MLAMBDA: A modified LAMBDA method for integer least-squares estimation

X.-W. Chang, X. Yang, T. Zhou
School of Computer Science, McGill University,
3480 University Street, Montreal, Quebec, Canada H3A 2A7
Tel: +1-514-398-8259, Fax: +1-514-398-3883
Email: chang@cs.mcgill.ca, xiaohua.yang@mcgill.ca, tianyang.zhou@mcgill.ca
Corresponding author: X.-W. Chang

Abstract

The LAMBDA method has been widely used in GNSS for fixing integer ambiguities. It can also solve any integer least squares (ILS) problem arising from other applications. For real time applications with high dimensions, the computational speed is crucial. A modified LAMBDA method (MLAMBDA) is presented. Several strategies are proposed to reduce the computational complexity of the LAMBDA method. Numerical simulations show that MLAMBDA is (much) faster than LAMBDA. The relations between the LAMBDA method and some relevant methods in the information theory literature are pointed out when we introduce its main procedures.

Keywords: Integer least squares estimation, the Lambda method, Decorrelation, Reduction, Search, Computational efficiency.

1 Introduction

A key computational component in high precision relative GNSS positioning is to resolve double differenced carrier phase ambiguities as integers. There are many methods of ambiguity resolution in the GNSS literature. Among them is the well-known LAMBDA (Least-squares AMBiguity Decorrelation Adjustment) method presented by Teunissen, see, e.g., Teunissen (1993, 1995a,b, 1998, 1999). A detailed description of the algorithms and implementation is given by De Jonge and Tiberius (1996). Its software (Fortran version and MATLAB version) is available from Delft University of Technology. Frequently asked questions and misunderstanding about the LAMBDA method are addressed by Joosten and Tiberius (2002).

The LAMBDA method solves an integer least squares (ILS) problem to obtain the estimates of the double differenced integer ambiguities. Note that ILS problems may also arise from other applications, such as communications, cryptography and lattice design et al, see, e.g., Agrell et al. (2002). It is interesting that the LAMBDA method and some ILS methods developed for those applications have some similarities, which will be pointed out in this paper.

Like many other ILS methods, the LAMBDA method consists of two stages. Its first stage is to transfer the original ILS problem to a new one by means of the so-called Z-transformations or unimodular transformations. Since this stage decorrelates the ambiguities in the GNSS context, it is called "decorrelation" in the GNSS literature. However, it actually does more than decorrelation (see Sect. 2.1), thus we prefer to call this stage "reduction", as the literature in information theory does, see, e.g., Agrell et al. (2002). Its second stage is to search the optimal estimate or a few optimal estimates of the parameter vector over a hyper-ellipsoidal region, and so is called the "search" stage. Different techniques have been proposed for the decorrelation of the integer ambiguities. For example, Hassibi and Boyed (1998) and Grafarend (2000) suggested to use the LLL reduction algorithm developed by Lenstra et al. (1982), which has actually been used in information theory for solving an ILS problem, see, e.g., Agrell et al. (2002). Liu et al. (1999) and Xu (2001) proposed the so-called united ambiguity decorrelation method and inverse integer Cholesky decorrelation method, respectively. Liu et al. (1999), Xu (2001), and Lou and Grafarend (2003) showed how these different decorrelation methods reduce the condition number of the covariance matrix involved in the ILS estimation problem. For different search techniques within the context of GPS ambiguity fixing, see also Frei and Beutler (1989) and Landau and Euler (1992) et al.

For real time kinematic GNSS applications and other applications with high dimensions, computational speed is crucial. In this paper, we present a modified LAMBDA method (MLAMBDA), which can significantly reduce the computational complexity of the LAMBDA method for high dimensional ILS problems. The new method improves the computational efficiency of both of the reduction stage and the search stage. Numerical results show that the MLAMBDA reduction is more computationally efficient than the LLL reduction too.

The rest of this paper is organized as follows. In Sect. 2, we introduce the LAMBDA method. In Sect. 3, we show how to effectively reduce the computational cost of the LAMBDA method by several strategies: symmetric pivoting, greedy selection, lazy transformation, and shrinking. And we present the new algorithm MLAMBDA in detail. In Sect. 4, we give numerical simulation results. Finally we give a summary in Sect. 5.

We now describe the notation to be used in this paper. The sets of all real and integer $m \times n$ matrices are denoted by $\mathbb{R}^{m\times n}$ and $\mathbb{Z}^{m\times n}$, respectively, and the set of real and integer n-vectors are denoted by $\mathbb{R}^n$ and $\mathbb{Z}^n$, respectively. The identity matrix is denoted by $\boldsymbol{I}$ and its ith column is denoted by $\boldsymbol{e}_i$. MATLAB notation is used to denote a submatrix. Specifically, if $\boldsymbol{A} = (a_{ij}) \in \mathbb{R}^{m\times n}$, then $\boldsymbol{A}(i,:)$ denotes the ith row, $\boldsymbol{A}(:,j)$ the jth column, and $\boldsymbol{A}(i_1\!:\!i_2, j_1\!:\!j_2)$

the submatrix formed by rows i_1 to i_2 and columns j_1 to j_2. For the (i,j) element of $\boldsymbol{A}$, we denote it by a_{ij} or $\boldsymbol{A}(i,j)$. For a scalar $z \in \mathbb{R}$, we use $\lfloor z \rceil$ to denote its nearest integer. If there is a tie, $\lfloor z \rceil$ denotes the one with smaller magnitude.

2 The LAMBDA method

Suppose $\hat{\boldsymbol{a}} \in \mathbb{R}^n$ is the real-valued least squares (LS) estimate of the integer parameter vector $\boldsymbol{a} \in \mathbb{Z}^n$ (i.e., the double differenced integer ambiguity vector in the GNSS context) and $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} \in \mathbb{R}^{n\times n}$ is its variance-covariance matrix, which is symmetric positive definite. The ILS estimate $\check{\boldsymbol{a}}$ is the solution of the minimization problem:

$$\min_{\boldsymbol{a}\in\mathbb{Z}^n} (\boldsymbol{a}-\hat{\boldsymbol{a}})^T \boldsymbol{Q}_{\hat{\boldsymbol{a}}}^{-1} (\boldsymbol{a}-\hat{\boldsymbol{a}}). \tag{1}$$

Although (1) is in the form of a quadratic optimization problem, it is easy to show that it can be rewritten in the form of a LS problem. So we refer to (1) as an ILS problem. The ILS problem has been proved to be an NP-hard problem (van Emde Boas, 1981). Thus all known algorithms for solving the problem have exponential complexity.

For the validation purpose, in addition to the optimal estimate $\check{\boldsymbol{a}}$, one often also requires the second optimal estimate, which gives the second smallest value of the objective function in (1). The LAMBDA package developed by Delft University of Technology gives an option to find a number of optimal estimates.

In the following we will introduce the two stages of the LAMBDA method: reduction and search. See De Jonge and Tiberius (1996) for more details. We will also point out the similarities between the ideas of the LAMBDA method and the ideas of some typical ILS methods developed for other applications in the literature.

2.1 Reduction process

The reduction process is to change the original ILS problem (1) to a new one by the so-called Z-transformations or unimodular transformations. Its purpose is to make the search process more efficient.

Let $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$ be unimodular, i.e., $|\det(\boldsymbol{Z})| = 1$. Obviously $\boldsymbol{Z}^{-1}$ is also an integer matrix. Define the following Z-transformations:

$$\boldsymbol{z} = \boldsymbol{Z}^T \boldsymbol{a}, \qquad \hat{\boldsymbol{z}} = \boldsymbol{Z}^T \hat{\boldsymbol{a}}, \qquad \boldsymbol{Q}_{\hat{\boldsymbol{z}}} = \boldsymbol{Z}^T \boldsymbol{Q}_{\hat{\boldsymbol{a}}} \boldsymbol{Z}. \tag{2}$$

Notice that if $\boldsymbol{a}$ is an integer vector, then $\boldsymbol{z}$ is too, and vice versa. Then by the above transformations, the ILS problem (1) is transformed to the new ILS problem

$$\min_{\boldsymbol{z}\in\boldsymbol{Z}^n} (\boldsymbol{z}-\hat{\boldsymbol{z}})^T \boldsymbol{Q}_{\hat{\boldsymbol{z}}}^{-1} (\boldsymbol{z}-\hat{\boldsymbol{z}}). \tag{3}$$

Let the $\mathrm{L^TDL}$ factorizations of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ and $\boldsymbol{Q}_{\hat{\boldsymbol{z}}}$, respectively, be

$$\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{L}^T \boldsymbol{D} \boldsymbol{L}, \qquad \boldsymbol{Q}_{\hat{\boldsymbol{z}}} = \boldsymbol{Z}^T \boldsymbol{L}^T \boldsymbol{D} \boldsymbol{L} \boldsymbol{Z} = \bar{\boldsymbol{L}}^T \bar{\boldsymbol{D}} \bar{\boldsymbol{L}}, \tag{4}$$

where $\boldsymbol{L}$ and $\bar{\boldsymbol{L}}$ are unit lower triangular, and $\boldsymbol{D} = \mathrm{diag}(d_1, \ldots, d_n)$ and $\bar{\boldsymbol{D}} = \mathrm{diag}(\bar{d}_1, \ldots, \bar{d}_n)$ with $d_i, \bar{d}_i > 0$. These factors have statistical meaning. For example, d_i is the conditional variance of $\hat{a}_i$ when $a_{i+1}, \ldots, a_n$ are fixed. The reduction process starts with the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ and updates the factors to give the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{z}}}$. In this process one tries

to find a unimodular matrix $\boldsymbol{Z}$ to reach two goals, which are crucial for the efficiency of the search process: (i) $\boldsymbol{Q}_{\hat{\boldsymbol{z}}}$ is as diagonal as possible (i.e., the off-diagonal entries of $\bar{\boldsymbol{L}}$ are as small as possible); (ii) The diagonal entries of $\bar{\boldsymbol{D}}$ are distributed in decreasing order if possible, i.e., one strives for

$$\bar{d}_1 \geq \bar{d}_2 \geq \cdots \geq \bar{d}_n. \tag{5}$$

Note that the first goal is to decorrelate the unknown parameters, and it is part of the reduction stage.

Here we make a remark. The LLL algorithm also pursues the above two goals. In fact, the LAMBDA reduction algorithm is based on the ideas from Lenstra (1981), which was modified and led to the LLL algorithm (Hassibi and Boyed, 1998). The approaches given in Liu et al. (1999) and Xu (2001) mainly pursue the first goal and the second goal is only partially achieved.

