DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No: DSIC-II/05/05 **Pages: 29**

Title: Solving Differential Riccati Equations by Using BDF Methods

Author (s): E.Arias, V. Hernández, J.J. Ibáñez and J. Peinado

Date: 31/05/05

Key Words: Differential Riccati Equations, Backward Differentiation Formula Methods, Algebraic Riccati Equations

VºBº **Author (s):**

Leader of Research Group

Vicente Hernández García

Solving Differential Riccati Equations Using BDF Methods

E. Arias, V. Hernández, J. J. Ibáñez and J. Peinado

Abstract

This technical report describes **three approaches for solving the Differential Riccati Equation (DRE),** by means of the Backward Differentiation Formula (BDF) and resolution of the corresponding implicit equation, using Newton's method. These approaches are based on: GMRES method, resolution of Sylvester equation and fixed point method. The role and use of **DRE** is especially important in optimal control, filtering, and estimation.

Keywords: Differential Riccati Equation, Backward Differentiation Formula Method, Algebraic Riccati Equation

1. Introduction

Consider the Differential Riccati Equation (DRE)

$$\begin{gathered}\dot{X}(t) = A_{21}(t) + A_{22}(t)X(t) - X(t)A_{11}(t) - X(t)A_{12}(t)X(t) = F(t,X), \\ X(t_0) = X_0, t_0 \le t \le t_f\end{gathered} \tag{1}$$

where $A_{11}(\cdot) \in R^{nxn}$, $A_{22}(\cdot) \in R^{mxm}$, $A_{12}(\cdot) \in R^{nxm}$, $A_{21}(\cdot) \in R^{mxn}$ and $X(\cdot) \in R^{mxn}$.

The DRE arises in several applications, in particular in control theory, for example the Linear Quadratic Optimal Control problem. In this case, the DRE has the following expression.

$$\dot{X}(t) = XA + A^T X + Q - XBR^{-1}B^T X, \tag{2}$$

where $A \in R^{nxn}$, $B \in R^{nxm}$, $Q = Q^T \in R^{nxn}$ is positive semidefinite, and $R = R^T \in R^{mxm}$ is positive definite, representing respectively, the state matrix, the input matrix, the state weight matrix and the input weight matrix, associated to the optimal control problem.

This report is concerned with the study and implementation of methods for solving the DRE (1) by numerical integration, using the BDF method. The result is an implicit scheme which solves the discrete version of the DRE according to the property that the discretization, of a polynomial differential equation, reduces it to a polynomial algebraic equation of the same degree.
An example of this methodology appeared in [Die92], known as DRESOL. This package is based on the IVP software LSODE, developed by Hindsmarsh [Hin82]. Several methods have been implemented for solving the Algebraic Riccati Equation (ARE); however, in the context of stiff DREs, one of the better choices for solving the associated ARE, is to apply implicit schemes based on Newton's or quasi-Newton's methods. In both cases, at each iteration step a Sylvester equation has to be solved

$$G_{11}Y - YG_{22} = H, \tag{3}$$

where $G_{11} \in R^{nxn}$, $G_{22} \in R^{mxm}$ and $H \in R^{nxm}$ change at each iteration, if Newton's method is used.

A standard method for solving the equation (3) is the Bartels-Stewart algorithm [BS72]. By using this approach, the matrices G_{11} and G_{22} are both reduced, via orthogonal matrices U and V, to Real Schur Form, obtaining the equivalent equation

$$(U^T G_{11} U)(U^T Y V) - (U^T Y V)(V^T G_{22} V) = U^T H V. \quad (4)$$

Once the quasi-triangular problem (4) is solved for $U^T Y V$, then Y is easily recovered.

This report is organized as follows. First, in Section 2, the state of the art on numerical methods for solving DREs is outlined. Section 3 describes the numerical integration methods considered in this work, that is, Luca Dieci methodology, and three additional approaches for solving the DRE. A theoretical study in terms of memory storage and flops requirements is included. Sequential implementations of the previous methods have been carried out using standard linear algebra libraries such as Basic Linear Algebra Subroutines (BLAS) [DDCHH88] and Linear Algebra PACKage (LAPACK) [ABB+94]. Section 4 presents the test battery and the experimental results. Finally, in Section 5 the conclusions and future work are outlined.

2. Numerical Methods for Solving DREs

The DRE occurs in several applications in different fields of science and engineering. However, it did not receive enough attention in the numerical literature until the mid seventies. Furthermore, the studies carried out consider time invariant systems far removed from the real world. Since then a great variety of methods have been proposed [KL85, Cho90]. These methods can be grouped into five classes

The vectorized approach. The idea is to vectorize the DRE and integrate the resulting system of differential equations using any numerical integration scheme. This approach is not suitable for large problems if an implicit method is used, because a large scale differential equation has to be solved.

Linearization. The linearization consists of transforming the DRE into a Linear Matrix Differential Equation (LMDE). Many numerical methods for DREs are based on this approach. In this case, if an invariant time system is considered, the linear associated differential system has constant coefficients. Thus, the numerical integration of the associated high dimension system can be replaced by the matrix exponential. In this group of methods, the works of Leipnik [Lei85] and Davison and Maki [DM73] are the most representative. However this approach does not work well if the differential equations are stiff (a detailed analysis can be found in [Vaug69]).

Chandrasekhar. This approach ([Chan76]) is applicable to symmetric time-invariant DREs. It is based on the transformation of (1) into two coupled system of nonlinear differential equations, called the Chandrasekhar system.

Superposition methods. This kind of method is based on the superposition rule of Riccati solutions [RW84,SW85]. The general solution of a DRE can be expressed as a nonlinear combination of at least five different solutions. This class of methods requires integration of the DRE several times with different initial conditions before applying the complex superposition formulas. Computational complexity is therefore too high to apply these formulae to large scale problems.

Matrix versions of ODE methods. These methods adapt known methods for solving ODEs, as Runge-Kutta methods or BDF methods, for solving DREs. Choi and Laub developed a set of algorithms presented in [CL90]. These algorithms reduced the computational cost using Newton's method. At each integration step these algorithms solve an ARE which is computed by solving several Sylvester equations [BS72,GNV79]. Luca Dieci [Die92] developed the DRESOL package for solving the DRE in a similar way to Choi and Laub. Because Luca Dieci's methodology is the starting point for the work presented in this technical report, further comments appear in the next Section.

Hamiltonian methods. By the end of the eighties and beginning of the nineties the Hamiltonian equation introduced by Hamilton in the 19th century was becoming relevant. Hamilton discovered that classical mechanics equations can be rewritten in a symmetric or Hamiltonian manner. Differential Hamiltonian equations have a set of attractive features. A family of methods based on Hamiltonian equation integration are known as symplectic methods [KMz87,SS92]. These methods are applied to the Hamiltonian form associated to the DRE [Gui99].

Magnus series. Recently the Magnus series method has become relevant [IN97]. Magnus [Mag54] showed that the solution of a lineal differential equation $\dot{y} = a(t)y$ can be approximated by the solution of a non lineal equation in $\sigma, y(t) = e^{\sigma(t)}$, by means of an integral expansion. However, until 1997 [IN97] the convergence of the solution was not shown. The Magnus series method is applied to the solution of several differential equations, DRE included.

3. Numerical Integration by Using BDF

In this Section, an introduction to the resolution of DRE by numerical integration using BDF is presented. Also, the Luca Dieci [Die92] methodology is introduced as a starting point for our implementations.

Let the Riccati equation

$$\dot{X}(t) = F(t, X),\ X(t_0) = X_0,\ t \in [t_0, t_f], \tag{5}$$

where

$$F(t, X) = A_{21}(t) + A_{22}(t)X(t) - X(t)A_{11}(t) - X(t)A_{12}(t)X(t),$$

with

$$A_{11}(t) \in R^{nxn},\ A_{12}(t) \in R^{nxm},\ A_{21}(t) \in R^{mxn},\ A_{22}(t) \in R^{mxm} \text{ and } X(t) \in R^{mxn}.$$

In the literature it is possible to find several methods for solving this stiff equation. One of these methods is the well known Backward Differentiation Formula (BDF) [AsPe98]. By using a BDF scheme, the integration interval $t \in [t_0, t_f]$ is divided so that the approximate solution in t_{k+1}, X_{k+1}, is obtained by solving the implicit equation

$$X_{k+1} - \sum_{j=0}^{p-1} \alpha_j X_{k-j} - h\beta F(t_{k+1}, X_{k+1}) = 0, \tag{6}$$

where coefficients α and β are in Table 1.

p	β	α_0	α_1	α_2	α_3	α_4
1	1	1				
2	2/3	4/3	-1/3			
3	6/11	18/11	-9/11	2/11		
4	12/25	48/25	-36/25	16/25	-3/25	
5	60/137	300/137	-300/137	200/137	-75/137	12/137

Table 1: Coefficients of BDF (p=1, 2, 3, 4 y 5).

