Algorithms for Computation of Reversible Integer Transform from Linear Float Transform and Minimization of Rounding Errors*

Alexei Hmelnov[1][0000−0002−0125−1130]

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, 134 Lermontov st. Irkutsk, Russia hmelnov@icc.ru
http://idstu.irk.ru

Abstract. Various compression algorithms use linear transforms to represent data vectors in different coordinate systems, where they can be compressed better. The matrices usually have float coefficients, and the data vectors are integer, so some rounding is required. And to make the compression lossless it is required to make these transforms reversible, i.e. to be able to exactly restore the original vectors from the results of their transform.
In this article we'll consider a straightforward algorithm for finding the decomposition of linear transform matrix and the approaches for estimation of the mean square approximation error and for finding the optimal decomposition, which minimizes the error.

Keywords: Lossless coding · Data compression · Invertible integer transform · Error estimation · Mean square error.

1 Introduction

When developing data formats for storing large volumes of multichannel data it is worth to take into account the correlation between the channels to avoid storing almost the same data again and achieve a better compression ratio. To measure the correlation and decorrelate the data we can use such methods as Karhunen-Loeve Transform (KLT) [1] to transfer the original values to a better coordinate system, where most part of coordinates would represent differences between some original channels, which are generally small. In this article we'll consider only integer input data. Because it is improbable, that the matrix of KLT transform would have all integer coefficients, the result of the transform will be represented by real values. But it is highly ineffective to immediately use

* work supported by RAS (projects: AAA-A17-117032210079-1, AAA-A19-119111990037-0), RFBR (projects:18-07-00758-a, 17-57-44006-Монг-а) and the project "Fundamentals, methods and technologies of digital monitoring and forecasting of the ecological situation in the Baikal natural territory" (2020-1902-01-071). The results were obtained using the core facilities center "Integrated Information Network of the Irkutsk Scientific and Educational Complex".

the floating point values to represent the real results of the transform, because the larger memory consumption of the floating point values would outweigh any advantages of the decorrelation.

So it is required to represent somehow the results of the transform by integer values to take advantage from the decorrelation. And it is not enough to just round the transform results, because it would cause information loss and may not allow to restore the original input values from the stored rounded values. Thus, we need to approximate the real-valued linear transform by some integer transform, that will be very close to the real-valued transform and will be reversible. I.e. it should be possible to restore the original values from the results of the integer approximation of the transform.

The problem of building reversible integer approximation of linear transform was thoroughly considered in the works of Hao and Shi [2–5]. The main idea of their approach is to represent the linear transform matrix A by a product of *elementary reversible matrices* (ERM), e.g. triangular matrices with diagonal elements of absolute value 1. The process of computation of the product of a vector by ERM with rounding resulting vector coordinates can be organized so, that it can be reversed. We'll consider the idea in more details in the next section.

The article [4] is the most close to our work: here the authors prove, that matrix A with $|detA| = 1$ of size $n \times n$ can be decomposed into the product of permutation matrix and at most $n + 1$ *single-row elementary reversible matrices* (SERM). The algorithm of SERM decomposition by Hao and Shi is based upon high-level matrix operations. They also develop some error estimation approach, which is based upon the infinity norm $\|\cdot\|_\infty$, i.e. they estimate and try to minimize the maximum error of vector coordinate.

In this article we'll consider a straightforward algorithm for finding the SERM decomposition of the transform matrix and the approaches for estimation of the mean square approximation error and for finding the optimal decomposition, which minimizes the error.

2 Reversible Integer Transforms

When developing some lossless data compression format, which performs a transform of integer vectors, say, the RGB color components, it is required to make the transform reversible. So, the developers of the formats use some reversible discrete approximations of the continuous transforms.

2.1 The Jpeg 2000 color transform

An example of simple reversible color transform is used in the JPEG 2000 file format. The JPEG 2000 uses the following "Reversible multiple component transformation" (RCT – forward transform of the RGB color space):

$$Yr = \lfloor (R + 2G + B)/4 \rfloor; Ur = R - G; Vr = B - G$$

which has the following reverse transform ("Inverse RCT"):

$$G = Yr - \lfloor (Ur + Vr)/4 \rfloor; R = Ur + G; B = Yr + G$$

This transform is rather rough approximation of the exact transform of the color space RGB to the color space YCrCb, which is defined by the CCIR 601 standard:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cr &= 0.5R - 0.4187G - 0.0813B + 128 \\ Cb &= -0.1687R - 0.3313G + 0.5B + 128 \end{aligned}$$

