LYAPACK

A MATLAB Toolbox for Large Lyapunov and Riccati Equations, Model Reduction Problems, and Linear–Quadratic Optimal Control Problems

Users' Guide (Version 1.0)

Thilo Penzl

Preface

Control theory is one of the most rapidly developing disciplines of mathematics and engineering in the second half of the 20th century. In the past decade, implementations of numerically robust algorithms for many types of dense problems in control theory have become available in software packages, such as SLICOT [7]. However, little research has been done on efficient numerical methods for control problems related to large sparse or structured dynamical systems before 1990. In the last few years, quite a number of approaches for several types of large control problems have been proposed, but, at present, it is often not clear, which of them are the more promising ones. It is needless to say that there is little software for large control problems available. In this situation, the author took the opportunity to implement the software package LYAPACK ("Lyapunov Package"), which covers one particular approach to a class of large problems in control theory. An efficient ADI-based solver for large Lyapunov equations is the "workhorse" of LYAPACK, which also contains implementations of two model reduction methods and modifications of the Newton method for the solution of large Riccati equations and linear-quadratic optimal control problems. Most of the underlying algorithms have been developed by the author in the past three years. A part of this research was done simultaneously and independently by Jing-Rebecca Li. A benefit of her work to LYAPACK is in particular an improvement in the efficiency of the Lyapunov solver.

LYAPACK aims at two goals. First, of course, the package will hopefully be used to solve problems that arise from practical applications. The availability of easy-to-use software is surely one step to make practitioners consider alternative numerical techniques: "unless mathematics is put into software, it will never be used" [The SIAM Report on Mathematics in Industry, 1996]. (This statement might be somewhat too strong. And, of course, the reverse statement is not necessarily true.) Second, SLICOT can be considered as a contribution to a fair and comprehensive comparison of the existing methods for large Lyapunov equations, model reduction problems, etc., which is yet to be done.

For several reasons LYAPACK has been implemented in MATLAB[1] rather than programming languages like FORTRAN, C, or JAVA. MATLAB codes are easier to understand, to modify, and to verify. On the other hand, their performance cannot compete with that of codes in the aforementioned programming languages. However, this does not mean that LYAPACK is restricted to the solution of "toy problems". Several measures, such as the use of global variables for large data structures, have been taken to enhance the computational performance of LYAPACK routines. To put this into the right perspective, Lyapunov equations of order larger than 12000 were solved by LYAPACK within few hours on a regular workstation. When using standard methods, supercomputers are needed to solve problems of this size.

LYAPACK was implemented and tested in a UNIX environment. Note, in particular, that the file names of some routines do not comply the DOS-like "xxxxxxxx.yyy" naming convention.

The author acknowledges the support of the DAAD (Deutscher Akademischer Austauschdienst = German Academic Exchange Service). He is grateful to Peter Benner, Peter Lancaster, Jing-Rebecca Li, Volker Mehrmann, Enrique Quintana-Orti, and Andras Varga for their direct or indirect help on the project. He also wants to thank the staff of

[1]MATLAB is a trademark of The MathWorks Inc.

"The First Cup" (University of Calgary), where the considerable quantity of coffee was produced, which was needed to realize the LYAPACK project.

Finally, it should be stressed that any kind of feedback from people who applied or tried to apply this package is highly appreciated.

Thilo Penzl
Calgary, November 1999

Addendum to Preface

This manuscript was mostly finished just before Thilo Penzl died in a tragic accident in December 1999, a few days before his return to work in the Numerical Analysis Group at TU Chemnitz where he also completed his PhD in 1998. I felt that this very nice piece of work should be made available to the scientific community and we therefore tested the codes, proofread the manuscript and performed minor corrections in the text. The MATLAB codes were tested by Falk Ebert and the corrections to the Users' Guide were performed by myself.

Any comments or questions concerning the package should be addressed to Volker Mehrmann *mehrmann@mathematik.tu-chemnitz.de*.

The LYAPACK codes are available at *http://www.tu-chemnitz.de/sfb393/lyapack*

Volker Mehrmann
Chemnitz, August 2000

Disclaimer and usage notes

- The author disclaims responsibility for any kind of damage sustained in context with the use of the software package LYAPACK.
- LYAPACK is restricted to non-commercial use.
- References to LYAPACK and/or to the publications on the underlying numerical methods must be provided in reports on numerical computations in which LYAPACK routines are involved.

Contents

1 Introduction

1.1 What is LYAPACK?

LYAPACK is the acronym for “Lyapunov Package”. It is a MATLAB toolbox (i.e., a set of MATLAB routines) for the solution of certain large scale problems in control theory, which are closely related to Lyapunov equations. Basically, LYAPACK works on realizations

$$\begin{aligned} \dot{x}(\tau) &= Ax(\tau) + Bu(\tau) \\ y(\tau) &= Cx(\tau) \end{aligned} \tag{1}$$

of continuous-time, time-invariant, linear, dynamical systems, where $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$, $C \in \mathbb{R}^{q,n}$, and $\tau \in \mathbb{R}$. n is the *order* of the system (1). LYAPACK is intended to solve problems of large scale (say $n > 500$). The matrices A, B, and C must fulfill certain conditions, which are discussed in more detail in §1.2. We call the entries of the vectors (or, more precisely, vector-valued functions) u, x, and y the *inputs*, *states*, and *outputs* of the dynamical system, respectively.

There are three types of problems LYAPACK can deal with.

- **Solution of Lyapunov equations.** Continuous-time algebraic *Lyapunov equations* (CALEs) play the central role in LYAPACK. Lyapunov equations are linear matrix equations of the type
$$FX + XF^T = -GG^T, \tag{2}$$
where $F \in \mathbb{R}^{n,n}$ and $G \in \mathbb{R}^{n,t}$ are given and $X \in \mathbb{R}^{n,n}$ is the solution. In some applications the solution X itself might be of interest, but mostly it is only an auxiliary matrix, which arises in the course of the numerical solution of another problem. Such problems are model reduction, Riccati equations, and linear-quadratic optimal control problems, for example.

- **Model reduction.** Roughly speaking, *model reduction* is the approximation of the dynamical system (1) by a system
$$\begin{aligned} \dot{\hat{x}}(\tau) &= \hat{A}\hat{x}(\tau) + \hat{B}u(\tau) \\ y(\tau) &= \hat{C}\hat{x}(\tau) \end{aligned} \tag{3}$$
of smaller order k, whose behavior is similar to that of the original one in some sense. There exist a large number of model reduction methods which rely on Lyapunov equations [2]. LYAPACK contains implementations of two such methods. Both are based on the Lyapunov equations
$$AX_B + X_BA^T = -BB^T \tag{4}$$
$$A^TX_C + X_CA = -C^TC. \tag{5}$$
Their solutions X_B and X_C are called *controllability Gramian* and *observability Gramian* of the system (1), respectively.

- **Riccati equations and linear-quadratic optimal control problems.** The minimization of
$$\mathcal{J}(u, y, x_0) = \frac{1}{2}\int_0^\infty y(\tau)^TQy(\tau) + u(\tau)^TRu(\tau)d\tau \tag{6}$$

subject to the dynamical system (1) and the initial condition $x(0) = x_0$ is called the *linear-quadratic optimal control problem* (LQOCP). Its optimal solution is described by the state-feedback

$$u(\tau) = -R^{-1}B^TXx(\tau) =: -K^Tx(\tau), \tag{7}$$

which can be computed by solving the (continuous-time algebraic) *Riccati equation* (CARE)

$$C^TQC + A^TX + XA - XBR^{-1}B^TX = 0. \tag{8}$$

Riccati equations also arise in further applications in control theory.

LYAPACK contains routines for these three types of problems. The underlying algorithms are efficient w.r.t. both memory and computation for many large scale problems.

1.2 When can LYAPACK be applied?

There exist a number of conditions, that must be fulfilled by the dynamical system (1) to guarantee applicability and usefulness of LYAPACK:

- **Stability.** In most cases, the matrix A must be stable, i.e., its spectrum must be a subset of the open left half of the complex plane. For the solution of Riccati equations and optimal control problems it is sufficient that a matrix $K^{(0)}$ is given, for which $A - BK^{(0)^T}$ is stable.

- **The number of the inputs and outputs must be small** compared to the number of states, i.e., $m \ll n$ and $q \ll n$. As a rule of thumb, we recommend $n/m, n/q \geq 100$. The larger these ratios are, the better is the performance of LYAPACK compared to implementations of standard methods.

- **The matrix A must have a structure, which allows the efficient solution of (shifted) systems of linear equations and the efficient realization of products with vectors.** Examples for such matrices are classes of sparse matrices, products of sparse matrices and inverses of sparse matrices, circulant matrices, Toeplitz matrices, etc.

At this point, it should be stressed that problems related to certain *generalized dynamical systems*

$$\begin{aligned} M\dot{\tilde{x}}(\tau) &= N\tilde{x}(\tau) + \tilde{B}u(\tau) \\ y(\tau) &= \tilde{C}\tilde{x}(\tau) \end{aligned} \tag{9}$$

where $M, N \in \mathbb{R}^{n,n}$, can be treated with LYAPACK as well. However, it is necessary that the generalized system can be transformed into a stable, standard system (1). This is the case when M is invertible and $M^{-1}N$ is stable. The transformation is done by an LU factorization (or a Cholesky factorization in the symmetric definite case) of M, i.e., $M = M_LM_U$. Then an equivalent standard system (1) is given by

$$A = M_L^{-1}NM_U^{-1}, \quad B = M_L^{-1}\tilde{B}, \quad C = \tilde{C}M_U^{-1}. \tag{10}$$

1.3 When can or should LYAPACK not be applied?

To avoid misunderstandings and to make the contents of the previous section more clear, it should be pointed out that the following problems cannot be solved or should not be attempted by LYAPACK routines.

- LYAPACK cannot solve Lyapunov equations and model reduction problems, where the system matrix A is not stable. It cannot solve Riccati equations and optimal control problems if no (initial) stabilizing feedback is provided.
- LYAPACK cannot be used to solve problems related to *singular systems*, i.e., generalized systems (9) where M is singular.
- LYAPACK is not able to solve problems efficiently which are **highly** "ill-conditioned" (in some sense). LYAPACK relies on iterative methods. Unlike direct methods, whose complexity does usually not depend on the conditioning of the problem, iterative methods generally perform poorly w.r.t. both accuracy and complexity if the problem to be solved is highly ill-conditioned.
- LYAPACK is inefficient if the system is of small order (say, $n \leq 500$). In this case, it is recommended to apply standard methods to solve the problem; see §7.
- LYAPACK is inefficient if the number of inputs and outputs is not much smaller than the system order. (For example, there is not much sense in applying LYAPACK to problems with, say, 1000 states, 100 inputs, and 100 outputs.)
- LYAPACK is not very efficient if it is not possible to realize basic matrix operations, such as products with vectors and the solution of certain (shifted) systems of linear equations with A, in an efficient way. For example, applying LYAPACK to systems with an unstructured, dense matrix A is dubious.
- LYAPACK is not intended to solve discrete-time problems. However, such problems can be transformed into continuous-time problems by the Cayley transformation. It is possible to implement the structured, Cayley-transformed problem in user-supplied functions; see §2.2.
- LYAPACK cannot handle more complicated types of problems, such as problems related to time-invariant or nonlinear dynamical systems.