When BDF formulation of Equation (6) is applied to the DRE (5), the following ARE is obtained

$$X_{k+1} - \sum_{j=0}^{p-1} \alpha_j X_{k-j} - h\beta\left[A_{21}(t_{k+1}) + A_{22}(t_{k+1})X_{k+1} - X_{k+1}A_{11}(t_{k+1}) - X_{k+1}A_{12}(t_{k+1})X_{k+1}\right] = 0\,. \qquad (7)$$

3.1 Luca Dieci Methodology

In [Die92], in view of implementation and efficiency considerations and BDF properties, a BDF scheme was considered for solving stiff DREs.

An ARE arises when a constant matrix is block upper-triangularized with a Riccati-type similarity transformation. More precisely, given

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

and the similarity transformation

$$S = \begin{pmatrix} I_n & 0 \\ Y & I_m \end{pmatrix},$$

then

$$S^{-1}CS = \begin{pmatrix} C_{11} + C_{12}Y & C_{12} \\ 0 & C_{22} - YC_{12} \end{pmatrix},$$

if and only if Y satisfies the ARE $C_{21} + C_{22}Y - YC_{11} - YC_{12}Y = 0$. Now, a matrix can be associated to (7). In particular this matrix is

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} h\beta A_{11} + \frac{1}{2} I_n & h\beta A_{12} \\ h\beta A_{21} + \sum_{j=0}^{p-1} \alpha_j X_{k-j} & h\beta A_{22} - \frac{1}{2} I_m \end{pmatrix},$$

in which case the BDF step from t_k to t_{k+1} is equivalent to the similarity transformation requirement

$$\begin{pmatrix} I_n & 0 \\ -X_{k+1} & I_m \end{pmatrix} C \begin{pmatrix} I_n & 0 \\ X_{k+1} & I_m \end{pmatrix}$$

where C is a block upper-triangular matrix.

Equation (7) is typically solved by a Newton's iteration. In matrix terms, this gives the following equation

$$\hat{G}'_{(l)}(X_{k+1}^{(l+1)} - X_{k+1}^{(l)}) = -X_{k+1}^{(l)} + h\beta F(t_{k+1}, X_{k+1}^{(l)}) + \sum_{j=0}^{p-1} \alpha_j X_{k-j} \,, \tag{8}$$

where $\hat{G}'_{(l)}$ approximates the Frechet derivative ([Gr81], page 310) $G'_{(l)}$,

$$\hat{G}'_{(l)} : Y \to (C_{22} - X_{k+1}^{(l)} C_{12})Y - Y(C_{11} + C_{12} X_{k+1}^{(l)}) \,. \tag{9}$$

From equations (8) and (9), the following Sylvester equation is obtained

$$\hat{C}_{22} Y - Y \hat{C}_{11} = H \,, \tag{10}$$

where

$$\hat{C}_{22} = C_{22} - X_{k+1}^{(l)} C_{12} \in R^{mxm} \,,$$

$$\hat{C}_{11} = C_{11} + C_{12} X_{k+1}^{(l)} \in R^{nxn} \,,$$

$$H = -X_{k+1}^{(l)} + h\beta F(t_{k+1}, X_{k+1}^{(l)}) + \sum_{j=0}^{p-1} \alpha_j X_{k-j} \in R^{nxm} \,,$$

and

$$Y = X_{k+1}^{(l+1)} - X_{k+1}^{(l)} \in R^{nxm} \,.$$

The standard solution process for (10) is the Bartels-Stewart algorithm as explained in section 1. Algorithm 1 shows this methodology's pseudo code.

Input Parameters:

Initial solution matrix $X_0 \in \Re^{mxn}$

Initial and final simulation time (t_0 and t_f)

Step size (h)

Tolerance of Newton iteration (tol)

Maximum number of Newton Iterations (*itmax*)

Output Parameters

DRE Solution (X)

1. $X = X_0$
2. Compute BDF coefficients according to Table 1 ($p = 5$).
3. $X_4 = X_0, X_3 = 0_{mxn}, X_2 = 0_{mxn}, X_1 = 0_{mxn}, X_0 = 0_{mxn}$
4. For $t = t_0 + h : t_f$
   a. Compute matrices $A_{11}(t)$, $A_{12}(t)$, $A_{21}(t)$, $A_{22}(t)$
   b. $iter = 1$
   c. *InWhile* =TRUE
   d. While (*iter* < *itmax*) AND (*InWhile* == TRUE)
      i. Compute matrices $\hat{C}_{22}$, $\hat{C}_{11}$ and H of equation (10) ($X_{k-j} \equiv X_{4-j,\, j\,=\,0\,:\,4}$)
      ii. Solve for Y the Sylvester equation (10), $\hat{C}_{22}Y - Y\hat{C}_{11} = H$
      iii. Update X: $X = X + Y$
      iv. If (norm(Y) < *tol*) *InWhile* = FALSE
      v. $iter = iter + 1$
   e. If (*iter*==*itmax*) then "Non convergence", exit program
   f. $X_0 = X_1, X_1 = X_2, X_2 = X_3, X_3 = X_4, X_4 = X$

Algorithm 1:Luca Dieci methodology's pseudo code for solving DRE.

The computational cost of each step of this method is $K_{NS}\left(25m^3 + 25n^3 + 13mn^2 + 9m^2n\right)$ flops, where K_{NS} represents the number of Newton's iterations for this approach.

3.2 A Methodology Based on the Newton-GMRES Method

It is possible to apply a Newton method based on a GMRES iterative solver [FGG97, Pein03] benefiting from the sparsity of the Jacobian matrix. From (6), the *vec* operator [HoJo99] is applied, obtaining the following equation

$$x_{k+1} - \sum_{j=0}^{p-1} \alpha_j x_{k-j} - h\beta f(t_{k+1}, x_{k+1}) = 0, \tag{11}$$

with

$$f(t_{k+1}, x_{k+1}) = vec\left(A_{21}(t_{k+1})\right) + \left(I_n \otimes A_{22}(t_{k+1})\right)x_{k+1} - \left(A_{11}^T(t_{k+1}) \otimes I_m\right)x_{k+1} - \left(I_n \otimes \left(X_{k+1}A_{12}(t_{k+1})\right)\right)x_{k+1}, \tag{12}$$

where $\otimes$ is the Kronecker product [Kron78] and $x_{k+1} = vec(X_{k+1})$.

Applying the Newton method to equation (11), the iteration

$$\frac{\partial g}{\partial x_{k+1}}(t_{k+1}, x_{k+1}^{(l)}) y^{(l+1)} = -g(t_{k+1}, x_{k+1}^{(l)}),\ l = 1,2,\cdots \tag{13}$$

is obtained, with

$$y^{(l+1)} = x_{k+1}^{(l+1)} - x_{k+1}^{(l)},\ x_{k+1}^{(1)} = x_k,$$

$$g(t_{k+1}, x_{k+1}) = x_{k+1} - \sum_{j=0}^{p-1} \alpha_j x_{k-j} - h\beta f(t_{k+1}, x_{k+1}),$$

and

$$\begin{aligned} \frac{\partial g}{\partial x_{k+1}}(t_{k+1}, x_{k+1}) &= I_{nxm} - h\beta\left[I_n \otimes \left(A_{22} - X_{k+1}A_{12}\right) + \left(-A_{11} - A_{12}X_{k+1}\right)^T \otimes I_m\right] = \\ &= I_n \otimes \left(-h\beta A_{22} + h\beta X_{k+1}A_{12}\right) - \left(h\beta A_{11} + h\beta A_{12}X_{k+1}\right)^T \otimes I_m = \\ &= I_n \otimes A - B^T \otimes I_m, \end{aligned} \tag{14}$$

where $A = -h\beta A_{22} + h\beta X_{k+1}A_{12}$ and $B = h\beta A_{11} + h\beta A_{12}X_{k+1}$.

At this point, the sparsity of Jacobian matrix (14) is exploited because the percentage of non-zero elements is

$$\frac{nm(n+m-1)}{n^2m^2}\% \cong \left(\frac{1}{n} + \frac{1}{m}\right)\% .$$

Moreover, numerical packages can be used to solve the equation (13), by the GMRES method [Saad94] as implemented in [FGG97] or [Pein03]. In these packages matrix-vector products are performed, where the matrix used is the Jacobian matrix that appears in expression (14). To save some storage cost, the above product is performed without explicitly building the Jacobian matrix; that is, compute

$$(I_n \otimes A - B^T \otimes I_m)v,$$

without building the matrix

$$I_n \otimes A - B^T \otimes I_m .$$

Algorithm 2 shows this methodology's pseudo code.

