 6907, Australia.

[3] Daubechies, I. and W. Sweldens, Factoring wavelet transforms into lifting steps, Journal of Fourier Analysis and Applications, 4(3): 245-267, 1998.

[4] Sweldens, W., The lifting Scheme: A construction of second-generation wavelets, SIAM Journal on Mathematical Analysis, 29(2):511-546, 1997.

[5] Sweldens, W., The lifting Scheme: A custom-design of biorthogonal wavelets, Journal of Applied and Computational Harmonic Analysis, 3:186-200, 1996.

[6] Cohen, A., I. Daubechies, and J. Feauveau, Biorthogonal bases compactly supported wavelets, Comm. Pure Appl. Math., 45, 485-560, 1992.