Numerical Solution of Differential Riccati Equations on Hybrid CPU-GPU Platforms

Peter Benner[1], Pablo Ezzatti[2], Hermann Mena[3], Enrique S. Quintana-Ortí[4], Alfredo Remón[4]

[1] *Fakultät für Mathematik, TU Chemnitz, D-09107 Chemnitz (Germany). E-mail:* `benner@mathematik.tu-chemnitz.de`.
[2] *Centro de Cálculo-Instituto de la Computación, Universidad de la República, 11.300-Montevideo (Uruguay). E-mail:* `pezzatti@fing.edu.uy`.
[3] *Departamento de Matemática, Escuela Politécnica Nacional, EC1701 Quito (Ecuador). E-mail:* `hermann.mena@epn.edu.ec`.
[4] *Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaime I, 12.071-Castellón (Spain). E-mails:* {`quintana,remon`}`@icc.uji.es`.

Abstract

In this paper we propose a differential Riccati equation (DRE) solver that uses a Lyapunov solver based on the matrix sign function. The algorithm combines an iterative solver with a refinement procedure, resulting in a mixed-precision algorithm that exploits the capabilities of both general-purpose multi-core processors and many-core GPUs, overlapping critical computations.

Keywords: Differential Riccati equations, Rosenbrock methods, matrix sing function, graphics processors, multi-core processors, control theory

1 Introduction

We consider the time-varying differential Riccati equations (DRE)

$$\begin{array}{rcl} \dot{X}(t) & = & Q(t) + X(t)A(t) + D(t)X(t) - X(t)S(t)X(t) \equiv F(t, X(t)), \\ X(t_0) & = & X_0, \end{array} \tag{1}$$

where $t \in [t_0, t_f]$, $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{m\times m}$, $Q(t) \in \mathbb{R}^{m\times n}$, $S(t) \in \mathbb{R}^{n\times m}$, and $X(t) \in \mathbb{R}^{m\times n}$. We assume that the coefficient matrices are piecewise continuous locally bounded matrix-valued functions which ensures existence of the solution and uniqueness of (1); see, e.g., [1, Thm. 4.1.6].

Symmetric DREs (i.e., with symmetric $Q(t)$, $S(t)$, and $D(t) = A(t)^T$ for all $t \in [t_0, T]$) arise in linear-quadratic optimal control problems such as LQR and LQG design with finite-time horizon, in H_∞ control of linear-time varying systems, as well as in differential games; see, e.g., [1, 5]. Unfortunately, in most control problems, fast and slow modes are present. This implies that the associated DRE will be fairly stiff which in turn demands for implicit methods to solve such DREs numerically. Matrix-valued algorithms based on generalizations of the BDF and Rosenbrock methods have proved to yield accurate solutions for large scale DREs arising in optimal control problems for parabolic partial differential equations [2, 6].

For the autonomous case, the Rosenbrock methods are more efficient than the BDFs mainly because they require only the solution of one Lyapunov equation per stage in each step, which is solved using a low-rank implementation of the ADI iteration [3, 4]. However, some applications require a large interval of integration and/or a small time step size, which turns these methods not feasible. Hence, we propose to use a hybrid CPU-GPU solver, based on the matrix sign function, to accelerate the computation of the Lyapunov equation in each step.

This paper is organized as follows. In Section 2, we briefly describe the application of the Rosenbrock method of order one to DREs. In Section 3, we review the sign function method for the solution of Lyapunov equations. In Section 4 numerical experiments are discussed. A brief summary and outlook on future work closes the paper.

2 Numerical solution of DREs

We focus on solving DREs arising in optimal control for parabolic partial differential equations. Typically the coefficient matrices of the DRE arising from these control problems have a certain structure (e.g. sparse, symmetric or low rank). The application of the Rosenbrock method of order one to an autonomous symmetric DRE of the form (1) yields:

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -Q - X_k S X_k - \frac{1}{h}X_k \tag{2}$$

where $X_k \approx X(t_k)$ and $\tilde{A}_k = A - SX_k - \frac{1}{2h}I$; see [4, 6] for details. In addition we assume,

$$\begin{array}{rcll} Q & = & C^TC, & C \in \mathbb{R}^{p\times n}, \\ S & = & BB^T, & B \in \mathbb{R}^{n\times m}, \\ X_k & = & Z_kZ_k^T, & Z_k \in \mathbb{R}^{n\times z_k}, \end{array} \tag{3}$$

with $p,\ m,\ z_k \ll n$. If we denote $N_k = [C^T\ Z_k(Z_k^TB)\ \sqrt{h^{-1}}Z_k]$, then the Lyapunov equation (2) results in

$$\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -N_kN_k^T, \tag{4}$$

where $\tilde{A}_k = A - B(Z_k(Z_k^TB))^T - \frac{1}{2h}I$. The procedure that is obtained from this ellaboration is given in Algorithm 2.1 Observing that $\mathrm{rank}(N_k) \leq p + m + z_k \ll n$, we can use the sign function method to solve (4), as described in the following subsection.

2.1 The sign function method

The matrix sign function is an efficient tool to solve stable Lyapunov equations. There are simple iterative schemes for the computation of the sign function. Among these, the Newton iteration described in Algorithm 2.2 is specially appealing for its simplicity, efficiency, parallel performance, and asymptotic quadratic convergence. However, even if A is sparse, $\{A_k\}_{k=1,2,\ldots}$ in general are full dense matrices and, thus, Algorithm 2.2 roughly requires $2n^3$ floating-point arithmetic operations (flops) per iteration.

Algorithm 2.1 Rosenbrock method of order one for DREs

```
Require: $A \in \mathbb{R}^{n\times n}$, $B$, $C$, $Z_0$ satisfying (3), $t \in [a,b]$, and $h$ step size.
Ensure: $(Z_k, t_k)$ such that $X_k \approx Z_k Z_k^T$, $Z_k \in \mathbb{R}^{n\times z_i}$ with $z_i \ll n$.
 1: $t_0 = a$.
 2: for $k = 0$ to $\lceil \frac{b-a}{h} \rceil$ do
 3:     $\tilde{A}_k = A - B(Z_k(Z_k^T B))^T - \frac{1}{2h} I$.
 4:     $N_k = [C^T \; Z_k(Z_k^T B) \; \sqrt{h^{-1}} Z_k]$.
 5:     Compute $Z_{k+1}$ by the sign function method such that the low rank factor product
        $Z_{k+1} Z_{k+1}^T$ approximates the solution of $\tilde{A}_k^T X_{k+1} + X_{k+1}\tilde{A}_k = -N_k N_k^T$ .
 6:     $t_{k+1} = t_k + h$.
 7: end for
```

On convergence $\tilde{S}$ satisfies $X \approx \tilde{S}^T\tilde{S}$. Initial convergence can be accelerated using several techniques. In our case, we employ a scaling defined by the parameter

$$c_k = \sqrt{\|A_k{}^{-1}\|_\infty / \|A_k\|_\infty}.$$

In the convergence test, τ is a tolerance threshold for the iteration that is usually set as a function of the problem dimension and the machine precision ϵ.

Algorithm 2.2 Matrix sign function for Lyapunov equations

```
Require: $A \in \mathbb{R}^{n\times n}$, $N \in \mathbb{R}^{j\times n}$.
Ensure: $\tilde{S}^T\tilde{S} \approx X$ such that $A^T X + XA = N * N^T$.
 1: $A_0 = A$, $\hat{S}_0 = N^T$.
 2: $k = 0$.
 3: repeat
 4:     $A_{k+1} = \frac{1}{\sqrt{2}}\left(A_k/c_k + c_k A_k{}^{-1}\right)$.
        Compute the rank-revealing QR (RRQR) decomposition
 5:     $\frac{1}{\sqrt{2c_k}}\left[\tilde{S}_k, \;\; c_k\tilde{S}_k(A_k^{-1})^T\right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$
 6:     $\tilde{S}_{k+1} \leftarrow U_s \Pi_s$
 7:     $k = k + 1$.
 8: until $\sqrt{\frac{\|A_{k+1}+I\|_\infty}{n}} < \tau \|A_k\|_\infty$
```

2.2 Matrix inversion

As shown in Algorithm 2.2, the application of Newton's method to the sign function requires, at each iteration, the computation of a matrix inverse.

In this section we introduce two different algorithms for matrix inversion: the traditional approach based on the LU factorization and the Gauss-Jordan elimination algorithm.

