MLAMBDA: A Modified LAMBDA Method for Integer Ambiguity Determination

Xiao-Wen Chang, Xiaohua Yang, Tianyang Zhou, *McGill University, Canada*

BIOGRAPHIES

Dr. Chang is Associate Professor of Computer Science at McGill University. He holds a B.Sc and an M.Sc in Computational Mathematics from Nanjing University, China, and a Ph.D in Computer Science from McGill University. His main research area is scientific computing with particular emphasis on matrix computations and applications. He has been involved in GNSS research since 1998 and his current interests include various estimation techniques for GNSS positioning and fast algorithms for ambiguity determination. His publications can be found at www.cs.mcgill.ca/∼chang.

Mr. Yang is currently a Ph.D student in Computer Science at McGill University. He holds a B.Sc and an M.Sc in Computational Mathematics from Nanjing University, China. His research interests include integer least squares estimation.

Mr. Zhou is currently an M.Sc student in Computer Science at McGill University. He holds a B.Sc in Computer Science from Nanjing University, China. He is interested in numerical algorithms for integer least squares estimation.

ABSTRACT

The celebrated LAMBDA method has been widely used in GNSS for fixing integer ambiguities. For real time kinematic GNSS applications with high dimensions, the computational speed is crucial. In this paper a modified LAMBDA method (MLAMBDA) is presented. Several strategies are proposed to reduce the computational complexity. Numerical simulations show that MLAMBDA is (much) faster than LAMBDA.

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 celebrated 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. ILS problems may also arise from other applications, such as communications, cryptography and lattice design et al, see, e.g., Agrell et al. (2002). So they can also be solved by the LAMBDA method.

For real time kinematic GNSS applications and other applications with high dimensions, computational speed is crucial. In this paper a modified LAMBDA method (MLAMBDA) is presented to improve the computational efficiency of the LAMBDA method. The LAMBDA method consists of two stages. The first stage is to transfer the original ILS problem to a new one by mean of the so called Z-transformations. Since this stage decorrelates the ambiguities, it is called "decorrelation" in the GNSS literature. But actually it does more than decorrelation (see section 2.1), thus we prefer to call this stage "reduction" instead. The second stage is to search the optimal estimate or a few optimal estimates of the parameter vector over a hyper ellipsoidal region, and so it is called "search". The goal of the first stage is to make the second stage

ION 61st Annual Meeting
The MITRE Corporation & Draper Laboratory,
27-29 June 2005, Cambridge, MA

more efficient. Our MLAMBDA method will improve both of these two stages.

The rest of this paper is organized as follows. In section 2 we introduce the LAMBDA method. In section 3 we show how to effectively reduce the computational cost of the LAMBDA method by several strategies: symmetric pivoting, greedy selection, lazy transformations, and shrinking. And we present the new algorithm in detail. In section 4 we give the numerical simulation results. Finally we give a summary in section 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 $\boldsymbol{A}$, sometimes we use a_{ij} and sometimes we use $\boldsymbol{A}(i,j)$. 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 $\boldsymbol{a} \in \mathbb{Z}^n$ (the double differenced integer ambiguity vector in the GNSS context) and $Q_{\hat{\boldsymbol{a}}} \in \mathbb{R}^{n\times n}$ 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.

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.

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. 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

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

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 integer matrix $\boldsymbol{Z}$ to reach two goals which are crucial for 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)—this is decorrelation; (ii) the diagonal entries of $\bar{\boldsymbol{D}}$ are distributed in nondecreasing order if possible, i.e., one strives for

$$\bar{d}_n \le \bar{d}_{n-1} \le \cdots \le \bar{d}_1. \tag{5}$$

Thus this reduction stage does more than decorrelation. 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 $|\bar{l}_{ij}| \leq 1/2$.

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 $|(\boldsymbol{LZ})(i, j)| \leq 1/2$, then computes $\boldsymbol{Z}_{ij}^T\hat{\boldsymbol{a}}$ and $\boldsymbol{ZZ}_{ij}$, 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 $\text{L}^\text{T}\text{DL}$ factorization have to be updated.