2.2 New color transform for remote sensing images

After analyzing correlation between color components of particular Earth remote sensing images We have suggested the following new reversible integer transform of the color components, which allows to obtain better compression ratio for this kind of images:

$$Cr_1 = \lfloor (R + G + B)/3 \rfloor; Cr_2 = \lfloor (R + B - 2G)/2 \rfloor; Cr_3 = B - R \qquad (1)$$

which has the following reverse transform:

$$\begin{aligned} C = 2Cr_2 + Cr_3 \bmod 2; G = Cr_1 - \lfloor C/3 \rfloor; \\ R = G + (C - Cr_3)/2; B = G + (C + Cr_3)/2 \end{aligned} \qquad (2)$$

Let's prove, that the transform (2) is indeed the inverse of the transform (1).

Theorem 1. *The color space transform for remote sensing images (1) is reversible, and its inverse is the transform (2).*

Proof. Here we consider the operation $\lfloor\ \rfloor$ as rounding down and the values like $X \bmod D$ are always positive. For example $-7 \bmod 5 = -2 + 5 = 3$. So

$$\lfloor X/D \rfloor = (X - X \bmod D)/D$$

Thus

$$\begin{aligned} C = 2Cr_2 + Cr_3 \bmod 2 &= 2\lfloor (R + B - 2G)/2 \rfloor + (B - R) \bmod 2 \\ &= (R + B - (R + B) \bmod 2) - 2G + (B + R - 2R) \bmod 2 \\ &= R + B - 2G \end{aligned}$$

Hence

$$\begin{aligned} Cr_1 - \lfloor C/3 \rfloor &= \lfloor (R + G + B)/3 \rfloor - \lfloor (R + B - 2G)/3 \rfloor \\ &= \lfloor (R + G + B)/3 \rfloor - \lfloor (R + B + G)/3 - G \rfloor = G \end{aligned}$$

$$G + (C - Cr_3)/2 = G + (R + B - 2G - (B - R))/2 = R$$

$$G + (C + Cr_3)/2 = G + (R + B - 2G + (B - R))/2 = B$$

□

In fact, development of this kind of reversible transforms is more like a mathematical trick and it is impossible to generalize this kind of results to integer approximations of arbitrary continuous transforms.

2.3 The Main Idea of the General Approach to Construction of Reversible Integer Approximations

To perform a reversible turn of a raster image by an angle a it is widely known the lifting scheme:

$$\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1-\cos\alpha}{\sin\alpha} & 1 \end{bmatrix} \begin{bmatrix} 1 & -\sin\alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{1-\cos\alpha}{\sin\alpha} & 1 \end{bmatrix}$$

which can be generalized to formulate the general approach to reversible integer approximation of transforms.

Let's consider the main idea of the approach, which can be used for almost any continuous transform, provided that it can be rewritten in the form required by the approach. We'll do it here in a little more general way, than in [4].

A continuous transform $\overline{y} = \overline{F}(\overline{x})$ should be split into steps of the form:

$$\begin{aligned} y_0 &= k_0 x_n + f_0(x_1, ..., x_{n-1}) \\ y_1 &= k_1 x_1 + f_1(y_0, x_2, ..., x_{n-1}) \\ y_i &= k_i x_i + f_i(y_0, ..., y_{i-1}, x_{i+1}, ..., x_{n-1}) \\ y_n &= k_n y_0 + f_n(y_1, ..., y_{n-1}) \end{aligned}$$

where $k_i \in \mathbb{Z}$, $k_i \neq 0$ and y_0 is an auxiliary intermediate value, which will not be included into the resulting vector $\overline{y}$. Usually the coefficients $k_i \in \{-1, 1\}$, because larger absolute values will extend the range of y_i, and it would make ineffective the following compression steps.

Therefore we can define reversible integer approximation of the transform $\overline{y'} = \overline{F'}(\overline{x})$, where the vectors $\overline{x}$ and $\overline{y'}$ have all integer components:

$$\begin{aligned} y'_0 &= k_0 x_n + [f_0(x_1, ..., x_{n-1})] \\ y'_1 &= k_1 x_1 + [f_1(y'_0, x_2, ..., x_{n-1})] \\ y'_i &= k_i x_i + \left[f_i(y'_0, ..., y'_{i-1}, x_{i+1}, ..., x_{n-1})\right] \\ y'_n &= k_n y_0 + \left[f_n(y'_1, ..., y'_{n-1})\right] \end{aligned} \tag{3}$$

here [ ] denotes some rounding operation, like *round*,*ceil*,*floor*.