Input Parameters:

Initial solution matrix $X_0 \in \Re^{mxn}$

Initial and final simulation time (t_0 and t_f)

Step size (h)

Tolerance of Newton iteration (tol)

Maximum number of Newton-GMRES iterations (*itmax*)

Output Parameters

DRE Solution (X)

1. Compute BDF coefficients according to Table 1.
2. $x_4 = x_0, x_3 = 0_{mn}, x_2 = 0_{mn}, x_1 = 0_{mn}, x_0 = vec(X_0)$
3. For $t = t_0 + h : t_f$
   a. Compute matrices $A_{11}(t)$, $A_{12}(t)$, $A_{21}(t)$, $A_{22}(t)$
   b. $iter = 1$
   c. *InWhile* = TRUE
   d. While (*iter* < *itmax*) AND (*InWhile* == TRUE)
      i. Compute A, B and $g(t,x)$ from expressions (13) ($X_{k-j} \equiv X_{4-j, j=0:4}$) and (14)
      ii. Solve the equation (13)
      $$\frac{\partial g}{\partial x}(t,x)y = -g(t,x)$$
      by GMRES method, computing the Jacobian matrix by vector products without forming the Jacobian
      iii. Update solution x: $x = x + y$
      iv. If (norm(y) < tol) *InWhile* = FALSE
      v. $iter = iter + 1$
   g. If (iter == *itmax*) then "Non convergence", exit program
   e. $x_0 = x_1, x_1 = x_2, x_2 = x_3, x_3 = x_4, x_4 = x$

Algorithm 2: Pseudo code of the approach for solving DRE by Newton-GMRES method.

The computational cost of this algorithm is

$$K_N\left[8m^2n + 4mn^2 + K_{GM}\left(2mn(m+n-1)\right)restart\right],$$

where K_N is the number of Newton's iterations, K_{GM} the number of GMRES iterations and *restart* the Krylov subspace dimension.

3.3 A Methodology Based on Sylvester Equation

The expression (7), provides the following ARE

$$AX_{k+1} + X_{k+1}B + X_{k+1}CX_{k+1} + D = 0, \tag{15}$$

where

$$A = -A_{22}(t_{k+1}) + \frac{1}{2h\beta}I_m,$$

$$B = A_{11}(t_{k+1}) + \frac{1}{2h\beta} I_n ,$$

$$C = A_{12}(t_{k+1}) ,$$

and

$$D = -\frac{1}{h\beta} \sum_{j=0}^{p-1} \alpha_j X_{k-j} - A_{21}(t_{k+1}) = 0 .$$

This new approach is equivalent to the one presented in Subsection 3.1, in terms of performance and formulation; however, the expressions for the Sylvester equation are different.

By applying Newton's method to equation (15), the approximated solution at stage $(l+1)$, $X_{k+1}^{(l+1)}$, is obtained by solving the following equation

$$G'_{(l)}\left(X_{k+1}^{(l)}\right) H^{(l)} = -AX_{k+1}^{(l)} - X_{k+1}^{(l)} B - X_{k+1}^{(l)} C X_{k+1}^{(l)} - D ,\ X_{k+1}^{(1)} = X_k , \tag{16}$$

where $G'_{(l)} H^{(l)}$ is the Frechet derivative in $H^{(l)} = X_{k+1}^{(l+1)} - X_{k+1}^{(l)}$, of the matrix function

$$G(Y) = AY + YB + YCY + D , \tag{17}$$

given by

$$\begin{aligned} G'(Y)H &= \lim_{h\to 0} \frac{G(Y+hH) - G(Y)}{h} = \\ &= \lim_{h\to 0} \frac{A(Y+hH) + (Y+hH)B + (Y+hH)C(Y+hH) + D - AY - YB - YCY - D}{h} = \\ &= \lim_{h\to 0} \frac{hAH + hHB + hYCH + hHCY + h^2 HCH}{h} = AH + HB + YCH + HCY = \\ &= \left(A + YC\right)H + H\left(B + CY\right). \end{aligned}$$

Applying this result to Equation (16), the following Sylvester equation is obtained

$$\overline{A} H^{(l)} + H^{(l)} \overline{B} = \overline{C} , \tag{18}$$

where

$$\overline{A} = A + X_{k+1}^{(l)} C = -A_{22}(t_{k+1}) + \frac{1}{2h\beta} I_m + X_{k+1}^{(l)} A_{12}(t_{k+1}) ,$$

$$\overline{B} = B + C X_{k+1}^{(l)} = A_{11}(t_{k+1}) + \frac{1}{2h\beta} I_n + A_{12}(t_{k+1}) X_{k+1}^{(l)} ,$$

$$\overline{C} = -AX_{k+1}^{(l)} - X_{k+1}^{(l)}B - X_{k+1}^{(l)}CX_{k+1}^{(l)} - D =$$

$$= \left[A_{22}(t_{k+1}) - \frac{1}{2h\beta} I_m \right] X_{k+1}^{(l)} - X_{k+1}^{(l)} \left[A_{11}(t_{k+1}) + \frac{1}{2h\beta} I_n \right] - X_{k+1}^{(l)} A_{12}(t_{k+1}) X_{k+1}^{(l)} +$$

$$+ \frac{1}{h\beta} \sum_{j=0}^{p-1} \alpha_j X_{k-j} + A_{21}(t_{k+1}) =$$

$$= A_{22}(t_{k+1}) X_{k+1}^{(l)} - X_{k+1}^{(l)} A_{11}(t_{k+1}) - X_{k+1}^{(l)} A_{12}(t_{k+1}) X_{k+1}^{(l)} + \frac{1}{h\beta} \left[\sum_{j=0}^{p-1} \alpha_j X_{k-j} - X_{k+1}^{(l)} \right] + A_{21}(t_{k+1}).$$

Once $H^{(l)}$ has been obtained, the solution at the $(l+1)$ iteration is given by $X_{k+1}^{(l+1)} = X_{k+1}^{(l)} + H^{(l)}$.

A pseudo code of the approach proposed in this section is shown in Algorithm 3.

Input Parameters:

Initial solution matrix $X_0 \in \Re^{mxn}$

Initial and final simulation time (t_0 and t_f)

Step size (h)

Tolerance of Newton iteration (tol)

Maximum number of Newton iterations (*itmax*)

Output Parameters

DRE Solution (X)

1. $X = X_0$
2. Compute BDF coefficients according to Table 1.
3. $X_4 = X_0, X_3 = 0_{mxn}, X_2 = 0_{mxn}, X_1 = 0_{mxn}, X_0 = 0_{mxn}$
4. For $t = t_0 + h : t_f$
   a. Compute matrices $A_{11}(t)$, $A_{12}(t)$, $A_{21}(t)$, $A_{22}(t)$
   b. $iter = 1$
   c. *InWhile* = TRUE
   d. While (*iter* < *itmax*) AND (*InWhile* == TRUE)
      i. Compute the matrices $\overline{A}$, $\overline{B}$ and $\overline{C}$ of the equation (18) ($X_{k-j} \equiv X_{4-j}, j = 0:4$)
      ii. Compute H, solving the Sylvester equation (18)
      iii. Update solution X, $X = X + H$
      iv. If (norm(H) < tol) end while
      v. $iter = iter + 1$
   e. If (*iter* == *itmax*) then "Non convergence", exit program
   f. $X_0 = X_1, X_1 = X_2, X_2 = X_3, X_3 = X_4, X_4 = X$

Algorithm 3: Pseudo code of Newton-Raphson method based on Sylvester Equation.

The computational cost of each step of this method is $K_{NS}\left(25m^3 + 25n^3 + 13mn^2 + 9m^2n\right)$ flops, where K_{NS} represents the number of Newton's iterations for this approach.

3.4 A Methodology Based on the Fixed Point Method

From (7), the following ARE is obtained

$$AX_{k+1} + X_{k+1}B + X_{k+1}CX_{k+1} + D = 0, \tag{19}$$

where

$$A = h\beta A_{22}(t_{k+1}) - I_m,$$

$$B = -h\beta A_{11}(t_{k+1}),$$

$$C = -h\beta A_{12}(t_{k+1}),$$

$$D = h\beta A_{21}(t_{k+1}) + \sum_{j=0}^{p-1} \alpha_j X_{k-j}.$$

Rewriting equation (19) as a recurrence of the matrix fixed point method gives the following equation

$$\left(A + X_{k+1}C\right)X_{k+1} + X_{k+1}B + D = 0 \Rightarrow \left(A + X_{k+1}C\right)X_{k+1} = -\left(X_{k+1}B + D\right). \tag{20}$$

So, by applying the fixed point iteration

$$\left(A + X_{k+1}^{(l)}C\right)X_{k+1}^{(l+1)} = -\left(X_{k+1}^{(l)}B + D\right),\ X_{k+1}^{(1)} = X_k. \tag{21}$$

Then, the solution at stage $(l+1)$ is obtained by solving the linear system (21).

