An efficient implementation of the block Gram–Schmidt method

Yoichi Matsuo[1] Takashi Nodera[2]

(Received 31 October 2012; revised 24 June 2013)

Abstract

The block Gram–Schmidt method computes the QR factorisation rapidly, but this is dependent on block size m. We endeavor to determine the optimal m automatically during one execution. Our algorithm determines m through observing the relationship between computation time and complexity. Numerical experiments show that our proposed algorithms compute approximately twice as fast as the block Gram–Schmidt method for some block sizes, and is a viable option for computing the QR factorisation in a more stable and rapid manner.

Subject class: 65F10, 65M12

Keywords: block Gram–Schmidt algorithm, optimal block size, parallel computing

http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/6327 gives this article, © Austral. Mathematical Soc. 2013. Published August 26, 2013, as part of the Proceedings of the 16th Biennial Computational Techniques and Applications Conference. ISSN 1446-8735. (Print two pages per sheet of paper.) Copies of this article must not be made otherwise available on the internet; instead link directly to this URL for this article.

Contents

1 Introduction

The orthogonalisation process or the QR factorisation by the Gram–Schmidt method is arguably one of the most important processes in a linear algebraic computation and there are numerous studies on this subject [3, 4, 6, 12, 13, 14, 15]. It is frequently used to solve least squares and eigenvalue problems which arise in signal processing, structural mechanics or magnetohydrodynamics [1, 2, 4, 9, 12]. Other applications that are commonly used for solving linear systems are Krylov subspace methods, like GMRES [10], which are derived from partial difference equations by using finite difference or finite element discretisations [7, 8, 11].

The GMRES method resorts to an orthogonalisation process with the modified Gram–Schmidt method (MGS), and hence we focus on a more efficient and faster implementation of the QR factorisation using the block Gram–Schmidt method (BGS) which is used to solve the least squares problem or singular value decomposition. The BGS enables us to compute the QR factorisation quickly by partitioning matrix X into columns and then orthogonalizing into blocks of size m. Stewart [13] illustrated how the computation time of the QR factorisation is shortened by employing the BGS. We find that the BGS is two or three times faster than the classical Gram–Schmidt (CGS) in some instances. This is because the BGS has matrix-matrix products and the algorithms are implemented with a level-3 BLAS (basic linear algebra subprogram). According to Yokozawa [15], a level-3 BLAS is approximately four times faster than a level-2 BLAS which is used to calculate the matrix-vector products in the CGS. Here we use the BGS algorithm with a level-3 BLAS [13].

The BGS has the same high parallelisation as the CGS [14]. Recent developments in supercomputers often make it mandatory to use parallelisation when employing various methods. The MGS is used in many methods and there is a lot of literature on parallel versions [3, 4]. However, the MGS is characterized by low parallelisation and it is difficult to attain high parallelisation.

The BGS is suitable for use in parallel environments, but has issues of its own. If a less than optimal block size is chosen, then the computation time of the BGS lengthens significantly. Moreover, since the optimal block size m is not consistent when employing the BGS, it is necessary to determine m. There is no unique m for any matrix X when using the BGS and it is necessary to determine m accurately through trial and error.

Section 2 is a summary of the BGS. In Section 3 new schemes for automatically determining optimal block sizes are proposed and Section 4 shows how the BGS is parallelized and adapted to these new schemes. Section 5 analyses data from our numerical experiments and conclude that our proposed methods are effective.

2 The block Gram–Schmidt algorithm

The algorithm and properties of the BGS are summarized in this section. Let X be an $n \times n$ matrix, let m be the block size (to simplify, divide n by m), let X_{block} be an $n \times m$ matrix, and let the orthogonal matrix Q be $n \times h$. The BGS step for k loops ($h = km$, $k = 1, \ldots, n/m - 1$) is

$$\begin{aligned} R_{12_k} &= Q_k^T X_{\text{block}_k}\,, && (1)\\ \widehat{Y}_k &= X_{\text{block}_k} - QR_{12_k}\,, && (2)\\ Y_k R_{22_k} &= \widehat{Y}_k\,. && (3)\end{aligned}$$