Then we'll be able to reverse this transform by expressing from the equations y_0 and x_i and rewriting the steps in backward order:

$$\begin{aligned} y_0 &= y'_n/k_n - \left[f_n(y'_1, ..., y'_{n-1})\right] \\ x_i &= y'_i/k_i - \left[f_i(y'_0, ..., y'_{i-1}, x_{i+1}, ..., x_{n-1})\right] \\ x_1 &= y'_1/k_1 - \left[f_1(y'_0, x_2, ..., x_{n-1})\right] \\ x_n &= y'_0/k_0 - \left[f_0(x_1, ..., x_{n-1})\right] \end{aligned} \tag{4}$$

The inversion is possible here, because f_i depends on the values available both on the forward and on the backward passes and doesn't depend on x_i and y'_i. The complexity of the functions f_i is not limited here, but hereinafter we will consider linear transforms only.

When $abs(k_i) > 1$ the integer transform becomes not perfectly reversible, because for some values of y'_i it will produce non-integer x_i, but this fact makes no problem for our purposes of being able to restore the values represented by the results of the forward transform.

The error of approximation $\overline{dy} = \overline{y'} - \overline{y}$ results from the combination of the rounding error of f_i and the errors induced by replacement of y_i by y'_i in the arguments of f_i.

3 Decomposition of Linear Transform into Single Coordinate Shift Matrices

The authors of [4] introduce the concepts of *elementary reversible matrices* (ERM), which can be computed in the reversible manner. They consider *triangular elementary reversible matrices* (TERM) and *single-row elementary reversible matrices* (ERM).

The SERM is different from identity matrix I in the non-diagonal cells of a single row, it can be considered as a linear shift along the corresponding to the row coordinate. They prove that the matrix A of the size $n \times n$ with $\det A = \pm 1$ can be factorized into the product of a row permutation matrix P and $n + 1$ SERM matrices. The proof uses TERM factorization as an intermediate step and it is hard to use the proof as a guide for implementation of the SERM factorization algorithm. That's why we are going to give in this section our own proof of an analogous theorem, which is straightforward and can be easily used for implementation.

3.1 Straightforward Constructive Proof of the Possibility of Matrix Factorization into the Shift Matrices

Let us show constructively that for a matrix of the size $n \times n$ with the absolute value of determinant 1 one can construct a factorization into $n + 1$ matrices $B^{(i)}$ of shifts along the axes of the coordinate system with unit elements on the diagonal and a single non-zero row:

$$M = P_L A P_R = B^{(n)} * ... * B^{(1)} * B^{(0)}$$

where

$$B^{(0)} = \begin{bmatrix} 1 & & \\ & \dots & \\ & & 1 \\ b_{01} & \dots & b_{0,n-1}\ k \end{bmatrix}$$

$$B^{(1)} = \begin{bmatrix} 1 & b_{12} & \dots & b_{1n} \\ & 1 & & \\ & & \dots & \\ & & & 1 \end{bmatrix} \dots B^{(n)} = \begin{bmatrix} 1 & & \\ & \dots & \\ & & 1 \\ b_{n1} & \dots & b_{n,n-1}\ 1 \end{bmatrix}$$

here $k = \det P_L A P_R = \det M = \pm 1$.

Of the rows of the shift matrix $B^{(i)}$ only the i-th row (and the n-th for the matrix $B^{(0)}$) is different from the corresponding row of the identity matrix I. So, when using this kind of decomposition each step changes a single vector coordinate. Total the transform performs $n+1$ coordinate change, and it requires $n+1$ rounding operation to implement integer approximation of the transform. The inversion of the integer approximation is performed the same way, as it was considered before for the more general case in (4).

Theorem 2. *To each result of permutations of rows and columns $M = P_L A P_R$ may correspond a particular factorization into $n+1$ shift by coordinate matrices.*

Proof. Let's show how can we find the single coordinate shift decomposition of the matrix M. Consider the transform $y = Mx$. Note, that the space of matrices M is $n^2 - 1$ - dimensional (because the matrix belongs to the subspace of the n^2 - dimensional space of coefficients, which is defined by the equation $\det M = 1$ or $\det M = -1$). The space of b_{ij} is also of the size $(n+1)(n-1) = n^2 - 1$ (where $n+1$ is the number of equations (matrices) and $n-1$ is the number of coefficients to be found in the equations). So the dimensions of both spaces are equal for any n. Because $\det B^{(i)} = 1$ for $i > 0$ and $\det B^{(0)} = k = \det M$, we have $\det M = \det B^{(n)} * \dots * B^{(1)} * B^{(0)}$.