2.2.1 Traditional approach

The traditional approach to compute the inverse of a matrix $A \in \mathbb{R}^{n\times n}$ is based on Gaussian elimination (i.e., the LU factorization), and consist of the following three steps:

1. Compute the LU factorization $PA = LU$, where $P \in \mathbb{R}^{n\times n}$ is a permutation matrix, and $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$ are, respectively, unit lower and upper triangular factors [7].
2. Invert the triangular factor $U \rightarrow U^{-1}$.
3. Solve the system $XL = U^{-1}$ for X.
4. Undo the permutations $A^{-1} := XP$.

LAPACK [8] is a high-performance linear algebra library which provides routines that cover the functionality required in the previous steps. In particular, routine `getrf` yields the LU factorization (with partial pivoting) of a nonsingular matrix (Step 1), while routine `getri` computes the inverse matrix of A using the LU factorization obtained by `getrf` (Steps 2–4).

The computational cost of computing a matrix inverse following the previous four steps is $2n^3$ flops.

2.2.2 The Gauss-Jordan elimination algorithm

The Gauss-Jordan elimination algorithm (GJE) for matrix inversion is, in essence, a reordering of the computation performed by matrix inversion methods based on Gaussian elimination, and hence requires the same arithmetic cost. The convenience of the GJE based methods relies on that all its computations are well suited for parallelization.

Algorithm 2.3 Blocked Gauss-Jordan elimination algorithm for matrix inversion

```
Require: $A \in \mathbb{R}^{n\times n}$
1: $t_0 = a$.
2: for $k = 1$ to $\frac{n}{b}\rceil$ do
3:     Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ where $A_{00} \in \mathbb{R}^{(k-1)b\times(k-1)b}$, $A_{11} \in \mathbb{R}^{b\times b}$
4:     $[A_{01},\ A_{11},\ A_{21}]^T \leftarrow$ GEINGJ($[A_{01},\ A_{11},\ A_{21}]^T$)
5:     $A_{00} \leftarrow A_{00} + A_{01}A_{10}$
6:     $A_{20} \leftarrow A_{20} + A_{21}A_{10}$
7:     $A_{10} \leftarrow A_{11}A_{10}$
8:     $A_{02} \leftarrow A_{02} + A_{01}A_{12}$
9:     $A_{22} \leftarrow A_{22} + A_{21}A_{12}$
10:    $A_{12} \leftarrow A_{11}A_{12}$
11: end for
```

Algorithm 2.3 illustrates a blocked version of the GJE procedure for matrix inversion using the FLAME notation [9, 10, 11]. There $m(A)$ stands for the number of rows of matrix

A. We believe the rest of the notation to be intuitive; for further details, see [9, 10]. (A description of the unblocked version, called from inside the blocked one, can be found in [12]; for simplicity, we hide the application of pivoting during the factorization, but details can be found there as well.) The bulk of the computations in the procedure can be cast in terms of the matrix-matrix product, an operation with a high parallelism. Therefore, GJE is a highly appealing method for matrix inversion on emerging architectures like GPUs, where many computational units are available, provided a highly-tuned implementation of the matrix-matrix product is available.

3 Implementation

The computational effort of Algorithm 2.1 for the solution of DREs is concentrated on the solution of a Lyapunov equation per iteration, and more specifically, in the matrix inversion required at each iteration of the sign function method (Algorithm 2.2). Matrix inversion is computed by an hybrid implementation of the GJE algorithm (2.3) that exploits the CPU and GPU capabilities. The rest of operations are mainly matrix-matrix products that involve small matrices, so they can be computed efficiently on the multicore invoquing a parallel implemetation of BLAS.

4 Numerical Results

In this section we evaluate the parallel performance of Algorithm 2.1 on a platform consisting of two INTEL Xeon QuadCore E5410 processors at 2.33GHz, connected to an NVIDIA Tesla C1060 via a PCI-e bus.

Two parallel implementations has been tested, one for the multicore and a more sofisticated hybrid version:

- LAPACK(CPU): all the computations are performed on the CPU using LAPACK and BLAS kernels (MKL 10.2). Matrix inversion is obtained using LAPACK (a routine is based on Gaussian elimintation). In this implementation, parallelism is extracted via a multithreaded implementation of BLAS.

- Hybrid(CPU+GPU): computations are executed on the most convinient architecture. The GJE (Algorithm 2.3) is employed for matrix inversion. This implementation is based on the use of computational kernels from MKL (on the CPU) and NVIDIA CUBLAS (version 2.1) for the GPU.