1.4 Highlights and features

LYAPACK consists of the following components (algorithms):

- **Lyapunov equations** are solved by the *Low Rank Cholesky Factor ADI* (LRCF-ADI) iteration. This iteration is implemented in the LYAPACK routine `lp_lradi`, which is the "workhorse" of the package.
- The performance of LRCF-ADI depends on certain parameters, so-called **ADI shift parameters**. These can be computed by a heuristic algorithm provided as routine `lp_para`.
- There are two **model reduction** algorithms in LYAPACK. Algorithm LRSRM, that is implemented in the routine `lp_lrsrm`, is a version of the well-known square-root method, which is a balanced truncation technique. Algorithm DSPMR provided as routine `lp_dspmr` is more heuristic in nature and related to dominant controllable and observable subspaces. Both algorithms heavily rely on low rank approximations to the system Gramians X_B and X_C provided by `lp_lradi`).
- **Riccati equations** and **linear-quadratic optimal control problems** are solved by the *Low Rank Cholesky Factor Newton Method* (LRCF-NM) or the *Implicit LRCF-NM* (LRCF-NM-I). Both algorithms are implemented in the routine `lp_lrnm`.

- LYAPACK contains some **supplementary routines**, such as routines for generating test examples or Bode plots, and a number of **demo programs**.
- A basic concept of LYAPACK is that matrix operations with A are implicitly realized by so-called **user-supplied functions** (USFs). For general problems, these routines must be written by the users themselves. However, for the most common problems such routines are provided in LYAPACK.

In particular, the concept of user-supplied functions, which relies on the storage of large data structures in global MATLAB variables, makes LYAPACK routines efficient, w.r.t. both memory and computation. Of course, LYAPACK could not compete with FORTRAN or C implementations of the code (if there were any). However, this package can be used to solve problems of quite large scale efficiently. The essential advantages of a MATLAB implementation are, of course, clarity and the simplicity of adapting and modifying the code.

Versatility is another feature of LYAPACK. The concept of user supplied functions does not only result in a relatively high degree of numerical efficiency, it also enables solving classes of problems with a complicated structure (in particular, problems related to systems, where the system matrix A is not given explicitly as a sparse matrix).

Typically, large scale problems are solved by iterative methods. In LYAPACK iterative methods are implemented in the routines `lp_lradi`, `lp_lrnm`, `lp_para`, and some user supplied functions. LYAPACK offers a variety of stopping criteria for these iterative methods.

2 Realization of basic matrix operations

In this section we describe in detail how operations with the structured system matrix A are realized in LYAPACK. Understanding this is important for using LYAPACK routines. However, this section can be skipped by readers who only want to get a general idea of the algorithms in LYAPACK.

2.1 Basic matrix operations

The efficiency of most LYAPACK routines strongly depends on the way how matrix operations with the structured matrix A are implemented. More precisely, in LYAPACK three types of such *basic matrix operations* (BMOs) are used. In this section, X denotes a complex $n \times t$ matrix, where $t \ll n$.

- **Multiplications** with A or A^T:

$$Y \longleftarrow AX \quad \text{or} \quad \longleftarrow A^T X.$$

- Solution of **systems of linear equations** (SLEs) with A or A^T:

$$Y \longleftarrow A^{-1}X \quad \text{or} \quad \longleftarrow A^{-T} X.$$

- Solution of **shifted systems of linear equations** (shifted SLEs) with A or A^T, where the shifts are the ADI parameters (see §3.2):

$$Y \longleftarrow (A + p_i I_n)^{-1} X \quad \text{or} \quad \longleftarrow (A^T + p_i I_n)^{-1} X.$$

2.2 The concept of user-supplied functions

All operations with the structured matrix A are realized by user supplied functions. Moreover, all data related to the matrix A is stored in "hidden" global variables for the sake of efficiency. One distinct merit of using global variables for storing large quantities of data is that MATLAB codes become considerably faster compared to the standard concept, where such variables are transfered as input or output arguments from one routine to another over and over again. The purpose of user supplied functions is to generate these "hidden" data structures, to realize basic matrix operations listed in §2.1, and destroy "hidden" data structures once they are not needed anymore. Moreover, pre- and postprocessing of the dynamical system (1) can be realized by user supplied functions. At first glance, the use of user supplied functions might seem a bit cumbersome compared to the explicit access to the matrix A, but this concept turns out to be a good means to attain a high degree of flexibility and efficiency. The two main advantages of this concept are the following:

- Adequate structures for storing the data, which corresponds to the matrix A, can be used. (In other words, one is not restricted to storing A explicitely in a sparse or dense array.)

- Adequate methods for solving linear systems can be used. (This means that one is not restricted to using "standard" LU factorizations. Instead, Cholesky factorizations, Krylov subspace methods, or even multi-grid methods can be used.)

In general, users have to implement user supplied functions themselves in a way that is as highly efficient w.r.t. both computation and memory demand. However, user supplied functions for the following most common types of matrices A (and ways to implement the corresponding basic matrix operations) are already contained in LYAPACK. Note that the *basis name*, which must be provided as input parameter `name` to many LYAPACK routines, is the first part of the name of the corresponding user supplied function.

- [basis name] = `as`: A in (1) is sparse and symmetric. (Shifted) linear systems are solved by sparse Cholesky factorization. In this case, the ADI shift parameters p_i must be real. **Note:** This is not guaranteed in the routine `lp_lrnm` for Riccati equations and optimal control problems. If this routine is used, the unsymmetric version `au` must be applied instead of `as`.

- [basis name] = `au`: A in (1) is sparse and (possibly) unsymmetric. (Shifted) linear systems are solved by sparse LU factorization.

- [basis name] = `au_qmr_ilu`: A in (1) is sparse and (possibly) unsymmetric. (Shifted) linear systems are solved iteratively by QMR using ILU preconditioning, [14].

- [basis name] = `msns`: Here, the system arises from a generalized system (9), where M and N are symmetric. Linear systems involved in all three types of basic matrix operations are solved by sparse Cholesky factorizations. In this case, the ADI shift parameters p_i must be real. **Note:** This is not guaranteed in the routine `lp_lrnm` for Riccati equations and optimal control problems. If this routine is used, the unsymmetric version `munu` must be applied instead of `msns`.

- [basis name] = `munu`: Here, the system arises from a generalized system (9), where M and N are sparse and possibly unsymmetric. Linear systems involved in all three types of basic matrix operations are solved by sparse LU factorizations.

Although, these classes of user supplied functions can be applied to a great variety of problems, users might want to write their user supplied functions themselves (or modify the user supplied functions contained in LYAPACK). For example, this might be the case if A is a dense Toeplitz or circulant matrix, or if alternative iterative solvers or preconditioners should be applied to solve linear systems. Obviously, it is impossible to provide user supplied functions in LYAPACK for all possible structures the matrix A can have.

For each type of problems listed above the following routines are needed. Here, one or two extensions are added to the basis name:

[basis name]_[extension 1] or [basis name]_[extension 1]_[extension 2]

Five different first extensions are possible. They have the following meaning:

- [extension 1] = `m`: matrix **m**ultiplication; see §2.1.
- [extension 1] = `l`: solution of systems of **l**inear equations; see §2.1.
- [extension 1] = `s`: solution of **s**hifted systems of linear equations; see §2.1.
- [extension 1] = `pre`: **pre**processing.
- [extension 1] = `pst`: **p**o**st**processing.

For some classes of user supplied functions preprocessing and postprocessing routines do not exist because they are not needed. There is no second extension if [extension 1] = `pre` or `pst`. If [extension 1] = `m`, `l`, or `s`, there are the following three possibilities w.r.t. the second extension:

- [extension 2] = `i`: **i**nitialization of the data needed for the corresponding basic matrix operations.
- no [extension 2]: the routine actually performs the basic matrix operations.
- [extension 2] = `d`: **d**estruction of the global data generated by the corresponding initialization routine ([extension 2] = `i`).

This concept is somewhat similar to that of constructors and destructors in object-oriented programming. Note that user supplied functions with [extension 1] = `pre` or `pst` will be called only in the main program (i.e., the program written by the user). user supplied functions with [extension 2] = `i` and `d` will be often (but not always) called in the main program. In contrast, the remaining three types of user supplied functions ([extension 1] = `m`, `l`, or `s`) will be used internally in LYAPACK main routines.

For example, `au_m_i` initializes the data for matrix multiplications with the unsymmetric matrix A in a global variable, `au_m_i` performs such multiplications, whereas `au_m_d` destroys the global data generated by `au_m_i` to save memory.

For more details and examples see §C.1.

Note: In the user supplied functions, that are contained in LYAPACK, the data for realizing basic matrix operations is stored in **fixed** global variables. This means that it is impossible to store data for more than one problem (in other words for more than one matrix A) at the same time. If, for example, several model reduction problems (with different matrices A should be solved, then these problems have to be treated one after another. The user supplied functions for initialization ([extension 2] = `i`) overwrite the data, that has been written to global variables in prior calls of these user supplied functions.

2.3 Preprocessing and postprocessing

In most cases, it is recommended or even necessary to perform a preprocessing step before initializing or generating global data structures by the routines [basis name]_{m, l, s}_i and before using LYAPACK main routines (see §B.1). Such preprocessing steps are implemented in the routines [basis name]_pre. There are no well-defined rules what has to be done in the preprocessing step, but in general this step consists of a transformation of the input data (for example, F and G for solving the Lyapunov equation (2), or A, B, and C for the model reduction problem, etc.), such that the transformed input data has an improved structure from the numerical point of view. For example, if a standard system (1) with a sparse matrix A is considered, then the preprocessing done by as_pre or au_pre is a reordering of the nonzero pattern of A for bandwidth reduction. If the problem is given in form of a generalized system (9) with sparse matrices M and N, then the preprocessing in msns_pre or munu_pre is done in two steps. First, the columns and rows of both matrices are reordered (using the same permutation). Second, the transformation (10) into a standard system is performed.

Although LYAPACK routines could often be applied to the original data, reordering of sparse matrices is most cases crucial to achieve a high efficiency, when sparse LU or Cholesky factorizations are computed in MATLAB. Figure 1 shows the nonzero pattern of the matrix M (which equals to that of N) for a system (9) arising from a finite element discretization of a two-dimensional partial differential equation.