Let's denote

$$v = \sum_{j=1}^{n-1} b_{0j} x_j + k x_n$$

the n-th coordinate of $R^{(0)}x$.

Next we'll compute the coordinates of the resulting vector y.

$$y_1 = x_1 + \sum_{j=2}^{n-1} b_{1j} x_j + b_{1n} v = (1 + b_{1n} b_{01})\, x_1 + \sum_{j=2}^{n-1} (b_{1j} + b_{1n} b_{0j})\, x_j + b_{1n} k x_n \quad (5)$$

On the other hand

$$y_i = \sum_{j=1}^{n} m_{ij} x_j \quad (6)$$

By comparing the coefficients at x_j in the expressions for y_1 (5) and (6) we have:

$$b_{1n} k = m_{1n},\ 1 + b_{1n} b_{01} = m_{11},\ b_{1j} + b_{1n} b_{0j} = m_{1j} \text{ for } 2 \le j < n$$

From the first equation we can find b_{1n}, then from the second we can find b_{01}, as a result the other equations become linear in the yet unknown values of b_{0j}.

Next we'll consider the i-th step:

$$y_i = \sum_{j=1}^{i-1} b_{ij} y_j + x_i + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v$$
$$= \sum_{j=0}^{i-1} b_{ij} \sum_{l=1}^{n} m_{jl} x_l + k x_n + x_i + b_{in} \left(\sum_{j=1}^{n-1} b_{0j} x_j + k x_n \right) \quad (7)$$

Here we use the values of y_j represented by the coefficients of the matrix M (6). By grouping the members with x_j we get:

$$y_i = \sum_{j=1}^{i-1} \left(\sum_{l=1}^{i-1} b_{il} m_{lj} + b_{in} b_{0j} \right) x_j + \left(\sum_{l=1}^{i-1} b_{il} m_{li} + b_{in} b_{0i} + 1 \right) x_i$$
$$+ \sum_{j=i+1}^{n-1} \left(\sum_{l=1}^{i-1} b_{il} m_{lj} + b_{in} b_{0j} + b_{ij} \right) x_j + \left(\sum_{l=1}^{i-1} b_{il} m_{\ln} + b_{in} \right) x_n \quad (8)$$

Let us suppose that as a result of analysis of the expressions for y_j for $j < i$ we will know the values of b_{0j} for $j < i$, then using the first and the last members of the expression for y_i (8) we will have the system of i linear equations:

$$\begin{cases} \sum\limits_{l=1}^{i-1} b_{il} m_{lj} + b_{in} b_{0j} = m_{ij} & \text{for } 1 \le j < i \\ \sum\limits_{l=1}^{i-1} b_{il} m_{ln} + b_{in} = m_{in} \end{cases}$$

for i unknown variables: b_{il} for $1 \le l < i$ and b_{in}.

After solving the system using the second term in the expression for y_i (8) we obtain:

$$\sum_{l=1}^{i-1} b_{il} m_{li} + b_{in} b_{0i} + 1 = m_{ii}$$

Hence $b_{0i} = \left(m_{ii} - \sum\limits_{l=1}^{i-1} b_{il} m_{li} - 1 \right) / b_{in}$.

Thus we have extended to the step i the inductive hypothesis that it is possible to find b_{0j} for $j < i$ using the analysis of the expressions for y_j for $j < i$. Note, that the induction base is also valid, because using the analysis of the expression for y_1 we have already computed the value of b_{01}.

From the third term in the expression for y_i (8) we have:

$$\sum_{l=1}^{i-1} b_{il} m_{lj} + b_{in} b_{0j} + b_{ij} = m_{ij} \text{ for } i+1 \le j < n \quad (9)$$

After substitution to the equations (9) of the already computed values b_{il} for $1 \leq l < i$ and b_{in} we can find the unknown variables b_{ij} :

$$b_{ij} = m_{ij} - \sum_{l=1}^{i-1} b_{il} m_{lj} - b_{in} b_{0j} \text{ for } i+1 \leq j < n \tag{10}$$

And after computation of b_{0i} in (9) we can exchange i and j and also find still unknown values:

$$b_{ji} = m_{ji} - \sum_{l=1}^{j-1} b_{jl} m_{li} - b_{jn} b_{0i} \text{ for } 1 \leq j < i \tag{11}$$