In the LAMBDA method, the unimodular matrix $\boldsymbol{Z}$ in (2) is constructed by a sequence of integer Gauss transformations and permutations. The integer Gauss transformations are used to make the off-diagonal entries of $\bar{\boldsymbol{L}}$ as small as possible, while permutations are used to strive for (5).

2.1.1 Integer Gauss transformations

An integer Gauss transformation $\boldsymbol{Z}_{ij}$ has the following form:

$$\boldsymbol{Z}_{ij} = \boldsymbol{I} - \mu \boldsymbol{e}_i \boldsymbol{e}_j^T, \qquad \mu \text{ is an integer.}$$

It is easy to show that $\boldsymbol{Z}_{ij}^{-1} = \boldsymbol{I} + \mu \boldsymbol{e}_i \boldsymbol{e}_j^T$. Applying $\boldsymbol{Z}_{ij}$ $(i > j)$ to $\boldsymbol{L}$ from the right gives

$$\bar{\boldsymbol{L}} = \boldsymbol{L}\boldsymbol{Z}_{ij} = \boldsymbol{L} - \mu \boldsymbol{L}\boldsymbol{e}_i \boldsymbol{e}_j^T.$$

Thus $\bar{\boldsymbol{L}}$ is the same as $\boldsymbol{L}$, except that

$$\bar{l}_{kj} = l_{kj} - \mu l_{ki}, \quad k = i, \ldots, n.$$

To make $\bar{l}_{ij}$ as small as possible, one takes $\mu = \lfloor l_{ij} \rceil$, ensuring

$$|\bar{l}_{ij}| \leq 1/2, \quad i > j. \tag{6}$$

When $\boldsymbol{Z}_{ij}$ is applied to $\boldsymbol{L}$ from the right, $\boldsymbol{Z}_{ij}^T$ should be applied to $\hat{\boldsymbol{a}}$ from the left simultaneously (cf. Eqn. 2). All transformations also need to be accumulated.

The following algorithm gives the process of applying the integer Gaussian transformation $\boldsymbol{Z}_{ij}$ to $\boldsymbol{L}$.

Algorithm 2.1 (Integer Gauss Transformations). Given a unit lower triangular $\boldsymbol{L} \in \mathbb{R}^{n\times n}$, index pair (i, j), $\hat{\boldsymbol{a}} \in \mathbb{R}^n$ and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$. This algorithm applies the integer Gauss transformation $\boldsymbol{Z}_{ij}$ to $\boldsymbol{L}$ such that $|(\boldsymbol{L}\boldsymbol{Z})(i, j)| \leq 1/2$, then computes $\boldsymbol{Z}_{ij}^T\hat{\boldsymbol{a}}$ and $\boldsymbol{Z}\boldsymbol{Z}_{ij}$, which overwrite $\hat{\boldsymbol{a}}$ and $\boldsymbol{Z}$, respectively.

function: $[\boldsymbol{L}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \text{GAUSS}(\boldsymbol{L}, i, j, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
$\mu = \lfloor \boldsymbol{L}(i, j) \rceil$
if $\mu \neq 0$
  $\boldsymbol{L}(i : n, j) = \boldsymbol{L}(i : n, j) - \mu \boldsymbol{L}(i : n, i)$
  $\boldsymbol{Z}(1 : n, j) = \boldsymbol{Z}(1 : n, j) - \mu \boldsymbol{Z}(1 : n, i)$
  $\hat{\boldsymbol{a}}(j) = \hat{\boldsymbol{a}}(j) - \mu \hat{\boldsymbol{a}}(i)$
end

2.1.2 Permutations

In order to strive for the order (5), symmetric permutations of the covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ are needed in the reduction process. When two diagonal elements of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ are interchanged, the factors $\boldsymbol{L}$ and $\boldsymbol{D}$ of its $\mathrm{L^TDL}$ factorization have to be updated.

Suppose we partition the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ as follows

$$\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{11}^T & \boldsymbol{L}_{21}^T & \boldsymbol{L}_{31}^T \\ & \boldsymbol{L}_{22}^T & \boldsymbol{L}_{32}^T \\ & & \boldsymbol{L}_{33}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_1 & & \\ & \boldsymbol{D}_2 & \\ & & \boldsymbol{D}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{11} & & \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} & \\ \boldsymbol{L}_{31} & \boldsymbol{L}_{32} & \boldsymbol{L}_{33} \\ k-1 & 2 & n-k-1 \end{bmatrix} \begin{matrix} k-1 \\ 2 \\ n-k-1 \end{matrix}.$$

Let

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \boldsymbol{P}_{k,k+1} = \begin{bmatrix} \boldsymbol{I}_{k-1} & & \\ & \boldsymbol{P} & \\ & & \boldsymbol{I}_{n-k-1} \end{bmatrix}.$$

It can be shown that $\boldsymbol{P}_{k,k+1}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{P}_{k,k+1}$ has the $\mathrm{L^TDL}$ factorization

$$\boldsymbol{P}_{k,k+1}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{P}_{k,k+1} = \begin{bmatrix} \boldsymbol{L}_{11}^T & \bar{\boldsymbol{L}}_{21}^T & \boldsymbol{L}_{31}^T \\ & \bar{\boldsymbol{L}}_{22}^T & \bar{\boldsymbol{L}}_{32}^T \\ & & \boldsymbol{L}_{33}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_1 & & \\ & \bar{\boldsymbol{D}}_2 & \\ & & \boldsymbol{D}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{11} & & \\ \bar{\boldsymbol{L}}_{21} & \bar{\boldsymbol{L}}_{22} & \\ \boldsymbol{L}_{31} & \bar{\boldsymbol{L}}_{32} & \boldsymbol{L}_{33} \end{bmatrix}, \tag{7}$$

where

$$\bar{\boldsymbol{D}}_2 = \begin{bmatrix} \bar{d}_k & \\ & \bar{d}_{k+1} \end{bmatrix}, \quad \bar{d}_{k+1} = d_k + l_{k+1,k}^2 d_{k+1}, \quad \bar{d}_k = \frac{d_k}{\bar{d}_{k+1}} d_{k+1}, \tag{8}$$

$$\bar{\boldsymbol{L}}_{22} \equiv \begin{bmatrix} 1 & \\ \bar{l}_{k+1,k} & 1 \end{bmatrix}, \quad \bar{l}_{k+1,k} = \frac{d_{k+1} l_{k+1,k}}{\bar{d}_{k+1}}, \tag{9}$$

$$\bar{\boldsymbol{L}}_{21} = \begin{bmatrix} -l_{k+1,k} & 1 \\ \frac{d_k}{\bar{d}_{k+1}} & \bar{l}_{k+1,k} \end{bmatrix} \boldsymbol{L}_{21} = \begin{bmatrix} -l_{k+1,k} & 1 \\ \frac{d_k}{\bar{d}_{k+1}} & \bar{l}_{k+1,k} \end{bmatrix} \boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1), \tag{10}$$

$$\bar{\boldsymbol{L}}_{32} = \boldsymbol{L}_{32}\boldsymbol{P} = \begin{bmatrix} \boldsymbol{L}(k+2\!:\!n, k+1) & \boldsymbol{L}(k+2\!:\!n, 1\!:\!k) \end{bmatrix}. \tag{11}$$

If we have

$$\bar{d}_{k+1} < d_{k+1} \tag{12}$$

(this implies that $d_k < d_{k+1}$), the permutation is performed. This does not guarantee that $\bar{d}_k \geq \bar{d}_{k+1}$, but it at least makes the gap between $\bar{d}_k$ and $\bar{d}_{k+1}$ smaller than that between d_k and d_{k+1}. Note that if the absolute values of the elements below l_{kk} and $l_{k+1,k+1}$ in $\boldsymbol{L}$ are bounded above by 1/2, the bounds still hold after the permutation, except that $\bar{l}_{k+1,k}$ (see Eqn. 9) may not be bounded by 1/2 any more.

Here we would like to point out an important property of the LAMBDA reduction. After the reduction process is finished, the inequality (12) will not hold for any k (otherwise a new permutation would be performed), thus for the $\bar{\boldsymbol{L}}$ and $\bar{\boldsymbol{D}}$ obtained at the end of the reduction process, we must have (cf. Eqn. 8)

$$\bar{d}_k + \bar{l}_{k+1,k}^2 \bar{d}_{k+1} \geq \bar{d}_{k+1}, \quad k = 1, 2, \ldots, n-1,$$

or

$$\bar{d}_k \geq (1 - \bar{l}_{k+1,k}^2)\bar{d}_{k+1}, \quad k = 1, 2, \ldots, n-1. \tag{13}$$

This is the order the LAMBDA reduction can guarantee, although it strives for the stronger order in Eqn. (5). It can easily be shown that Eqs. (6) and (13) are equivalent to the so-called LLL reduction criteria, which are satisfied by the $\bar{\boldsymbol{L}}$ and $\bar{\boldsymbol{D}}$ obtained by the LLL reduction algorithm, see Lenstra et al. (1982) and Agrell et al. (2002).

Now we write the above operations as an algorithm.

Algorithm 2.2 (Permutations). Given the $\boldsymbol{L}$ and $\boldsymbol{D}$ factors of the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} \in \mathbb{R}^{n\times n}$, index k, scalar δ which is equal to $\bar{d}_{k+1}$ in Eqn. (8), $\hat{\boldsymbol{a}} \in \mathbb{R}^n$, and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$. This algorithm computes the updated $\boldsymbol{L}$ and $\boldsymbol{D}$ factors in (7) after $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$'s kth row and $(k+1)$th row, and kth column and $(k+1)$th column are interchanged, respectively. It also interchanges $\hat{\boldsymbol{a}}$'s kth entry and $(k+1)$th entry and $\boldsymbol{Z}$'s kth column and $(k+1)$th column.

function: $[\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \mathrm{PERMUTE}(\boldsymbol{L}, \boldsymbol{D}, k, \delta, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
$\eta = \boldsymbol{D}(k,k)/\delta$ // see Eqn. (8)
$\lambda = \boldsymbol{D}(k+1,k+1)\boldsymbol{L}(k+1,k)/\delta$ // see Eqn. (9)
$\boldsymbol{D}(k,k) = \eta \boldsymbol{D}(k+1,k+1)$ // see Eqn. (8)
$\boldsymbol{D}(k+1,k+1) = \delta$
$\boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1) = \begin{bmatrix} -\boldsymbol{L}(k+1,k) & 1 \\ \eta & \lambda \end{bmatrix} \boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1)$ // see Eqn. (10)
$\boldsymbol{L}(k+1,k) = \lambda$
swap columns $\boldsymbol{L}(k+2\!:\!n, k)$ and $\boldsymbol{L}(k+2\!:\!n, k+1)$ // see Eqn. (11)
swap columns $\boldsymbol{Z}(1\!:\!n, k)$ and $\boldsymbol{Z}(1\!:\!n, k+1)$
swap entries $\hat{\boldsymbol{a}}(k)$ and $\hat{\boldsymbol{a}}(k+1)$

2.1.3 The reduction algorithm

The reduction process starts with the second to the last column of $\boldsymbol{L}$ and the last pair of the diagonal entries of $\boldsymbol{D}$ and tries to reach the first column of $\boldsymbol{L}$ and the first pair of the diagonal entries of $\boldsymbol{D}$. For simplicity, later we just say that the order of this process is from right to left. When the algorithm encounters an index k in the first time, the algorithm first performs an integer Gauss transformation on $\boldsymbol{L}$ such that the absolute values of the elements below l_{kk} are bounded above by 1/2 and then a permutation takes place for the pair $(k, k+1)$ if the condition in Eqn. (12) is satisfied. After a permutation, the algorithm restarts, i.e., it goes back to the initial position. The algorithm uses a variable ($k1$ in Algorithm 2.3) to track down those columns whose off-diagonal entries in magnitude are already bounded above by 1/2 due to previous integer Gauss transformations so that no new transformations will be performed any more for those columns in a restart process.

