Innovativity in Modeling and Analytics Journal of Research
vol. 1, 2016, pp.34-44,
http://imajor.info/

ISSN XXXX-XXXX

Iteratively computing the stabilizing solution of the discrete time Riccati equation with an indefinite quadratic part

Ivelin G. Ivanov
College of Dobrich, Shoumen University
Shoumen, Bulgaria
iwelin.ivanow@gmail.com

Abstract. The problem of computation of the stabilizing solution of a class of discrete-time Riccati equation with an indefinite sign of the quadratic term. Two iterative methods are considered and numerically compared. The performances of the proposed algorithm are illustrated on some numerical examples and Python implementations are commented. We apply Anaconda with Python 2.7 which includes the fundamental packages for scientific computing.

Key Words: Discrete-time Riccati equations, H_∞ problem, stabilizing solution, Python implementations.

1 Introduction

We consider the problem for solving a class of discrete-time Riccati equation with an indefinite sign of the quadratic term. The motivation behind this problematic is that such equation is closely related to the so called full information H_∞ control of discrete-time systems. More specifically, we will consider some reliable procedures for numerical computation of the stabilizing solution of such an equation. The so called stabilizing solution of discrete-time systems has a crucial role in solving a linear quadratic optimal control problem on infinite time horizon. This is due to non definiteness (of the sign) of the quadratic term of the considered Riccati equations, making most of the existing methods in the literature (which are built on the assumption of definite sign of the quadratic term) useless. The considered iterative methods can be viewed as an extension of the results in [2, 3, 1] for the deterministic continuous-time time-invariant case.

2 Two iterative methods

Consider the following discrete time Riccati equation (DTRE):

$$\begin{aligned} X \quad &= \mathcal{R}(X) := A^T X A - (A^T X B + C^T D) \\ &\quad \times \left[R_\gamma + B^T X B\right]^{-1} (A^T X B + C^T D)^T + C^T C \,, \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, $B = (B_1 \quad B_2), B_i \in \mathbb{R}^{n\times m_i}$, $C \in \mathbb{R}^{p\times n}$, $D = (D_1 \quad D_2), D_i \in \mathbb{R}^{p\times m_i}, i = 1,2$, $R_\gamma = D^T D + \begin{pmatrix} -\gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{m\times m}$, $m = m_1 + m_2, \gamma > 0$ is a given scalar.

In our investigation we are interested by all solutions X satisfying equation (1) and the following two sign conditions:

$$\begin{aligned} &D_2^T D_2 + B_2^T X B_2 > 0\,, \\ &D_1^T D_1 + B_1^T X B_1 - (B_1^T X B_2 + D_1^T D_2) \\ &\times \left[B_2^T X B_2 + D_2^T D_2\right]^{-1} (B_1^T X B_2 + D_1^T D_2)^T - \gamma^2 I_{m_1} < 0\,. \end{aligned} \tag{2}$$

Following [6] we can formulate a definition for the stabilizing solution to (1):

Definition 1 *A solution $\tilde{X}$ is called* **stabilizing solution**, *if the zero state equilibrium of the discrete - time linear system on $\mathbb{R}^n$:*

$$x(t+1) = [A + B\tilde{F}]x(t) \tag{3}$$

is exponentially stable, where

$$\tilde{F} = -[R_\gamma + B^T \tilde{X} B]^{-1} [B^T \tilde{X} A + D^T C]\,. \tag{4}$$

The stabilizing solution of DTRE (1) is important to find the solution of H_∞ control problem. This problem associated to the discrete - time linear system:

$$x(t+1) = A(t)x(t) + B_1(t)w(t) + B_2(t)u(t) \tag{5}$$

and the cost functional

$$J(w(\cdot), u(\cdot)) = \sum_{t=-\infty}^{\infty} \left(\|C(t)x(t) + D_1(t)w(t) + D_2(t)u(t)\|^2 - \gamma^2 \|w(t)\|^2\right) \tag{6}$$

with level of attenuation γ. Here $u(\cdot) \in \mathbb{R}^{m_2}$ are the control parameters and $w(\cdot) \in \ell_2(\mathbb{Z}, \mathbb{R}^{m_1})$ model the exogenous disturbances whose effect should be attenuated [6]

In this paper we consider and numerically compare two iterative procedures for the numerical computation of the stabilizing solution $\tilde{X}$ of (1). These methods are introduced in [6] and they extend to the discrete-time time-varying case the method developed in [1] for the deterministic continuous-time time - invariant case and in [2] for the stochastic continuous-time case.