Finally, on the n-th step we have:

$$y_n = \sum_{j=1}^{n-1} b_{nj} y_j + v = \sum_{j=0}^{n-1} b_{nj} \sum_{l=1}^{n} m_{jl} x_l + \left(\sum_{j=1}^{n-1} b_{0j} x_j + k x_n \right) \tag{12}$$

Similarly to the already considered general case, we can group the tems with x_j to obtain:

$$y_n = \sum_{j=1}^{n-1} \left(\sum_{l=1}^{n-1} b_{nl} m_{lj} + b_{0j} \right) x_j + \left(\sum_{l=1}^{n-1} b_{nl} m_{ln} + k \right) x_n \tag{13}$$

As a result we have the following system of n linear equations:

$$\begin{cases} \sum\limits_{l=1}^{n-1} b_{nl} m_{lj} + b_{0j} = m_{nj} & \text{for } 1 \leq j < n \\ \sum\limits_{l=1}^{n-1} b_{nl} m_{ln} + k = m_{nn} \end{cases} \tag{14}$$

for $n-1$ unknown variable b_{nl} , $1 \leq l < n$. I.e. one of the equations (14) will be redundant. The reason for the redundancy is that the condition $\det M = \pm 1 = k$ is automatically satisfied due to the structure of the matrices involved in the decomposition.

Since the redundancy can consist in the fact that all the coefficients in the equation with unknowns b_{nl}, will be equal to 0 or that one of the equations will be expressed through others, we should solve the set of equations (14) by the version of Gauss algorithm for overdetermined sets of equations.

Some of the computations involved in the proof include division by values, which may be zero for some M. And solving of the systems of linear equations here may also fail. That's why we say "*may correspond*" in the theorem statement. But, once we have found a solution by the algorithm, we can be sure, that it is the only solution of the problem of finding decomposition for the matrix M. □

Later on we will consider the process of finding the optimal decomposition of the matrix A by looking through all the combinations of row and column permutations P_L and P_R. So, when the process of finding the decomposition for a particular matrix M will fail, it will be just required to consider another version of M.

4 Finding the Best Computation Order

Let's recall, that $M = P_L A P_R$, i.e. it is a result of permutation of rows and columns of the matrix A. Thus, it is required to find the best order of computation of the coefficients b_{ij} . We will minimize the mean square deviation of the results of the integer approximation from the results of the original continuous transformation. For the small dimensions n we can afford to use the exhaustive search of all the $(n!)^2$ variants of permutations. The exhaustive search will simply skip the variants, which cause division by zero during computation of the matrix M decomposition.

To select the best of the variants we should be able to estimate and compare the errors, caused by the variants of approximations. Let us consider a method for estimating the mean square difference of the results of the integer approximation of the transform from the results of the original continuous transform.

4.1 Mean Square Error of the Rounding Operation for Linear Expressions with Real Coefficients

Let's denote by v' the integer version of v.

Hereinafter, the expression $[x]$ denotes the operation of rounding x, and

$$\{x\} = [x] - x \tag{15}$$

denotes the difference between the result of rounding and the original value.

When rounding a linear expression on integer variables with rational coefficients the size of the set of possible results of rounding depends on the least common multiple (LCM) of the denominators of the coefficients. Subsequently we'll call *real* rational coefficients with large denominators. And the term *rational* will be used for the rational coefficients with small denominators.

Assuming that rounding of an expression x having some real coefficients is performed to the nearest integer for large ranges of x, we can consider the value $\{x\}$ as a random variable, uniformly distributed over the interval $[-0.5, 0.5)$. Then

$$\overline{\{x\}^2} = \int_{-0.5}^{0.5} x^2 dx = \frac{1}{12} \tag{16}$$

When we deal with the matrices A produced from some real-world data, say, as a result of Karhunen-Loeve Transform (KLT), the estimate (16) will always be in effect, because it is highly improbable to get rational elements here.

4.2 Estimation of integer approximation errors

When some expression uses several rounded values, we can consider the rounding errors of the values as independent random variables when the rounded values differ from each other.

Once U and V - are independent random variables, and $W = aU + bV$, then

$$\overline{W^2} = \overline{(aU + bV)^2} = a^2\overline{U^2} + b^2\overline{V^2}$$

But if $U = V$ then the rounding error will be higher:

$$\overline{W^2} = \overline{(aV + bV)^2} = (a+b)^2\,\overline{V^2}$$

Thus, to calculate the mean square rounding errors of linear expressions it is required to sum the squares of the coefficients at the sub-expression errors δ_i multiplied by $\overline{\delta_i^2}$.