Suppose we partition the $\text{L}^\text{T}\text{DL}$ factorization of $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ as follows

$$\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L},$$

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

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 $\text{L}^\text{T}\text{DL}$ factorization

$$\boldsymbol{P}_{k,k+1}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{P}_{k,k+1} = \bar{\boldsymbol{L}}^T\bar{\boldsymbol{D}}\bar{\boldsymbol{L}}$$

where

$$\bar{\boldsymbol{L}} = \begin{bmatrix} \boldsymbol{L}_{11} & & \\ \bar{\boldsymbol{L}}_{21} & \bar{\boldsymbol{L}}_{22} & \\ \boldsymbol{L}_{31} & \bar{\boldsymbol{L}}_{32} & \boldsymbol{L}_{33} \end{bmatrix}, \ \bar{\boldsymbol{D}} = \begin{bmatrix} \boldsymbol{D}_1 & & \\ & \bar{\boldsymbol{D}}_2 & \\ & & \boldsymbol{D}_3 \end{bmatrix}, \quad (6)$$

$$\bar{\boldsymbol{D}}_2 = \begin{bmatrix} \bar{d}_k & \\ & \bar{d}_{k+1} \end{bmatrix},$$

$$\bar{d}_{k+1} = d_k + l_{k+1,k}^2 d_{k+1}, \quad \bar{d}_k = \frac{d_k}{\bar{d}_{k+1}} d_{k+1}, \quad (7)$$

$$\bar{\boldsymbol{L}}_{22} \equiv \begin{bmatrix} 1 & \\ \bar{l}_{k+1,k} & 1 \end{bmatrix}, \quad \bar{l}_{k+1,k} = \frac{d_{k+1} l_{k+1,k}}{\bar{d}_{k+1}}, \quad (8)$$

$$\bar{\boldsymbol{L}}_{21} = \begin{bmatrix} -l_{k+1,k} & 1 \\ \frac{d_k}{\bar{d}_{k+1}} & \bar{l}_{k+1,k} \end{bmatrix} \boldsymbol{L}_{21}$$
$$= \begin{bmatrix} -l_{k+1,k} & 1 \\ \frac{d_k}{\bar{d}_{k+1}} & \bar{l}_{k+1,k} \end{bmatrix} \boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1), \quad (9)$$

$$\bar{\boldsymbol{L}}_{32} = \boldsymbol{L}_{32}\boldsymbol{P}$$
$$= \begin{bmatrix} \boldsymbol{L}(k+2\!:\!n, k+1) & \boldsymbol{L}(k+2\!:\!n, 1\!:\!k) \end{bmatrix}. \quad (10)$$

If we have

$$\bar{d}_{k+1} < d_{k+1} \quad (11)$$

(this implies that $d_k < d_{k+1}$), the permutation is performed. This does not guarantee that $\bar{d}_{k+1} \leq \bar{d}_k$, 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 $1/2$, after the permutation the bounds still hold, except that $\bar{l}_{k+1,k}$ (see Eqn. (8)) may not be bounded by $1/2$ any more.

Now we write the above operations as an algorithm.

Algorithm 2.2 (Permutations). Given the $\boldsymbol{L}$ and $\boldsymbol{D}$ factors of the $\text{L}^\text{T}\text{DL}$ 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. (7), $\hat{\boldsymbol{a}} \in \mathbb{R}^n$, and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$. This algorithm computes the updated $\boldsymbol{L}$ and $\boldsymbol{D}$ factors in (6) after $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$'s two rows k and $k+1$ and two columns k and $k+1$ are interchanged. It also interchanges the kth element and the $(k+1)$th element of $\hat{\boldsymbol{a}}$ and $\boldsymbol{Z}$'s two columns k and $k+1$.

function: $[\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \text{PERMUTE}(\boldsymbol{L}, \boldsymbol{D}, k, \delta, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
$\eta = \boldsymbol{D}(k, k)/\delta$ // see Eqn. (7)
$\lambda = \boldsymbol{D}(k+1, k+1)\boldsymbol{L}(k+1, k)/\delta$ // see Eqn. (8)
$\boldsymbol{D}(k, k) = \eta\boldsymbol{D}(k+1, k+1)$ // see Eqn. (7)
$\boldsymbol{D}(k+1, k+1) = \delta$
$\boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1)$ // see Eqn. (9)
$= \begin{bmatrix} -\boldsymbol{L}(k+1, k) & 1 \\ \eta & \lambda \end{bmatrix} \boldsymbol{L}(k\!:\!k+1, 1\!:\!k-1)$
$\boldsymbol{L}(k+1, k) = \lambda$
swap columns $\boldsymbol{L}(k+2\!:\!n, k)$ and $\boldsymbol{L}(k+2\!:\!n, k+1)$
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 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 (11) 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 of the covariance matrix $\boldsymbol{Q}_{\hat{\boldsymbol{z}}} = \boldsymbol{Z}^T\boldsymbol{Q}_{\hat{\boldsymbol{a}}}\boldsymbol{Z} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$, 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 $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$
$\boldsymbol{Z} = \boldsymbol{I}$
$k = n-1$; $k1 = n-1$
while $k > 0$
  if $k \le k1$
    for $i = k+1 : n$
      $[\boldsymbol{L}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \mathrm{GAUSS}(\boldsymbol{L}, i, k, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
    end
  end
  $\bar{\boldsymbol{D}}(k+1, k+1)$
  $= \boldsymbol{D}(k,k) + \boldsymbol{L}(k+1,k)^2\boldsymbol{D}(k+1,k+1)$
  if $\bar{\boldsymbol{D}}(k+1,k+1) < \boldsymbol{D}(k+1,k+1)$
    $[\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] =$
    $\mathrm{PERMUTE}(\boldsymbol{L}, \boldsymbol{D}, k, \bar{\boldsymbol{D}}(k+1,k+1), \hat{\boldsymbol{a}}, \boldsymbol{Z})$
    $k1 = k$; $k = n-1$
  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):