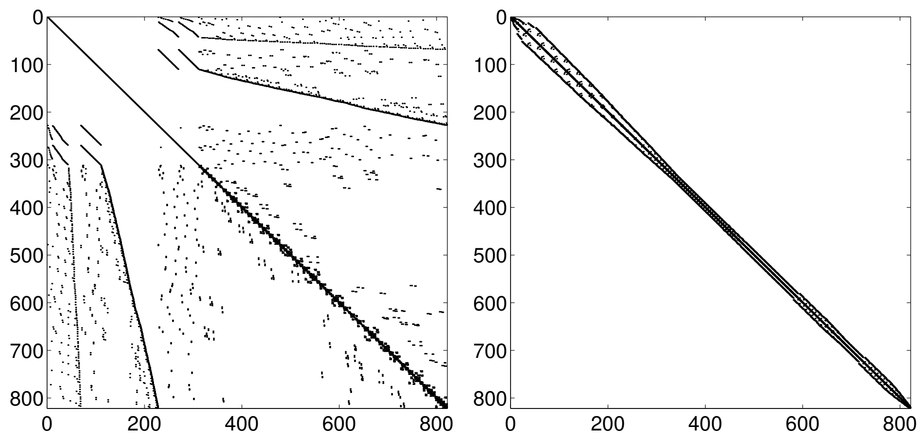

Figure 1: Nonzero pattern before (left) and after (right) reordering.

There are a few situations, when preprocessing is not necessary. Examples are standard systems (1), where A is a tridiagonal matrix and (shifted) linear systems are solved directly (Here, reordering would be superfluous.), or where A is sparse and (shifted) linear systems are solved by QMR [14].

Usually, the preprocessing step consists of an equivalence transformation of the system. In rare cases not only the system matrices, but also further matrices must be transformed. In particular, this applies to **nonzero** initial stabilizing state-feedback matrices K_0 when a Riccati equation or an optimal control problems should be solved.

It is important for users to understand, what is done during the preprocessing and to distinguish carefully between "original" and "transformed" (preprocessed) data. Often the output data of LYAPACK routines must be backtransformed (postprocessed) in order to obtain the solution of the original problem. Such data are, for example, the low rank Cholesky factor Z that describes the (approximate) solution of a Lyapunov equation or a

Riccati equation, or the (approximate) state-feedback K for solving the optimal control problems. For instance, if `as_pre` or `au_pre` have been applied for preprocessing, then the rows of Z or K must be reordered by the inverse permutation. If `msns_pre` or `munu_pre` are used, these quantities must be transformed with the inverse of the Cholesky factor M_U and subsequently re-reordered. These backtransformations are implemented in the corresponding user supplied functions [basis name]`_pst` for postprocessing.

In some cases, postprocessing can be omitted, despite preprocessing has been done. This is the case, when the output data does not depend on what has been done as preprocessing (which is usually an equivalence transformation of the system). An example is model reduction by LRSRM or DSPMR. Here, the reduced systems are invariant w.r.t. equivalence transformations of the original system.

2.4 Organization of user-supplied functions for basic matrix operations and guidelines for their implementation

In the first part of this section we explain how user supplied functions are organized and how they work. We take a standard system (1), where A is sparse, and the corresponding user supplied functions `au_*` as an illustrative example. The order in which these user supplied functions are invoked is important. A typical sequence is shown below. Note that this is a scheme displaying the chronological order rather than a "main program". For example, Steps 6–13 could be executed inside the routine `lp_lrnm` for the Newton iteration.

```
...
[A0,B0,C0,prm,iprm] = au_pre(A,B,C);    % Step 1
au_m_i(A0);                             % Step 2
Y0 = au_m('N',X0);                      % Step 3
...
au_l_i;                                 % Step 4
Y0 = au_l('N',X0);                      % Step 5
...
p = lp_para(...);                       % Step 6
au_s_i(p);                              % Step 7
...
Y0 = au_s('N',X0,i);                    % Step 8
...
au_s_d(p);                              % Step 9
...
p = lp_para(...);                       % Step 10
...
au_s_i(p);                              % Step 11
...
Y0 = au_s('N',X0,i);                    % Step 12
...
au_s_d(p);                              % Step 13
...
Z = au_pst(Z0,iprm);                    % Step 14
au_l_d;                                 % Step 15
au_m_d;                                 % Step 16
...
```

Note, in particular, that the user supplied functions `au_m` (multiplication), `au_l` (solution of linear systems), and `au_s` (solution of shifted linear systems) can be called anywhere between the following steps:

`au_m`:	between Steps 2 and 16,
`au_l`:	between Steps 4 and 15,
`au_s`:	between Steps 7 and 9, Steps 11 and 13, etc.

Next, we describe what is done in the single steps.

Step 1: Preprocessing, which has been discussed in §2.3. The system matrices A, B, and C are transformed into A_0, B_0 and C_0 (by a simultaneous reordering of columns and rows).

Step 2: Initialization of data for multiplications with A_0. Here, the input parameter `A0` is stored in the "hidden" global variable `LP_A`.

Step 3: Matrix multiplication with A_0. `au_m` has access to the global variable `LP_A`.

Step 4: Initialization of data for the solution of linear systems with A_0. Here, an LU factorization of the matrix A_0 (provided as `LP_A`) is computed and stored in the global variables `LP_L` and `LP_U`.

Step 5: Solution of linear system $A_0 Y_0 = X_0$. `au_l` has access to the global variables `LP_L` and `LP_U`.

Step 6: Compute shift parameters $\{p_1, \ldots, p_l\}$.

Step 7: Initialization of data for the solution of shifted linear systems with A_0. Here, the LU factors of the matrices $A_0 + p_1 I$, …, $A_0 + p_l I$ (A_0 is provided in `LP_A`) are computed and stored in the global variables `LP_L1`, `LP_U1`, …, `LP_L`l, `LP_U`l.

Step 8: Solution of shifted linear system $(A_0 + p_i I)Y_0 = X_0$. `au_s` has access to the global variables `LP_L`i and `LP_U`i.

Step 9: Delete the global variables `LP_L1`, `LP_U1`, …, `LP_L`l, `LP_U`l.

Step 10: Possibly, a new set of shift parameters is computed, which is used for a further run of the LRCF-ADI iteration. (This is the case within the routine `lp_lrnm`, but typically not for model reduction problems.)

Step 11: (Re)initialization of data for the solution of shifted linear systems with A_0 and the new shift parameters. Again, the LU factors are stored in the global variables `LP_L1`, `LP_U1`, …, `LP_L`l, `LP_U`l. Here, the value of l may differ from that in Step 7.

Step 12: Solve shifted linear system.

Step 13: Delete the data generated in Step 11, i.e., clear the global variables `LP_L1`, `LP_U1`, …, `LP_L`l, `LP_U`l. (Steps 9–13 can be repeated several times.)

Step 14: Postprocessing, which has been discussed in §2.3. The result Z_0 of the preprocessed problem is backtransformed into Z.

Step 15: Delete the data generated in Step 4, i.e., clear the global variables `LP_L` and `LP_U`.

Step 16: Delete the data generated in Step 2, i.e., clear the global variable `LP_A`.

The other user supplied functions, which are contained in LYAPACK, are organized in a similar way. Consult the corresponding m-files for details.

The following table shows which user supplied functions are invoked within the single LYAPACK main routines. [b.n.] means [basis name].

main routine	**invoked USFs**
`lp_para`	[b.n.]`_m`, [b.n.]`_l`.
`lp_lradi`	[b.n.]`_m`, [b.n.]`_s`.
`lp_lrsrm`	[b.n.]`_m`.
`lp_dspmr`	[b.n.]`_m`.
`lp_lrnm`	[b.n.]`_m`, [b.n.]`_l`, [b.n.]`_s_i`, [b.n.]`_s`, [b.n.]`_s_d`.

The calling sequences for these user supplied functions are fixed. It is mandatory to stick to these sequences when implementing new user supplied functions. The calling sequences are shown below. There it is assumed that X_0 (parameter `X0`) is a complex $n \times t$ matrix, `p` is a vector containing shift parameters, and the flag `tr` is either `'N'` ("not transposed") or `'T'` ("transposed").

- [**basis name**]`_m`. Calling sequences: `Y0 =` [b.n.]`_m(tr,X0)` or `n =` [b.n.]`_m`. In the first case, the result is $Y_0 = A_0 X_0$ for `tr = 'N'` and $Y_0 = A_0^T X_0$ for `tr = 'T'`. The parameter `tr` must also be provided (and is ignored) if A_0 is symmetric. In the second case, where the user supplied function is called without input parameters, only the problem dimension n is returned.

- [**basis name**]`_l`. Calling sequence: `Y0 =` [b.n.]`_l(tr,X0)`. The result is $Y_0 = A_0^{-1} X_0$ for `tr = 'N'` and $Y_0 = A_0^{-T} X_0$ for `tr = 'T'`.

- [**basis name**]`_i_p`. Calling sequence: [b.n.]`_s_i(p)`.

- [**basis name**]`_s`. Calling sequence: `Y0 =` [b.n.]`_s(tr,X0,i)`. The result is $Y_0 = (A_0 + p_i I)^{-1} X_0$ for `tr = 'N'` and $Y_0 = (A_0^T + p_i I)^{-1} X_0$ for `tr = 'T'`.

- [**basis name**]`_s_d`. Calling sequence: [b.n.]`_s_d(p)`.

2.5 Case studies

See §C.1.

3 Lyapunov equations

3.1 Low Rank Cholesky Factor ADI

3.1.1 Theory and algorithm

This section gives a brief introduction to the solution technique for continuous time Lyapunov equations used in LYAPACK. For more details, the reader is referred to[31, 6, 33, 37].

We consider the continuous time Lyapunov equation

$$FX + XF^T = -GG^T, \qquad (11)$$

where $F \in \mathbb{R}^{n,n}$ is stable, $G \in \mathbb{R}^{n,t}$ and $t \ll n$. It is well-known that such continuous time Lyapunov equations have a unique solution X, which is symmetric and positive semidefinite. Moreover, in many cases, the eigenvalues of X decay very fast, which is discussed for symmetric matrices F in [40]. Thus, there exist often very accurate approximations of a rank, that is much smaller than n. This property is most important for the efficiency of LYAPACK.

The ADI iteration [36, 50] for the Lyapunov equation (11) is given by $X_0 = 0$ and

$$\begin{aligned} (F + p_i I_n) X_{i-1/2} &= -GG^T - X_{i-1}(F^T - p_i I_n) \\ (F + \bar{p}_i I_n) X_i{}^T &= -GG^T - X_{i-1/2}{}^T (F^T - \bar{p}_i I_n), \end{aligned} \tag{12}$$

for $i = 1, 2, \ldots$ It is one of the most popular iterative techniques for solving Lyapunov equations. This method generates a sequence of matrices X_i which often converges very fast towards the solution, provided that the ADI shift parameters p_i are chosen (sub)optimally. The basic idea for a more efficient implementation of the ADI method is to replace the ADI iterates by their Cholesky factors, i.e., $X_i = Z_i Z_i^H$ and to reformulate in terms of the factors Z_i. Generally, these factors have ti columns. For this reason, we call them *low rank Cholesky factors* (LRCFs) and their products, which are equal to the ADI iterates, *low rank Cholesky factor products* (LRCFPs). Obviously, the low rank Cholesky factors Z_i are not uniquely determined. Different ways to generate them exist; see [31, 37]. The following algorithm, which we refer to as *Low Rank Cholesky Factor ADI* (LRCF-ADI), is the most efficient of these ways. It is a slight modification of the iteration proposed in [31]. Note that the number of iteration steps i_{max} needs not be fixed a priori. Instead, several stopping criteria, which are described in §3.1.2 can be applied.