The first iterative method constructs two matrix sequences of positive semidefinite matrices $\{X^{(k)}\}_{k=0}^{\infty}$, $\{Z^{(k)}\}_{k=0}^{\infty}$. We take $X^{(0)} = 0$ and then $X^{(k+1)} = X^{(k)} + Z^{(k)}$ $k = 0, 1, 2 \ldots$. Each iteration consists in the computation of the stabilizing solution of a suitable DTRE with defined sign of its quadratic part. The proposed algorithm may be described in the following steps:

Step 0. We compute $Z^{(0)}$ as the stabilizing solution of the DTRE:

$$\begin{aligned} Z^{(0)} \;&= A^T Z^{(0)} A - (A^T Z^{(0)} B_2 + C^T D_2) \\ &\times \left[B_2^T Z^{(0)} B_2 + D_2^T D_2\right]^{-1} (A^T Z^{(0)} B_2 + C^T D_2)^T + C^T C\,. \end{aligned} \tag{7}$$

Step k. ($k \geq 1$). We take $X^{(k)} = Z^{(k-1)} + X^{(k-1)}$ and compute $Z^{(k)}(\cdot)$ as the stabilizing solution of the DTRE with defined sign:

$$\begin{aligned} Z^{(k)} &= (A + BF^{(k)})^T Z^{(k)} (A + BF^{(k)}) - ((A + BF^{(k)})^T Z^{(k)} B_2) \\ &\times \left[R_2^{(k)} + B_2 Z^{(k)} B_2 \right]^{-1} ((A + BF^{(k)})^T Z^{(k)} B_2)^T + M^{(k)} \end{aligned} \tag{8}$$

where

$$\begin{cases} R_2^{(k)} = D_2^T D_2 + B_2^T X^{(k)} B_2(t)\,, \\ F^{(k)} = -\left[R_\gamma + B^T X^{(k)} B \right]^{-1} (B^T X^{(k)} A + D^T C)\,, \\ M^{(k)} = A^T X^{(k)} A - (A^T X^{(k)} B + C^T D) \\ \quad \times \left[R_\gamma + B X^{(k)} B \right]^{-1} (A^T X^{(k)} B + C^T D)^T + C^T C - X^{(k)} \end{cases} \tag{9}$$

Following iteration (9) derived in [9] we build an internal iterative process to compute $Z^{(k)}, k = 1, 2, \ldots$, where a discrete Lyapunov equation is solved at each iteration step.

The main iterative process stops with $\|X^{(k_0)} - \mathcal{R}(X^{(k_0)})\| \leq tol$.

Theorem 3.1 [6] proves the convergence of the matrix sequence defined by (7) - (9) to the stabilizing solution to (1).

We consider the second iterative method presented by (15) in [5]:

$$\begin{aligned} X^{(k+1)} &= (A^{(k)})^T X^{(k+1)} (A^{(k)}) - ((A^{(k)})^T X^{(k+1)} B_2 + (C^{(k)})^T D_2) \\ &\times \left[D_2^T D_2 + B_2^T X^{(k+1)} B_2 \right]^{-1} ((A^{(k)})^T X^{(k+1)} B_2 + (C^{(k)})^T D_2)^T + Q^{(k)} \end{aligned} \tag{10}$$

where

$$\begin{cases} A^{(k)} = A + B_1 F_{1,X^{(k)}}\,, \\ C^{(k)} = C + D_1 F_{1,X^{(k)}}\,, \\ \tilde{F}_{2,X^{(k+1)}} = -(D_2^T D_2 + B_2^T X^{(k+1)} B_2)^{-1} (A^{(k)^T} X^{(k+1)} B_2 + C^{(k)^T} D_2)^T\,, \\ Q^{(k)} = C^{(k)^T} C^{(k)} - \gamma^2 F_{1,X^{(k)}}^T F_{1,X^{(k)}}\,. \end{cases} \tag{11}$$

Theorem 3.2 [5] proves the convergence of the matrix sequence defined by (10) - (11) to the stabilizing solution to (1).