Let's denote

$$dv = v' - v = \left\{ \sum_{j=1}^{n-1} b_{0j} x_j \right\} = \delta_0$$

$$y'_i = x_i + \left[\sum_{j=1}^{i-1} b_{ij} y'_j + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v' \right] \tag{17}$$

$$y_i = x_i + \sum_{j=1}^{i-1} b_{ij} y_j + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v =$$

$$x_i + \sum_{j=1}^{i-1} b_{ij} y'_j + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v' - \sum_{j=1}^{i-1} b_{ij} dy_j - b_{in} dv$$

$$dy_i = y'_i - y_i = \left\{ \sum_{j=1}^{i-1} b_{ij} y'_j + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v' \right\} + \sum_{j=1}^{i-1} b_{ij} dy_j + b_{in} dv \tag{18}$$

Let's call

$$\delta_i = \left\{ \sum_{j=1}^{i-1} b_{ij} y'_j + \sum_{j=i+1}^{n-1} b_{ij} x_j + b_{in} v' \right\} \tag{19}$$

From (18) we can see that dy_i depends on dy_j for j from 0 to $i-1$, and also on δ_i and δ_0 . We can substitute into (18) for dy_j the corresponding expressions (18) to finally get:

$$dy_i = \delta_i + \sum_{j=0}^{i-1} D_{ij} \delta_j \tag{20}$$

Here D_{ij} are the coefficients to be found.

Let's find the coefficients D_{ij} by substituting into (18) the expressions for dy_j from (20)

$$dy_i = \delta_i + \sum_{j=1}^{i-1} b_{ij} \left(\delta_j + \sum_{l=0}^{j-1} D_{jl}\delta_l \right) + b_{in}\delta_0$$

$$= \delta_i + \left(\sum_{j=1}^{i-1} b_{ij}D_{j0} + b_{in} \right) \delta_0 + \sum_{j=1}^{i-1} b_{ij}\delta_j + \sum_{l=1}^{i-2} \delta_l \sum_{j=l+1}^{i-1} b_{ij}D_{jl}$$

$$dy_i = \delta_i + \left(\sum_{j=1}^{i-1} b_{ij}D_{j0} + b_{in} \right) \delta_0 + b_{i,i-1}\delta_{i-1} + \sum_{j=1}^{i-2} b_{ij}\delta_j + \sum_{j=1}^{i-2} \delta_j \sum_{l=j+1}^{i-1} b_{il}D_{lj}$$

As a result we have:

$$dy_i = \delta_i + \left(\sum_{j=1}^{i-1} b_{ij}D_{j0} + b_{in} \right) \delta_0 + b_{i,i-1}\delta_{i-1} + \sum_{j=1}^{i-2} \left(b_{ij} + \sum_{l=j+1}^{i-1} b_{il}D_{lj} \right) \delta_j \quad (21)$$

The formula (21) describes the process of computation of D_{ij} using the values already computed for j from 0 to $i-1$.

It is also required to consider separately the initial value of i:

$$dy_1 = \delta_1 + b_{1n}\delta_0$$

When computing the average square of the expressions (21) we will assume that $\overline{\delta_i^2} = \frac{1}{12}$ because we suppose here, that the coefficients b_{ij} are real.

5 Tests

Let's consider the results of the algorithm usage for random rotation matrices. Table 1 shows some values, computed by the algorithms for a 3×3 matrix. The columns of the table correspond to the vector components. The rows of the table are:

- NErr - the number of differences of the coordinates of source vectors from the coordinates of the results of the forward transformation of the vectors by the obtained reversible integer approximation and back by its inverse detected in the loop over an area in the space of the vectors;
- sqD - root mean square error of the results of continuous transform from the results of its reversible integer approximation computed in the loop over the area in the space of the vectors;
- sqDNet - root mean square of deviation of the coordinates of the results of continuous transform from their rounded values (it allows us to estimate the lower bound for sqD);

- DCalc - root mean square error of the results of continuous transform from the results of its reversible integer approximation estimated by our algorithm;
- Top# - the permutation of columns, which gives the minimum of DCalc;
- Left# - the permutation of columns, which gives the minimum of DCalc;
- A - the transform matrix;