$$f(\boldsymbol{z}) \stackrel{\mathrm{def}}{=} (\boldsymbol{z}-\hat{\boldsymbol{z}})^T\boldsymbol{Q}_{\hat{\boldsymbol{z}}}^{-1}(\boldsymbol{z}-\hat{\boldsymbol{z}}) \le \chi^2. \tag{12}$$

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. (12) gives

$$f(\boldsymbol{z}) = (\boldsymbol{z}-\hat{\boldsymbol{z}})^T\bar{\boldsymbol{L}}^{-1}\bar{\boldsymbol{D}}^{-1}\bar{\boldsymbol{L}}^{-T}(\boldsymbol{z}-\hat{\boldsymbol{z}}) \le \chi^2. \tag{13}$$

Define

$$\bar{\boldsymbol{z}} = \boldsymbol{z} - \bar{\boldsymbol{L}}^{-T}(\boldsymbol{z}-\hat{\boldsymbol{z}}), \tag{14}$$

giving

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

or equivalently

$$\bar{z}_n = \hat{z}_n, \bar{z}_i = \hat{z}_i + \sum_{j=i+1}^{n}(z_j - \bar{z}_j)\bar{l}_{ji}, i = n-1, n-2, \ldots, 1, \tag{15}$$

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. (13) with Eqn. (14) that

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

or equivalently

$$f(\boldsymbol{z}) = \frac{(z_1-\bar{z}_1)^2}{\bar{d}_1} + \frac{(z_2-\bar{z}_2)^2}{\bar{d}_2} + \cdots + \frac{(z_n-\bar{z}_n)^2}{\bar{d}_n} \le \chi^2. \tag{16}$$

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

$$\mathrm{lb}(z_n) \le z_n \le \mathrm{ub}(z_n), \tag{17}$$

$$\vdots$$

$$\mathrm{lb}(z_i) \le z_i \le \mathrm{ub}(z_i), \tag{18}$$

$$\vdots$$

$$\mathrm{lb}(z_1) \le z_1 \le \mathrm{ub}(z_1) \tag{19}$$

where

$$\begin{aligned}
&\mathrm{lb}(z_n) = \bar{z}_n - \bar{d}_n^{1/2}\chi,\\
&\mathrm{ub}(z_n) = \bar{z}_n + \bar{d}_n^{1/2}\chi,\\
&\mathrm{lb}(z_i) = \bar{z}_i - \bar{d}_i^{1/2}\Big[\chi^2 - \sum_{j=i+1}^{n}(z_j-\bar{z}_j)^2/\bar{d}_j\Big]^{1/2},\\
&\mathrm{ub}(z_i) = \bar{z}_i + \bar{d}_i^{1/2}\Big[\chi^2 - \sum_{j=i+1}^{n}(z_j-\bar{z}_j)^2/\bar{d}_j\Big]^{1/2},
\end{aligned}$$

for $i = n-1, n-2, \ldots, 1$. Note that (19) is equivalent to (16).

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 section 3.2.

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 is not greater than $n+1$, 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 (15), producing the first integer vector $\boldsymbol{z}^{(1)}$. 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, $\check{\boldsymbol{a}} = \boldsymbol{Z}^{-1}\hat{\boldsymbol{z}}$ (cf. Eqn. (2)) is needed. For the details about this computation, see De Jonge and Tiberius (1996), sections 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 section 3.1 we show how to improve the reduction process and in section 3.2 we show how to improve the search process. The combined two new processes forms our modified LAMBDA method—MLAMBDA.

3.1 Modified reduction process

3.1.1 Symmetric pivoting strategy

In order to strive for (5), the reduction algorithm (Algorithm 2.3) in section 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 $Q_{\hat{a}}$ 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 L^{T}DL 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 L^{T}DL 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

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

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

giving the L^{T}DL factorization of $\boldsymbol{Q}$ with symmetric pivoting. Note that the L^{T}DL 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), section 4.2.9. But in the latter, the pivot element is chosen to be the largest element, rather than the smallest one.