A pseudo code of the approach proposed in this section is shown in Algorithm 4.

Input Parameters:

Initial and final simulation time (t_0 and t_f)

Step size (h)

Tolerance (tol)

Maximum number of Newton iterations ($itmax$)

Output Parameters

DRE Solution (X)

1. Compute BDF coefficients according to table 1.
2. $X_4 = X_0, X_3 = 0_{mxn}, X_2 = 0_{mxn}, X_1 = 0_{mxn}, X_0 = 0_{mxn}$
3. For $t = t_0 + h : t_f$
   a. Compute matrices $A_{11}(t)$, $A_{12}(t)$, $A_{21}(t)$, $A_{22}(t)$
   b. Compute matrices A, B, C and D of equation (19) ($X_{k-j} \equiv X_{4-j, j=0:4}$)
   c. $X_0 = X$
   d. $iter = 1$
   e. $InWhile$ = TRUE
   f. While ($iter < itmax$) AND ($InWhile$ == TRUE)
      i. Compute X, solving the linear system (21)
      ii. If (norm($X - X0$) < tol) InWhile = FALSE
      iii. $iter = iter + 1$
      iv. $X0=X$
   g. If ($iter==itmax$) then "Non convergence", exit program
   h. $X_0 = X_1, X_1 = X_2, X_2 = X_3, X_3 = X_4, X_4 = X$

Algorithm 4: Pseudo code of the method based on an iteration of the fixed point method for solving DRE.

The computational cost of this method is $K_{FP}\left(\frac{2}{3}m^3 + 4m^2n + 2mn^2\right)$, where K_{PF} represents the number of Newton's iterations for the matrix fixed point method.

3.4 Computational Costs

Table 2 summarizes the cost of all the algorithms described in the previous sections, in terms of memory storage requirements and computational cost (number of flops).

Method	Memory storage	Computational cost (flops)
Luca Dieci	$mnp + 2m^2 + 2n^2 + 2mn + 5\max(m,n)$	$K_{NS}\left(25m^3 + 25n^3 + 13mn^2 + 9m^2n\right)$
GMRES	$1 + restart^2 + restart(nm+5) + + mnp + 3mm + 12mn$	$K_N\left[8m^2n + 4mn^2 + K_{GM}\left(2mn(m+n-1)\right)restart\right]$
Sylvester equation	$mnp + 2m^2 + 2n^2 + 2mn + 5\max(m,n)$	$K_{NS}\left(25m^3 + 25n^3 + 13mn^2 + 9m^2n\right)$
Matrix Fixed Point	$mnp + 2m^2 + n^2 + 3mn + m$	$K_{PF}\left(\frac{2}{3}m^3 + 4m^2n + 2mn^2\right)$

Table 2: Costs of different methods in terms of memory storage and flops.

It can be appreciated that the computational cost of the matrix fixed point method is lower than the other methods. This will be demonstrated in the next section, where for all the problems, the results obtained confirm the estimations shown in Table 2.

4 Experimental Results

4.1 Platform

The tests have been carried out on a node of a SGI Altix 3700 [SGIAltix] with 48 processors. This machine with a NUMA architecture belongs to the SGI 3000 series [SGI3000] where the memory is physically distributed and resides with each individual processor, but the combination of the operating system and extra hardware controllers maintain the single shared-memory image at user level. This architecture is becoming increasingly popular for top-of-the-range HPC servers [CBS].

The standard processor is an Intel Itanium II CPU [SGIAltix] working with 1.3Ghz and 3MB cache (known as "Madison"). The total memory is 44 Gb. The hardware is modular and is divided in bricks: C-Bricks (Computational bricks), R-bricks (Router bricks), IX-Bricks (Peripheral bricks) and M-Bricks (Memory Bricks). Each C-Brick has a bandwidth of 3.2Gb/s and it has 4 processors.

The Operating System (O.S.) is a modified version of Red Hat Linux for the SGI hardware. It has some of the well known SGI tools used in other O.S., such as IRIX.

The implemented algorithms use the FORTRAN77, C and C++ programming languages. Mathematical libraries such as BLAS and LAPACK are also used. LAPACK (Linear Algebra PACKage) is a set of routines to solve linear systems, the minimum least squares problem, eigenvalue problems and singular values related problems. These routines obtain high performance when used in combination with an optimized BLAS package, making use of routines using the matrix product (matrix-matrix, or matrix-vector) when possible. BLAS (Basic Linear Algebra Subroutines) is a set of routines for computing basic operations, such as the above mentioned matrix-matrix or matrix-vector products. There are optimized versions of BLAS for every architecture.

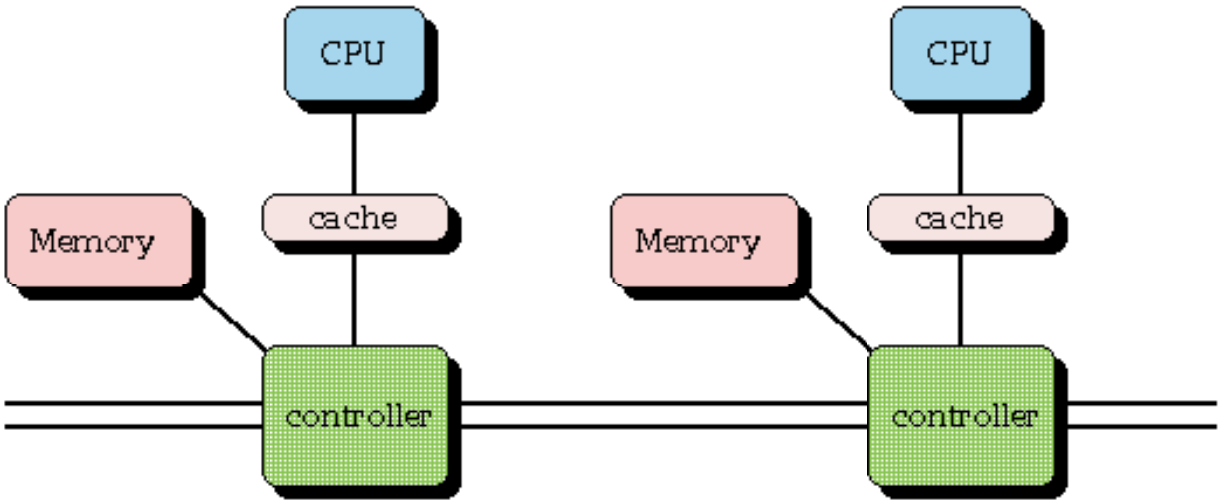


Figure 1: SGI Altix 3700 architecture.

The compilers used are the optimized Intel versions of FORTRAN77, C and C++. These versions are much more optimized than other standard compilers, such as the GNU compilers. We used the latest version of all the compilers (release 8.1). The mathematical libraries used are optimized versions of BLAS and LAPACK [ABB+94] which have two optimized alternatives, Intel MKL [MKL] release 7.0, and SGI SCSL (Scientific Computing Software Library), release 1.5.1. The SCSL version is slightly more optimized than Intel MKL library.

4.2 Case Studies

In this section two case studies have been considered to design the tests that are described later.

4.2.1 Case Study 1

The first case study has been described by [Meyer73], and consists of the following DRE

$$\dot{X}(t) = A_{21}(t) + A_{22}(t)X(t) - X(t)A_{11}(t) - X(t)A_{12}(t)X(t),\ 0 \le t \le t_f,$$

where $A_{11} = 0_n$, $A_{12} = A_{21} = \alpha I_n$ $(\alpha > 0)$, $A_{22} = 0_n$, and $X_0 \in \Re^{nxn}$ represents a matrix with all entries to 1.

This problem has been chosen because the dimension *n* and the stiffness α of the problem can be changed.

The exact solution is given by

$$X(t) = (\alpha(X_0 + I)e^{\alpha t} - \alpha(X_0 - I)e^{-\alpha t})^{-1}(\alpha(X_0 + I)e^{\alpha t} + \alpha(X_0 - I)e^{-\alpha t}),$$

which allows the approaches presented in this document to be compared in terms of accuracy.

Several tests by changing several parameter values have been considered. The parameters and their values are the following:

n = 128, 256, 512, 1024;

h = 1e-2 ,5e-3, 1e-3;

tol = 1e-6, 1e-12, 1e-15;

tf=1 and 20 seconds;

itmax=1000 iterations.