In order to compute the solution $X^{(k+1)}$ to (10) Dragan et.al. have proposed the iteration (40) with notations (41)-(42) from [5]. Using the approach introduced by Ivanov in [9] we derive the following identity for each symmetric matrix W:

$$\begin{aligned} X^{(k+1)} &= (A^{(k)} + B_2 \tilde{F}_{2,W})^T X^{(k+1)} (A^{(k)} + B_2 \tilde{F}_{2,W}) + T_{2,W,X^{(k)}} \\ &-(\tilde{F}_{2,X^{(k+1)}} - \tilde{F}_{2,W})^T \left[D_2^T D_2 + B_2^T X^{(k+1)} B_2 \right] (\tilde{F}_{2,X^{(k+1)}} - \tilde{F}_{2,W})\,, \end{aligned} \tag{12}$$

with

$$T_{2,W,X^{(k)}} = \begin{pmatrix} I_n \\ \tilde{F}_{2,W} \end{pmatrix}^T \begin{pmatrix} Q^{(k)} & C^{(k)^T} D_2 \\ D_2^T C^{(k)} & D_2^T D_2 \end{pmatrix} \begin{pmatrix} I_n \\ \tilde{F}_{2,W} \end{pmatrix}. \tag{13}$$

The validity of identity 12 is confirmed by direct manipulations. Using identity 12 we obtain an iterative process to compute $X^{(k+1)}$. We construct a matrix sequence $Y^{(0)}, Y^{(1)}, \ldots,$ $Y^{(s)}, \ldots$ and $Y^{(s+1)}$ is a solution of the matrix equation:

$$Y^{(s+1)} = (A^{(k)} + B_2\tilde{F}_{2,Y^{(s)}})^T Y^{(s+1)}(A^{(k)} + B_2\tilde{F}_{2,Y^{(s)}}) + T_{2,Y^{(s)},X^{(k)}}) \,. \tag{14}$$

It is easy to verify that if the matrix sequence $\{Y^{(s)}\}$ is a convergent one, then it converges to $X^{(k+1)}$. In fact (14) is the iteration (40) from [5]. The convergence properties of (14) is derived in [9].

Further on, we apply the Larin algorithm described in [11] for computing the stabilizing solution to (1). However, we slightly modify this algorithm in this paper. The algorithm begins with the consideration of the matrix pencil $M - \mu F$ to find the stabilizing solution to (1), where M and F are $2n \times 2n$ block matrices

$$M = \begin{pmatrix} A - BW^{-1}(B^TA + N^T) & 0 \\ NW^{-1}(B^TA + N^T) - Q & I \end{pmatrix}, \quad F = \begin{pmatrix} I - BW^{-1}B^T & BW^{-1}B^T \\ NW^{-1}B^T & A - NW^{-1}B^T \end{pmatrix}$$

with

$$\begin{cases} W = R + B^TB \\ N = C^TD \\ Q = C^TC \end{cases} .$$

The Hamiltonian matrix is determined by Cayley transformation on the matrix pencil $M - \mu F$:

$$H = (M - F)^{-1}(M + F) = \begin{pmatrix} U & -T \\ -G & -U^T \end{pmatrix} . \tag{15}$$

The algebraic Riccati equation

$$YU + U^TY - YTY + G = 0 \tag{16}$$

corresponds to the Hamiltonian matrix H. The stabilizing solution to (16) is the required solution to (1).

Now, we explain our modification. We transform the matrix H in the Schur form S or the diagonal matrix D as is described in [7]

$$H\,E = E\,S\,, \quad (\,H\,V = V\,D\,)\,.$$

Note that the above transformation is a similarity transformation, i.e. $V^{-1}H\,V = D$. This transformation can be done via a QR algorithm [8]. Since H is a Hamiltonian matrix then there exits a permutation matrix such that $\tilde{D} = P\,D\,P = diag[\tilde{U}, -\tilde{U}]\,,\ (P*P = I)$, where $\tilde{U}$ contains n eigenvalues of H with negative real parts,i.e. $\tilde{U}$ is a stable matrix. Thus

$$V^{-1}H\,V = D = P\,\tilde{D}\,P$$

or

$$\tilde{V}^{-1}\,H\,\tilde{V} = \tilde{D} = diag[\tilde{U}, -\tilde{U}]\,, \qquad \tilde{V} = V\,P. \tag{17}$$

The columns of the matrix $\tilde{V} = \begin{pmatrix} \tilde{V}_{11} & \tilde{V}_{12} \\ \tilde{V}_{21} & \tilde{V}_{22} \end{pmatrix}$, ($\tilde{V}_{11}$ is an $n \times n$ matrix) are the eigenvectors corresponding to eigenvalues of $\tilde{D}$. Thus $\begin{pmatrix} \tilde{V}_{11} \\ \tilde{V}_{21} \end{pmatrix}$ is an invariant linear subspace for the matrix H. Thus, $X_S = \tilde{V}_{21}\,\tilde{V}_{11}^{-1}$ is the stabilizing solution to (16) and moreover the stabilizing solution to (1).