Here is the complete reduction process of the LAMBDA method:

Algorithm 2.3 (Reduction). Given the variance-covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ and real-valued LS estimate $\hat{\boldsymbol{a}}$ of $\boldsymbol{a}$. This algorithm computes an integer unimodular matrix $\boldsymbol{Z}$ and the $\mathrm{L^TDL}$ factorization $\boldsymbol{Q}_{\hat{\boldsymbol{z}}} = \boldsymbol{Z}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{Z} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$, where $\boldsymbol{L}$ and $\boldsymbol{D}$ are updated from the factors of the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$. This algorithm also computes $\hat{\boldsymbol{z}} = \boldsymbol{Z}^T\hat{\boldsymbol{a}}$, which overwrites $\hat{\boldsymbol{a}}$.

function: $[\boldsymbol{Z}, \boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}] = \mathrm{REDUCTION}(\boldsymbol{Q}_{\hat{a}}, \hat{\boldsymbol{a}})$
Compute the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$: $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$
$\boldsymbol{Z} = \boldsymbol{I}$
$k = n-1$; $k1 = n-1$
while $k > 0$
    if $k \le k1$

```
        for i = k+1 : n
            [L, â, Z] = GAUSS(L, i, k, â, Z)
        end
    end
    D̄(k+1, k+1) = D(k, k) + L(k+1, k)^2 D(k+1, k+1)
    if D̄(k+1, k+1) < D(k+1, k+1)
        [L, D, â, Z] = PERMUTE(L, D, k, D̄(k+1, k+1), â, Z)
        k1 = k; k = n − 1
    else
        k = k − 1
    end
end
```

2.2 Discrete search process

After the reduction process, one starts the discrete search process. To solve the ILS problem (3), a discrete search strategy is used to enumerate a subspace in $\mathbb{Z}^n$ which contains the solution. Suppose one has the following bound on the objective function in Eqn. (3):

$$f(\boldsymbol{z}) \stackrel{\text{def}}{=} (\boldsymbol{z}-\hat{\boldsymbol{z}})^T \boldsymbol{Q}_{\hat{\boldsymbol{z}}}^{-1}(\boldsymbol{z}-\hat{\boldsymbol{z}}) \leq \chi^2. \tag{14}$$

Notice that this is a hyper-ellipsoid. The solution will be searched over this hyper-ellipsoid.

Substituting the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{z}}}$ in Eqn. (4) into Eqn. (14) gives

$$f(\boldsymbol{z}) = (\boldsymbol{z}-\hat{\boldsymbol{z}})^T \bar{\boldsymbol{L}}^{-1} \bar{\boldsymbol{D}}^{-1} \bar{\boldsymbol{L}}^{-T} (\boldsymbol{z}-\hat{\boldsymbol{z}}) \leq \chi^2. \tag{15}$$

Define

$$\bar{\boldsymbol{z}} = \boldsymbol{z} - \bar{\boldsymbol{L}}^{-T}(\boldsymbol{z}-\hat{\boldsymbol{z}}), \tag{16}$$

giving

$$\bar{\boldsymbol{L}}^T(\boldsymbol{z}-\bar{\boldsymbol{z}}) = \boldsymbol{z}-\hat{\boldsymbol{z}},$$

or equivalently

$$\bar{z}_n = \hat{z}_n, \qquad \bar{z}_i = \hat{z}_i + \sum_{j=i+1}^{n} (z_j - \bar{z}_j)\bar{l}_{ji}, \qquad i = n-1, n-2, \ldots, 1, \tag{17}$$

where we observe that $\bar{z}_i$ depends on $z_{i+1}, \cdots, z_n$ and the former is determined when the latter are fixed. Then it follows from Eqn. (15) with Eqn. (16) that

$$f(\boldsymbol{z}) = (\boldsymbol{z}-\bar{\boldsymbol{z}})^T \bar{\boldsymbol{D}}^{-1}(\boldsymbol{z}-\bar{\boldsymbol{z}}) \leq \chi^2,$$

or equivalently

$$f(\boldsymbol{z}) = \frac{(z_1-\bar{z}_1)^2}{\bar{d}_1} + \frac{(z_2-\bar{z}_2)^2}{\bar{d}_2} + \cdots + \frac{(z_n-\bar{z}_n)^2}{\bar{d}_n} \leq \chi^2. \tag{18}$$

Obviously any $\boldsymbol{z}$ satisfying this bound must also satisfy the following individual bounds:

$$\bar{z}_n - \bar{d}_n^{1/2}\chi \le z_n \le \bar{z}_n + \bar{d}_n^{1/2}\chi, \tag{19}$$

$$\vdots$$

$$\bar{z}_i - \bar{d}_i^{1/2}\Big[\chi^2 - \sum_{j=i+1}^{n}(z_j - \bar{z}_j)^2/\bar{d}_j\Big]^{1/2} \le z_i \le \bar{z}_i + \bar{d}_i^{1/2}\Big[\chi^2 - \sum_{j=i+1}^{n}(z_j - \bar{z}_j)^2/\bar{d}_j\Big]^{1/2}, \tag{20}$$

$$\vdots$$

$$\bar{z}_1 - \bar{d}_1^{1/2}\Big[\chi^2 - \sum_{j=2}^{n}(z_j - \bar{z}_j)^2/\bar{d}_j\Big]^{1/2} \le z_1 \le \bar{z}_1 + \bar{d}_1^{1/2}\Big[\chi^2 - \sum_{j=2}^{n}(z_j - \bar{z}_j)^2/\bar{d}_j\Big]^{1/2}. \tag{21}$$

Note that the inequalities in Eqn. (21) are equivalent to the inequality in Eqn. (18).

Based on these bounds, a search procedure can be derived. The lower and upper bounds on z_i define an interval, which we call level i. The integers at this level are searched through in a straightforward manner from the smallest to the largest. Each valid integer at this level will be tried, one at a time. Once z_i is determined at level i, one proceeds with level $i-1$ to determine z_{i-1}. If no integer can be found at level i, one returns to the previous level $i+1$ to take the next valid integer for z_{i+1}, and then moves to level i again. Once z_1 is determined at level 1, a full integer vector $\boldsymbol{z}$ is found. Then we start to search new integer vectors. The new process starts at level 1 to search through all other valid integers from the smallest to the largest. The whole search process terminates when all valid integer encountered have been treated. To save space, we will not give a detailed description of a search algorithm here. But we will give it in Sect. 3.2. The idea of the above search strategy is similar to that of the so-called Pohst enumeration strategy in information theory, see Fincke and Pohst (1985), Phost (1981), Viterbo and E. Biglieri (1993), and Viterbo and Boutros (1999).

One important issue in the search process is setting the positive constant χ^2 which controls the size of the ellipsoidal region. In the LAMBDA package, during the search process the size of the ellipsoidal region stays the same. Therefore, the performance of the search process will highly depend on the value of χ^2. A small value for χ^2 may result in an ellipsoidal region that fails to contain the minimizer of (1), while a too large value for χ^2 may result in a region for which the search for the minimizer becomes too time-consuming. The LAMBDA package sets the value of χ^2 in the following way. Suppose p optimal ILS estimates are required. If $p \le n+1$, one takes $z_i = \lfloor \bar{z}_i \rceil$ for $i = n, n-1, \ldots, 1$ (i.e., rounding each $\bar{z}_i$ to its nearest integer) in Eqn. (17), producing the first integer vector $\boldsymbol{z}^{(1)}$, which corresponds to the so-called Babai point in information theory literature, see Babai (1986) and Agrell et al. (2002). Then for each i ($i = 1, \ldots, n$), one rounds the obtained $\bar{z}_i$ to the next-nearest integer while keeping all other entries of $\boldsymbol{z}^{(1)}$ unchanged, producing a new integer vector. Based on these $n+1$ integer vectors, χ^2 is set to be the pth smallest value of the objective function $f(\boldsymbol{z})$, which will guarantee at least p candidates in the ellipsoidal region. If $p > n+1$, the volume of the ellipsoid is set to be p and then χ^2 is determined.

Before the end of the section, let us make two remarks. The implementation of the search process in the LAMBDA package is actually based on the $\mathrm{LDL^T}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}^{-1}$, which is computed from the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$, i.e., the lower triangular factor of the former is obtained by inverting the lower triangular factor of the latter. When the optimal estimate of $\boldsymbol{z}$ denoted by $\check{\boldsymbol{z}}$ is found, a back transformation, $\check{\boldsymbol{a}} = \boldsymbol{Z}^{-1}\check{\boldsymbol{z}}$ (cf. Eqn. 2) is needed. For the details about this computation, see De Jonge and Tiberius (1996), Sects. 3.9 and 4.13.

3 Modifying the LAMBDA method

In this section we present several strategies to effectively reduce the computational complexity of the LAMBDA method. In Sect. 3.1, we show how to improve the reduction process and in Sect. 3.2, we show how to improve the search process. The combined two new processes form the modified LAMBDA method (MLAMBDA).

3.1 Modified reduction

3.1.1 Symmetric pivoting strategy

In order to strive for the inequalities in Eqn. (5) or to achieve the inequalities in Eqn. (13), the reduction algorithm (Algorithm 2.3) in Sect. 2.1 performs the permutations. In general the computation caused by the permutations is likely to dominate the cost of the whole reduction process. The less the number of permutations, the less the cost of Algorithm 2.3. Motivated by this, we propose to incorporate a symmetric pivoting strategy in computing the $\mathrm{L^TDL}$ factorization of the covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ at the beginning of the reduction process.