For the tests carried out, the best value of *h* is *1e-2*. Therefore results using other values of *h* are not shown.

4.2.2 Case Study 2

The second case study is described in [CL90]. It consists of the following non-symmetric DRE

$$\dot{X}(t) = X(t) + T_{2^n}X(t) - X(t)T_{2^n}X(t) + k^2 T_{2^n};\ X(0) = I_{2^n}, \mathrm{t} \ge 0,$$

where the values of n and k vary during the experiments, and $X(t), T_{2^n} \in \Re^{2^n}$. Matrices T_2 are generated as follows

$$T_2 = \begin{bmatrix} -1 & 1 \\ k^2 & 1 \end{bmatrix},$$

$$T_{2^{n+1}} = \begin{bmatrix} -T_{2^n} & T_{2^n} \\ k^2 T_{2^n} & T_{2^n} \end{bmatrix}, \mathrm{n} \geq 1.$$

The analytic solution of this DRE is given by

$$X(t) = I_{2^n} + \frac{(k^2+1)}{\omega} \tanh \omega t T_{2^n},$$

where $\omega = (k^2+1)^{\frac{n+1}{2}}$.

Again we developed several tests for different values of the above mentioned parameters:

n =2^n = 128, 256, 512, 1024;

h = 1e-2 ,5e-3;

tol = 1e-6, 1e-10, 1e-12;

tf=1 second;

itmax = 1000 iterations.

The tests showed that the best value of *h* for this problem is *h = 5e-3*. Therefore results using other values of *h* are not shown here.

4.3 Experimental results

In this subsection we analyze three methods for solving the DREs by numerical. All these methods are compared with our implementation of the Diecci Method, based on the standard Dieci algorithm ([Die92]) using double precision LAPACK routines.

The routines have names of the form XYYZZZVVV, where

- X indicates the data type (d= Double).
- YY indicates the type of matrix (ge= GEneral).
- ZZZ indicates the problem (drc= Differential Riccati equation time Constant).
- VVV indicates the method used (bdi= Bdf and DIecci methodology, bgm= Bdf and GMres method, bfp= Bdf and Fixed Point method, bsy= BDF with resolution of the SYlvester equation).

The following routines have been implemented:

1. Diecci method (dgedrcbdi).
2. Newton-Raphson method with a Sylvester equation solver (dgedrcbsy).
3. Newton-Raphson method with a GMRES solver (dgedrcbgm).
4. Fixed Point method (dgedrcbfp).

For each test we show a figure (a) with its execution time for different problem sizes. We also show a table (b) including the same results, and a table (c) including the error values. Table (c) shows the difference between the analytical solution and the obtained solution.

4.3.1 Experimental results for case study 1

TEST 1.1

In this test the following parameter values are considered:

n = 128, 256, 512, 1024;

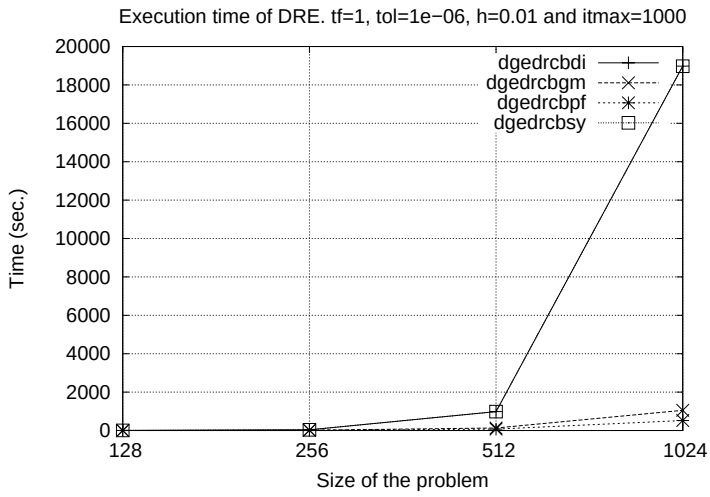
h =1e-2;

tol = 1e-6;

tf=1 second;

itmax=1000 iterations.

Table 1 shows the execution time and accuracy for this test. The best results are obtained when using the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than using the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	3,5	37,107	985,363	18975,643
	dgedrcbdi	3,494	37,087	986,444	18980,324
	dgedrcbfp	1,126	8,51	74,49	520,166
	dgedrcbgm	1,866	14,395	129,008	1053,277

(b)

Error		Size of the problem			
		128	256	512	1024
Method	Dgedrcbsy	0.00276774	0.00276774	0.00276774	0.00276774
	Dgedrcbdi	0.00276774	0.00276774	0.00276774	0.00276774
	Dgedrcbfp	0.00276772	0.00276772	0.00276772	0.00276772
	Dgedrcbgm	0.00272557	0.00272557	0.00272557	0.00272557

(c)

Table 1: Figure of the execution time for solving the DRE for test 1.1 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

Timing is very similar in `dgedrcbsy and dgedrcbdi`. In fact, it is impossible to distinguish the curves in the graphical view. This is true for all the graphical views in this work.

TEST 1.2

The following parameter values are considered for this test:

n = 128, 256, 512, 1024;

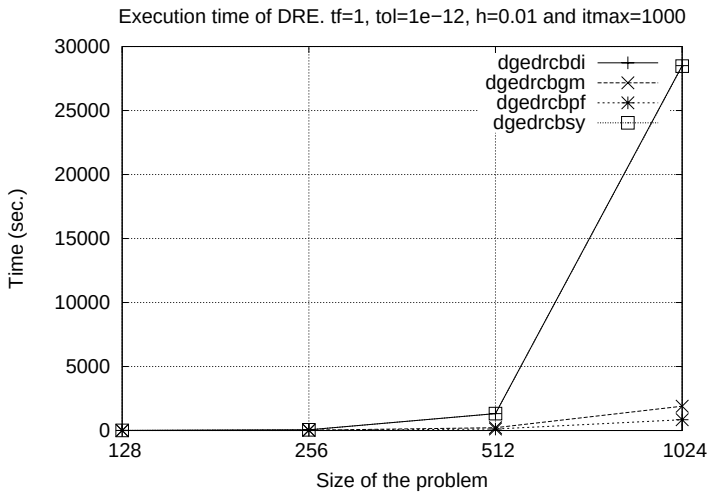
h =1e-2;

tol = 1e-12;

tf =1 second.

itmax=1000 iterations.

Table 2 shows execution time and accuracy obtained for this test. Again, the best results are obtained with the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification implemented in this work, `dgedrcbsy`.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	5,14	55,335	1323,83	28468,678
	dgedrcbdi	5,12	55,254	1323,18	28511,260
	dgedrcbfp	1,831	13,762	119,776	843,867
	dgedrcbgm	3,336	25,482	221,559	1900,213

(b)

Error		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbdi	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbfp	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbgm	0.00276774	0.00276774	0.00276774	0.00276774

(c)

Table 2: Figure of the execution time for solving the DRE for test 1.2 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

TEST 1.3

The following parameter values are considered for this test:

n = 128, 256, 512, 1024;

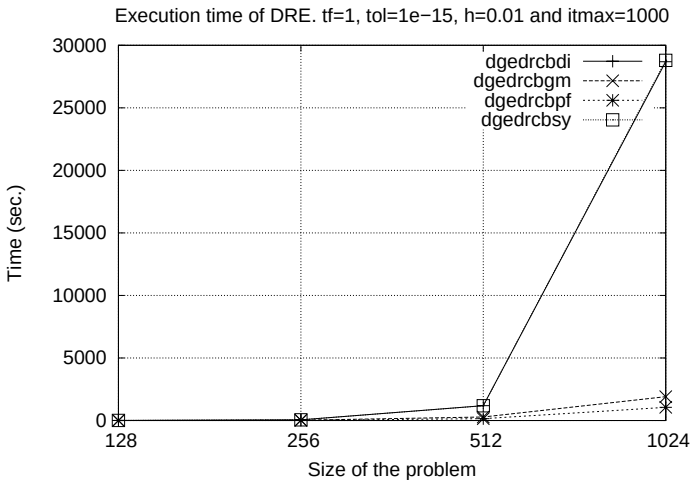
h =1e-2;

tol = 1e-15;

tf=1 second;

itmax=1000 iterations.