To orthogonalize $\widehat{Y}_k$ the CGS is employed in (3). From (2) and (3), matrix X_{block} satisfies

$$X_{\text{block}_k} = Q_k R_{12_k} + Y_k R_{22_k}\,. \qquad (4)$$

Sometimes Y_k lacks orthogonality in the BGS. In such a case, applying (4) to matrix Y_k gives

$$\begin{aligned} Y_k &= Q_k S_{12_k} + Z_k S_{22_k}\,, && (5)\\ X_{\text{block}_k} &= Q_k R_{12_k} + (Q_k S_{12_k} + Z_k S_{22_k}) R_{22_k}\,. && (6)\end{aligned}$$

According to Stewart [13], usually one reorthogonalisation is enough for Y_k. Equation (6) is one loop of the BGS with reorthogonalisation. Repeating this process for every X_{block_k} results in the QR factorisation.

Let one multiplication be one unit of computational complexity. Assume that every column requires reorthogonalisation. At this point, the computational complexity for one loop is considered. Matsuo and Nodera [6] determined that the number of multiplications required to calculate one loop, that is the one loop computational complexity, is

$$f(h) = 4nmh + 2nm\,(m-1) + (h+m)\,m^2\,. \qquad (7)$$

The computational complexity in its entirety, S_{BGS}, is the sum over all k of the number of multiplications in one loop,

$$S_{\text{BGS}} = \sum_{k=1}^{n/m-1} \left[4nm^2k + 2nm\,(m-1) + (k+1)\,m^3\right] . \tag{8}$$

The computation time of the BGS increases linearly with the number of loops $k = h/m$. We ascertain this by observing the computational complexity of the BGS. Consider the function $f(h)$ in (7); $f(h)$ is a linear function in h and thus the increase in the complexity of the BGS will be linear in k. Matsuo and Nodera [6] present a numerical example which showed that the graph of the computation time is very similar to a linear function.

Figure 1 illustrates the relationship between computation time and block size. It shows that the computation time of the BGS is different for each block size, which is a typical characteristic of the BGS. The performance of the BLAS is affected by block size [15].

3 Optimal block size

The computation speed of the BGS changes significantly depending on block size. Conventionally, optimal block size is determined through trial and error. To address this issue we propose two methods, Scheme A and Scheme B, that allow us to determine optimal block size automatically.

Scheme A uses (7), determined by Matsuo and Nodera [6], to calculate the computation time c_i from the one loop computational complexity:

$$c_i = \frac{4nm_ih + 2nm_i(m_i-1) + (h+m_i)m_i^2}{t_i} , \tag{9}$$

where m_i ($m_i = 2^i$, $i = 2, \ldots, 10$) is a sample block size and t_i is the computation time of the BGS for one loop. In this scheme we choose m from