We can summarize the above as follows:

Algorithm 3.

1. Compute matrices $2n \times 2n$ matrices M, N.
2. Compute the Hamiltonian matrix H defined by (15).
3. Compute the eigen factorization of H (17).
4. Compute the stabilizing solution $X_S = \tilde{V}_{21}\,\tilde{V}_{11}^{-1}$ of (16).

Moreover, there are two weakness- Algorithm 3 uses matrices with 2 times bigger dimensions and it works in complex arithmetic. It needs to compute eigen factorization (17) in general case, i.e. this factorization is executed in complex arithmetic because the matrix H has usually complex eigenvalues.

3 Numerical experiments

We carry out some numerical experiments for computing the stabilizing solution to discrete time Riccati equation (1). We operate with the iterative methods - the first one is presented by (7) - (9) and the second one is given by (10) - (11), and Algorithm 3.

Continuum Analytics [12] offers the Python dsitribution through Anaconda. Python is an object-oriented and high-level programming language with dynamic semantics. Python is free and open source. One can found the common scientific Python packages and many ones related to data analytics [10]. The Anaconda is a free distribution with hundreds of cross-platform tested and optimized packages for Mac OS X, Windows, and Linux users. There are free licenses for academics and researchers. Fundamental array processing possibilities are provided by the NumPy library. Our experiments are executed in Anaconda with Python 2.7 on a 2,16GHz Intel(R) Duo CPU computer.

Here we describe how to implement the considered methods in Python. Iterative method (7) - (9) is implemented by the following Python function:

```
def DTRE_IQP_3(a,b1,b2,d1,d2,c,gam,n,tol,eps):
  d=np.concatenate((d1, d2),axis=1)
  b=np.concatenate((b1, b2),axis=1)
  X0=0*np.identity(n); Z0=2*np.identity(n)
  for j in range(0, 4):
    Fz=-inv(d2.T*d2 + b2.T*Z0*b2)
    Fz=Fz*(b2.T*Z0*a + d2.T*c)
    tA=a+b2*Fz
    Tz=c.T*c + Fz.T*d2.T*c+c.T*d2*Fz+Fz.T*d2.T*d2*Fz
    Z0=tA.T*Z0*tA+Tz

  X0=X0 + Z0
```

```
y0=np.concatenate((-gam*gam*np.identity(2), 0*np.identity(2)),axis=1)
y1=np.concatenate((0*np.identity(2), 0*np.identity(2)),axis=1)
Rg0=d.T*d+np.concatenate((y0, y1))
errE=1; k=0;
while errE > tol :
  Zk= 2*np.identity(n)
  Fk0=-inv(Rg0+b.T*X0*b)*(b.T*X0*a+d.T*c)
  Ak=a+b*Fk0
  thatR2=d2.T*d2+b2.T*X0*b2
  SM0=a.T*X0*b+c.T*d
  hatM0=a.T*X0*a-SM0*inv(Rg0+b.T*X0*b)*SM0.T+c.T*c-X0;
  j=0; errZ=1;
  while errZ > eps:
    Sk=Ak.T*Zk*b2
    FFz=-inv(thatR2+b2.T*Zk*b2)*Sk.T
    ttA=Ak+b2*FFz
    TTz=hatM0 + FFz.T*thatR2*FFz
    # discrete_lyapunov
    # ttA'*Zk*ttA - Zk + TTz = 0
    Zk=linalg.solve_discrete_lyapunov(ttA.T, TTz)
    Sk=Ak.T*Zk*b2
    rezRR=Ak.T*Zk*Ak-Sk*inv(thatR2+b2.T*Zk*b2)*Sk.T+hatM0-Zk
    errZ=linalg.norm(rezRR)
    j=j+1
    #   end while
  X0=X0 + Zk
  S0=a.T*X0*b+c.T*d
  rezX0=X0-a.T*X0*a+S0*inv(Rg0+b.T*X0*b)*S0.T-c.T*c
  errE=linalg.norm(rezX0)
  k=k+1
  #  end while
return X0
```