Table 1. Test results for a random 3×3 matrix of rotations

	1	2	3
NErr	0	0	0
sqD	0.381290171321118	0.337411519990808	0.350700141728564
sqDNet	0.288674515065362	0.288675136393005	0.288675146079434
DCalc	0.381289822912763	0.337400208005412	0.404471841500362
Top#	3	1	2
Left#	1	2	3
A1	0.501140304449431	0.0653853697257877	-0.862892315808962
A2	0.671762008580112	0.59919861058942	0.4355419944117
A3	0.545521951056733	-0.797925922936165	0.25635916683771

For the matrix 3×3 from Table 1 we have the root mean square error

$$Error = 0.650244800045056$$

and the following reversible integer approximation

$$\begin{aligned} v &= x_2 + [0.578125087465738 * x_3 - 0.53671762651506 * x_1] \\ y_1 &= x_3 + [-0.397744145953282 * x_1 - 0.862892315808962 * v] \\ y_2 &= x_1 + [0.419964254965846 * y_1 + 0.797925922936165 * v] \\ y_3 &= v + [0.608264013914563 * y_1 - 0.502304008291562 * y_2] \end{aligned}$$

For the matrix 5×5 from Table 2 we have the root mean square error

$$Error = 0.768078871487727$$

and the following reversible integer approximation

$$\begin{aligned} v &= x_4 + [0.2739172 * x_3 - 0.8386267 * x_2 - 0.00005832195 * x_1 + 0.1419508 * x_5] \\ y_4 &= x_3 + [-0.5215370 * x_2 - 0.006188384 * x_1 - 0.04225347 * x_5 - 0.3529485 * v] \\ y_1 &= x_2 + [0.1916207 * y_4 - 0.002794371 * x_1 + 0.28758131 * x_5 + 0.8573734 * v] \\ y_5 &= x_1 + [0.006326542 * y_4 + 0.01397231 * y_1 - 0.005955142 * x_5 - 0.01004929 * v] \\ y_3 &= x_5 + [0.05669100 * y_4 - 0.774736 * y_1 + 0.006511915 * y_5 + 0.5056881 * v] \\ y_2 &= v + [-0.07052526 * y_4 - 0.7576750 * y_1 + 0.007944399 * y_5 - 0.4683665 * y_3] \end{aligned}$$

Table 2. Test results for a random 5×5 matrix of rotations

	1	2	3	4	5
NErr	0	0	0	0	0
sqD	0.30612224853	0.37192683870	0.22420601944	0.36796009227	0.36985305466
sqDNet	0.28867676698	0.28867526034	0.22420595547	0.28867576619	0.28866127050
DCalc	0.30612805880	0.37197777795	0.28871523643	0.36792735991	0.37300974146
Top#	3	2	1	5	4
Left#	4	1	5	3	2
A1	0.90332131205	-0.2255448942	-0.0061677990092	-0.092354780375	-0.35294852408
A2	0.40794441137	0.23776469377	-0.0040262521598	0.39158903935	0.78974117353
A3	0.008662154017	0.010322802042	0.99990530920	-0.0024945274352	-0.001247718504
A4	-0.12626603596	-0.62100756867	0.0092514304866	0.76315266428	-0.1261701622
A5	-0.039671395903	-0.7119273070	0.0070378409831	-0.50568805317	0.48560864030

Table 3. Test results for a random 7×7 matrix of rotations

	1	2	3	4	5	6	7
NErr	0	0	0	0	0	0	0
sqD	0.3846963	0.3202919	0.3138482	0.3830634	0.394112	0.312817	0.4199523
sqDNet	0.2886750	0.2886750	0.2886689	0.2886647	0.2886753	0.2884511	0.2886757
DCalc	0.3846966	0.3202547	0.3138602	0.3830685	0.3941317	0.31305	0.5059632
Top#	7	3	5	4	2	6	1
Left#	7	6	4	3	2	1	5
M1	0.1839351	-0.3193916	0.1334853	0.113963	-0.2213811	-0.09188760	0.8808507
M2	-0.3417553	0.6397698	-0.1804105	-0.1702700	0.3857258	0.2043536	0.4709703
M3	-0.02518445	0.4264799	0.8528390	-0.08033195	-0.2374124	-0.1614647	-0.03546056
M4	-0.8741814	-0.439128	0.2010590	-0.01850904	0.04347172	-0.01722685	0.004371392
M5	0.2798122	-0.3364512	0.3273608	-0.566333	0.5675667	0.241562	0.01108168
M6	-0.02110229	-0.0127339	0.04954787	-0.03171776	-0.4384360	0.8962660	-0.02031064
M7	-0.07618058	0.02455606	-0.26880	-0.793401	-0.4791590	-0.2485626	0.02184048