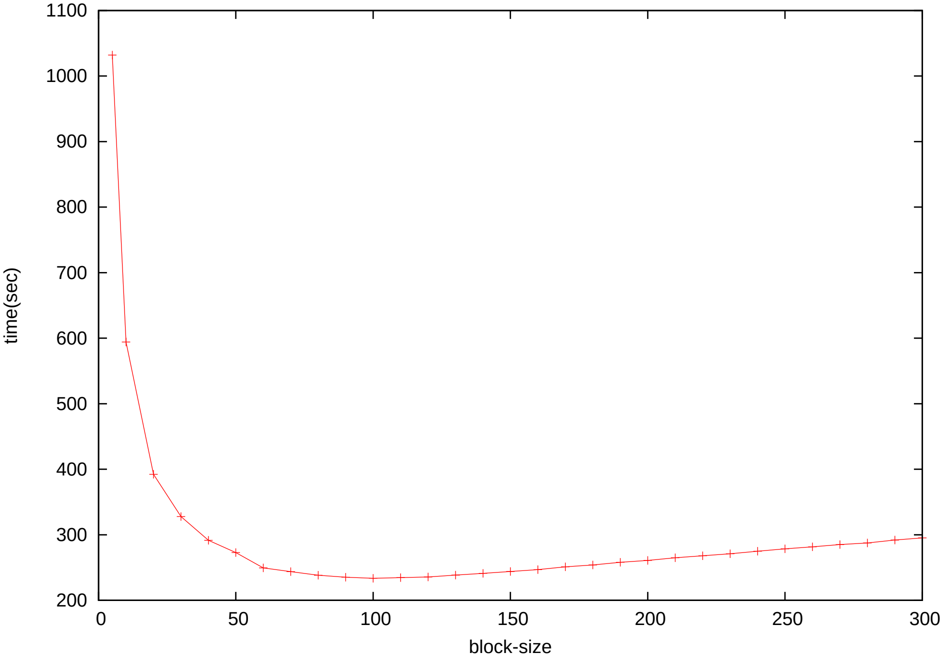

Figure 1: Relationship between block size and computation time

sample block sizes m_i. The m that minimizes c_i results in the shortest total computation time for the BGS. The block size which minimizes c_i is selected and named m_A.

Scheme B, as shown by Matsuo and Nodera [6], determines m by approximating the curve in Figure 1. From (8) and (9) the total computation time T_i is estimated by

$$T_i = c_i S_{\text{BGS}_{m_i}} . \tag{10}$$

Let $m_i = 4, 8, 16$, let A be a matrix and let $\mathbf{x}$ and $\mathbf{b}$ be vectors such that

$$A[i,j] = m_i^{4-j} , \quad \mathbf{x} = (x_1, x_2, x_3)^T , \quad \mathbf{b} = (T_1, T_2, T_3)^T .$$

By solving $A\mathbf{x} = \mathbf{b}$ we approximate the curve in Figure 1 with a quadratic function. The optimal block size for Scheme B is determined from

$$m_B = \min_{s\in[0,\frac{1}{2}n]} x_1 s^2 + x_2 s + x_3 \,. \tag{11}$$

One issue is that sometimes these schemes find an inadequate block size that is either very large or small. This is because these schemes use c_i, defined in (9). Since t_i is very small, c_i can be affected significantly by t_i. Calculating T_i in (10) produces a poor estimate. In conclusion, these schemes are unable to identify the optimal block size. We improve on these methods by implementing two loops of the BGS to compute sample points and a quartic function. As mentioned in Section 2, the computation time of the BGS increases linearly. By using two loops, the slope of the linear function (7) is calculated and a better estimation of T_i is obtained. Let $m_i = 2, 4, 8, 16, 32$. For each m_i, two loops are executed with the BGS. Then, the computation time for the first loop t_{i_0} and the computation time for the second loop t_{i_1}, are measured. The increased computation time for one loop is constant and is approximated by

$$a = (t_{i0} - t_{i1})/m_i \,, \tag{12}$$

The computation time of the BGS is approximated by

$$T_i := \frac{1}{2} n^2 a + t_{i0} - a(h - m_i) \,. \tag{13}$$

Let A be a matrix and let $\mathbf{x}$ and $\mathbf{b}$ be vectors such that

$$A[i,j] = m_i^{6-j} \,, \quad \mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T \,, \quad \mathbf{b} = (T_1, T_2, T_3, T_4, T_5)^T \,. \tag{14}$$

In the same manner as Scheme B, the linear equation $A\mathbf{x} = \mathbf{b}$ is used to solve the problem. The curve in Figure 1 is approximated with a quartic function. Hence, in this scheme, which we name Scheme C, the optimal block size m is determined from

$$m = \min_{s\in[0,\frac{1}{2}n]} x_1 s^4 + x_2 s^3 + x_3 s^2 + x_4 s + x_5 \,. \tag{15}$$

Algorithm 1: The new method, Scheme C, for estimating optimal block size

```
Data: $X \in \mathbb{R}^{n\times n}$
Result: $m$
begin
    $m_1 \longleftarrow 1$ ;
    for $i = 1 : 5$ do
        $m_i \longleftarrow m_i * 2$ ;
        for $j = 0 : 1$ do
            $t_{ij} \longleftarrow$computation time of the BGS one loop;
        end
        $a \longleftarrow (t_{i0} - t_{i1})/m_i$ ;
        $\mathbf{T}[i] \longleftarrow \frac{1}{2}n^2a + t_{i0} - a(h - m)$ ;
        for  $j = 5 : 1$ do
            $A[ij] = m_i^{j-1}$ ;
        end
        $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$ ;
        $\mathbf{b} = (T_1, T_2, T_3, T_4, T_5)^T$ ;
    end
    solve $A\mathbf{x} = \mathbf{b}$ ;
    $m \longleftarrow \min_{s\in[0,\frac{1}{2}n]} x_1s^4 + x_2s^3 + x_3s^2 + x_4s + x_5$ ;
end
```