Algorithm 3.1 (L^{T}DL 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 $\boldsymbol{Q}(:,q)$
  $\boldsymbol{Q}(k, 1:k-1) = \boldsymbol{Q}(k, 1:k-1)/\boldsymbol{Q}(k,k)$
  $\mathrm{Tmp} = \boldsymbol{Q}(k, 1:k-1)^T * \boldsymbol{Q}(k,k) * \boldsymbol{Q}(k, 1:k-1)$
  $\boldsymbol{Q}(1:k-1, 1:k-1) = \boldsymbol{Q}(1:k-1, 1:k-1) - \mathrm{Tmp}$
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 L^{T}DL factorization with symmetric pivoting, we start reduction. In order to make reduction more efficient, we would like to further reduce the number of permutations. As we have seen in section 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. (11)) 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 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

$$k = \arg \min_{1 \le j \le n-1} \{\bar{d}_{j+1}/d_{j+1} : \bar{d}_{j+1} < d_{j+1}\}. \quad (20)$$

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

Here is a simple example, for which a reader can check that the LAMBDA's reduction method will perform 4 permutations, while the above greedy selection method will perform only 2:

$$\boldsymbol{Q}_{\hat{\boldsymbol{a}}} = \begin{bmatrix} 1 & 0 & 0.04 \\ & 1 & 0.04 \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 999 & \\ & & 1001 \end{bmatrix} \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 0.04 & 0.04 & 1 \end{bmatrix}.$$

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. (9)) are changed and the two columns of $\boldsymbol{L}(k+1:n, k:k+1)$ (see Eqn. (10)) 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. (8)), 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. (7)). 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 takes place, we will 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:

$$\text{Criterion 1: } \frac{d_k}{d_{k+1}} < 1.$$

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 an integer Gauss transformation is applied to $l_{k+1,k}$ when both Criterion 1 and the following Criterion 2 are satisfied:

Criterion 2: $l_{k+1,k}$ is changed by last permutation.

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 $l_{k,k-1}$, $l_{k+1,k}$, $l_{k+2,k+1}$. 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 sections 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 $k = 1 : n-1$
    //Check if Criterion 1 is satisfied
    if $\frac{\boldsymbol{D}(k,k)}{\boldsymbol{D}(k+1,k+1)} < 1$
      //Check if Criterion 2 is satisfied
      if ChangeFlag$(k+1) = 1$
        $[\boldsymbol{L}, \hat{\boldsymbol{a}}, \boldsymbol{Z}]$
        $= \mathrm{GAUSS}(\boldsymbol{L}, k+1, k, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
        $\bar{\boldsymbol{D}}(k+1,k+1) = \boldsymbol{D}(k,k)$
        $+\boldsymbol{L}(k+1,k)^2\boldsymbol{D}(k+1,k+1)$
        ChangeFlag$(k+1) = 0$
      end
      tmp $= \frac{\bar{\boldsymbol{D}}(k+1,k+1)}{\boldsymbol{D}(k+1,k+1)}$
      if tmp < minratio
        $i = k$ // see Eqn. (20)
        minratio = tmp
      end
    end
  end
  if minratio = 1
    break while loop
  end
  $[\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{a}}, \boldsymbol{Z}]$
  $= \mathrm{PERMUTE}(\boldsymbol{L}, \boldsymbol{D}, i, \bar{\boldsymbol{D}}(k+1,k+1), \hat{\boldsymbol{a}}, \boldsymbol{Z})$
  Set ChangeFlag(i:i+2) to ones
end
//Apply integer Gauss transformation to $\boldsymbol{L}$'s
//strictly lower triangular entries
for $k = 1 : n-1$
  for $i = k+1 : n$
    $[\boldsymbol{L}, \hat{\boldsymbol{a}}, \boldsymbol{Z}] = \mathrm{GAUSS}(\boldsymbol{L}, i, k, \hat{\boldsymbol{a}}, \boldsymbol{Z})$
  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 ellip-

soidal region is found, the corresponding $f(\boldsymbol{z})$ in (13) 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 ((Teunissen, 1995b, section 2.4) and (De Jonge and Tiberius, 1996, section 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 (18) at level i. Specifically, if $\bar{z}_i \leq \lfloor \bar{z}_i \rceil$, we use the following search order:

$$\lfloor \bar{z}_i \rceil, \lfloor \bar{z}_i \rceil - 1, \lfloor \bar{z}_i \rceil + 1, \lfloor \bar{z}_i \rceil - 2, \cdots, \qquad (21)$$

otherwise, we use

$$\lfloor \bar{z}_i \rceil, \lfloor \bar{z}_i \rceil + 1, \lfloor \bar{z}_i \rceil - 1, \lfloor \bar{z}_i \rceil + 2, \cdots. \qquad (22)$$