We first look at the derivation of the algorithm for the $\mathrm{L^TDL}$ factorization without pivoting. Suppose $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is symmetric positive definite. We partition $\boldsymbol{Q} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$ as follows

$$\begin{bmatrix} \tilde{\boldsymbol{Q}} & \boldsymbol{q} \\ \boldsymbol{q}^T & q_{nn} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{L}}^T & \boldsymbol{l} \\ & 1 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{D}} & \\ & d_n \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{L}} & \\ \boldsymbol{l}^T & 1 \end{bmatrix}.$$

Therefor

$$d_n = q_{nn}, \quad \boldsymbol{l} = \boldsymbol{q}/d_n, \quad \tilde{\boldsymbol{Q}} - \boldsymbol{l}d_n\boldsymbol{l}^T = \tilde{\boldsymbol{L}}^T\tilde{\boldsymbol{D}}\tilde{\boldsymbol{L}}^T.$$

These equations shows clearly how to find d_n, $\boldsymbol{l}$, $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{D}}$.

Since we strive for the inequalities in Eqn. (5), we first symmetrically permutate the smallest diagonal entry of $\boldsymbol{Q}$ to the (n,n) position and then find d_n, $\boldsymbol{l}$ and apply the same approach to $\tilde{\boldsymbol{Q}} - \boldsymbol{l}d_n\boldsymbol{l}^T$. Finally we obtain the $\mathrm{L^TDL}$ factorization of a permutated $\boldsymbol{Q}$. In fact, suppose after the first symmetric permutation $\boldsymbol{P}_1$ we have

$$\boldsymbol{P}_1^T\boldsymbol{Q}\boldsymbol{P}_1 = \begin{bmatrix} \tilde{\boldsymbol{Q}} & \boldsymbol{q} \\ \boldsymbol{q}^T & q_{nn} \end{bmatrix}.$$

Define $d_n = q_{nn}$ and $\boldsymbol{l} = \boldsymbol{q}/d_n$. Let $\tilde{\boldsymbol{Q}} - \boldsymbol{l}d_n\boldsymbol{l}^T$ have the following $\mathrm{L^TDL}$ factorization with symmetric pivoting

$$\tilde{\boldsymbol{P}}^T(\tilde{\boldsymbol{Q}} - \boldsymbol{l}d_n\boldsymbol{l}^T)\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{L}}^T\tilde{\boldsymbol{D}}\tilde{\boldsymbol{L}},$$

where $\tilde{\boldsymbol{P}}$ is the product of permutation matrices. Then it is easy to verify that

$$\boldsymbol{P}^T\boldsymbol{Q}\boldsymbol{P} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}, \quad \boldsymbol{P} = \boldsymbol{P}_1\begin{bmatrix} \tilde{\boldsymbol{P}} & \\ & 1 \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} \tilde{\boldsymbol{L}} & \\ \boldsymbol{l}^T\tilde{\boldsymbol{P}} & 1 \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} \tilde{\boldsymbol{D}} & \\ & d_n \end{bmatrix},$$

giving the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}$ with symmetric pivoting. Note that the $\mathrm{L^TDL}$ factorization with symmetric pivoting is similar to the Cholesky factorization of a symmetric nonnegative definite matrix with symmetric pivoting, see, e.g., Golub and Van Loan (1996), Sect. 4.2.9. But in the latter, the pivot element is chosen to be the largest element, rather than the smallest one. We need to point out that this symmetric pivoting strategy was also used in Xu et al. (1995) and Liu et al. (1999) for different motivations.

Algorithm 3.1 ($\mathrm{L^TDL}$ factorization with symmetric pivoting). Suppose $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is symmetric positive definite. This algorithm computes a permutation $\boldsymbol{P}$, a unit lower triangular matrix $\boldsymbol{L}$ and a diagonal $\boldsymbol{D}$ such that $\boldsymbol{P}^T\boldsymbol{Q}\boldsymbol{P} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$. The strict lower triangular part of $\boldsymbol{Q}$ is overwritten by that of $\boldsymbol{L}$ and the diagonal part of $\boldsymbol{Q}$ is overwritten by that of $\boldsymbol{D}$.

$\boldsymbol{P} = \boldsymbol{I}_n$
for $k = n:-1:1$
  $q = \arg\min_{1\le j\le k} \boldsymbol{Q}(j,j)$
  swap $\boldsymbol{P}(:,k)$ and $\boldsymbol{P}(:,q)$
  swap $\boldsymbol{Q}(k,:)$ and $\boldsymbol{Q}(q,:)$
  swap $\boldsymbol{Q}(:,k)$ and $\boldsymbol{Q}(:,q)$
  $\boldsymbol{Q}(k,1:k-1) = \boldsymbol{Q}(k,1:k-1)/\boldsymbol{Q}(k,k)$
  $\boldsymbol{Q}(1:k-1,1:k-1) = \boldsymbol{Q}(1:k-1,1:k-1) - \boldsymbol{Q}(k,1:k-1)^T * \boldsymbol{Q}(k,k) * \boldsymbol{Q}(k,1:k-1)$
end

The above algorithm can be made more efficient. In fact, we need only to compute the lower triangular part of $\boldsymbol{Q}(1:k-1,1:k-1)$ in the kth step, since it is symmetric.

3.1.2 Greedy selection strategy

After the $\mathrm{L^TDL}$ factorization with symmetric pivoting, we start the reduction process. In order to make the reduction more efficient, we would like to further reduce the number of permutations. As we have seen in Sect. 2.1, the reduction process in the LAMBDA method is done in a sequential order (from right to left). When the condition $\bar{d}_{k+1} < d_{k+1}$ (see Eqn. 12) is met, a permutation for the pair $(k, k+1)$ takes place and then we go back to the initial position, i.e., $k = n-1$. Intuitively, it is unlikely very efficient to do the reduction in this way. When we reach a *critical index* k, i.e., $d_{k+1} \gg d_k$ and $\bar{d}_{k+1} < d_{k+1}$, a permutation is then performed at this position, but it is likely that we will end up with $d_{k+2} \gg d_{k+1}$ and $\bar{d}_{k+2} < d_{k+2}$, thus $k+1$ becomes a *critical index* and so on. Therefore the permutations which had been performed before we reached the index k are likely wasted.

One solution for this problem is to apply what we call a greedy selection strategy. Instead of looping k from $n-1$ to 1 in Algorithm 2.3, we always choose the index k such that d_{k+1} will decrease most when a permutation for the pair $(k, k+1)$ is performed, i.e., k is defined by

$$k = \arg\min_{1\le j\le n-1} \{\bar{d}_{j+1}/d_{j+1} : \bar{d}_{j+1} < d_{j+1}\}. \tag{22}$$

If no such k is found, no any permutation can be done.

3.1.3 Lazy transformation strategy

The permutations in the reduction process of the LAMBDA method may change the magnitudes of the off-diagonal elements of $\boldsymbol{L}$. So it may happen that integer Gauss transformations are applied to the same elements in $\boldsymbol{L}$ many times due to permutations. Specifically, if a permutation for the pair $(k, k+1)$ is performed, $\boldsymbol{L}(k:k+1, 1:k-1)$ (see Eqn. (10)) are changed and the two columns of $\boldsymbol{L}(k+1:n, k:k+1)$ (see Eqn. 11) are swaped. If the absolute values of the elements of $\boldsymbol{L}(k:k+1, 1:k-1)$ are bounded above by 1/2 before this permutation, then after permuting, these bounds are not guaranteed to hold any more. Thus, for those elements of $\boldsymbol{L}(k:k+1, 1:k-1)$ which are now larger than 1/2 in magnitude, the corresponding integer Gauss transformations which were applied to make these elements bounded above by 1/2 before

this permutation are wasted. The permutation also affects $l_{k+1,k}$ (see Eqn. 9), whose absolute value may not be bounded above by 1/2 any more either. But any integer Gauss transformation which was applied to ensure $|l_{k+1,k}| \leq 1/2$ before the permutation is not wasted, since it is necessary to do this transformation for doing this permutation. The reason is as follows. Note that $\bar{d}_{k+1} = d_k + l_{k+1,k}^2 d_{k+1}$ (see Eqn. 8). The goal of a permutation is to make $\bar{d}_{k+1}$ smaller, so $l_{k+1,k}$ should be as small as possible, which is realized by an integer Gauss transformation.

We propose to apply integer Gauss transformations only to some of the subdiagonal elements of $\boldsymbol{L}$ first, then do permutations. During the permutation process, integer Gauss transformations will be applied only to some of the changed subdiagonal elements of $\boldsymbol{L}$. When no permutation will take place, we apply the transformations to the off-diagonal elements of $\boldsymbol{L}$. We call this strategy a "lazy" transformation strategy. Specifically, at the beginning of the reduction process, an integer Gauss transformation is applied to $l_{k+1,k}$ when the following criterion is satisfied:

$$\text{Criterion 1:} \quad d_k < d_{k+1}. \tag{23}$$

When this criterion is not satisfied, d_k and d_{k+1} have been in the correct order, so we do not need to do a permutation for the pair $(k, k+1)$. Later on an integer Gauss transformation is applied to $l_{k+1,k}$ when both Criterion 1 and the following Criterion 2 are satisfied:

$$\text{Criterion 2:} \quad l_{k+1,k} \text{ is changed by the last permutation.} \tag{24}$$

This criterion is used to skip the unchanged subdiagonal elements of $\boldsymbol{L}$. After a permutation takes place for the pair $(k, k+1)$, three elements in sub-diagonal of $\boldsymbol{L}$ are changed. They are $l_{k,k-1}$, $l_{k+1,k}$, and $l_{k+2,k+1}$. So after a permutation, at most three integer Gauss transformations are applied to these elements. Finally, when no permutation will be performed, integer Gauss transformations are applied to all elements in the strictly lower triangular part of $\boldsymbol{L}$.

3.1.4 Modified reduction algorithm

Now we can combine the three strategies given in Sects. 3.1.1–3.1.3 together, leading to the following modified reduction algorithm.