The new Scheme C is detailed in Algorithm 1. In Algorithm 1 the changes in the BGS's computation time are approximated by polynomial functions using sample points T_i which are estimated by observing the complexity of the BGS. Instead of c_i, we estimated T_i more precisely by using a in (12). This algorithm is easily applied to the QR decomposition. To achieve this, Algorithm 1 is simply implemented before the BGS algorithm. This algorithm needs some additional computation time to determine m, because a number of calculations with small block size m_i must be evaluated.

4 The parallel block Gram–Schmidt algorithm

When dealing with large scale problems in a parallel environment, it is integral to parallelize the BGS to speed up computation time. Vanderstraeten [14], Gudula [3] and Katagiri [4] developed a parallelized Gram–Schmidt process. Here we parallelize the BGS (PBGS) by employing a column-wise distribution. In the column-wise distribution, all processing elements (PE) have columns of matrix Q and X_{block} [4].

We parallelize the new scheme, detailed in Section 3, for the PBGS, and name this parallelized scheme PBGS-m, or Scheme D. The PBGS-m is different from the PBGS in that the PBGS-m determines the optimal block size automatically. The critical points of this method are summarized as follows. Firstly, one PE sends the X_{block} to the other PEs, and the PBGS with sample point m_i for two loops, is executed. Next, the total computation time T_i of the PBGS is approximated from (12) and (13) according to the method proposed in Section 3, using the information from samples T_i and m_i. Finally, m is determined from (15). Algorithm 2 details the PBGS-m algorithm, many parts of which are similar to Scheme C, detailed in Algorithm 1. In Algorithm 2, “MyPID” refers to the PE number.

5 Numerical experiments

In this section the BGS is evaluated with Algorithm 1 and the PBGS is evaluated with Algorithm 2. The test matrices selected for these experiments were BCSSTK02, 06, 15 and 19 from the Matrix Market [5], with sizes 112×112, 420×420, 3948×3948 and 11948×11948, respectively. These are nonsymmetric real matrices which arise from generalized eigenvalue problems in structural engineering. The QR factorisation for these matrices are often used to find eigenpairs. Algorithms were programmed in C-language with double precision. Block size m_{TRIAL} is determined by trial and error. Block

Algorithm 2: PBGS-m method, Scheme D, for estimating optimal block size

```
Data: $X \in \mathbb{R}^{n\times n}$
Result: $m$
begin
    $m_1 \longleftarrow 1$ ;
    for $i = 1 : 5$ do
        $m_i \longleftarrow m_i * 2$ ;
        for $j = 0 : 1$ do
            $t_{ij} \longleftarrow$ computation time of the PBGS loop;
        end
    end
    if MyPID=0 then
        $a \longleftarrow (t_{i0} - t_{i1})/m_i$ ;
        $\mathbf{T}[i] \longleftarrow \frac{1}{2}n^2 a + t_{i0} - a(h - m)$ ;
        for  $j = 5 : 1$ do
            $A[ij] \longleftarrow m_i^{j-1}$ ;
        end
        $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$ ;
        $\mathbf{b} = (T_1, T_2, T_3, T_4, T_5)^T$ ;
        solve $A\mathbf{x} = \mathbf{b}$ ;
        $m \longleftarrow \min_{m\in[0,\frac{1}{2}n]} x_1 m^4 + x_2 m^3 + x_3 m^2 + x_4 m + x_5$ ;
    end
end
```

Table 1: Trial and error BGS compared to BGS with Schemes A, B and C, using BCSSTK matrices.