Algorithm 1 (Low rank Cholesky factor ADI iteration (LRCF-ADI))

INPUT: F, G, $\{p_1, p_2, \ldots, p_{i_{max}}\}$

OUTPUT: $Z = Z_{i_{max}} \in \mathbb{C}^{n, t i_{max}}$, such that $ZZ^H \approx X$.

1. $V_1 = \sqrt{-2 \operatorname{Re} p_1} (F + p_1 I_n)^{-1} G$
2. $Z_1 = V_1$

FOR $i = 2, 3, \ldots, i_{max}$

3. $V_i = \sqrt{\operatorname{Re} p_i / \operatorname{Re} p_{i-1}} \left(V_{i-1} - (p_i + \bar{p}_{i-1})(F + p_i I_n)^{-1} V_{i-1} \right)$
4. $Z_i = \left[\begin{array}{cc} Z_{i-1} & V_i \end{array} \right]$

END

Let $\mathcal{P}_j$ be either a negative real number or a pair of complex conjugate numbers with negative real part and nonzero imaginary part. We call a parameter set of type $\{p_1, p_2, \ldots, p_i\} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_j\}$ a *proper* parameter set. The LYAPACK implementation of LRCF-ADI requires proper parameter sets $\{p_1, p_2, \ldots, p_{i_{max}}\}$. If $X_i = Z_i Z_i^H$ is generated by a proper parameter set $\{p_1, \ldots, p_i\}$, then X_i is real, which follows from (12). However, if there are nonreal parameters in this subsequence, Z_i is not real. A more complicated modification of Algorithm 1 for generating real LRCFs has been proposed in [6]. However, in the LYAPACK implementation of Algorithm 1, real LRCFs can be derived from the complex factors computed by this algorithm (at the price of additional computation). That means for the delivered complex low rank Cholesky factor $Z = Z_{i_{max}}$ a real low rank Cholesky factor $\tilde{Z}$ is computed in a certain way, such that $ZZ^H = \tilde{Z}\tilde{Z}^T$. The low rank Cholesky factor $\tilde{Z}$ is returned as output parameter of the corresponding routine `lp_lradi`.

3.1.2 Stopping criteria

The LYAPACK implementation of the LRCF-ADI iteration in the routine `lp_lradi` offers the following stopping criteria:

- maximal number of iteration steps;
- tolerance for the *normalized residual norm* (NRN);
- stagnation of the normalized residual norm (most likely caused by round-off errors);
- smallness of the values $\|V_i\|_F$.

Here, the normalized residual norm corresponding to the low rank Cholesky factor Z is defined as

$$\mathrm{NRN}(Z) = \frac{\|FZZ^T + ZZ^TF^T + GG^T\|_F}{\|GG^T\|_F}. \tag{13}$$

Note: In LYAPACK, a quite efficient method for the computation of this quantity is applied. See [37] for details. However, the computation of the values $\mathrm{NRN}(Z_i)$ in the course of the iteration can still be very expensive. Sometimes, this amount of computation can exceed the computational cost for the actual iteration itself! Besides this, computing the normalized residual norms can require a considerable amount of memory. This amount is about proportional to ti. For this reason, it can be preferable to avoid stopping criteria based on the normalized residual norm (tolerance for the NRN, stagnation of the NRN) and to use cheaper, possibly heuristical criteria instead.

In the sequel, we discuss the above stopping criteria and show some sample convergence histories (in terms of the normalized residual norm) for LRCF-ADI runs. Here, this method is applied to a given test example, but the iterations are stopped by different stopping criteria and different values of the corresponding stopping parameters. It should be noted that the convergence history plotted in Figures 2–5 is quite typical for LRCF-ADI provided that shift parameters generated by `lp_para` are used in the given order. In the first stage of the iteration, the logarithm of the normalized residual norm decreases about linearly. Typically, this slope becomes less steep, when more "ill-conditioned" problems are considered. (Such problems are in particular Lyapunov equations, where many eigenvalues of F are located near the imaginary axis, but far away from the real one. In contrast, symmetric problems, where the condition number of F is quite large, can usually be solved by LYAPACK within a reasonable number of iteration steps.) In the second stage, the normalized residual norm curve nearly stagnates on a relatively small level (mostly, between 10^{-12} and 10^{-15}), which is caused by round-off errors. That means the accuracy (in terms of the NRN) of the low rank cholesky factor product $Z_i Z_i^H$ cannot be improved after a certain number of steps. Note, however, that the stagnation of the error norm $\|Z_i Z_i^H - X\|_F$ can occur a number of iteration steps later. Unfortunately, the error cannot be measured in practice because the exact solution X is unknown.

Each of the four stopping criteria can be "activated" or "avoided" by the choice of the corresponding input argument (stopping parameter) of the routine `lp_lradi`. If more than one criterion is activated, the LRCF-ADI iteration is stopped as soon as (at least) one of the "activated" criteria is fulfilled.

- **Stopping criterion: maximal number of iteration steps.** This criterion is represented by the input parameter `max_it` in the routine `lp_lradi`. The iteration is stopped by this criterion after `max_it` iterations steps. This criterion can be avoided by setting `max_it = +Inf` (i.e., `max_it` $= \infty$). Obviously, no additional computations need to be performed to evaluate it. The drawback of this stopping criterion is, that it is not related to the attainable accuracy of the delivered low rank cholesky factor product ZZ^H. This is illustrated by Figure 2.

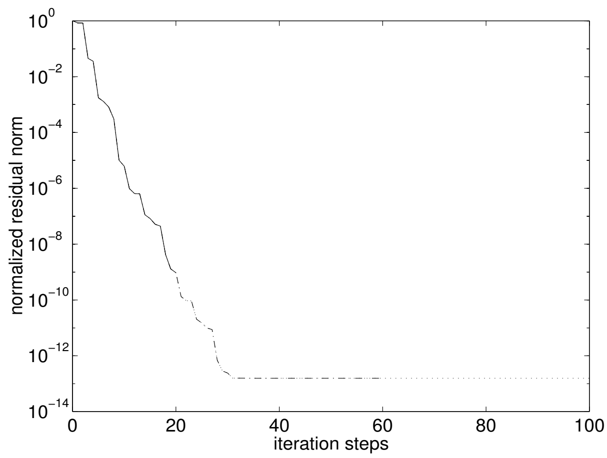

Figure 2: Stopping criterion: maximal number of iteration steps. Solid line: `max_it` $= 20$; dashdotted line: `max_it` $= 60$; dotted line: `max_it` $= 100$. The other three criteria are avoided.

- **Stopping criterion: tolerance for the normalized residual norms.** This criterion is represented by the input parameter `min_res` in the routine `lp_lradi`. The iteration is stopped by this criterion as soon as

$$\mathrm{NRN}(Z_i) \leq \texttt{min\_res}.$$

  This criterion can be avoided by setting `min_res` $= 0$. (Because of round-off errors it is practically impossible to attain $\mathrm{NRN}(Z_i) = 0$.) It requires the computation of normalized residual norms and is computationally expensive. A further drawback of this criterion is that it will either stop the iteration before the maximal accuracy is attained (see `min_res` $= 10^{-5}$, 10^{-10} in Figure 3) or it will not stop the iteration at all (see `min_res` $= 10^{-15}$ in Figure 3). If one wants to avoid this criterion, but compute the convergence history provided by the output vector `res`, one should set `min_res` to a value much smaller than the machine precision (say, `min_res` $= 10^{-100}$).

- **Stopping criterion: stagnation of the normalized residual norm.** This criterion is represented by the input parameter `with_rs` in the routine `lp_lradi`. It is activated if `with_rs` = `'S'` and avoided if `with_rs` = `'N'`. The iteration is stopped by this criterion when a stagnation of the normalized residual norm curve is detected. We do not discuss the implementation of this criterion in detail here, but, roughly speaking, the normalized residual norm curve is considered as "stagnating", when no noticeable decrease of the normalized residual norms is observed in 10 consecutive iteration steps. In extreme cases, where the shape of the normalized residual norm curve is not so clearly subdivided in the linearly decreasing and the stagnating part as in Figure 4, this criterion might terminate the iteration prematurely. However, it works well in practice. It requires the computation of normalized residual norms and is computationally expensive. Note that the delay between stagnation and stopping of the curve is 10 iteration steps.

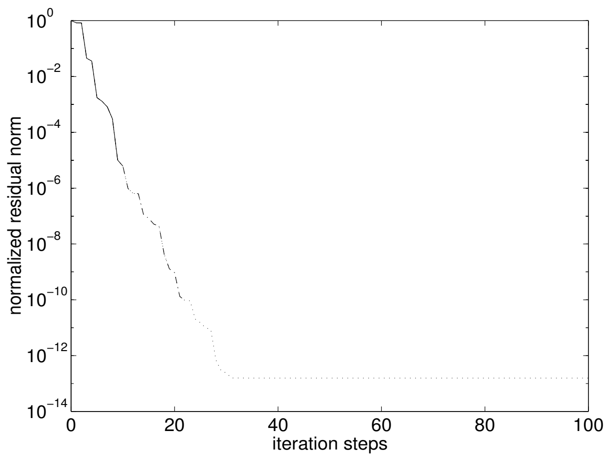

Figure 3: Stopping criterion: tolerance for the normalized residual norm. Solid line: `min_res` $= 10^{-5}$; dashdotted line: `min_res` $= 10^{-10}$; dotted line: `min_res` $= 10^{-15}$. The other three criteria are avoided.

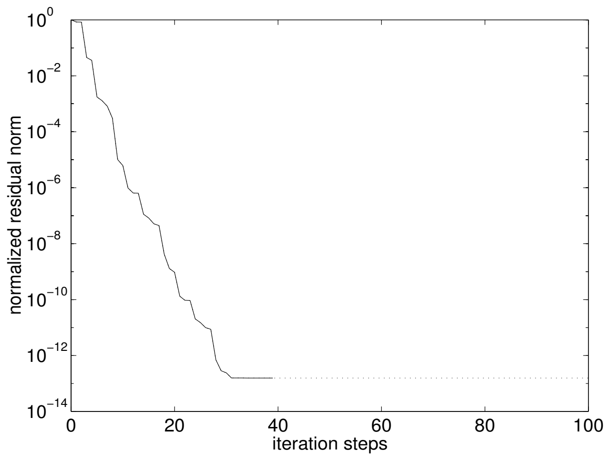

Figure 4: Stopping criterion: Stagnation of the normalized residual norm. Solid line: `with_rs` = `'S'`; dotted line: `with_rs` = `'N'`. The other three criteria are avoided.