Table 3 shows execution time and accuracy obtained for this test. Again, the best results are obtained with the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than using either the Diecci method (`dgedrcbdi`) or the modification implemented in this work, `dgedrcbsy`.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	5,362	55,809	1183,05	28799,793
	dgedrcbdi	5,339	55,705	1184,105	28689,622
	dgedrcbfp	2,271	16,955	148,895	1052,852
	dgedrcbgm	3,339	25,636	282,794	1907,294

(b)

Error		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbdi	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbfp	0.00276774	0.00276774	0.00276774	0.00276774
	dgedrcbgm	0.00276774	0.00276774	0.00276774	0.00276774

(c)

Table 3: Figure of the execution time for solving the DRE for test 1.3 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

In these tests 1.1, 1.2 and 1.3, when the *tol* parameter is changed to *tol =1e-12* and *Tol=1e-15*, the errors for all the methods are similar.

The routine `dgedrcbgm` behaves worse when using *tol = 1e-6*. This is due to the fact that `dgedrcbgm` method is very sensitive to the *tol* parameter. This can be appreciated better in the following tests, where parameter *tf = 20*. This time is necessary for the method to converge with a low error.

In addition, *N=1024* was eliminated from the tests. This size used too much time and there was not enough CPU time to test it.

TEST 1.4

The following parameter values are considered for this test:

n = 128, 256, 512;

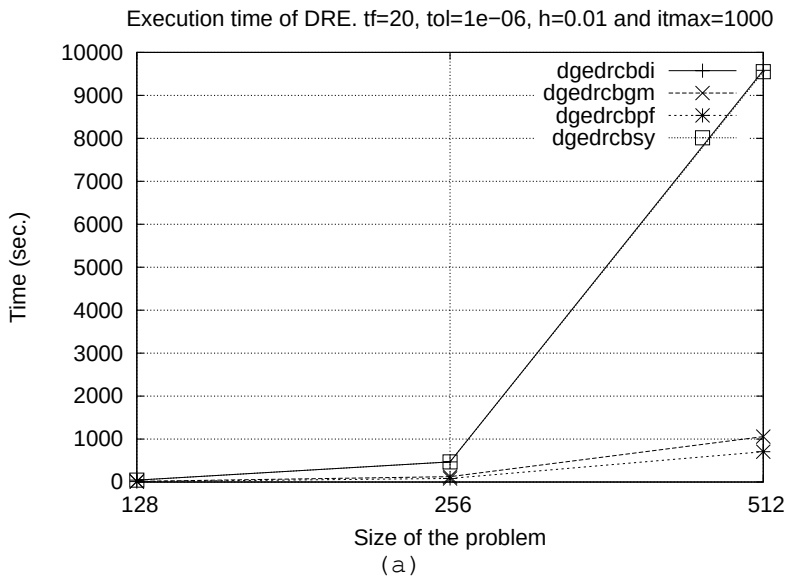
h =1e-2;

tol = 1e-6;

tf = 20 seconds;

itmax=1000 iterations.

Table 4 summarizes the results obtained for the test 1.4. The best results are obtained with the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`. But in terms of accuracy, the `dgedrcbgm` method does not obtain low error unless the parameter *tol* is small enough.



(a)

Time (sec.)		Size of the problem		
		128	256	512
Method	dgedrcbsy	43,675	470,206	9552,66
	dgedrcbdi	43,487	469,514	9583,64
	dgedrcbfp	10,698	83,383	707,372
	dgedrcbgm	15,246	124,817	1056,39

(b)

Error		Size of the problem		
		128	256	512
Method	dgedrcbsy	6.32827e-15	6.32827e-15	6.32827e-15
	dgedrcbdi	4.10783e-15	4.10783e-15	4.10783e-15
	dgedrcbfp	1.06581e-14	1.06581e-14	1.06581e-14
	dgedrcbgm	5.03215e-05	5.03215e-05	5.03215e-05

(c)

Table 4: Figure of the execution time for solving the DRE for test 1.4 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

TEST 1.5

The following parameter values are considered for this test:

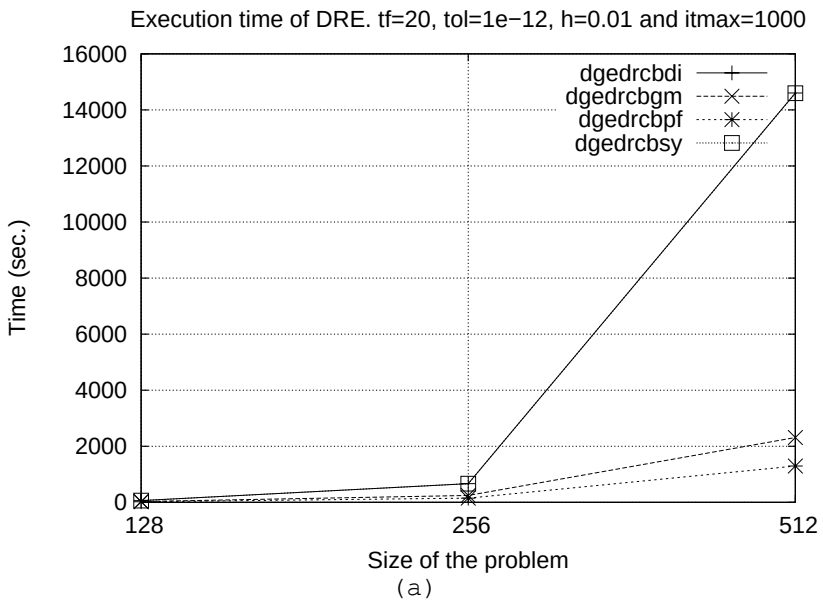
n = 128, 256, 512;

h =1e-2;

tol = 1e-12;

tf = 20 seconds;

itmax=1000 iterations.



(a)

Time (sec.)		Size of the problem		
		128	256	512
Method	dgedrcbsy	62,511	666,272	14594
	dgedrcbdi	62,265	665,205	14605,5
	dgedrcbfp	19,882	150,572	1297,85
	dgedrcbgm	31,406	244,44	2312,57

(b)

Error		Size of the problem		
		128	256	512
Method	dgedrcbsy	6.32827e-15	6.32827e-15	6.32827e-15
	dgedrcbdi	4.10783e-15	4.10783e-15	4.10783e-15
	dgedrcbfp	1.06581e-14	1.06581e-14	1.06581e-14
	dgedrcbgm	5.01074e-11	5.01074e-11	5.01074e-11

(c)

Table 5: Figure of the execution time for solving the DRE for test 1.5 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

Table 5 shows the results obtained for test 1.5. Again, the best results are obtained with the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`. The `dgedrcbgm` method shows improved accuracy with a low value of parameter *tol*. In any case, this method is less accurate than the others.

TEST 1.6

The following parameter values are considered for this test:

n = 128, 256, 512;

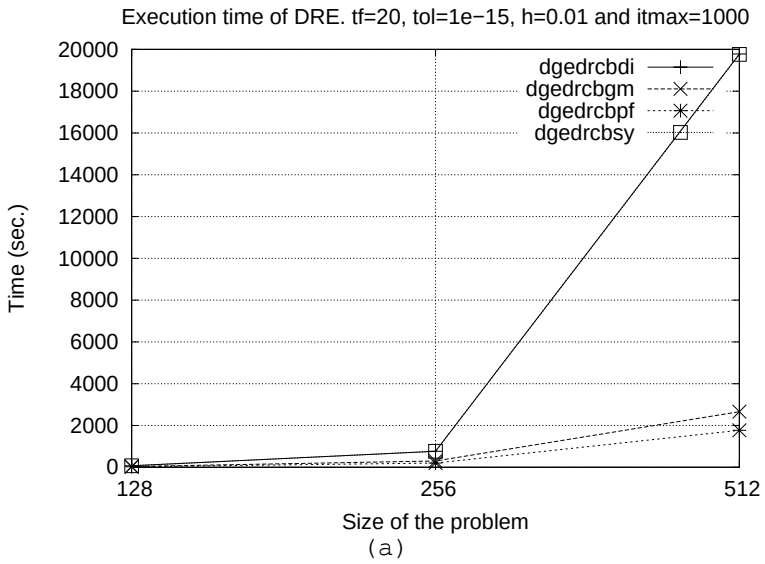
h =1e-2;

tol = 1e-15;

tf = 20 seconds;

itmax=1000 iterations.

Table 6 shows the results for test 1.6. The best results are obtained with the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`. Now the accuracy of `dgedrcbgm` method is comparable with that of the other methods because the value of the parameter *tol* is so small. Thus, the `dgedrcbgm` method must use a very low value of the parameter *tol*.