Iterative method (10) - (11) is implemented by the following Python function:

```
def DTRE_IQP_4(a,b1,b2,d1,d2,c,gam,n,tol,eps):
  d=np.concatenate((d1, d2),axis=1)
  b=np.concatenate((b1, b2),axis=1)
  y0=np.concatenate((-gam*gam*np.identity(2), 0*np.identity(2)),axis=1)
  y1=np.concatenate((0*np.identity(2), 0*np.identity(2)),axis=1)
  Rg0=d.T*d+np.concatenate((y0, y1))
  X0=0*np.identity(n); errE=1; j=0;
  while errE > tol :
    Fk0=-inv(Rg0+b.T*X0*b)*(b.T*X0*a+d.T*c)
    Fk10=Fk0[0:2,:]
```

```
    Ak=a+b1*Fk10
    Ck=c+d1*Fk10
    Qk=Ck.T*Ck-gam*gam*Fk10.T*Fk10
    #
    Yk=2*np.identity(n)
    i=0; errZ=1;
    # internal iteration (14)
    while errZ > eps:
      thatR2=d2.T*d2+b2.T*Yk*b2
      FFy=-inv(thatR2)*(Ak.T*Yk*b2+Ck.T*d2).T
      ttA=Ak+b2*FFy
      TTy=Qk + FFy.T*d2.T*Ck + Ck.T*d2*FFy + FFy.T*d2.T*d2*FFy
      # discrete_lyapunov
      # ttA'*Yk*ttA - Yk + TTy = 0
      Yk=linalg.solve_discrete_lyapunov(ttA.T, TTy)
      Sk=Ak.T*Yk*b2+Ck.T*d2
      rezRR=Ak.T*Yk*Ak-Sk*inv(d2.T*d2+b2.T*Yk*b2)*Sk.T+Qk
      rezRR=Yk-rezRR
      errZ=linalg.norm(rezRR)
      i=i+1

      # end while
    X0=Yk
    S0=a.T*X0*b+c.T*d;
    rezX0=X0-a.T*X0*a+S0*inv(Rg0+b.T*X0*b)*S0.T-c.T*c
    errE=linalg.norm(rezX0)
    j=j+1

  # end while
  return X0
```

Algorithm 3 is implemented by the following Python function:

```
def DTRE_IQP_5(a,b1,b2,d1,d2,c,gam,n):
  d=np.concatenate((d1, d2),axis=1)
  b=np.concatenate((b1, b2),axis=1)
  y0=np.concatenate((-gam*gam*np.identity(2), 0*np.identity(2)),axis=1)
  y1=np.concatenate((0*np.identity(2), 0*np.identity(2)),axis=1)
  Rg0=d.T*d+np.concatenate((y0, y1))
  N=c.T*d;    Q=c.T*c
  W=Rg0+b.T*b;    iW=inv(W)
  bN=b.T*a+N.T
  M1=np.concatenate(( a-b*iW*bN, np.matlib.zeros((n,n))),axis=1)
  M2=np.concatenate(( N*iW*bN - Q, np.matlib.identity(n) ),axis=1)
  M=np.concatenate(( M1,M2),axis=0)
  bW=b*iW*b.T
```

```
nW=N*iW*b.T
F1 = np.concatenate(( np.matlib.identity(n)-bW, bW),axis=1)
F2 = np.concatenate(( nW, a.T-nW),axis=1)
F=np.concatenate(( F1,F2),axis=0)
H=inv(M-F)*(M+F)
w,v=eig(H)
# rearrangement
ind = argsort(w)
v = v[:,ind]
u11=v[0:n,0:n]
u21=v[n:2*n,0:n]
# the solution XS
XS=u21*inv(u11)
return XS
```

In this section we apply the introduced the procedures DTRE_IQP_3, DTRE_IQP_4 and DTRE_IQP_5 to compute the stabilizing solution to (1). We execute different numerical simulations in order to compare the considered procedures. We use two variables *tol* and *eps* for small positive numbers to control the accuracy of the computations. We use $tol = eps = 1.0e-7$.