Algorithm 3.2 (Modified reduction). Given the variance-covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} \in \mathbb{R}^{n\times n}$ and real-valued LS estimate $\hat{\boldsymbol{a}} \in \mathbb{R}^n$ of $\boldsymbol{a}$. This algorithm computes an integer unimodular matrix $\boldsymbol{Z}$ and the $\mathrm{L^TDL}$ factorization $\boldsymbol{Q}_{\hat{\boldsymbol{z}}} = \boldsymbol{Z}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{Z} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$, where $\boldsymbol{L}$ and $\boldsymbol{D}$ are updated from the factors of the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ with symmetric pivoting. This algorithm also computes $\hat{\boldsymbol{z}} = \boldsymbol{Z}^T\hat{\boldsymbol{a}}$, which overwrites $\hat{\boldsymbol{a}}$.

function: $[\boldsymbol{Z}, \boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}] = \mathrm{MREDUCTION}(\boldsymbol{Q}_{\hat{\boldsymbol{a}}}, \hat{\boldsymbol{a}})$
Compute the $\mathrm{L^TDL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ with symmetric pivoting: $\boldsymbol{P}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{P} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$
$\boldsymbol{Z} = \boldsymbol{P}$
Set all elements of ChangeFlag(1:n+1) to ones
while true
  minratio = 1
  for $k = 1 : n-1$
    if $\frac{\boldsymbol{D}(k,k)}{\boldsymbol{D}(k+1,k+1)} < 1$ // Criterion 1
      if ChangeFlag$(k+1) = 1$ // Criterion 2
        $[\boldsymbol{L}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \mathrm{GAUSS}(\boldsymbol{L}, k+1, k, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
        $\bar{\boldsymbol{D}}(k+1,k+1) = \boldsymbol{D}(k,k) + \boldsymbol{L}(k+1,k)^2\boldsymbol{D}(k+1,k+1)$
        ChangeFlag$(k+1) = 0$

```
            end
            tmp = D̄(k+1,k+1) / D(k+1,k+1)
            if tmp < minratio
                i = k        // see Eqn. (22)
                minratio = tmp
            end
        end
    end
    if minratio = 1
        break while loop
    end
    [L, D, â, Z] = PERMUTE(L, D, i, D̄(k + 1, k + 1), â, Z)
    Set ChangeFlag(i:i+2) to ones
end
for k = 1 : n − 1        // apply GAUSS to L's strictly lower triangular entries
    for i = k + 1 : n
        [L, â, Z] = GAUSS(L, i, k, â, Z)
    end
end
```

The number of subdiagonal entries of $\boldsymbol{L}$ is $n-1$, but in this algorithm we set ChangeFlag to be an $n+1$ dimensional vector in order to easily handle two extreme cases: $k=1$ and $k=n-1$.

3.2 Modified search process

In section 2.2 we describe the search process of the LAMBDA method. As we know, χ^2, which controls the volume of the search region, plays an important role in the search process. For finding the (single) optimal ILS estimate, Teunissen (1993) proposed to use a shrinking strategy to reduce the search region. As soon as a candidate integer vector $\boldsymbol{z}$ in the ellipsoidal region is found, the corresponding $f(\boldsymbol{z})$ in (15) is taken as a new value for χ^2. So the ellipsoidal region is shrunk. As De Jonge and Tiberius (1996) point out, the shrinking strategy can greatly benefit the search process. However, this strategy is not used in the LAMBDA package, which can find several optimal ILS estimates. To make the search process more efficient, we propose to extend the shrinking strategy to the case where more than one optimal estimates are required.

Before proceeding, we describe an alternative (see (Teunissen, 1995b, Sect. 2.4) and (De Jonge and Tiberius, 1996, Sect. 4.7)) to the straightforward search from the smallest to the largest at a level given in section 2.2. We search for integers according to nondecreasing distance to $\bar{z}_i$ in the interval defined by Eqn. (20) at level i. Specifically, if $\bar{z}_i \leq \lfloor \bar{z}_i \rceil$, we use the following search order:

$$\lfloor \bar{z}_i \rceil, \lfloor \bar{z}_i \rceil - 1, \lfloor \bar{z}_i \rceil + 1, \lfloor \bar{z}_i \rceil - 2, \cdots, \tag{25}$$

otherwise, we use

$$\lfloor \bar{z}_i \rceil, \lfloor \bar{z}_i \rceil + 1, \lfloor \bar{z}_i \rceil - 1, \lfloor \bar{z}_i \rceil + 2, \cdots. \tag{26}$$

This search strategy was also proposed independently by Schnorr and Euchner (1994).

Now we describe how to apply a shrinking strategy to the search process when more than one optimal ILS estimates are required. Suppose we require p optimal ILS estimates. At the

beginning we set χ^2 to be infinity. Obviously the first candidate obtained by the search process is

$$\boldsymbol{z}^{(1)} = \left[\lfloor \bar{z}_1 \rceil, \lfloor \bar{z}_2 \rceil, \cdots, \lfloor \bar{z}_n \rceil\right]^T.$$

Note that $\boldsymbol{z}^{(1)}$ here is obtained by the search process, rather than by a separate process as the LAMBDA package does (cf. Sect. 2.2, paragraph 4). We take the second candidate $\boldsymbol{z}^{(2)}$ identical to $\boldsymbol{z}^{(1)}$ except that the first entry in $\boldsymbol{z}^{(2)}$ is taken as the second nearest integer to $\bar{z}_1$. And the third $\boldsymbol{z}^{(3)}$ is the same as $\boldsymbol{z}^{(1)}$ except that its first entry is taken as the third nearest integer to $\bar{z}_1$, and so on. In this way we obtain p candidates $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \cdots, \boldsymbol{z}^{(p)}$. Obviously we have $f(\boldsymbol{z}^{(1)}) \leq f(\boldsymbol{z}^{(2)}) \cdots \leq f(\boldsymbol{z}^{(p)})$ (cf. Eqn. 18). Then the ellipsoidal region is shrunk by setting $\chi^2 = f(\boldsymbol{z}^{(p)})$. This is an alterative to the method used by the LAMBDA method for setting χ^2 and its main advantage is that it is simpler to determine χ^2. Also if $p = 2$, it is likely the value of χ^2 determined by this method is smaller than that determined by the LAMBDA method since d_1 is likely larger than other d_i after the reduction process (cf. Eqn. 18). Then we start to search a new candidate. We return to level 2 and take the next valid integer for z_2. Continue the search process until we find a new candidate at level 1. Now we replace the candidate $\boldsymbol{z}^{(j)}$ which satisfies $f(\boldsymbol{z}^{(j)}) = \chi^2$ with the new one. Again we shrink the ellipsoidal region. The newer χ^2 is taken as $\max_{1 \leq i \leq p} f(\boldsymbol{z}^{(i)})$. Then we continue the above process until we cannot find a new candidate. Finally we end up with the p optimal ILS estimates.

The above modified search process is described by Algorithm 3.3.

Algorithm 3.3 (Modified search). Given the unit lower triangular matrix $\boldsymbol{L} \in \mathbb{R}^{n \times n}$, the diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, and the real vector $\hat{\boldsymbol{z}} \in \mathbb{R}^n$ obtained from the reduction process. This algorithm finds the p optimal ILS estimates for the ILS problem in Eqn. (3), which are stored in an $n \times p$ array Optis. (Note: the operation $\mathrm{sgn}(x)$ returns -1 if $x \leq 0$ and 1 if $x > 0$)

function: Optis = MSEARCH($\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{z}}, p$)
maxDist = $+\infty$ // maxDist: current χ^2
$k = n$; dist(k) = 0
endSearch = false
count = 0 // count: the number of candidates
Initialize an $n \times n$ zero matrix $\boldsymbol{S}$ // $\boldsymbol{S}$ will be used for computing $\bar{\boldsymbol{z}}(k)$
$\bar{\boldsymbol{z}}(n) = \hat{\boldsymbol{z}}(n)$ // see Eqn. (17)
$\boldsymbol{z}(n) = \lfloor \bar{\boldsymbol{z}}(n) \rceil$; $y = \bar{\boldsymbol{z}}(n) - \boldsymbol{z}(n)$; step($n$) = sgn($y$); imax = p
while endSearch = false
  newDist = dist(k) + $y^2/\boldsymbol{D}(k,k)$ // newDist = $\sum_{j=k}^{n}(z_j - \bar{z}_j)^2/d_j$
  if newDist < maxDist
    if $k \neq 1$ //**Case 1:** move down
      $k = k - 1$
      dist(k) = newDist // dist(k) = $\sum_{j=k+1}^{n}(z_j - \bar{z}_j)^2/d_j$
      $\boldsymbol{S}(k, 1\!:\!k) = \boldsymbol{S}(k+1, 1\!:\!k) + (\boldsymbol{z}(k+1) - \bar{\boldsymbol{z}}(k+1)) * \boldsymbol{L}(k+1, 1\!:\!k)$
        // $\boldsymbol{S}(k, 1:k) = \sum_{j=k+1}^{n}(z_j - \bar{z}_j)\boldsymbol{L}(j, 1:k)$
      $\bar{\boldsymbol{z}}(k) = \hat{\boldsymbol{z}}(k) + \boldsymbol{S}(k,k)$ // see Eqn. (17)
      $\boldsymbol{z}(k) = \lfloor \bar{\boldsymbol{z}}(k) \rceil$; $y = \bar{\boldsymbol{z}}(k) - \boldsymbol{z}(k)$; step($k$) = sgn($y$)
    else //**Case 2:** store the found candidate and try next valid integer
      if count < $p - 1$ // store the first $p - 1$ initial points
        count = count + 1
        Optis(:, count) = $\boldsymbol{z}(1:n)$
        fun(count) = newDist // store $f(\boldsymbol{z})$

```
            else
                Optis(:, imax) = z(1 : n)
                fun(imax) = newDist
                imax = arg max_{1≤i≤p} fun(i)
                maxDist = fun(imax)
            end
            z(1) = z(1) + step(1)        //next valid integer
            y = z̄(1) − z(1)
            step(1) = −step(1) − sgn(step(1))        // cf. Eqs. (25) and (26)
        end
    else      //Case 3: exit or move up
        if k = n
            endSearch = true
        else
            k = k + 1        //move up
            z(k) = z(k) + step(k)        //next valid integer
            y = z̄(k) − z(k)
            step(k) = −step(k) − sgn(step(k))        // cf. Eqs. (25) and (26)
        end
    end
end
```

In this algorithm, suppose k is the current level. When newDist is less than the current χ^2, i.e., the value of the objective function f at the "worst" candidate, the algorithm moves down to level $k-1$. This is done in Case 1. On the other hand, as soon as newDist is greater than the current χ^2, the algorithm moves up to level $k+1$, which is done in Case 3. Case 2 is invoked when the algorithm has successfully moved through all the levels to level 1 without exceeding the current χ^2. Then this found candidate is stored as a potential optimal candidate and the χ^2 is updated and the algorithm tries the next valid integer for level 1. To some extent, the description of the algorithm is similar to the one given in Agrell et al. (2002), which finds only the (single) optimal estimate.