(a)

Time		Size of the problem		
		128	256	512
Method	dgedrcbsy	72,173	767,55	19761,3
	dgedrcbdi	71,9	766,662	19782,8
	dgedrcbfp	27,09	204,742	1770,32
	dgedrcbgm	39,055	300,346	2660,41

(b)

Error		Size of the problem		
		128	256	512
Method	dgedrcbsy	6.32827e-15	6.32827e-15	6.32827e-15
	dgedrcbdi	4.10783e-15	4.10783e-15	4.10783e-15
	dgedrcbfp	1.06581e-14	1.06581e-14	1.06581e-14
	dgedrcbgm	5.67324e-14	5.67324e-14	5.67324e-14

(c)

Table 6: Figure of the execution time for solving the DRE for test 1.6. (a). Tables reporting the execution time (b) and the accuracy (c) for the same case.

4.3.2 Experimental results for case study 2

Tables 7, 8 and 9, summarize the results for tests 2.1, 2.2 and 2.3 which show the behaviour of case study 2.

TEST 2.1

The following values of the parameters are considered for this test:

n = 128, 256, 512, 1024;

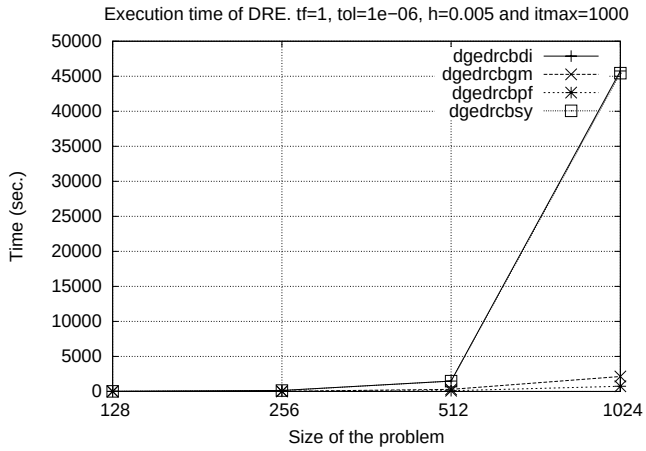
h =5.e-3;

tol = 1e-6;

tf = 1 second;

itmax=1000 iterations.

Table 7 shows the results obtained for test 2.1. The best results are obtained when using the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`. However, the accuracy of the `dgedrcbgm` method is the worst, because the value of *tol* parameter is not small enough.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	19.4033	132.824	1474.27	45440.6
	dgedrcbdi	22.8692	147.905	1487.1	45753.2
	dgedrcbfp	1.98602	13.509	99.4336	743.041
	dgedrcbgm	4.68176	29.0657	295.351	2131.88

(b)

Error		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	3.36412e-13	2.0352e-15	3.30964e-16	2.80317e-15
	dgedrcbdi	3.37037e-13	2.17558e-15	2.48959e-16	3.10245e-15
	dgedrcbfp	7.66936e-14	2.23079e-15	2.52849e-15	2.65316e-15
	dgedrcbgm	3.40494e-07	1.74888e-07	8.80272e-08	1.32983e-08

(c)

Table 7: Figure of the execution time for solving the DRE for test 2.1 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

TEST 2.2

The following values of the parameters are considered for this test:

n = 128, 256, 512, 1024;

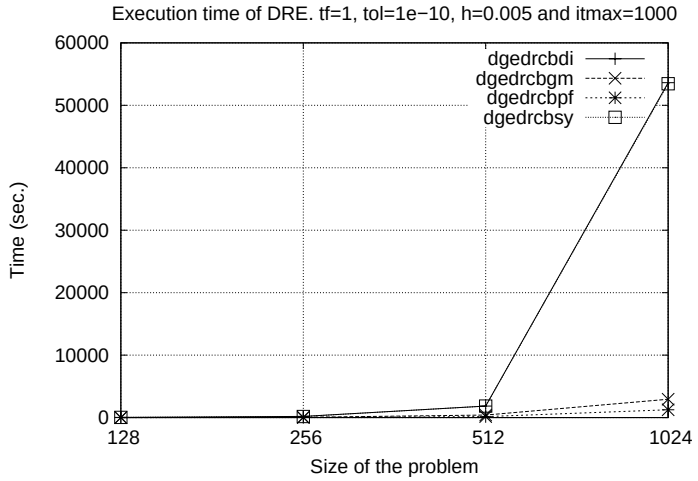
h =5e-3;

tol = 1e-10;

tf = 1 second;

itmax=1000 iterations.

Table 8 shows the results obtained for test 2.2. The best results are obtained when using the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method (`dgedrcbdi`) or the modification used in this work, `dgedrcbsy`. The `dgedrcbgm` method shows improved accuracy due to the low value of *tol* parameter. However, it is the least accurate of all the methods.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	25.2087	169.044	1845.22	53454.5
	dgedrcbdi	29.9956	190.283	1851.41	53624.8
	dgedrcbfp	3.52668	22.6953	160.357	1252.49
	dgedrcbgm	7.55606	44.28	405.094	2938.13

(b)

Error		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	3.36365e-13	2.05576e-15	3.3559e-16	2.79312e-15
	dgedrcbdi	3.37062e-13	2.17969e-15	2.26881e-16	3.19741e-15
	dgedrcbfp	2.11476e-13	2.36412e-15	2.51861e-15	2.64971e-15
	dgedrcbgm	3.72593e-11	2.05448e-11	8.3594e-12	1.21813e-12

(c)

Table 8: Figure of the execution time for solving the DRE for test 2.2 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

TEST 2.3

The following values of the parameters are considered for this test:

n = 128, 256, 512, 1024;

h =5.e-3;

tol = 1e-12;

tf = 1 second;

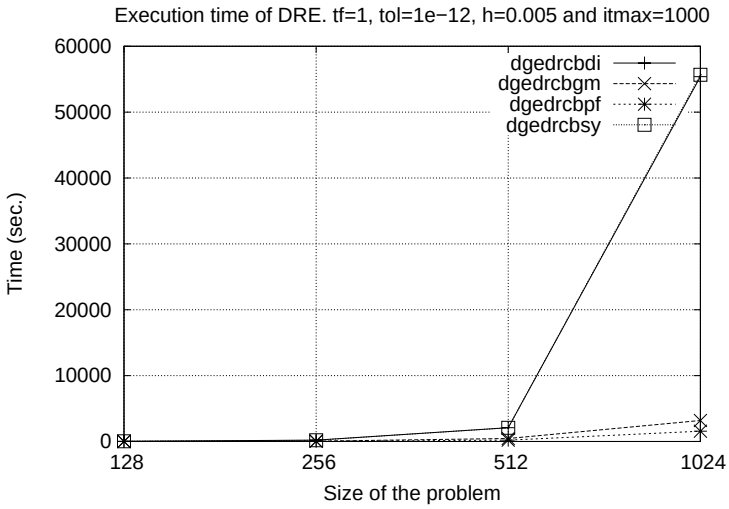
itmax=1000 iterations.

Table 9 shows the results obtained for test 2.3. Again, the best results are obtained when using the `dgedrcbfp` method. The `dgedrcbgm` method is also faster than either the Diecci method

(dgedrcbdi) or the modification used in this work, dgedrcbsy. Now, the accuracy of dgedrcbgm method is comparable to that of the others.

To sum up, the fastest method is dgedrcbfp followed by dgedrcbgm. These differences are greater in the tests for case study 1. Table 7 shows that dgedrcbfp is 61 times faster than dgedrcbdi, and dgedrcbgm is 42 times faster than dgedrcbdi. In both cases, $n = 1024$.

Once again the parameter *tol* influences error in the dgedrcbgm method. The lowest value possible of *to*l must be used, but in this problem its influence is less significant than in case study 1.



(a)

Time (sec.)		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	27.5522	185.66	2075.68	55671.4
	dgedrcbdi	32.7962	208.655	2085.96	55396.7
	dgedrcbfp	4.52513	28.4135	198.928	1562.44
	dgedrcbgm	8.72893	50.6385	456.791	3183.47

(b)

Error		Size of the problem			
		128	256	512	1024
Method	dgedrcbsy	3.36251e-13	2.04577e-15	3.33908e-16	2.81187e-15
	dgedrcbdi	3.37066e-13	2.17675e-15	2.22886e-16	3.1584e-15
	dgedrcbfp	3.41338e-13	2.32947e-15	2.51125e-15	2.64701e-15
	dgedrcbgm	3.36319e-13	1.76914e-13	9.61872e-14	1.16086e-14

(c)

Table 9: Figure of the execution time for solving the DRE for test 2.3 (a). Tables reporting execution time (b) and accuracy (c) for the same case.

5 Conclusions and Future Work

In this work three approaches for solving DRE's are presented. One of them (dgedrcbsy), is analogous to the Dieci methodology, and obtains similar experimental results than the Diecci method (dgedrcbdi). The other two approaches are new, the fixed point approach (dgedrcbfp), and the GMRES approach (dgedrcbgm). These two methods are faster than Dieci method. This conclusion is confirmed with a theoretical study of the memory storage and flops performed by each method.