- **Stopping criterion: smallness of the values $\|V_i\|_F$.** This criterion is represented by the input parameter `min_in` in the routine `lp_lradi`. It is based on the observation that the values $\|V_i\|_F$ tend to decrease very fast. Note in this context that $V_i V_i^H$ is the difference between the ADI iterates X_i and X_{i-1}, and that the sequence of the matrices X_i is monotonically converging (i.e., $X_i \geq X_{i-1}$). Loosely speaking, this means the following. When $\|V_i\|_F^2$ and consequently $\|V_i V_i^H\|_F \approx \|V_i\|_F^2$ become nearly as small as the machine precision, then the "contribution" from iteration step $i-1$ to i is almost completely corrupted by round-off errors and, thus, there is no point in continuing the iteration. However, since $\|V_i\|_F$ is not monotonically decreasing, it is required in `lp_lradi` that

$$\frac{\|V_i\|_F^2}{\|Z_i\|_F^2} \leq \texttt{min\_in}$$

is fulfilled in 10 consecutive iteration steps before the iteration is stopped to keep the risk of a premature termination very small. The evaluation of this criterion is inexpensive (see also [6]) compared to both criteria based on the normalized residual norms. Moreover, it is less "static" as the criterion based on the number of iteration steps. Unfortunately, it is not clear how the accuracy of the approximate solution $Z_i Z_i^T$ is related to the ratio of $\|V_i\|_F$ and $\|Z_i\|_F$. Thus, the criterion is not absolutely safe. However, if the Lyapunov equation should be solved as accurate as possible, good results are usually achieved for values of `min_in`, that are slightly larger than the machine precision (say, `min_in` $= 10^{-12}$). The criterion can be avoided by setting `min_in` $= 0$. See Figure 5.

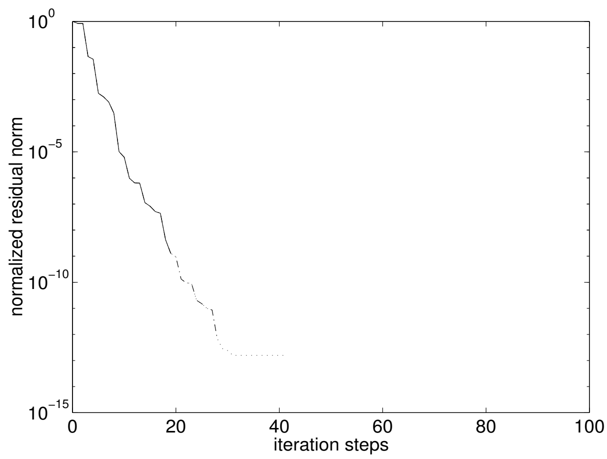

Figure 5: Stopping criterion: smallness of the values $\|V_i\|_F$. Solid line: `min_in` $= 10^{-5}$; dashdotted line: `min_in` $= 10^{-10}$; dotted line: `min_in` $= 10^{-15}$. The other three criteria are avoided.

We recommend to use (only) the stopping criterion related to `with_rs` if Lyapunov solutions of high accuracy should be computed and if it is affordable to compute residual norms. If the computation of the residual norms must be avoided, the criterion related to `min_in` is probably the best choice.

3.1.3 The routine `lp_lradi`

The LRCF-ADI iteration is implemented in the LYAPACK routine `lp_lradi`. Here, we provide a brief description of this routine. For more details see the inline documentation which is displayed by typing the MATLAB command `>> help lp_lradi`.

The routine can solve either the continuous time Lyapunov equation

$$FX + XF^T = -GG^T \tag{14}$$

or to the "dual" continuous time Lyapunov equation

$$F^T X + XF = -G^T G. \tag{15}$$

Here, $F = A - B_f K_f^T$. Basic matrix operations must be supplied by user-supplied functions for the matrix A (not F!). G in (14) or G^T in (15) should contain very few columns compared to the system order. B_f and K_f are matrices, that represent a so-called state feedback. They are needed in the routine `lp_lrnm` for the Newton method, in which `lp_lradi` is invoked. In general, users will not use this option, which means $B_f = K_f = 0$. However, if this is not the case, the matrices B_f and K_f must contain very few columns to guarantee the efficiency of the routine.

The approximate solution of either Lyapunov equation is given by the low rank Cholesky factor Z, for which $ZZ^H \approx X$. Z has typically fewer columns than rows. (Otherwise, this routine and LYAPACK itself are useless!) In general, Z can be a complex matrix, but the product ZZ^H is real. `lp_lradi` can perform an optional internal post-processing step, which guarantees that the delivered low rank cholesky factor Z is real. More precisely, the complex low rank Cholesky factor delivered by the LRCF-ADI iteration is transformed into a real low rank Cholesky factor of the same size, such that both low rank Cholesky factor products are identical. However, doing this requires additional computation. (This option is not related to the "real version" of LRCF-ADI described in [6].)

Furthermore, there exists an option for directly generating the product of the (approximate) solution with a matrix, i.e., $K_{out} = ZZ^H K_{in}$ is computed without forming the low rank Cholesky factor Z. Here, K_{in} must contain only few columns. However, this option should not be used by the user. It is needed in the implicit version of the Newton method. If this mode is used, stopping criteria based on the residual cannot be applied.

Calling sequences:

Depending on the choice of the mode parameter `zk`, the following two calling sequences exist. However, **it is recommended to use only the first mode.**

- `zk` = `'Z'`:

```
[Z, flag, res, flp] = lp_lradi( tp, zk, rc, name, Bf, Kf, G, p, ...
max_it, min_res, with_rs, min_in, info)
```

- `zk` = `'K'`:

```
[K_out, flag, flp] = lp_lradi( tp, zk, rc, name, Bf, Kf, G, p, ...
K_in, max_it, min_in, info)
```

Input parameters:

`tp`: Mode parameter, which is either `'B'` or `'C'`. If `tp = 'B'`, CALE (14) is solved. Otherwise, CALE (15) is solved.

`zk`: Mode parameter, which is either `'Z'` or `'K'`. If `zk = 'Z'`, the low rank Cholesky factor Z is computed. Otherwise, $K_{out} = ZZ^H K_{in}$ is computed directly.

`rc`: Mode parameter, which is either `'R'` or `'C'`. If `rc = 'C'`, the routine delivers a low rank Cholesky factor, which is not real when non-real shift parameters are used. Otherwise, the low rank Cholesky factor resulting from the LRCF-ADI iteration is transformed into a real low rank Cholesky factor $\tilde{Z}$, which describes the identical approximate solution $\tilde{Z}\tilde{Z}^T$. $\tilde{Z}$ is returned instead of Z.

`name`: The basis name of the USFs that realize BMOs with A.

`Bf`: Feedback matrix B_f, which is not used explicitely in general. For $B_f = 0$, set `Bf = []`.

Kf: Feedback matrix K_f, which is not used explicitely in general. For $K_f = 0$, set Kf = [].

G: The matrix G.

p: Vector containing the suitably ordered ADI shift parameters $\mathcal{P} = \{p_1, \ldots, p_l\}$, which are delivered by the routine lp_para. If the number l of distinct parameters is smaller than i_{max} in Algorithm 1, shift parameters are used cyclically. That means, $p_{l+1} = p_1$, $p_{l+2} = p_2$, …, $p_{2l} = p_l$, $p_{2l+1} = p_1$, …

K_in: The matrix K_{in}, which is only used in the mode zk = 'K'.

max_it: Stopping parameter. See §3.1.2.

min_res: Stopping parameter. See §3.1.2.

with_rs: Stopping parameter. See §3.1.2.

min_in: Stopping parameter. See §3.1.2.

info: Parameter, which determines the "amount" of information that is provided as text and/or residual history plot. The following values are possible: info = 0 (no information), 1, 2, and 3 (most possible information)

Output parameters:

Z: The low rank Cholesky factor Z, which is complex if rc = 'C' and p is not a real vector.

K_out: The matrix K_{out}, which is only returned in the mode zk = 'K'.

flag: A flag, that shows by which stopping criterion (or stopping parameter) the iteration has been stopped. Possible values are 'I' (for max_it), 'R' (for min_res), 'S' (for with_rs), and 'N' (for min_in).

res: A vector containing the history of the normalized residual norms. res(1) = 1 and res($i+1$) is the normalized residual norm w.r.t. the iteration step i. If the stopping criteria are chosen, so that the normalized residual norms need not be computed, res = [] is returned.

flp: A vector containing the history of the flops needed for the iteration. flp(1) = 0 and flp($i+1$) is the number of flops required for the iteration steps 1 to i. flp displays only the number of flops required for the actual iteration. The numerical costs for initializing and generating data by USFs, the computation of ADI shift parameters, and the computation of normalized residual norms are not included.

3.2 Computation of ADI shift parameters

3.2.1 Theory and algorithm

In this section, we briefly describe a practical algorithm to compute a set $\mathcal{P} = \{p_1, \ldots, p_l\}$ of suboptimal shift parameters, which are needed in the LRCF-ADI iteration. This algorithm [37] is implemented in the routine lp_para, whose output is an ordered set of l distinct shift parameters.

The determination of (sub)optimal ADI shift parameters is closely connected with a rational minimax problem (e.g., [46, 49, 51]) related to the function

$$s_{\mathcal{P}}(t) = \frac{|(t-p_1)\cdot\ldots\cdot(t-p_l)|}{|(t+p_1)\cdot\ldots\cdot(t+p_l)|}.$$

This minimax problem can be stated as the choice of $\mathcal{P}$, such that

$$\max_{t\in\sigma(F)} s_{\mathcal{P}}(t)$$

is minimized. Unfortunately, the spectrum $\sigma(F)$ is not known in general and it cannot be computed inexpensively if F is very large. Furthermore, even if the spectrum or bounds for the spectrum are known, no algorithms are available to compute the optimal parameters p_i.

Our algorithm for the computation of a set of **sub**optimal shift parameters is numerically inexpensive and heuristic. It is based on two ideas. First, we generate a discrete set, which "approximates" the spectrum. This is done by a pair of Arnoldi processes; e.g., [19]. The first process w.r.t. F delivers k_+ values that tend to approximate "outer" eigenvalues, which are generally not close to the origin, well. The second process w.r.t. F^{-1} is used to get k_- approximations of eigenvalues near the origin, whose consideration in the ADI minimax problem is crucial. The eigenvalue approximations delivered by the Arnoldi processes are called *Ritz values*. Second, we choose a set of shift parameters, which is a subset of the set of Ritz values $\mathcal{R}$. This is done by a heuristic, that delivers a suboptimal solution for the resulting discrete optimization problem. Note that the order in which this heuristic delivers the parameters is advantageous. Loosely speaking, the parameters are ordered such that parameters, which are related to a strong reduction in the ADI error, are applied first. For more details about the parameter algorithm, see [37].