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. section 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. (16)). 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. (16)). 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 of (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($\boldsymbol{L}, \boldsymbol{D}, \hat{\boldsymbol{z}}, p$)
$k = n$; dist(k) = 0
endSearch = false
maxDist = $+\infty$ // maxDist: current χ^2
count = 0 // count: the number of candidates
Initialize an $n \times n$ zero matrix $\boldsymbol{S}$ for computing $\bar{\boldsymbol{z}}(k)$
$\bar{\boldsymbol{z}}(n) = \hat{\boldsymbol{z}}(n)$ // see Eqn. (15)
$\boldsymbol{z}(n) = \lfloor \bar{\boldsymbol{z}}(n) \rceil$; $y = \bar{\boldsymbol{z}}(n) - \boldsymbol{z}(n)$
step(n) = sgn(y); imax = p
while endSearch = false
//newDist = $\sum_{j=k}^{n}(z_j - \bar{z}_j)^2/d_j$
newDist = dist(k) + $y^2/\boldsymbol{D}(k,k)$
if newDist < maxDist
if $k \neq 1$
// **Case 1:** move down
$k = k - 1$
// dist(k) = $\sum_{j=k+1}^{n}(z_j - \bar{z}_j)^2/d_j$
dist(k) = newDist
$T = (\boldsymbol{z}(k+1) - \bar{\boldsymbol{z}}(k+1))\boldsymbol{L}(k+1, 1\!:\!k)$
//$\boldsymbol{S}(k, 1\!:\!k) = \sum_{j=k+1}^{n}(z_j - \bar{z}_j)\boldsymbol{L}(j, 1:k)$
$\boldsymbol{S}(k, 1\!:\!k) = \boldsymbol{S}(k+1, 1\!:\!k) + T$
$\bar{\boldsymbol{z}}(k) = \hat{\boldsymbol{z}}(k) + \boldsymbol{S}(k,k)$ // see Eqn. (15)
$\boldsymbol{z}(k) = \lfloor \bar{\boldsymbol{z}}(k) \rceil$; $y = \bar{\boldsymbol{z}}(k) - \boldsymbol{z}(k)$
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) = $\boldsymbol{z}(1:n)$
// Store $f(\boldsymbol{z})$
fun(count) = newDist
else
Optis(:, imax) = $\boldsymbol{z}(1:n)$
fun(imax) = newDist
imax = arg $\max_{1\leq i\leq p}$fun(i)
maxDist = fun(imax)
end
$\boldsymbol{z}(1) = \boldsymbol{z}(1) + \mathrm{step}(1)$ //next valid integer
$y = \bar{\boldsymbol{z}}(1) - \boldsymbol{z}(1)$

```
        step(1) = −step(1) − sgn(step(1))
        //cf. Eqs (21) and (22)
      end
  else
      // Case 3: exit or move up
      if k = n
        endSearch = true
      else
        k = k + 1        //move up
        //next valid integer
        z(k) = z(k) + step(k) //next valid integer
        y = z̄(k) − z(k)