For the matrix 7×7 from Table 3 we have the root mean square error

$$Error = 1.0025704864004$$

and the following reversible integer approximation

$$v = x_1 + [-0.92645 * x_7 - 0.85131 * x_3 + 0.39344 * x_5 - 1.1726 * x_4 - 0.33471 * x_2 + 0.22740 * x_6]$$
$$y_7 = x_7 + [0.43048 * x_3 - 0.21308 * x_5 + 1.1468 * x_4 + 0.073452 * x_2 - 0.29219 * x_6 + 0.88085 * v]$$
$$y_6 = x_3 + [0.094575 * y_7 - 0.34555 * x_5 + 0.27354 * x_4 + 0.53641 * x_2 + 0.12488 * x_6 + 0.38766 * v]$$
$$y_4 = x_5 + [-0.097879 * y_7 + 0.42127 * y_6 - 0.17059 * x_4 - 0.47099 * x_2 - 0.22297 * x_6 - 0.14765 * v]$$
$$y_3 = x_4 + [-0.86507 * y_7 - 0.057838 * y_6 - 0.0070872 * y_4 + 0.13813 * x_2 - 0.26645 * x_6 + 0.79336 * v]$$
$$y_2 = x_2 + [-0.27060 * y_7 - 0.59006 * y_6 + 0.22352 * y_4 - 0.72340 * y_3 + 0.23846 * x_6 + 0.53843 * v]$$
$$y_1 = x_6 + [-0.14534 * y_7 - 0.27618 * y_6 + 0.13728 * y_4 - 0.29855 * y_3 - 0.41459 * y_2 + 0.24856 * v]$$
$$y_5 = v + [-0.95442 * y_7 - 0.28899 * y_6 - 0.097297 * y_4 - 1.2704 * y_3 - 0.69614 * y_2 - 0.41733 * y_1]$$

The examples considered demonstrate, that the root mean square error grows steadily with the increase of the matrix size. The largest error we usually have for the vector component, which is computed last. And even for the matrix 7×7 it was slightly over 0.5, which is quite acceptable for most use cases.

6 Conclusion

In contrast to the previous works on the reversible integer transforms we give a straightforward proof of the possibility to construct a reversible integer approximation of a linear transform, represented by matrix with unit determinant.

We have developed a method to estimate the mean square of rounding errors for the transform matrices with real coefficients and using the estimates we perform a brute-force search of the best computation order, which minimizes the error estimation.

For the matrices of rational coefficients with small denominators it is still required to rectify the error estimates, but a thorough examination of the related questions would substantially increase the size of this article beyond the prescribed limits.

The algorithms suggested in the article are detailed enough to be easily implemented in software. And the tests performed show, that they allow us to compute high quality reversible approximations of linear transforms.

We have a successful experience of using the reversible integer transforms in conjunction with a specialized compression algorithm [6] for compression of multichannel images.

References

1. Dony, R. D.: Karhunen-Loève Transform. In: The Transform and Data Compression Handbook Ed. K. R. Rao and P.C. Yip. Boca Raton, CRC Press LLC, 2001

2. Pengwei Hao and Qingyun Shi, Invertible Linear Transforms Implemented by Integer Mapping (Published in Chinese, Science in China, Series E, April 2000, V30, N2, pp132-141)
3. Pengwei Hao and Qingyun Shi, "Comparative Study of Color Transforms for Image Coding and Derivation of Integer Reversible Color Transform", 15th International Conference on Pattern Recognition (ICPR), 2000, Vol. 3, pp. 224-227, Sept 3-8, 2000, Barcelona, Spain.
4. Pengwei Hao and Qingyun Shi, "Matrix factorizations for reversible integer mapping," in IEEE Transactions on Signal Processing, vol. 49, no. 10, pp. 2314-2324, Oct. 2001, doi: 10.1109/78.950787.
5. P. Hao and Q. Shi, "Reversible integer KLT for progressive-to-lossless compression of multiple component images," Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429), Barcelona, Spain, 2003, pp. I-633, doi: 10.1109/ICIP.2003.1247041.
6. Hmelnov, A.E. A lossless compression algorithm for integer differences sequences by optimization of their division into intervals of constant bit depth values // Computational technologies. 2015. V. 20. № 3. P. 75-98 (in Russian)