Experiments employ single precision and execution time always includes the cost of data transfers between the host and the device (GPU) memory spaces.

We evaluate both implementations employing model STEEL_I from the Oberwolfach benchmark colletion at the University of Freiburg. This model arises in a manufacturing method for steel profiles. The goal is to design a control that yields moderate temperature gradients when the rail is cooled down. The mathematical model corresponds to the boundary control for a 2-D heat equation. A finite element discretization, followed by adaptive refinement of the mesh results in the examples in this benchmark. The dimensions of this problem are n = 5,177, m = 7, and p = 6.

Table 1 presents the total time in seconds required for solving the DRE associated with model STEEL_I on the interval $[0, 1]$. Three different step sizes (h) are tested: 0.1, 0.01 and 0.001, that resulted in 11, 101 and 1001 iterations on algorithm 2.1 respectively.

Step size (h)	LAPACK(CPU)		Hybrid(CPU+GPU)		Speed-up
	Inversion time	Total time	Inversion time	Total time	
0.1	2.13178e+02	2.13687e+02	1.65416e+02	1.65894e+02	1.288
0.01	1.63974e+03	1.64495e+03	1.29111e+03	1.29592e+03	1.269
0.001	1.65748e+04	1.66226e+04	1.30865e+04	1.31427e+04	1.264

Table 1: Performance obtained for the STEEL_I benchmark.

Results show that time dedicated to computations out of matrix inversions is residual, and that the presented hybrid implementation is approximately a 25% faster than the traditional LAPACK implementation.

Acknowledgment

Enrique S. Quintana-Ortí and Alfredo Remón were supported by projects PROMETEO 2009/013 and CICYT TIN2008-06570-C04. This work was partially done while Hermann Mena was visiting the Universidad Jaime I with the support from the program ”Pla de suport a la investigació 2009” from Universidad Jaime I.

References

[1] H. Abou-Kandil, G. Freiling, V. Ionescu and G. Jank, *Matrix Riccati Equations in Control systems Theory*, Birkhäuser, Basel, Switzerland, 2003.

[2] P. Benner and H. Mena, *BDF methods for large-scale differential Riccati equations*, In B. De Moor, B. Motmans, J. Willems, P. Van Dooren and V. Blondel editors. Proc. of Mathematical Theory of Network and Systems, MTNS 2004, 2004.

[3] P. Benner and H. Mena, *Rosenbrock methods for solving differential Riccati equations*, Tech. rep., Chemnitz Scientific Computing, TU Chemnitz, 2010, to appear.

[4] P. Benner and H. Mena, *Numerical solution of large scale differential Riccati Equations arising in optimal control problems*, Tech. rep., Chemnitz Scientific Computing, TU Chemnitz, 2010, to appear.

[5] A. Ichikawa and H. Katayama, *Remarks on the time-varying H_∞ Riccati equations*, Sys. Cont. Lett. 37(5):335-345, 1999.

[6] H. Mena, *Numerical Solution of Differential Riccati Equations Arising in Optimal Control Problems for Parabolic Partial Differential Equations*, PhD thesis, Escuela Politécnica Nacional, 2007 .

[7] G. Golub and C. V. Loan *Matrix Computations, 3rd Edition*, The Johns Hopkins University Press, Baltimore, 1996.

[8] E. Anderson and Z. Bai and J. Demmel and J. E. Dongarra and J. DuCroz and A. Greenbaum and S. Hammarling and A. E. McKenney and S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide*, SIAM 1992

[9] J. A. Gunnels and F. G. Gustavson and G. M. Henry and R. A. van de Geijn, *FLAME: Formal Linear Algebra Methods Environment*, ACM Trans. Math. Soft. vol. 27, num. 4, 2001.

[10] P. Bientinesi and J. A. Gunnels and M. E. Myers and E. S. Quintana-Ortí and R. A. van de Geijn, *The Science of Deriving Dense Linear Algebra Algorithms*, ACM Trans. Math. Soft. vol. 31, num. 1, 2005.

[11] University of Texas *http://www.cs.utexas.edu/~flame/*.

[12] E.S. Quintana-Ortí and G. Quintana-Ortí and X. Sun and R.A. van de Geijn *A note on parallel matrix inversion*, SIAM J. Sci. Comput., vol. 22, 2001.