        step(k) = −step(k) − sgn(step(k))
        //cf. Eqs (21) and (22)
      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 MLAMBDA method given in section 3 and did numerical simulations to compare its running time with that of the original 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/). All our computations were performed in MATLAB 7.0.1 on a Pentium 4, 3.20GHz PC with 1GB memory running Windows XP Professional.

The real vector $\hat{\boldsymbol{a}}$ and the symmetric positive definite matrix $\boldsymbol{Q}_{\hat{\boldsymbol{a}}}$ were constructed as follows:

$$\hat{\boldsymbol{a}} = \texttt{randn}(n,1), \boldsymbol{Q}_{\hat{a}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}, l_{ij} = \texttt{randn},\ i > j, \tag{23}$$

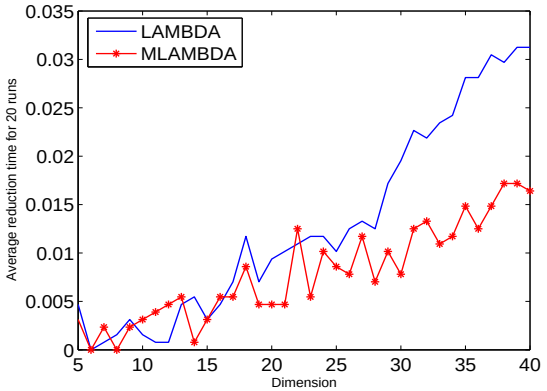
where $\texttt{randn}(n,1)$ is a MATLAB built-in function to generate a vector of $\boldsymbol{n}$ random entries which are normally distributed, $\boldsymbol{L}$ is a unit lower triangular matrix, and $\boldsymbol{D}$ is generated in four different ways:

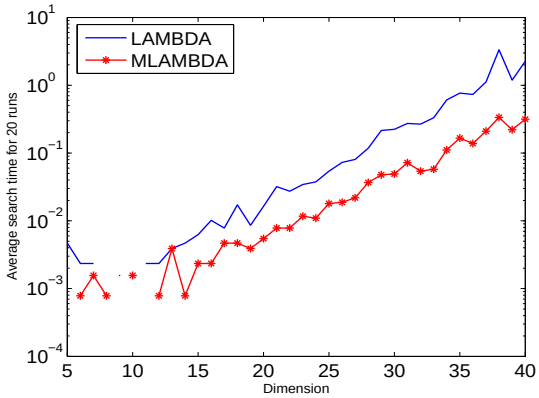
- Case 1 : $\boldsymbol{D} = \text{diag}(d_i),\ d_i = |\texttt{randn}|$
- Case 2 : $\boldsymbol{D} = \text{diag}(n^{-1}, (n-1)^{-1}, \ldots, 1^{-1})$
- Case 3 : $\boldsymbol{D} = \text{diag}(1^{-1}, 2^{-1}, \ldots, n^{-1})$
- Case 4 : $\boldsymbol{D} = \text{diag}(100, 100, 100, 0.1, 0.1, \ldots, 0.1)$

We took dimension $n = 5:40$. For each n, we gave 20 runs and in each run we computed two optimal ILS estimates by both the LAMBDA package and our MLAMBDA method. The results about average running time (in seconds) are given in Figure 1–Figure 4. For each case we give three plots, corresponding average reduction time, average search time, and average time (i.e., total time including both reduction time and search time), respectively. In the figures which show the average running time for the search process, there are no values for some lower dimensional cases. The reason is as follows. For these cases, the running time of each of the 20 runs is lower than 1 millisecond and MATLAB just counts it as zero, and so the value of the average running time is zero, but the logarithm of zero is not defined. For clarity, we also give the average running time for dimension $n = 40$ in Table 1.