Sequential implementation has also been done using linear algebra libraries in order to improve the portability, efficiency and robustness of all the implementations.

Next, the main conclusions of the experimental results, which confirm the theoretical analysis, are commented.

Also, the experimental results show that the `dgedrcbsy` method behaves in a similar way to the `dgedrcbdi` method. In fact, both methods are based on the same mathematical formulation. Therefore, the conclusions for methods 2 and 3 are to be emphasized.

Firstly, the conclusions for each problem used are shown. Secondly, the common conclusions for both problems are shown.

Case study 1:

- The fastest method is `dgedrcbfp`, the Fixed Point method. This method is much faster than the other methods used. Compared to the `dgedrcbdi` method, it is between 2.5 times and 28 times faster.
- `dgedrcbgm` is also much faster than the `dgedrcbdi` method. Compared to the `dgedrcbdi` method it is between 2 and 15 times faster..
- Normally both methods, `dgedrcbfp` and `dgedrcbgm`, are much faster than the `dgedrcbdi` method as the size of the problem increases.
- The error (difference between the analytical and obtained solution) is very similar for `dgedrcbdi` and `dgedrcbfp`. It is slightly better in the `dgedrcbdi` method, and worse in `dgedrcbgm` method. To avoid this problem the smallest possible *tol* values must be used.

Case study 2:

- Again `dgedrcbfp` is much faster than the other methods. Compared to `dgedrcbdi` it is between 7 and 61 times faster.
- `Dgedrcbgm` is also much faster, between 4 and 21 times faster. It depends on the case to be solved.
- Normally both `dgedrcbfp` and `dgedrcbgm`, are much faster than `dgedrcbdi` as the size of the problem increases.
- Again, the error is very similar for `dgedrcbdi` and `dgedrcbfp`. It is a little better in `dgedrcbdi`, and it is very similar using `dgedrcbfp` and `dgedrcbgm`. To obtain low error with `dgedrcbgm`, the smallest possible *tol* values must be used.

Common conclusions:

Three Approaches for solving Differential Riccati Equations Based on Numerical Integration with BDF have been implemented. Two of them are much faster than the standard method used known as the Dieci method:

- Best method is fixed point (`dgedrcbfp`). It is much faster than Dieci method (`dgedrcbdi`).
- Newton-GMRES method (`dgedrcbgm`) is also faster than `dgedrcbdi`, but not as fast as `dgedrcbfp`. Moreover, the error in this method can be influenced by the *to*l parameter.

- Normally, both methods, `dgedrcbfp` and `dgedrcbgm` are faster than `dgedrcbdi` as the size of the problem increases.

In the future, parallel implementation of the algorithms presented in this work will be carried out in a distributed memory platform, using the message passing paradigm; MPI [GLS94] and BLACS [DVDG91] for communications, and PBLAS [CDO+95] and ScaLAPACK [BCC+97] for computations.

References

[ABB+94] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, 2 edition, 1994.

[AsPe98] U. M. Ascher and L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. Siam, 1998.

[AW97] K. J. Astrom and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

[BS72] R.H. Bartels and G.W. Stewart. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Comm. ACM*, 15:820–826, 1972.

[CBS] Computing for Business and Science - http://www.epcc.ed.ac.uk/HPCinfo/hware-smp.html#dsm]

[Chan76] Generalized Chandrasekhar algorithms: Time-variyng models. IEEE Trans. Automatic Control, 21, 728-732, 1976

[CDO+95] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, and D. Walker. A proposal for a set of Parallel Basic Linear Algebra Subprograms. Technical Report UT-CS-95-292, Department of Computer Science, University of Tennessee, May 1995.

[Cho90] Chiu H. Choi. A survey of numerical methods for solving matrix Riccati differential equations. In *Southeastcon'90 Proceedings*, pages 696–700, 1990.

[CL90] Chiu H. Choi and Alan J. Laub. Efficient matrix-valued algorithms for solving stiff Riccati differential equations. *IEEE Trans. on Automatic Control*, 35(7):770–776, 1990.

[DDCHH88] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subroutines. *ACM Trans. Math. Software*, 1988.

[Die92] L. Dieci. Numerical integration of the differential Riccati equation and some related issues. *SIAM*, 29(3):781–815, 1992.

[DM73] E. J. Davison and M. C. Maki. The numerical solution of the matrix Riccati differential equation. *IEEE Trans. on Automatic Control*, 18:71–73, 1973.

[DVDG91] J. J. Dongarra and R. A. Van De Geijn. Two dimensional Basic Linear Algebra Communications Subprograms. Technical Report UT-CS-91-138, Department of Computer Science, University of Tennessee, October 1991.

[FGG97] Frayss N. V. , Giraud L., Gratton S. A set of GMRES routines for Real and Complex Arithmetics. CERFACS Technical Report TR/PA/97/49, 1997.

[GNV79] G. H. Golub, S. Nash and C. F. Van Loan. A Hessenberf-Schur method for the problem AX+XB=C. Transactions on Automatic Control 24, 909-913, 1979

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.

[Gui99] Guiomar Martín-Herran. Symplectic methods for the solution to Riccati matrix equations related to macroeconomic models. Computational Economis, 12, 61-91, 1999.

[Hin82] A. C. Hindmarsh. Brief description of ODEPACK- A systematized collection of ODE solvers. Technical report, www.netlib.org/odepack/doc, 1982.

[HoJo99] R. A. Horn and C. R. Johnson, Topics in Matrix Analysis. Cambridge University Press, 1999.

[Gr81] D. H. Griffel. Applied Functional Analysis. Ellis Hordwood Series, Mathematics and its applications, 1981.

[IN97] A. Iserles and S. P. Norsett. On the solution of linear differential equations in Lie groups. Technical report, Cambrigde University, United Kingdom, 1997. DAMTP NA3/1997.

[Isi89] A Isidori. *Nonlinear Control Systems: An Introduction*. Springer-Verlag, 1989.

[KL85] C. S. Kenney and R. B. Leipnik. Numerical integration of the differential matrix Riccati equation. *IEEE Trans. Automat. Control*, AC-:962–970, 1985.

[KMz87] Feng Kang and Qin Meng-zhao. The symplectic methods for the computation of Hamiltonian equations. In *Proceedings of the 1st Chinese Conference for Numerical Methods for PDE's*, pages 1–37, 1987.

[Kron78] J. W. Brewer. Kronecker product and matrix calculus in system theory. IEEE transactions on circuits and systems, 25 (9), 772-781, September, 1978.

[Lai76] D. G. Lainiotis. Partitioned Riccati solutions and integration-free doubling algorithms. *IEEE Trans. on Automatic Control*, 21:677–689, 1976.

[Lei85] R. B. Leipnik. A canonical form and solution for the matrix Riccati differential equation. *Bull Australian Math. Soc*, 26, pp- 355-361, 1985.

[Mag54] W. Magnus. In the exponential solution of differential equations for a linear operator. Comm. Pure nd Appl. Math., 7, 649-673, 1954..

[MKL] Mathematic Kernel Library. [http://www.intel.com/software/products/mkl/].

[Pein2003] J. Peinado. Parallel Resolution of Nonlinear Systems of Equations. PhD Thesis. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, España, July 2003.

[RW84] D. W. Rand and P. Winternitz. Nonlinear superposition principles: a new numerical method for solving matrix Riccati equations. *Comp. Phys. Commun.*, 33:305–328, 1984.

[Saad94] Yousef Saad. SPARSKIT: a basic tool kit for sparse matrix computations. CSDR, University of Illinois and NASA Ames Research Center, 1994.

[SGIAltix] SGI Altix Applications Development and Optimization - http://www.sgi.com/products/servers/altix/]

[SGI3000] SGI 3000 series [http://www.ccs.ornl.gov/Ram/sgi-tutorial.pdf]

[SL84] J. J. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, 1984.

[SS92] J. M. Sanz-Serna. Symplectic integrators for Hamiltonian problems: An overview. *Acta Numerica*, 1:243–286, 1992.

[SS99] J. Schiff. and S. Schneider. A natural approach to the numerical integration of Riccati differential equations. *SIAM*, 36(5):1392–1413, 1999.

[SV89] M.W. Spong and M. Vidyasagar. *Robot dynamics and control*. John Whiley and Sons, 1989.

[SW85] M Sorine and P. Winternitz. Superposition laws for the solution of differential Riccati equations. *IEEE*.

[Vaug69] D.R. Vaughan, "A negative exponential solution for the matrix Riccati equation". IEEE Trans. Automat. Control, vol AC-14, pp. 72-75, 1969.