Problem	m_{TRIAL}	t_m	m_A	t_{m_A}	m_B	t_{m_B}	m_C	t_{m_C}
BCSSTK02	60	0.0008	32	0.0013	41	0.0014	51	0.0009
BCSSTK06	40	0.065	128	0.076	41	0.066	15	0.077
BCSSTK15	80	35.0	1024	64.9	43	39.2	53	37.3
BCSSTK19	100	872	1024	1137	43	1035	55	959

sizes m_A, m_B, m_C and m_D are from the BGS with Scheme A by Matsuo and Nodera [6], the BGS with Scheme B by Matsuo and Nodera [6], the BGS with Scheme C detailed in Algorithm 1, and the PBGS with Scheme D detailed in Algorithm 2, respectively. The t is computation time in seconds and PE is the number of processor elements.

5.1 The BGS with optimal block size

We compare the QR decomposition of a conventional BGS versus a BGS with Schemes A, B and C to illustrate the effectiveness of the new method. The algorithms were run on a Sun Fire X2250 with a 4 Gigabyte main memory.

The data from these numerical experiments are shown in Table 1. Block sizes m_A in Table 1 are very large as the sizes of the matrices are large. In contrast, m_B has similar values for each problem. Block sizes m_A, m_B and m_C took the nearest value m for BCSSTK02, 15 and 19. The computation times t_{m_C} were faster than t_{m_A} and t_{m_B}. Scheme C's performance was the best in this experiment.

Table 2: Computation times of PBGS and PBGS-$\mathfrak{m}$, or Scheme D, with eight PE and for BCSSTK15 with RATIO $=$ (t for PBGS)/(t for PBGS-$\mathfrak{m}$) .

Algorithm	$\mathfrak{m}$	t (sec)	RATIO
PBGS	50	35.7	1.96
PBGS	100	21.2	1.17
PBGS	200	15.6	0.86
PBGS	300	16.2	0.89
PBGS-$\mathfrak{m}$	180	18.2	1.00

5.2 The PBGS with optimal block size

Numerical experiments were implemented to illustrate the effectiveness of the PBGS-$\mathfrak{m}$ Scheme D. The derived algorithm was run on a six core AMD Opteron processor 2439 SE with 12 processors and an AuthenticAMD with a 2.8 GHz CPU. We used eight PEs to execute our program. The results of the numerical experiments are shown in Table 2.

The data in Table 2 for the PBGS-$\mathfrak{m}$ was not optimal. However, the PBGS-$\mathfrak{m}$ was twice as fast as the PBGS when $\mathfrak{m} = 50$. Moreover, when $\mathfrak{m} = 100$, the PBGS-$\mathfrak{m}$ was a little bit faster than the PBGS. When $\mathfrak{m} = 200$ and 300, the PBGS-$\mathfrak{m}$ performed approximately 10% slower than the PBGS.

5.3 Least squares problems

We now show the results of our proposed methods for least squares problems and compare the accuracy and computation times of BGS and Scheme C. The numerical environment is the same as in Subsection 5.1. Test matrices are taken from the Harwell–Boeing test collection [1, pp. 266–268], namely WELL1850 and ILLC1850. Both matrices are of size 1850×712 . WELL1850 is well conditioned and ILLC1850 is a moderately conditioned problem. The least

Table 3: Computation times of BGS and Scheme C for WELL1850 and ILLC1850.

	WELL1850		ILLC1850	
Algorithm	m	t (sec)	m	t (sec)
BGS	50	0.63	50	0.63
BGS	100	0.75	100	0.74
Scheme C	78	0.70	76	0.69

square problems are defined by taking the exact solution to be $\mathbf{x} = (1, \ldots, 1)^T$, and $\mathbf{b} = A\mathbf{x}$. The main step of the algorithms [1, 9] is solving $R\mathbf{x} = Q^T\mathbf{b}$ after calculating the QR decomposition, $A = QR$. The derived algorithm was run on a six core AMD Opteron processor 2439 SE with a 2.8 GHz CPU. We present the numerical results in Table 3. Here all methods provide high accuracy with a relative error of order 10^{-15}. Our proposed method finds an appropriate block size m, and the computation time of BGS is shorter than Scheme C when $m = 100$ and slightly longer than Scheme C when $m = 50$.

6 Conclusion

The block Gram–Schmidt and parallel block Gram–Schmidt methods compute the QR factorisation rapidly. However, determining optimal block size has always been an issue. To address this issue we developed a new method that automatically determines block size, and the results of our numerical experiments, tabulated in Section 5, suggest that it is effective. The proposed algorithms are based on the computation time and the complexity of the BGS and are not influenced by the properties of the matrices, and this enables us to compute the QR factorisation in a stable and efficient manner.