In Case 1 (see Figure 1), each $\boldsymbol{D}$ is random. We see that MLAMBDA is faster than LAMBDA except for a few lower dimensional cases where both methods are very fast. When the dimension n reaches 40, MLAMBDA is about seven times as fast as LAMBDA. In Case 2 (see Figure 2), $\boldsymbol{D}$ are in the order opposite to that we strive for (see Eqn. (5)). We see that the search process of each method takes extremely longer time than that in Case 1 and dominates the total computational cost when $n > 20$. But MLAMBDA is faster than LAMBDA and when $n = 40$ the former is more than 20 times faster than the latter. In Case 3 (see Figure 3), $\boldsymbol{D}$ are in the right order, and both methods take much less time than in Case 2. When $n = 40$, MLAMBDA is about 15 times as fast as LAMBDA. In Case 4 (see Figure 4), $\boldsymbol{D}$ has a big gap between the third and the fourth element. We observe that MLAMBDA is much faster than LAMBDA even for lower dimensional cases. When $n = 40$, the former is about 18 times as fast as the latter.

We observed in our simulations that usually both the reduction process and the search process of MLAMBDA are faster than those of LAMBDA except in some lower dimensional cases where the cost of either method is small. As the dimension increases, usually the search time becomes more and more dominant of the total time, and the improvement of search process becomes more obvious. It appears that the search time increases exponentially while the reduction time increases linearly.





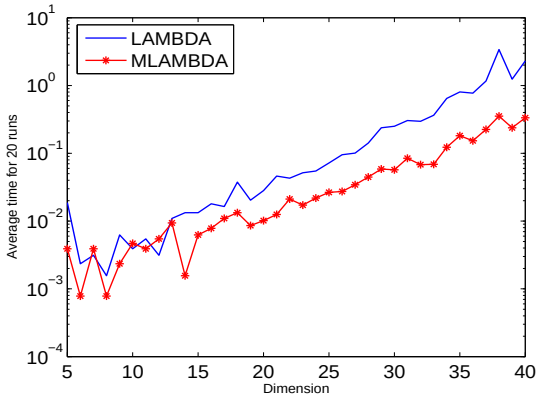
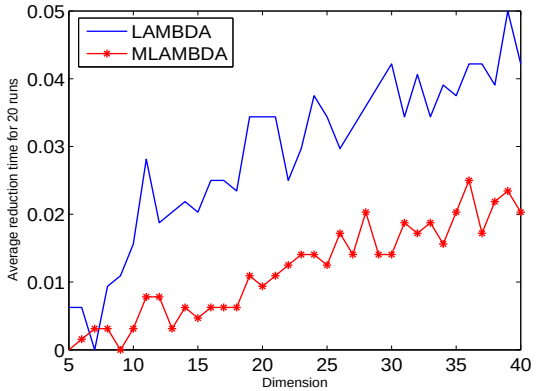
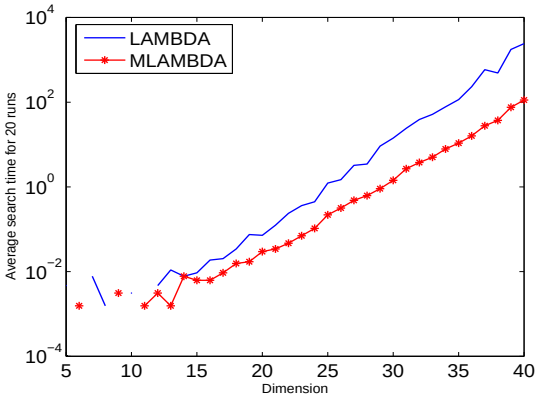


Figure 1: Running time for Case 1





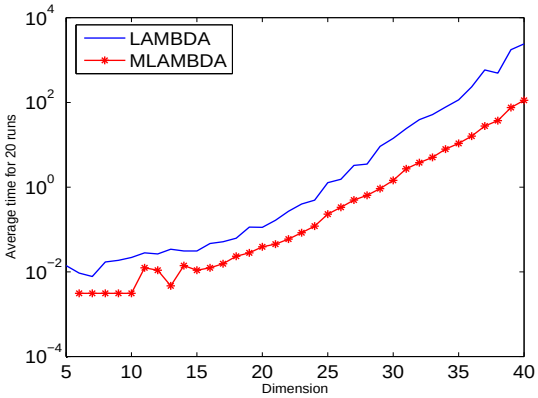
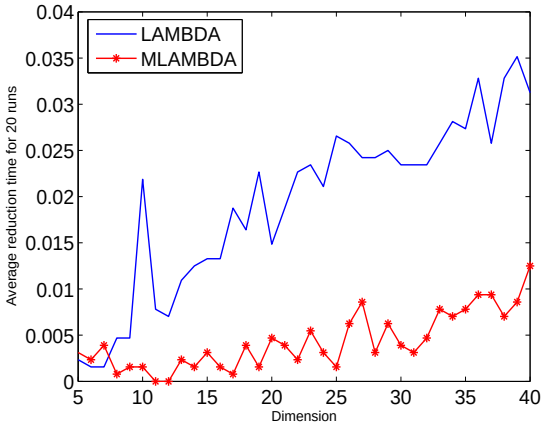
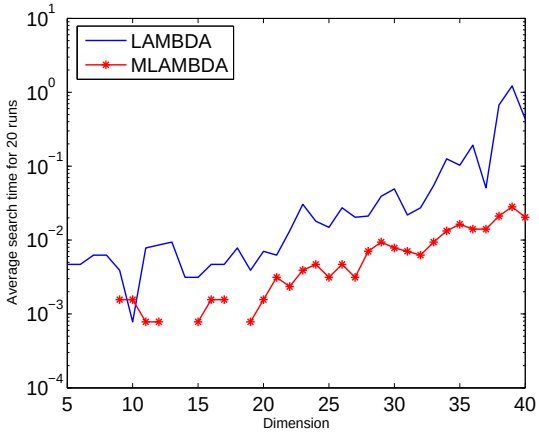


Figure 2: Running time for Case 2

	LAMBDA			MLAMBDA		
	reduction	search	total	reduction	search	total
Case 1	0.0313	2.2625	2.2938	0.0164	0.3164	0.3328
Case 2	0.0422	2419.8	2419.8	0.0203	112.6	112.62
Case 3	0.0313	0.4336	0.4649	0.0125	0.0203	0.0328
Case 4	0.0195	3.5821	3.6016	0.0102	0.2055	0.2157