Algorithm 2 (Suboptimal ADI parameters)

INPUT: F, l_0, k_+, k_-

OUTPUT: $\mathcal{P} = \{p_1,\ldots,p_l\}$, where $l = l_0$ or l_0+1

1. Choose $b_0 \in \mathbb{R}^n$ at random.

2. Perform k_+ steps of the Arnoldi process w.r.t. (F, b_0) and compute the set of Ritz values $\mathcal{R}_+$.

3. Perform k_- steps of the Arnoldi process w.r.t. (F^{-1}, b_0) and compute the set of Ritz values $\mathcal{R}_-$.

4. $\mathcal{R} = \{\rho_1,\ldots,\rho_{k_++k_-}\} := \mathcal{R}_+ \cup (1/\mathcal{R}_-)$

5. IF $\mathcal{R} \not\subset \mathbb{C}_-$, remove unstable elements from $\mathcal{R}$ and display a warning.

6. Detect i with $\max_{t\in\mathcal{R}} s_{\{\rho_i\}}(t) = \min_{\rho\in\mathcal{R}} \max_{t\in\mathcal{R}} s_{\{\rho\}}(t)$ and initialize
$$\mathcal{P} := \begin{cases} \{\rho_i\} & : \ \rho_i \text{ real} \\ \{\rho_i, \bar\rho_i\} & : \ \text{otherwise} \end{cases}.$$

WHILE $\mathrm{card}(\mathcal{P}) < l_0$

7. Detect i with $s_{\mathcal{P}}(\rho_i) = \max_{t\in\mathcal{R}} s_{\mathcal{P}}(t)$ and set
$$\mathcal{P} := \begin{cases} \mathcal{P}\cup\{\rho_i\} & : \ \rho_i \text{ real} \\ \mathcal{P}\cup\{\rho_i, \bar\rho_i\} & : \ \text{otherwise} \end{cases}.$$

END WHILE

Obviously, the output of this algorithm is a proper parameter set; see §3.1.1. The number of shift parameters is either l_0 or $l_0 + 1$. Larger values of k_+ and k_- lead to better approximations of the spectrum, but increase also the computational cost, because k_+ matrix-vector multiplications with F must be computed in the first Arnoldi algorithm and k_- systems of linear equations with F must be solved in the second one. A typical choice of the triple (l_0, k_+, k_-) is (20,50,25). For "tough" problems these values should be increased. For "easy" ones they can be decreased. Note that decreasing l_0 will reduce the memory demand if shifted SLEs are solved directly, because in this case the amount of the memory needed to store the matrix factors is proportional to l.

Steps 6 and 7 require that $\mathcal{R}$ is contained in $\mathbb{C}_-$. However, this can only be guaranteed if $F + F^T$ is negative definite and exact machine precision is used. If F is unstable, than LYAPACK cannot be applied anyway, because the ADI iteration diverges or, at least, stagnates. Experience shows that also in the case, when F is stable but $F + F^T$ is not definite, the Ritz values tend to be contained in the left half of the complex plane. If this is not the case, unstable Ritz values are removed in Step 5, which is more or less a not very elegant emergency measure. If LRCF-ADI run with the resulting parameters diverges despite this measure, the matrix F is most likely unstable. In connection with the LRCF-NM or LRCF-NM-I applied to ill-conditioned CAREs, this might be caused by round-off errors. There the so-called closed-loop matrix $A - B_f K_f^T$ can be proved to be stable (in exact arithmetics), but the closed-loop poles (i.e, the eigenvalues of $A - B_f K_f^T$) can be extremely sensitive to perturbations, so that stability is not guaranteed in practice.

Figure 6 shows the result of the parameter algorithm for a random example of order $n = 500$. The triple (l_0, k_+, k_-) is chosen as (20,50,25). 21 shift parameters were returned. The picture shows the eigenvalues of F, the set $\mathcal{R}$ of Ritz values, and the set $\mathcal{P}$ of shift parameters. Note that the majority of the shift parameters is close to the imaginary axis.

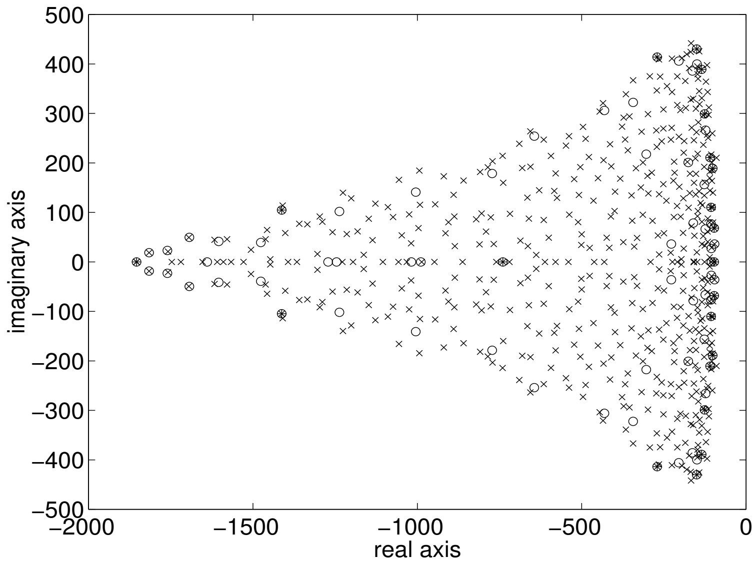

Figure 6: Results of Algorithm 2. $\times$: eigenvalues of F; $\bigcirc$: elements of $\mathcal{R}$; $*$: elements of $\mathcal{P} \subset \mathcal{R}$.

3.2.2 The routine lp_para

Calling sequences:

The following two calling sequences are possible:

```
[p,err_code,rw,Hp,Hm] = lp_para(name,Bf,Kf,l0,kp,km)

[p,err_code,rw,Hp,Hm] = lp_para(name,Bf,Kf,l0,kp,km,b0)
```

However, usually one is only interested in the first output parameter `p`.

Input parameters:

`name`: The basis name of the USFs that realize BMOs with A.

`Bf`: Feedback matrix B_f, which is not used explicitely in general. For $B_f = 0$, set `Bf = []`.

`Kf`: Feedback matrix K_f, which is not used explicitely in general. For $K_f = 0$, set `Kf = []`.

`l0`: Parameter l_0. Note that $k_+ + k_- > 2l_0$ is required.

`kp`: Parameter k_+.

`km`: Parameter k_-.

`b0`: This optional argument is an n-vector, that is used as starting vector in both Arnoldi processes. If `b0` is not provided, this vector is chosen at random, which means that different results can be returned by `lp_para` in two different runs with identical input parameters.

Output parameters:

`p`: A vector containing the ADI shift parameters $\mathcal{P} = \{p_1, \dots, p_l\}$, where either $l = l_0$ or $l = l_0 + 1$. It is recommended to apply the shift parameters in the same order in the routine `lp_lradi` as they are returned by this routine.

`err_code`: This parameter is an error flag, which is either `0` or `1`. If `err_code` $=$ `1`, the routine encountered Ritz values in the right half of the complex plane, which are removed in Step 5 of Algorithm 2. `err_code` $=$ `0` is the standard return value.

`rw`: A vector containing the Ritz value set $\mathcal{R}$.

`Hp`: The Hessenberg matrix produced by the Arnoldi process w.r.t. F.

`Hm`: The Hessenberg matrix produced by the Arnoldi process w.r.t. F^{-1}.

3.3 Case studies

See §C.2.1.

4 Model reduction

4.1 Preliminaries

Roughly speaking, model reduction is the approximation of the dynamical system

$$\begin{aligned} \dot{x}(\tau) &= Ax(\tau) + Bu(\tau) \\ y(\tau) &= Cx(\tau) \end{aligned} \tag{16}$$

with $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$, and $C \in \mathbb{R}^{q,n}$ by a *reduced system*

$$\begin{aligned} \dot{\hat{x}}(\tau) &= \hat{A}\hat{x}(\tau) + \hat{B}u(\tau) \\ y(\tau) &= \hat{C}\hat{x}(\tau) \end{aligned} \tag{17}$$

with $\hat{A} \in \mathbb{R}^{k,k}$, $\hat{B} \in \mathbb{R}^{k,m}$, $\hat{C} \in \mathbb{R}^{q,k}$ (or, possibly, $\hat{A} \in \mathbb{C}^{k,k}$, $\hat{B} \in \mathbb{C}^{k,m}$, $\hat{C} \in \mathbb{C}^{q,k}$), and $k < n$. In particular, we consider the case where the system order n is large, and m and q are much smaller than n. Furthermore, we assume that A is stable. Several ways exist to evaluate the approximation error between the original system and the reduced system. Frequently, the difference between the systems (16) and (17) measured in the L_∞ norm

$$\|G - \hat{G}\|_{L_\infty} = \sup_{\omega \in \mathbb{R}} \|G(\jmath\omega) - \hat{G}(\jmath\omega)\| \tag{18}$$

is used to do this, where $\jmath = \sqrt{-1}$ and $\|\cdot\|$ is the spectral norm of a matrix. Moreover, G and $\hat{G}$ are the *transfer functions* of the systems (16) and (17), which are defined as $G(s) = C(sI_n - A)^{-1}B$ and $\hat{G}(s) = \hat{C}(sI_k - \hat{A})^{-1}\hat{B}$.

LYAPACK contains implementations of two algorithms (LRSRM and DSPMR) for computing reduced systems. Both model reduction algorithms belong to the class of *state space projection methods*, where the reduced system is given as

$$\hat{A} = S_C^H A S_B, \quad \hat{B} = S_C^H B, \quad \hat{C} = C S_B. \tag{19}$$

Here, $S_B, S_C \in \mathbb{C}^{n,k}$ are certain projection matrices, which fulfill the biorthogonality condition

$$S_C^H S_B = I_k.$$

Furthermore, both model reduction algorithms rely on low rank approximations to the solutions (Gramians) of the continuous time Lyapunov equations

$$AX_B + X_B A^T = -BB^T \tag{20}$$

$$A^T X_C + X_C A = -C^T C. \tag{21}$$

This means that we assume that low rank Cholesky factors $Z_B \in \mathbb{C}^{n,r_B}$ and $Z_C \in \mathbb{C}^{n,r_C}$ with $r_B, r_C \ll n$ are available, such that $Z_B Z_B^H \approx X_B$ and $Z_C Z_C^H \approx X_C$. In LYAPACK these low rank Cholesky factors are computed by the LRCF-ADI iteration; see §3.1. In §§4.2 and 4.3 we will briefly describe the two model reduction algorithms LRSRM [39, 33] and DSPMR [32, 39]. In [39] a third method called LRSM (low rank Schur method) is proposed. However, this less efficient method is not implemented in LYAPACK. The distinct merit of LRSRM and DSPMR compared to standard model reduction algorithms, such as standard balanced truncation methods [47, 44, 48] or all-optimal Hankel norm approximation [18]), is their low numerical cost w.r.t. both memory and computation. On the other hand, unlike some standard methods, the algorithms implemented in LYAPACK do generally not guarantee the stability of the reduced system. If stability is crucial, this property must be checked numerically after running LRSRM or DSPMR. If the reduced system is not stable, several measures can be tried. For example, one can simply remove the unstable modes by modal truncation [11]. Another option is to run LRSRM or DSPMR again using more accurate low rank Cholesky factors Z_B and Z_C. Note that for some problems the error function $\|G(\jmath\omega) - \hat{G}(\jmath\omega)\|$ in ω, which characterizes the frequency response of the difference of both systems, can be evaluated by supplementary LYAPACK routines; see §6.