4 Numerical simulations

We implemented the modified LAMBDA method (MLAMBDA) given in section 3 and did numerical simulations to compare the running time (i.e., CPU time) of MLAMBDA with that of the LAMBDA package (MATLAB, version 2.0), which is available from the Mathematical Geodesy and Positioning of Delft University of Technology (http://enterprise.lr.tudelft.nl/mgp/). In addition, we also compare the reduction time of the LLL algorithm and the MLAMBDA method.

All our computations were performed in MATLAB 7.0.1 on a Pentium 4, 3.20GHz PC with 1GB memory running Windows XP Professional.

We performed simulations for different cases. The real vector $\hat{\boldsymbol{a}}$ was constructed as follows:

$$\hat{\boldsymbol{a}} = 100 * \texttt{randn}(n, 1), \tag{27}$$

where $\texttt{randn}(n, 1)$ is a MATLAB built-in function to generate a vector of $\boldsymbol{n}$ random entries which are normally distributed.

The first four cases are based on $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$ where $\boldsymbol{L}$ is a unit lower triangular matrix with each l_{ij} (for $i > j$) being a random number generated by `randn`, and $\boldsymbol{D}$ is generated in four different ways:

- Case 1: $\boldsymbol{D} = \mathrm{diag}(d_i),\ d_i = \texttt{rand}$, where `rand` is a MATLAB built-in function to generate uniformly distributed random numbers in $(0, 1)$.
- Case 2: $\boldsymbol{D} = \mathrm{diag}(n^{-1}, (n-1)^{-1}, \ldots, 1^{-1})$.
- Case 3: $\boldsymbol{D} = \mathrm{diag}(1^{-1}, 2^{-1}, \ldots, n^{-1})$.
- Case 4: $\boldsymbol{D} = \mathrm{diag}(200, 200, 200, 0.1, 0.1, \ldots, 0.1)$.

The other four cases are as follows:

- Case 5: $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, $\boldsymbol{U}$ is a random orthogonal matrix obtained by the QR factorization of a random matrix generated by $\texttt{randn}(n, n)$, $\boldsymbol{D} = \mathrm{diag}(d_i),\ d_i = \texttt{rand}$.
- Case 6: $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, $\boldsymbol{U}$ is generated in the same way as in Case 5, $d_1 = 2^{-\frac{n}{4}}$, $d_n = 2^{\frac{n}{4}}$, other diagonal elements of $\boldsymbol{D}$ is randomly distributed between d_1 and d_n, n is the dimension of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$. Thus the condition number of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ is $2^{\frac{n}{2}}$
- Case 7: $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{A}^T\boldsymbol{A}$, $\boldsymbol{A} = \texttt{randn}(n, n)$.
- Case 8: $\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, the dimension of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ is fixed to 20, $\boldsymbol{U}$ is generated in the same way as in Case 5, $d_1 = 2^{-\frac{k}{2}}$, $d_n = 2^{\frac{k}{2}}$, other diagonal elements of $\boldsymbol{D}$ are randomly distributed between d_1 and d_n, $k = 5, 6, \ldots, 20$. Thus the range of the condition number of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ is from 2^5 to 2^{20}.

Case 4 is motivated by the fact that the covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ in GPS usually has a large gap between the third conditioned standard deviation and the forth one (Teunissen, 1998, Sect. 8.3.3). We took dimension $n = 5, 6, \ldots, 40$ for the first seven cases, and performed 100 runs for the first four cases, 20 runs for the last four cases. In each run we computed the first and second optimal ILS estimates by both the LAMBDA package and our MLAMBDA method. The results about the average running time (in seconds) are given in Figure 1–Figure 8. For each case we give three plots, corresponding to the average reduction time, the average search time, and the average time (including both the reduction time and the search time), respectively. In the reduction time plots, we also include the time of the LLL algorithm. A reader may notice that in the average search time plots in Figure 3 and Figure 7, the values of the average search times taken by MLAMBDA for some lower dimensional problems are missing. This is because MATLAB counts the CPU time of a computation as zero if it is lower than 1 millisecond, but the logarithm of zero is not defined. For clarity, we also give the average running time for dimension $n = 40$ for all cases (except Case 8 in which $n = 20$) in Table 1.

From the simulation results, we observe that MLAMBDA is faster or much faster than LAMBDA. Usually the improvement becomes more and more significant when the dimension n increases (see the bottom plots of Figures 1–7, each uses a logarithmic scale for the vertical axis). Table 1 shows that when $n = 40$, averagely MLAMBDA is about 11 times, 22 times, 13 times, 34 times, 5 times, 16 times, 38 times as fast as LAMBDA for Cases 1–7, respectively.

MLAMBDA improves the computational efficiency for both reduction and search. For reduction, when the dimension n is small, usually LAMBDA, LLL and MLAMBDA take more or less the same reduction time, but when n increases, MLAMBDA becomes faster and faster than LAMBDA and LLL. Sometimes, the improvement is significant, for example, in Case 5, when

$n = 40$ (see Table 1), MLAMBDA is about 10 times as fast as LAMBDA and LLL. We also observe that usually there is no big reduction time difference between LAMBDA and LLL, although occasionally (see Figure 4) the latter may be significantly faster than the former. As the dimension n increases, the reduction time of all the three algorithms increases in a polynomial way.

For search, MLAMBDA is faster than LAMBDA for almost all cases. Usually the time difference between the two algorithms becomes bigger and bigger as the dimension n increases (note that a logarithmic scale for the vertical axis is used in each average search time plot). When the dimension n small, both reduction and search are fast for the two algorithms, and one may take more time than the other. But as the dimension increases, the search time becomes more and more dominant and actually increases exponentially for both algorithms, so the improvement of computational efficiency becomes more and more important. For example, in Case 2, where $\boldsymbol{D}$ are in the order opposite to that we strive for (see Eqn. 5), when $n = 40$ (see Table 1), almost 100% time is spent on search for either algorithm. LAMBDA takes 1894 seconds for search, while MLAMBDA takes about only 88 seconds.

In Case 8, where $n = 20$, when the condition number of the covariance matrix increases dramatically, the search time for either algorithm increases slightly, and the reduction time does too. This indicates that the condition number of the original covariance matrix has only minor effect on the computational time for both algorithms. From the simulation results, we observe that for different cases with the same dimension, the search time may vary dramatically.

We would like to make two other remarks. In the simulations, we found LAMBDA and MLAMBDA gave the same computed solution for the same ILS problem. Thus, for the tested problems, there is no difference between the two algorithms in terms of accuracy. In our simulations, after we transformed an ILS problem using the LLL reduction algorithm, we applied the MLAMBDA search algorithm to the transformed ILS problem. We found it took almost the same amount of time as it did to the transformed ILS problem obtained by the MLAMBDA reduction. The explanation is that the properties in Eqs. (6) and (13), which are crucial to the search speed, are satisfied by the transformed problem obtained either by the LLL reduction or by the MLAMBDA reduction.

	LLL	LAMBDA			MLAMBDA		
	reduction	reduction	search	total	reduction	search	total
Case 1	0.0345	0.033	5.800	5.833	0.0158	0.5195	0.5353
Case 2	0.0623	0.0455	1894	1894	0.0230	87.57	87.60
Case 3	0.0164	0.0156	0.4191	0.4347	0.0080	0.0261	0.0341
Case 4	0.0222	0.0234	6.546	6.569	0.0105	0.1806	0.1911
Case 5	0.0565	0.0551	9.887	9.942	0.0055	2.203	2.208
Case 6	0.1282	0.1252	233.2	233.3	0.0315	14.65	14.68
Case 7	0.0711	0.0813	983.9	984.0	0.0164	25.84	25.86

Table 1: Average running time for dimension $n = 40$

5 Summary

The well-known LAMBDA method has been widely used for integer least-squares estimation problems in positioning and navigation. We pointed out its close relations with the ILS methods developed for other applications in the literature. The LAMBDA method consists of two