In future studies we will apply our proposed algorithm to larger least squares problems for the analysis of large data that scientists regularly encounter.

References

[1] Björck, Å., *Numerical Methods for Least Squares Problems*, SIAM, (1996). C477, C487, C488

[2] Elden, L., and Park, H., Block Downdating of Least Squares Solutions, *SIAM J. Matrix Anal. Appl.*, 15:1018–1034 (1994). doi:10.1137/S089547989223691X C477

[3] Rünger, G., and Schwind, M., Comparison of Different Parallel Modified Gram–Schmidt Algorithms, *Euro-Par 2005, LNCS* 3648:826–836 (2005). doi:10.1007/11549468_90 C477, C478, C484

[4] Katagiri, T., Performance Evaluation of Parallel Gram–Schmidt Re-orthogonalization Methods, *VECPAR 2002, LNCS* 2565:302–314 (2003). doi:10.1007/3-540-36569-9_19 C477, C478, C484

[5] Matrix Market, Mathematical and Computational Sciences Division, Information Technology Laboratory of the National Institute of Standards and Technology, USA. http://math.nist.gov/MatrixMarket/ C484

[6] Matsuo, Y. and Nodera, T., The Optimal Block-Size for the Block Gram–Schmidt Orthogonalization, *J. Sci. Tech*, 49:348–354 (2011). C477, C479, C480, C481, C486

[7] Moriya, K. and Nodera, T., The DEFLATED-GMRES(m, k) Method with Switching the Restart Frequency Dynamically, *Numer. Linear Alg. Appl.*, 7:569–584 (2000). doi:10.1002/1099-1506(200010/12)7:7/8<569::AID-NLA213>3.0.CO;2-8 C477

[8] Moriya, K. and Nodera, T., Usage of the convergence test of the residual norm in the Tsuno–Nodera version of the GMRES algorithm, *ANZIAM J.*, 49:293–308 (2007). doi:10.1017/S1446181100012852 C477

[9] Liu, Q., Modified Gram–Schmidt-based Methods for Block Downdating the Cholesky Factorization, *J. Comput. Appl. Math.*, 235:1897–1905 (2011). doi:10.1016/j.cam.2010.09.003 C477, C488

[10] Saad, Y. and Schultz, M. H., GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.*, 7:856–869 (1986). doi:10.1137/0907058 C477

[11] Shiroishi, J. and Nodera, T., A GMRES(m) Method with Two Stage Deflated Preconditioners, *ANZIAM J.*, 52:C222–C236 (2011). `http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/3984` C477

[12] Leon, S. J., Björck, Å., and Gander, W., Gram–Schmidt Orthogonalization: 100 years and more, *Numer. Linear Algebra Appl.*, 20:492–532 (2013). doi:10.1002/nla.1839 C477

[13] Stewart, G. W., *Block Gram–Schmidt Orthogonalization*, SIAM J. Sci. Comput., 31:761–775 (2008). doi:10.1137/070682563 C477, C478, C479

[14] Vanderstraeten, D., An Accurate Parallel Block Gram-Schmidt Algorithm without Reorthogonalization, *Numer. Lin. Alg. Appl.*, 7:219–236 (2000). doi:10.1002/1099-1506(200005)7:4<219::AID-NLA196>3.0.CO;2-L C477, C478, C484

[15] Yokozawa, T., Takahashi, T., Boku, T. and Sato, M., Efficient Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication, (in Japanese) *Information Processing Society of Japan (IPSJ), Computing System*, 1:61–72 (2008). C477, C478, C480

Author addresses

1. **Yoichi Matsuo**, School of Fundamental Science and Technology, Graduate School of Science and Technology, Keio University, 3-14-1

Hiyoshi, Kohoku, Yokohama, Kanagawa, 223-8522, Japan.
mailto:matsuo@math.keio.ac.jp

2. **Takashi Nodera**, Department of Mathematics, Faculty of Science and Technology, Keio University, 3-14-1 Hiyoshi, Kohoku, Yokohama, Kanagawa, 223-8522, Japan.
mailto:nodera@math.keio.ac.jp