If the low rank Cholesky factors Z_B and Z_C delivered by the LRCF-ADI iteration are not real, then the reduced systems are not guaranteed to be real. This problem is discussed more detailed in [33] for the low rank square root method. If the reduced system needs to be real, it is recommended to check a posteriori whether the result of low rank square root method or dominat subspace projection model reduction is real. It is possible to transform a reduced complex system into a real one by a unitary equivalence transformation; see [33]. A much simpler way, of course, is using the option `rc = 'R'` for which the routine `lp_lradi` delivers real low rank Cholesky factors (at the price of a somewhat increased numerical cost).

4.2 Low rank square root method

4.2.1 Theory and algorithm

The *low rank square root method* (This algorithm is named *SLA* in [33].) (LRSRM) [39, 33] is only a slight modification of the classical square root method [47], which in turn is a numerically advantageous version of the balanced truncation technique [35]. The following algorithm is implemented in the LYAPACK routine `lp_lrsrm`:

Algorithm 3 (Low rank square root method (LRSRM))

INPUT: A, B, C, Z_B, Z_C, k

OUTPUT: $\hat{A}$, $\hat{B}$, $\hat{C}$

1. $U_C \Sigma U_B^H := Z_C^H Z_B$ ("thin" SVD with descending ordered singular values)
2. $S_B = Z_B U_{B\,(:,1:k)} \Sigma_{(1:k,1:k)}^{-1/2}, \quad S_C = Z_C U_{C\,(:,1:k)} \Sigma_{(1:k,1:k)}^{-1/2}$
3. $\hat{A} = S_C^H A S_B, \quad \hat{B} = S_C^H B, \quad \hat{C} = C S_B$

The only difference between the classical square root method and this algorithm is, that here (approximate) low rank Cholesky factors Z_B and Z_C are used instead of exact Cholesky factors of the Gramians, which have possibly full rank. This reduces in particular the numerical cost for the singular value decomposition in Step 1 considerably.

However, there are two basic drawbacks of LRSRM compared to the "exact" square root method. Unlike LRSRM, the latter delivers stable reduced systems under mild conditions. Furthermore, there exists an upper error bound for (18) for the standard square root method, which does not apply to the low rank square root method. Thus, it is not surprising that the performance of Algorithm 3 depends on the accuracy of the approximate low rank Cholesky factor products $Z_B Z_B^H$ and $Z_C Z_C^H$ and the value k, where $k \leq \operatorname{rank} Z_C^H Z_B$. This makes the choice of the quantities r_B, r_C, and k a trade-off. Large values of r_B and r_C, and values of k much smaller than $\operatorname{rank} Z_C^H Z_B$ tend to keep the deviation of the low rank square root method from the standard square root method small. On the other hand the computational efficiency of the low rank square roo method is decreased in this way. However, LRCF-ADI often delivers low rank Cholesky factors Z_B and Z_C, whose products approximate the system Gramians nearly up to machine precision. In this case the results of the LYAPACK implementation of the low rank square root method will be about as good as those by any standard implementation of the balanced truncation technique, which, however, can be still numerically much more expensive.

Finally, note that the classical square root method is well-suited to compute (numerically) minimal realizations; e.g., [48]. LRSRM (as well as DSPMR) can be used to compute such realizations for large systems. The term "numerically minimal realization" is not well-defined. Loosely speaking, it is rather the concept of computing a reduced system, for which the (relative) approximation error (18) is of magnitude of the machine precision. See Figure 14 in §C.3.3.

4.2.2 Choice of reduced order

In the LYAPACK implementation of the low rank square root method, the reduced order k can be chosen a priori or in dependence of the descending ordered singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$ computed in Step 1, where $r = \operatorname{rank} Z_C^H Z_B$.

- **Maximal reduced order.** The input parameters `max_ord` of the routine `lp_lrsrm` prescribes the maximal admissible value for the reduced order k, i.e., $k \leq$ `max_ord` is required. If the choice of this value should be avoided, one can set `max_ord` = n or `max_ord = []`.

- **Maximal ratio σ_k/σ_1.** The input parameter `tol` prescribes the maximal admissible value for the ratio σ_k/σ_1. That means k is chosen as the largest index for which $\sigma_k/\sigma_1 \geq$ `tol`. This means that one will generally choose a value of `tol` between the machine precision an 1.

In general, both parameters will determine different values of k. The routine `lp_lrsrm` uses the **smaller** value.

4.2.3 The routine `lp_lrsrm`

Algorithm LRSRM is implemented in the LYAPACK routine `lp_lrsrm`. We provide a brief description of this routine. For more details see the inline documentation which is displayed by typing the MATLAB command `>> help lp_lrsrm`.

Calling sequence:

```
[Ar ,Br, Cr, SB, SC, sigma] = lp_lrsrm( name, B, C, ZB, ZC, ...
max_ord, tol)
```

Input parameters:

`name`: The basis name of the user supplied functions that realize basic matrix operations with A.

`B`: System matrix B.

`C`: System matrix C.

`ZB`: LRCF $Z_B \in \mathbb{C}^{n,r_B}$. This routine is only efficient if $r_B \ll n$.

`ZC`: LRCF $Z_C \in \mathbb{C}^{n,r_C}$. This routine is only efficient if $r_C \ll n$.

`max_ord`: A parameter for the choice of the reduced order k; see §4.2.2.

`tol`: A parameter for the choice of the reduced order k; see §4.2.2.

Output parameters:

`Ar`: Matrix $\hat{A} \in \mathbb{C}^{k,k}$ of reduced system.

`Br`: Matrix $\hat{B} \in \mathbb{C}^{k,m}$ of reduced system.

`Cr`: Matrix $\hat{C} \in \mathbb{C}^{q,k}$ of reduced system.

`SB`: Projection matrix S_B.

`SC`: Projection matrix S_C.

`sigma`: Vector containing the singular values computed in Step 1.

Usually, one is only interested in the first three output parameters.

4.2.4 Case studies

See §C.3.

4.3 Dominant subspaces projection model reduction

4.3.1 Theory and algorithms

The *dominant subspaces projection model reduction* (DSPMR) [32, 39], which is provided as LYAPACK routine `lp_dspmr`, is more heuristic in nature. The basic idea behind this method is that the input-state behaviour and the state-output behavior of the system (16) tend to be dominated by states, which have a strong component w.r.t. the dominant invariant subspaces of the Gramians X_B and X_C. These dominant invariant subspaces are approximated by the left singular vectors of Z_B and Z_C provided that $X_B \approx Z_B Z_B^H$ and $X_C \approx Z_C Z_C^H$. The motivation of the dominant subspace correction method is discussed at length in [39]. Compared to the low rank square root method, the approximation properties of the reduced systems by DSPMR are often less satisfactory, i.e., the error function $\|G(s) - \hat{G}(s)\|$ tends to be less small. On the other hand, DSPMR sometimes delivers a stable reduced system, when that by LRSRM is not stable. In DSPMR, the stability of the reduced system is guaranteed at least if $A + A^T$ is negative definite. Note also that DSPMR uses an orthoprojection, whereas LRSRM is based on an oblique projection. For this reason, DSPMR is also advantageous w.r.t. preserving passivity.

Algorithm 4 (Dominant subspaces projection model reduction (DSPMR))

INPUT: A, B, C, Z_B, Z_C, k

OUTPUT: $\hat{A}$, $\hat{B}$, $\hat{C}$

1. $Z = \left[\begin{array}{cc} \frac{1}{\|Z_B\|_F} Z_B & \frac{1}{\|Z_C\|_F} Z_C \end{array} \right]$

2. $U \Sigma V^H := Z$ ("thin" SVD with descending ordered singular values)

3. $S = U_{(:,1:k)}$

4. $\hat{A} = S^H A S, \quad \hat{B} = S^H B, \quad \hat{C} = C S$

4.3.2 Choice of reduced order

In the LYAPACK implementation of DSPMR, the reduced order k can be chosen a priori or in dependence of the descending ordered singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$ computed in Step 2, where $r = \operatorname{rank} Z$.

- **Maximal reduced order.** The input parameter `max_ord` of the routine `lp_dspmr` prescribes the maximal admissible value for the reduced order k, i.e., $k \leq$ `max_ord` is required. To avoid this choice, one can set `max_ord` = n or `max_ord = []`.

- **Maximal ratio σ_k/σ_1.** The input parameter `tol` determines the maximal admissible value for the ratio σ_k/σ_1. More precisely, k is chosen as the largest index for which $\sigma_k/\sigma_1 \geq \sqrt{\texttt{tol}}$. Note that here the square root of `tol` is used in contrast to LRSRM. (Note that the values σ_i have somewhat different meanings in LRSRM and DSPMR.)

In general, both parameters will determine different values of k. The routine `lp_dspmr` uses the **smaller** value. Finally, it should be mentioned, that, at least in exact arithmetics, both LRSRM and DSPMR (run with identical values `max_ord` and `tol`) deliver the same result for state-space symmetric systems (i.e., systems, where $A = A^T$ and $C = B^T$).

4.3.3 The routine `lp_dspmr`

Algorithm DSPMR is implemented in the LYAPACK routine `lp_dspmr`. We provide a brief description of this routine. For more details see the inline documentation which is displayed by typing the MATLAB command `>> help lp_dspmr`.

Calling sequence:

```
[Ar ,Br, Cr, S] = lp_dspmr( name, B, C, ZB, ZC, max_ord, tol)
```

Input parameters:

`name`: The basis name of the user supplied functions that realize basic matrix operations with A.

`B`: System matrix B.

`C`: System matrix C.

`ZB`: LRCF $Z_B \in \mathbb{C}^{n,r_B}$. This routine is only efficient if $r_B \ll n$.

`ZC`: LRCF $Z_C \in \mathbb{C}^{n,r_C}$. This routine is only efficient if $r_C \ll n$.

`max_ord`: A parameter for the choice of the reduced order k; see §4.3.2.

`tol`: A parameter for the choice of the reduced order k; see §4.3.2.

Output parameters:

`Ar`: Matrix $\hat{A} \in \mathbb{C}^{k,k}$ of reduced system.

`Br`: Matrix $\hat{B} \in \mathbb{C}^{k,m}$ of reduced system.

`Cr`: Matrix $\hat{C} \in \mathbb{C}^{q,k}$ of reduced system.

`S`: Projection matrix S.

4.3.4 Case studies

See §C.3.