Example 1. We construct the matrix coefficients of (1) as follows:

The matrix coefficients are:

$$A = \begin{pmatrix} -0.48 & 0. & 0.16 \\ 1.6 & 0.8 & 0. \\ 0.048 & -3.2 & 0.64 \end{pmatrix}, \quad B_1 = \begin{pmatrix} -1.8048 & 0.5877 \\ -0.3916 & -0.0175 \\ 1.2778 & 0.5082 \end{pmatrix}, \quad D_1 = \begin{pmatrix} 0.35 & -0.45 \\ 0.28 & -0.13 \end{pmatrix},$$

$$C = \begin{pmatrix} 0.233 & 1.5 & 0.3 \\ 0.2 & 1.5 & 0.28 \end{pmatrix}, \quad B_2 = \begin{pmatrix} 0.4089 & -1.0077 \\ -0.1932 & -0.4189 \\ 0.4144 & -1.1287 \end{pmatrix}, \quad D_2 = \begin{pmatrix} 0.15 & 0.75 \\ -0.18 & -0.25 \end{pmatrix}.$$

We apply the procedure DTRE_IQP_3 and the DTRE_IQP_4(a,b1,b2,d1,d2,c) to compute the stabilizing solution to (1). The procedure DTRE_IQP_3 requires 3 main iterations and it computes the stabilizing solution with $\|\tilde{X}^{(3)} - \mathcal{R}(\tilde{X}^{(3)})\| = 6.44e-15 < tol$ with $tol = 1.0e-9; eps = 1.0e-11$. It computes the stabilizing solution for 0.96 seconds for 100 runs.

The procedure DTRE_IQP_4 requires 3 main iterations and it computes the stabilizing solution with $\|\tilde{X}^{(3)} - \mathcal{R}(\tilde{X}^{(3)})\| = 8.40e-15 < tol$ with $tol = 1.0e-9; eps = 1.0e-11$. It computes the stabilizing solution for 2.11 seconds for 100 runs.

The procedure DTRE_IQP_5 computes the stabilizing solution X_S with $\|X_S - \mathcal{R}(X_S)\| = 3.3e-13$. It computes the stabilizing solution for 0.032 seconds for 100 runs.

Example 2. We construct more general matrix coefficients. The matrix coefficients D_1, D_2 to (1) as in Example 1. The matrix A is an $n \times n$ matrix, B_1 and B_2 are $n \times 2$

Table 1: Example 2. Results from 50 runs for each value of n.

	DTRE_IQP_3			DTRE_IQP_4			DTRE_IQP_5	
n	It	CPU	max Err	It	CPU	max Err	CPU	max Err
12	2	0.29s	9.5e-8	1	0.29s	9.3e-8	0.06s	1.5e-12
24	2	0.61s	8.6e-8	1	0.49s	9.7e-8	0.15s	2.2e-12
120	2	4.68s	9.2e-8	1	3.56s	8.4e-8	2.28s	8.8e-7
240	2	33.24s	4.2e-8	1	26.59s	6.16e-8	10.4s	2.1e-5
500	2	444.55s	4.6e-10	1	418.8s	5.66e-8	73.4s	1.7e-3

matrices and C is a $2 \times n$ matrix. They are described as follows:

$$A_{n\times n} = \begin{pmatrix} 0.33 & 0 & \dots & 0 \\ 0 & 0.33 & \dots & 0 \\ \vdots & & \vdots & \vdots \\ 2.75 & 0 & \dots & 0.33 \end{pmatrix}, \quad C_{2\times n} = \begin{pmatrix} 0 & \dots & 0 \\ 0.4 & \dots & 0.4 \end{pmatrix},$$

$$B_{1,n\times 2} = \begin{pmatrix} uniform(-1.5, 0.5) & uniform(-1.5, 0.5) \\ \vdots & \vdots \\ uniform(-1.5, 0.5) & uniform(-1.5, 0.5) \end{pmatrix},$$

$$B_{2,n\times 2} = \begin{pmatrix} uniform(-1.5, 0.5) & uniform(-1.5, 0.5) \\ \vdots & \vdots \\ uniform(-1.5, 0.5) & uniform(-1.5, 0.5) \end{pmatrix}.$$