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





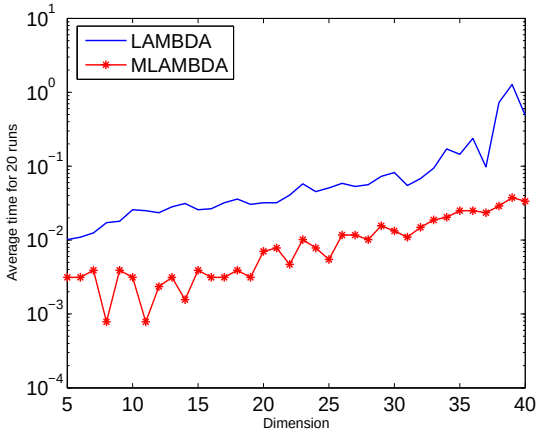
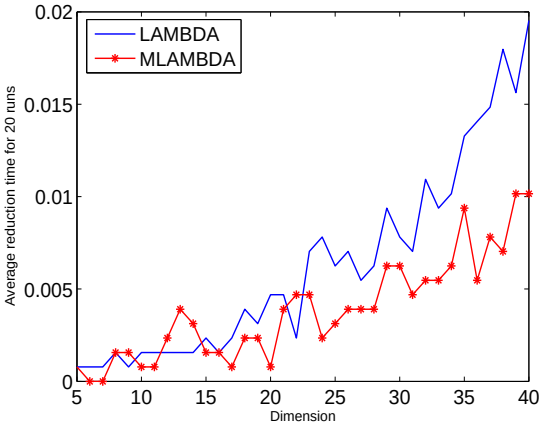
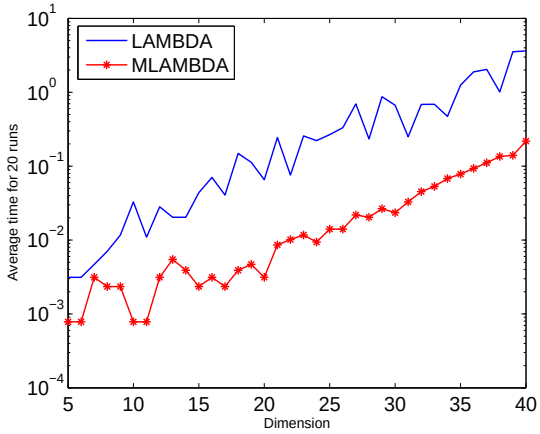


Figure 3: Running time for Case 3





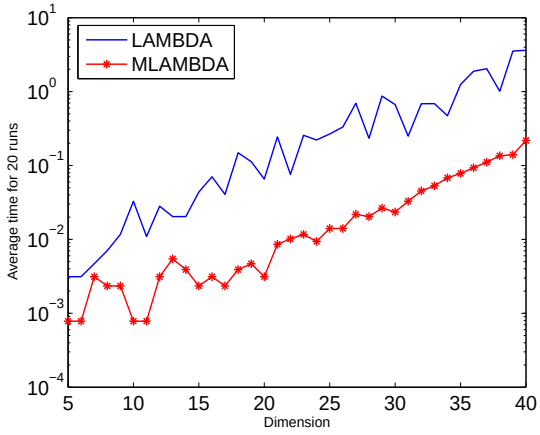


Figure 4: Running time for Case 4

5 SUMMARY

The well-known LAMBDA method has been widely used for integer least-squares estimation problems in positioning and navigation. It consists of two stages, reduction and search. In this paper we presented a modified LAMBDA method—MLAMBA, which improves both of those two stages. The keys to our algorithm are to compute the L^TDL factorization with symmetric pivoting, do the reduction by greedy selection and lazy transformation strategies, and shrink the ellipsoidal region during the search process. Numerical simulation showed that the MLAMBDA method is usually faster than the LAMBDA method implemented in Delft's LAMBDA package (MATLAB version 2.0), and in the high dimensional cases, the former

is much faster than the latter. 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 complete computational details for our new method. Hopefully a reader can implement the algorithms without difficulty.

ACKNOWLEDGMENTS

This research was supported by NSERC of Canada Grant RGPIN217191-03. The authors are grateful to Chris Paige for his encouragement and help in this work.

References

Agrell E, Eriksson T, Vardy A, and Zeger K (2002). Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214

De Jonge P and Tiberius, CCJM (1996). LAMBDA method for integer ambiguity estimation: implementation aspects. In *Delft Geodetic Computing Center LGR-Series, No.12*

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

Joosten P and Tiberius CCJM (2002). LAMBDA: FAQs. *GPS Solutions*, 6(1-2):109–114

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. *Manuscripta Geodatetica*, 20(6):489–497

Teunissen PJG (1995b). The least-squares ambiguity decorrelation adjustment: a method for fast GPS ambiguity estitmation. *J. of Geodesy*, 70(1-2):65–82

Teunissen PJG (1998). GPS carrier phase ambiguity fixing concepts. In Teunissen PJG 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. of Geodesy*, 73(11):587–593