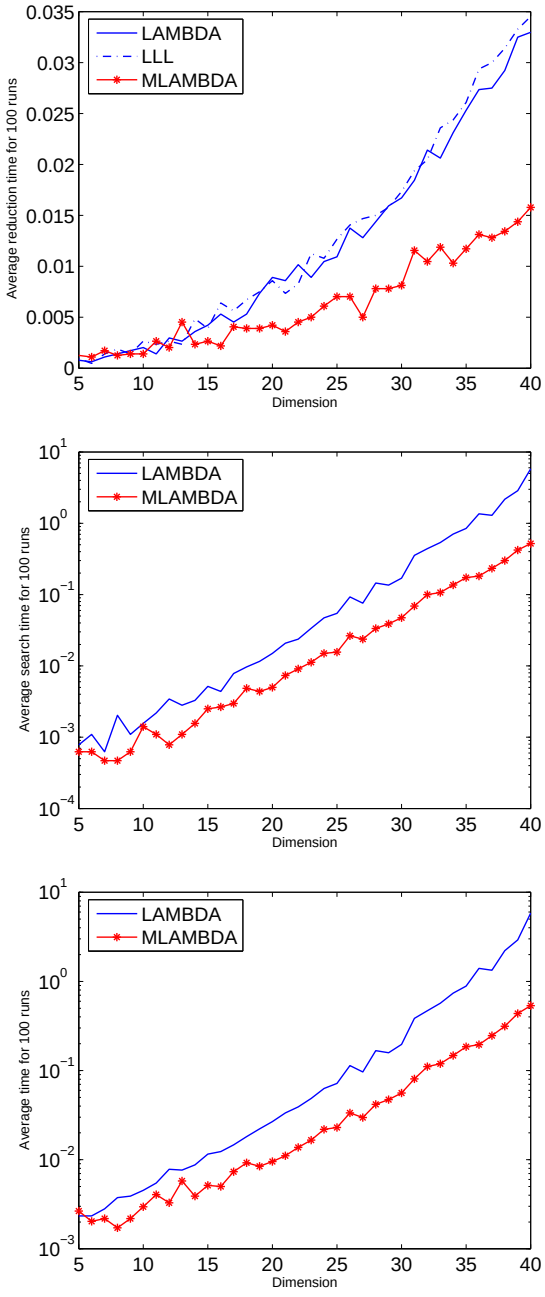


Figure 1: Running time for Case 1

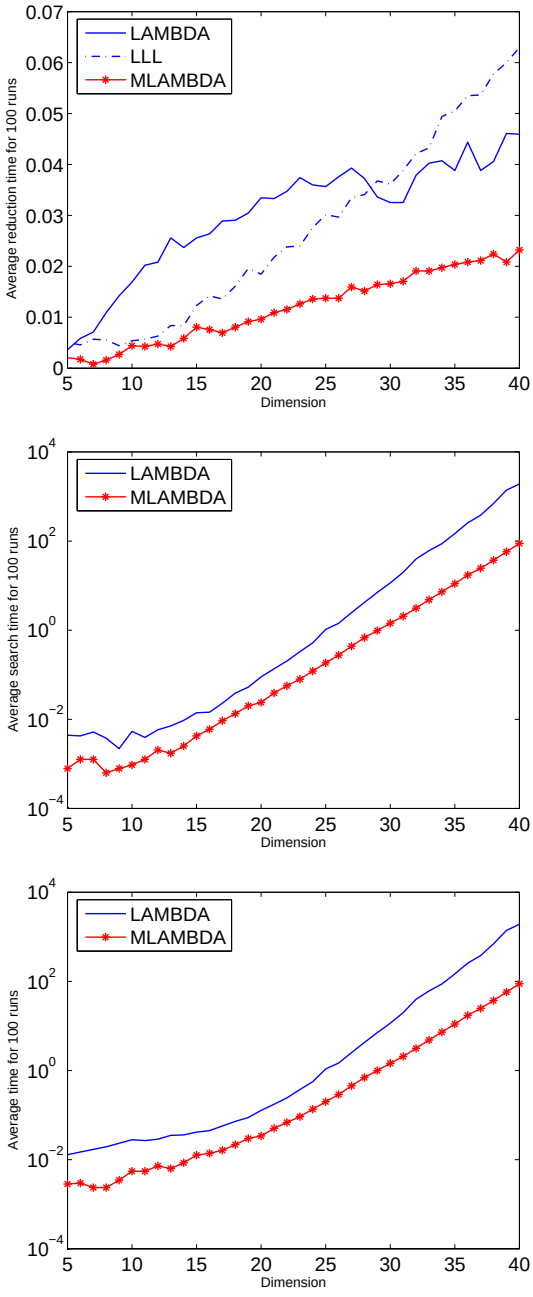


Figure 2: Running time for Case 2

stages, reduction and search. We presented a modified LAMBDA method (MLAMBDA) which improves both of these two stages. The key to our algorithm is to compute the L^TDL factorization with symmetric pivoting, decorrelate the parameters by greedy selection and lazy transformations, and shrink the ellipsoidal region during the search process. Numerical simulation showed that MLAMBDA can be much faster than LAMBDA implemented in Delft's LAMBDA package (MATLAB, version 2.0) for high dimensional problems. This will be particularly useful to integer ambiguity determination when there are more GNSS satellites visible simultaneously, with carrier phase observations on three frequencies in the future. We gave

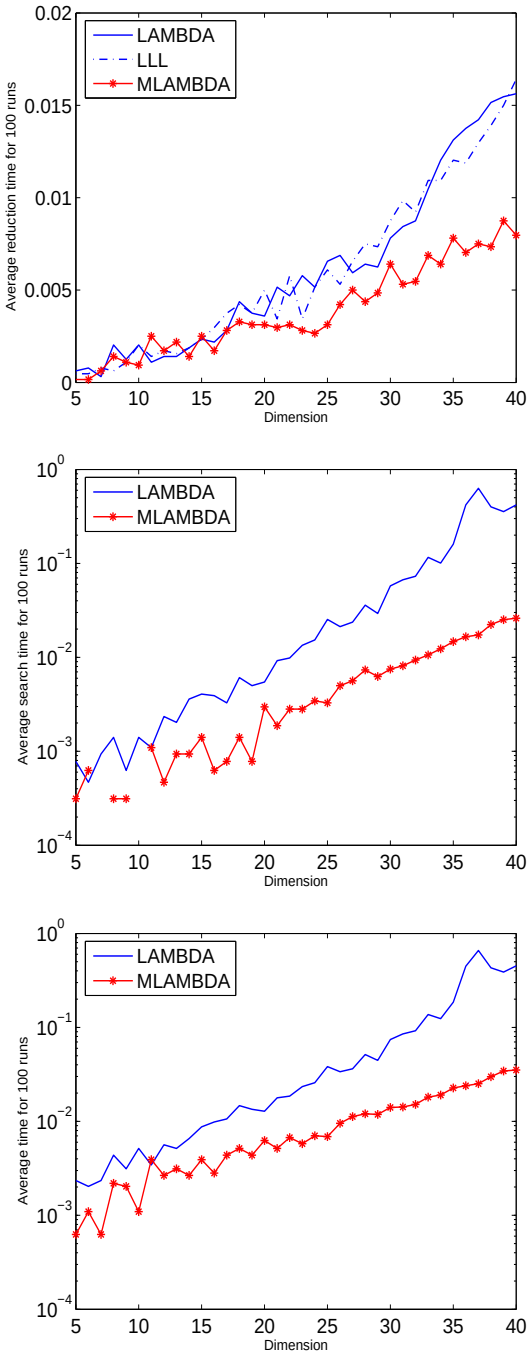


Figure 3: Running time for Case 3

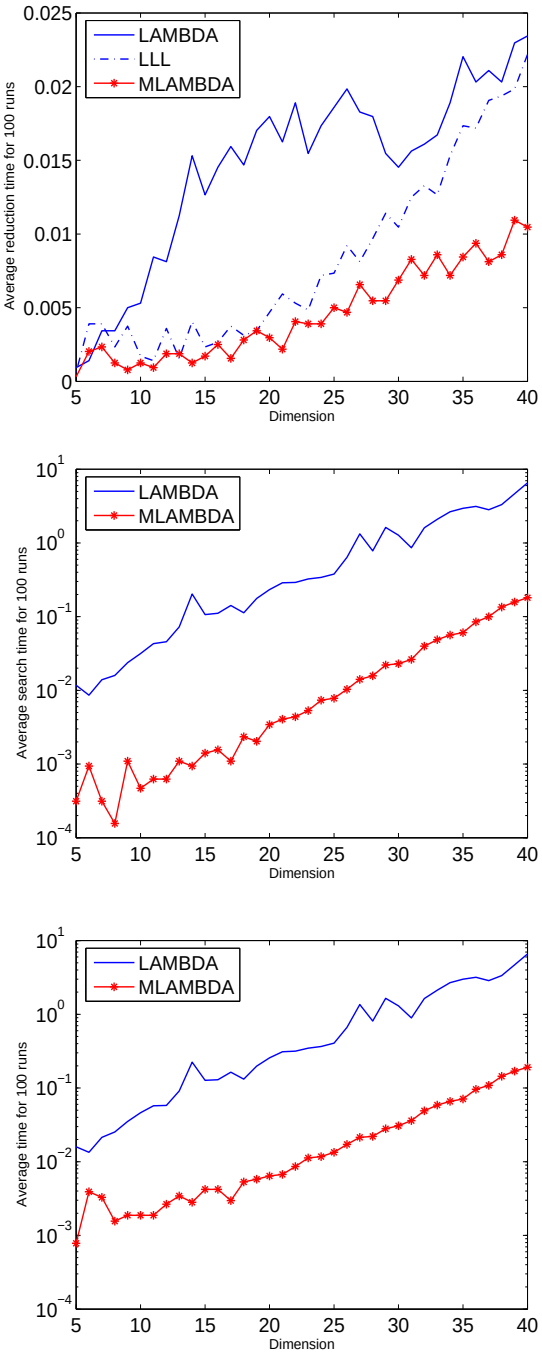


Figure 4: Running time for Case 4

complete computational details for our new method. Hopefully a reader can implement the algorithms without difficulty.