We compute the stabilizing solution to (1) for different values of n. We execute 50 different runs for each value of n. We create new matrices B_1, B_2 for each run. The results from experiments are presented in Table 1. In addition, the column "CPU" presents the CPU time for execution of the corresponding procedures for all 50 runs. We define the variable $max\,Err = max_{1\le k\le 50} \|\tilde{X}^{(k_0)} - \mathcal{R}(\tilde{X}^{(k_0)})\|$ and we write the value of $max\,Err$ in the column "max Err". Note that the procedures DTRE_IQP_3 and DTRE_IQP_4 stop when $max\,Err \le tol$. The column "max Err" for the procedure DTRE_IQP_5 provides the norm $\|X_S - \mathcal{R}(X_S)\|$, where X_S is the computed stabilizing solution. In fact this norm estimate the accuracy of this procedure. The results show that that the procedure DTRE_IQP_5 is faster than other procedures. However, the procedure DTRE_IQP_5 achieves less accuracy than other procedures when the value of n increases (see the last column of Table 1.

4 Conclusion

We have studied two iterative procedures for finding the stabilizing solution to discrete time Riccati equation (1). Numerical experiments are carried out and the obtained results are used for comparison purposes. In order to implement the numerical computations we use the platform Anaconda with Python 2.7 which contains open source packages for

scientific computations. Thus, the following conclusions might be outlined. On one hand, the effectiveness of the considered iterative methods (7) - (9) and (10) - (11) is confirmed. On the other hand the third procedure DTRE_IQP_5, based on the computation of the eigenvalues and the eigenvectors of an $2n \times 2n$ block matrix H, is found to be faster than the iterative methods. This procedure uses least computational time for computing the stabilizing solution. This conclusion is retained true in the case of Riccati equations with large dimensions of matrix coefficients. Moreover, the third procedure computes the solution X_S the third procedure computes the solution X_S under the rule when the dimension n of X_S increases then the computed accuracy decreases.

References

[1] A. Lanzon, Y. Feng, B. D. O. Anderson and M. Rotkowitz, Computing the Positive Stabilizing Solution to Algebraic Riccati Equations with an Indefinite Quadratic Term via a Recursive Method, *IEEE Trans. Automat. Contr.* , 53, 10, 2008, 2280–2291.

[2] Y. Feng, B.D.O. Anderson, An iterative algorithm to solve state-perturbed stochastic algebraic Riccati equations in LQ zero-sum games, *Systems and Control Letters*, 59, 2010, 50–56.

[3] Y. Feng, B. D. O. Anderson, Weitian Chen, Solving Discrete Algebraic Riccati Equations: A New Recursive Method, *Proceedings of joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference Shanghai, P.R. China, December 16-18, 2009*

[4] J. F. do Amaral, T. P. de Lima, M. S. Silva, *Positive Solutions of a Discrete-time System and the Leontief Price-Model*, LNCIS, 2006, 341, 65–72, Springer, in Positive Systems, C.Commault and N. Marchand (editors) Proceedings of the Second Multidisciplinary International Symposium on Positive Systems.

[5] V. Dragan, S. Aberkane, I. Ivanov, An iterative procedure for computing the stabilizing solution of discrete-time periodic Riccati equations with an indefinite sign, 21st International Symposium on Mathematical Theory of Networks and Systems July 7-11, 2014. Groningen.

[6] V. Dragan, S. Aberkane, I. Ivanov, On computing the stabilizing solution of a class of discrete-time periodic Riccati equations, International Journal of Robust and Nonlinear Control, vol.25, 7, 2015, 1066-1093, doi: 10.1002/rnc.3131

[7] H.D. Ikramov, The Numerical Solution of Matrix Equations. Nauka, Moscow (1984) (in Russian).

[8] G.H. Golub, C. F. Van Loan, Matrix Computations, forth edition, Johns Hopkins University Press, 2012.

[9] I. Ivanov, Properties of Stein (Lyapunov) iterations for solving a general Riccati equation, Nonlinear Analysis 67, 2007, 1155–1166.

[10] H. P. Langtange, A Primer on Scientific Programming with Python, University of Oslo, 2014, https://hplgit.github.io/primer.html/doc/pub/half/book.pdf, December, 2016.

[11] V.B. Larin, High-Accurasy Algorithms for Solving of Discrete Periodic Riccati Equation. Appl. Comput. Math., 6, 1, 2007, 10–17.

[12] https://www.continuum.io/ December 2016