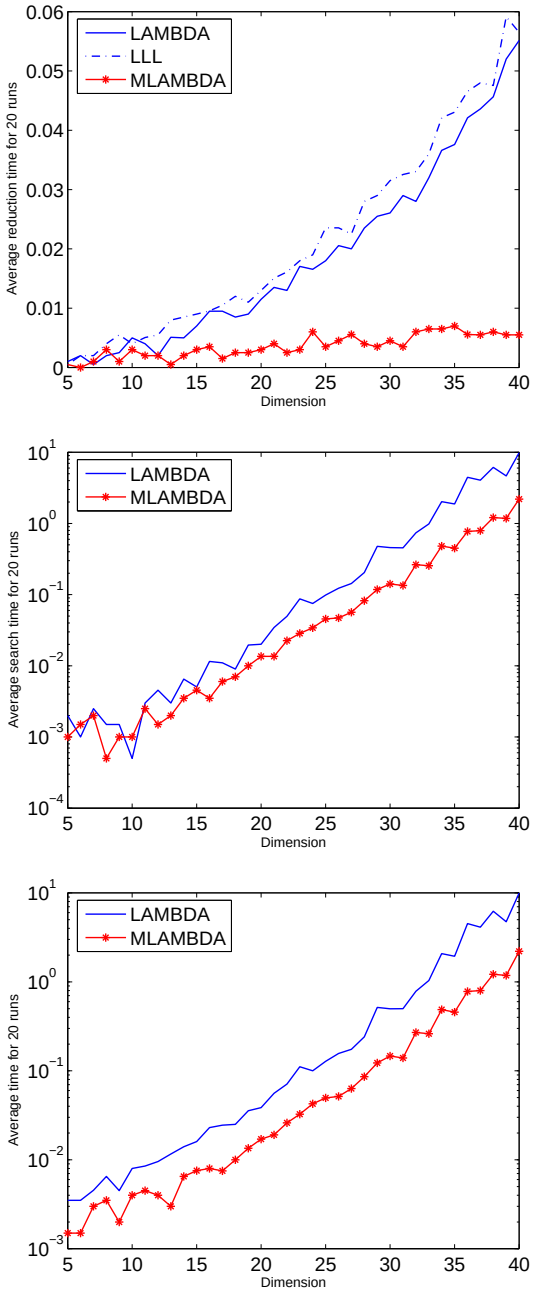


Figure 5: Running time for Case 5

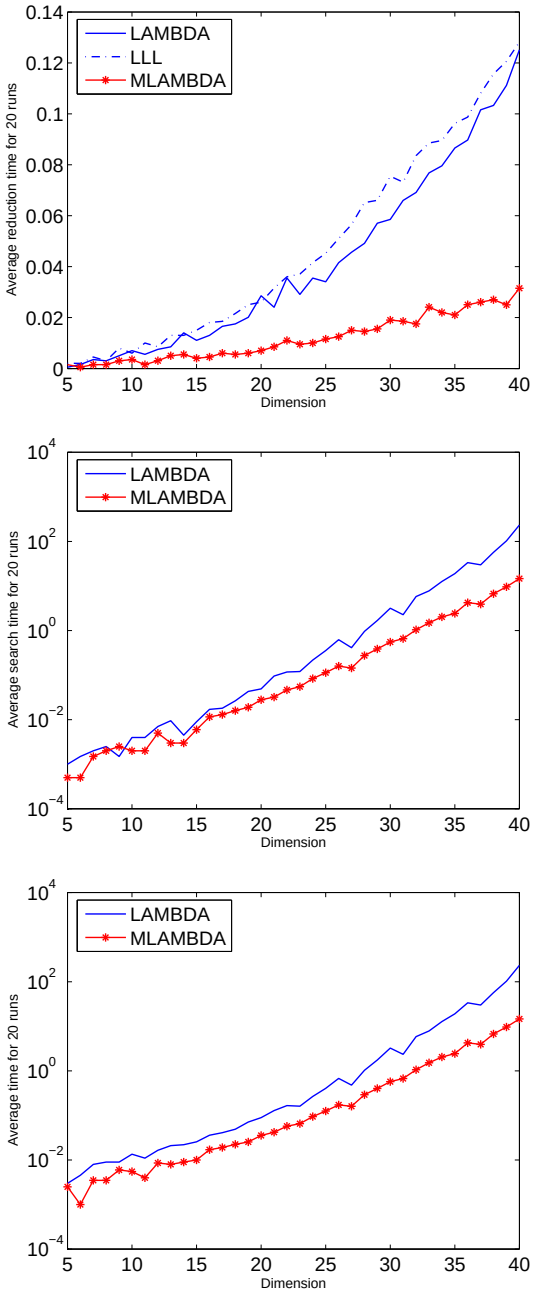
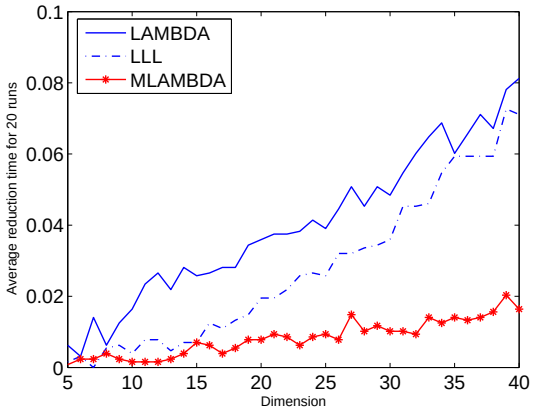
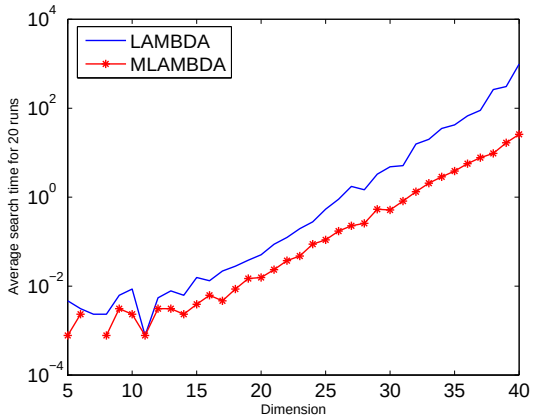


Figure 6: Running time for Case 6

Acknowledgement

This research was supported by NSERC of Canada Grant RGPIN217191-03. The authors would like to thank Erik Grafarend, Peter Joosten, Peiliang Xu and an anonymous referee for their very helpful comments and suggestions, and for pointing out some references.





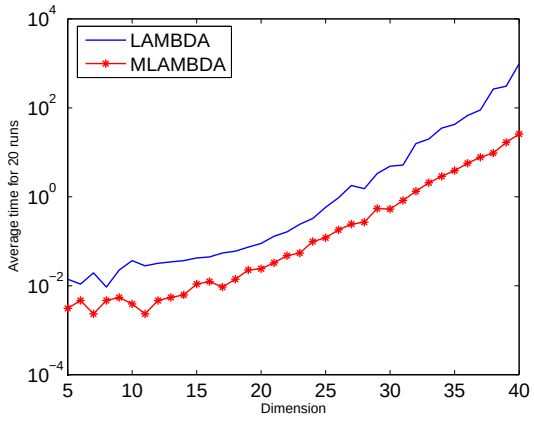
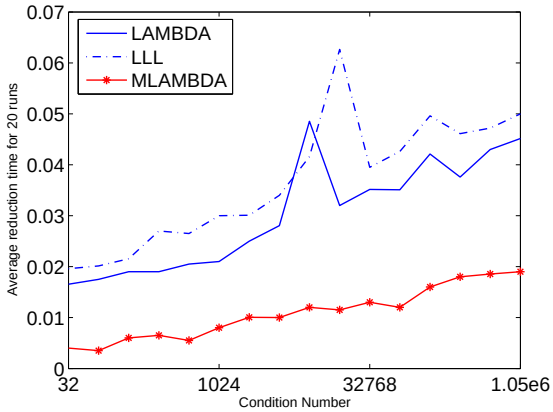
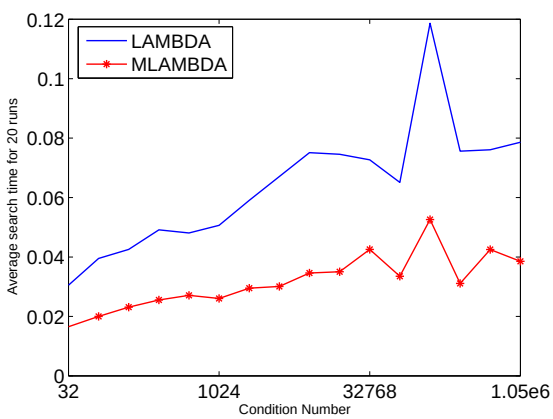


Figure 7: Running time for Case 7





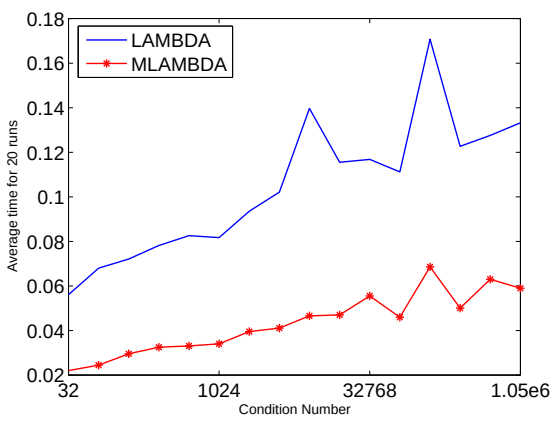


Figure 8: Running time for Case 8

References

Agrell E, Eriksson T, Vardy A, Zeger K (2002) Closest point search in lattices. IEEE Trans Inform Theory 48:2201–2214

Babai L (1986) On Lovász' lattice reduction and the nearesr lattice point problem. Combinatorica 6:1–13

De Jonge P, Tiberius, CCJM (1996) LAMBDA method for integer ambiguity estimation: im-

plementation aspects. In Delft Geodetic Computing Center LGR-Series, No.12

Fincke U, Pohst M (1985) Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Mathematics of Computation 44:463–471

Frei E, Beutler G (1990) Rapid static positioning based on the fast ambiguity resolution approach "FARA": theory and first results. Manus Geod 15:325–356

Golub GH, Van Loan CF (1996) Matrix Computations, 3rd edition. The Johns Hopkins University Press, Baltimore, Maryland

Grafarend EW (2000) Mixed integer-real valued adjustment(IRA) problems. GPS Solut 4:31–45

Hassibi A, Boyed S (1998) Integer parameter estimation in linear models with applications to GPS. IEEE Trans Signal Proc, 46:2938–2952

Joosten P, Tiberius CCJM (2002) LAMBDA: FAQs. GPS Solut 6:109–114

Landau H, Euler H (1992) On-the-fly ambiguity resolution for precise differential positioning. Proc ION GPS-92, pp. 607–613

Lenstra HW Jr (1981) Integer programming with a fixed number of variables. Tech Rep 81-03, Dept Math, Univ Amsterdam, The Netherlands

Lenstra AK, Lenstra HW Jr, Lovász L (1982) Factoring polynomials with rational coefficients. Math Ann 261:515–534

Liu LT, Hsu HT, Zhu YZ, Ou JK (1999). A new approach to GPS ambiguity decorrelation. J Geod 73:478–490

Lou L, Grafarend EW (2003) GPS integer ambituity resolution by various decorrelation methods. Zeitschrift fur Vermessungswesen 128:203–211

Pohst M (1981) On the computation of lattice vector of minimal length, successive minima and reduced bases with applications. ACM SIGSAM Bulletin, 15:37–44

Schnorr CP, Euchner M (1994) Lattice basis reduction: improved practical algorithms and solving subset sum problems. Mathematical Programming, 66:181–199

Teunissen PJG (1993) Least-squares estimation of the integer GPS ambiguities. Invited lecture, Section IV Thery and Methodology, IAG General Meeting, Beijing, China. Also in Delft Geodetic Computing Centre LGR series, No. 6, 16pp

Teunissen PJG (1995a) The invertibe GPS ambiguity transformation. Manus Geod 20:489–497

Teunissen PJG (1995b) The least-squares ambiguity decorrelation adjustment: a method for fast GPS ambiguity estitmation. J Geod 70:65–82

Teunissen PJG (1998) GPS carrier phase ambiguity fixing concepts. In Teunissen P and Kleusberg A (eds), GPS for Geodesy, 2nd edition, Springer-Verlag, New York, pp 317–388

Teunissen PJG (1999) An optimality property of the integer least-squares estimator. J Geod 73:587–593

van Emde Boas P (1981) Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Rep 81-04, Mathematisch Instituut, Amsterdam, The Netherlands.

Viterbo E, Biglieri E (1993) A universal decoding algorithm for lattice codes. Quatorzieme colloque GRETSI, 611–614, Juan-Les-Pins, France, Sept 1993

Viterbo E, Boutros J (1999) A universal lattice code decoder for fading channels. IEEE Trans Inform Theory, 45:1639–1642

Xu P (2001) Random simulation and GPS decorrelation. J Geod 75:408–423

Xu P, Cannon E, Lachapelle G (1995) Mixed integer programming for the resolution of GPS carrier phase ambiguities. IUGG95 Assembly, Boulder, 2-14 July. Also in Technical Report Nr. 2000.2, Department of Geodesy and Geoinformatics, Universität Stuttgart, Germany.