5 Riccati equations and linear-quadratic optimal control problems

5.1 Preliminaries

This section mainly deals with the efficient numerical solution of continuous time algebraic Riccati equationss of the type

$$C^TQC + A^TX + XA - XBR^{-1}B^TX = 0, \tag{22}$$

where $A \in \mathbb{R}^{n,n}$, $B \in \mathbb{R}^{n,m}$, and $C \in \mathbb{R}^{q,n}$ with $m, q \ll n$. Moreover, we assume that $Q \in \mathbb{R}^{q,q}$ is symmetric, positive semidefinite and $R \in \mathbb{R}^{m,m}$ is symmetric, positive definite. Unlike in the other sections of this document, we do not assume here that A is stable, but it is required that a matrix $K^{(0)}$ is given, such that $A - BK^{(0)^T}$ is stable. Such a matrix $K^{(0)}$ can be computed by partial pole placement algorithms [21], for example.

In general, the solution of (22) is not unique. However, under the above assumptions, a unique, stabilizing solution X exists, which is the solution of interest in most applications; e.g., [34, 29]. A solution X is called *stabilizing* if the closed-loop matrix $A - BR^{-1}B^TX$ is stable.

Algebraic Riccati equations arise from numerous problems in control theory, such as robust control or certain balancing and model reduction techniques for unstable systems. Another application, for which algorithms are provided by LYAPACK, is the solution of the linear quadratic optimal control problem. In this paragraph, we briefly describe the connection between linear quadratic optimal control problems and algebraic Riccati equations. The linear quadratic optimal control problem is a constrained optimization problem. The cost functional to be minimized, is

$$\mathcal{J}(u, y, x_0) = \frac{1}{2}\int_0^\infty y(\tau)^TQy(\tau) + u(\tau)^TRu(\tau)d\tau\,, \tag{23}$$

where $Q = Q^T \geq 0$ and $R = R^T > 0$. The constraints are given by the dynamical system

$$\begin{aligned} \dot{x}(\tau) &= Ax(\tau) + Bu(\tau) \\ y(\tau) &= Cx(\tau) \end{aligned} \tag{24}$$

and the initial condition

$$x(0) = x_0. \tag{25}$$

The solution of this optimization problem is described by the feedback matrix K, that is defined as

$$K = XBR^{-1}, \tag{26}$$

where X is the stabilizing solution of the algebraic Riccati equation (22). The corresponding control function is given by the state-feedback

$$u(\tau) = -K^Tx(\tau)$$

and the initial condition (25).

To sum up, we consider two problems in this section. The first one is the numerical computation of the stabilizing solution of the continuous time algebraic Riccati equations (22). The second problem is the solution of the linear quadratic optimal control problem (23,24,25), which is a particular application of algebraic Riccati equations. Its solution can be described by the stabilizing solution X, from which the optimal state-feedback can easily be computed via (26), or by the feedback K itself.

LYAPACK contains implementations of the *low rank Cholesky factor Newton method* (LRCF-NM) and the *implicit low rank Cholesky factor Newton method* (LRCF-NM-I) proposed in [6]. LRCF-NM delivers a LRCF Z, such that the product ZZ^H approximates the Riccati solution X. This means that LRCF-NM can be used to solve both continuous time algebraic Riccati equations and linear quadratic optimal control problems. The implicit version LRCF-NM-I, which directly computes an approximation to K without forming Z or X, can only be used to solve the linear quadratic optimal control problem in a more memory efficient way.

Both LRCF-NM and LRCF-NM-I are modifications of the classical Newton method for algebraic Riccati equations [28], or more precisely, combinations of the Newton method with the LRCF-ADI iteration. We will describe these combinations in §§5.2 and 5.3. The classical formulation of the Newton method is given by the double step iteration

$$\begin{array}{l} \text{Solve Lyapunov equation} \\ (A^T - K^{(k-1)}B^T)X^{(k)} + X^{(k)}(A - BK^{(k-1)^T}) = -C^TQC - K^{(k-1)}RK^{(k-1)^T} \\ \text{for } \; X^{(k)}, \\ \\ K^{(k)} = X^{(k)}BR^{-1} \end{array} \tag{27}$$

for $k = 1, 2, 3, \ldots$, which generates a sequence of iterates $X^{(k)}$. This sequence converges towards the stabilizing solution X if the initial feedback K_0 is stabilizing, i.e., $A - BK^{(0)^T}$ is stable. Then, the convergence is global and quadratic.

5.2 Low rank Cholesky factor Newton method

Due to the symmetry and definiteness assumptions, the matrices Q and R can be factored (by a Cholesky factorization, for example) as

$$Q = \tilde{Q}\tilde{Q}^T \qquad \text{and} \qquad R = \tilde{R}\tilde{R}^T, \tag{28}$$

where the matrices $\tilde{Q} \in \mathbb{R}^{q,h}$ $(h \leq q)$ and $\tilde{R} \in \mathbb{R}^{m,m}$ have full rank. Thus, the Lyapunov equations to be solved in (27) have the structure

$$F^{(k)}X^{(k)} + X^{(k)}F^{(k)^T} = -G^{(k)}G^{(k)^T}$$

where $F^{(k)} = A^T - K^{(k-1)}B^T$ and $G^{(k)} = \left[\begin{array}{cc} C^T\tilde{Q} & K^{(k-1)}\tilde{R} \end{array}\right]$. Note that $G^{(k)}$ contains only $t = m + h \ll n$ columns. Hence, these Lyapunov can be solved efficiently by the LRCF-ADI iteration. The Lyapunov solutions form a sequence of approximate solutions to the algebraic Riccati equations (22). Therefore, the inclusion of Algorithm 1 into the Newton iteration (27) can be utilized to determine low rank Cholesky factor products which approximate the solution of the algebraic Riccati equation (22). The resulting algorithm low rank Cholesky factor Newton method is described below.

Algorithm 5 (Low rank Cholesky factor Newton method (LRCF-NM))

INPUT: A, B, C, Q, R, $K^{(0)}$ for which $A - BK^{(0)^T}$ is stable (e.g., $K^{(0)} = 0$ if A is stable)

OUTPUT: $Z = Z^{(k_{max})}$, such that ZZ^H approximates the solution X of the algebraic Riccati equation (8)

FOR $k = 1, 2, \ldots, k_{max}$

1. Determine (sub)optimal ADI shift parameters $p_1^{(k)}, p_2^{(k)}, \ldots$ with respect to the matrix $F^{(k)} = A^T - K^{(k-1)}B^T$.

2. $G^{(k)} = \left[\begin{array}{cc} C^T\tilde{Q} & K^{(k-1)}\tilde{R} \end{array}\right]$

3. Compute matrix $Z^{(k)}$ by Algorithm 1, such that the low rank Cholesky factor product $Z^{(k)}Z^{(k)^H}$ approximates the solution of $F^{(k)}X^{(k)} + X^{(k)}F^{(k)^T} = -G^{(k)}G^{(k)^T}$.

4. $K^{(k)} = Z^{(k)}(Z^{(k)^H}BR^{-1})$

END

Similar to the LRCF-ADI iteration for the solution of Lyapunov equations, the distinct merit of this algorithm is that the (approximate) solution of the algebraic Riccati equations is provided as a low rank Cholesky factor product rather than an explicit dense matrix. In particular, this allows the application of the algorithm to problems of large order n, where dense $n \times n$ matrices cannot be stored in the computer memory. Moreover, the LRCF-NM requires often much less computation compared to the standard implementation, where Lyapunov are solved directly by the Bartels-Stewart or the Hammarling method; see §7. See [6] for more technical details of the LRCF-NM.

5.3 Implicit low rank Cholesky factor Newton method

The idea behind the implicit version of LRCF-NM is that the solution of the linear quadratic optimal control problem is described by the state feedback matrix K, which generally contains much less columns than the low rank Cholesky factor Z delivered by LRCF-NM or even the exact solution X. LRCF-NM-I is mathematically equivalent to LRCF-NM. It computes an approximation to K without forming LRCF-NM iterates $Z^{(k)}$ and LRCF-ADI iterates $Z_i^{(k)}$ at all. The trick is to generate the matrix $K^{(k)}$ itself in Step 3 of Algorithm 5 instead of solving the Lyapunov equation for $Z^{(k)}$ and computing the product $K^{(k)} = Z^{(k)}Z^{(k)^H}BR^{-1}$ in Step 4. Note that the matrix $K^{(k)}$ can be accumulated in the course of the "inner" LRCF-ADI iteration as

$$K^{(k)} = \lim_{i \to \infty} K_i^{(k)},$$

where

$$K_i^{(k)} := Z_i^{(k)}Z_i^{(k)^H}BR^{-1} = \sum_{j=1}^{i} V_j^{(k)} \left(V_j^{(k)^H} BR^{-1} \right). \tag{29}$$

This means, that the (exact) matrix K is the limit of the matrices $K_i^{(k)}$ for $k, i \to \infty$. This consideration motivates the following Algorithm 6, which is best understood as a version of the LRCF-NM with an inner loop (Steps 4 and 5) consisting of interlaced sequences based on Step 3 in Algorithm 1 and the partial sums given by the right hand term in (29).

Algorithm 6 (Implicit low rank Cholesky factor Newton method (LRCF-NM-I))

INPUT: A, B, C, Q, R, $K^{(0)}$ for which $A - BK^{(0)^T}$ is stable (e.g., $K^{(0)} = 0$, if A is stable)

OUTPUT: $K^{(k_{max})}$, which approximates K given by (26)

FOR $k = 1, 2, \ldots, k_{max}$

1. Determine (sub)optimal ADI shift parameters $p_1^{(k)}, p_2^{(k)}, \ldots$ with respect to the matrix $F^{(k)} = A^T - K^{(k-1)} B^T$.

2. $G^{(k)} = \left[\begin{array}{cc} C^T \tilde{Q} & K^{(k-1)} \tilde{R} \end{array} \right]$

3. $V_1^{(k)} = \sqrt{-2 \operatorname{Re} p_1^{(k)}} (F^{(k)} + p_1^{(k)} I_n)^{-1} G^{(k)}$

FOR $i = 2, 3, \ldots, i_{max}^{(k)}$

4. $V_i^{(k)} = \sqrt{\operatorname{Re} p_i^{(k)} / \operatorname{Re} p_{i-1}^{(k)}} \left(V_{i-1}^{(k)} - (p_i^{(k)} + \bar{p}_{i-1}^{(k)})(F^{(k)} + p_i^{(k)} I_n)^{-1} V_{i-1}^{(k)} \right)$

5. $K_i^{(k)} = K_{i-1}^{(k)} + V_i^{(k)} \left(V_i^{(k)^H} B R^{-1} \right)$

END

6. $K^{(k)} = K_{i_{max}^{(k)}}^{(k)}$

END

Again, see [6] for more implementational details.

5.4 Stopping criteria

As far as possible, the same stopping criteria are used in LRCF-NM and LRCF-NM-I for terminating the (outer) Newton iteration. The LYAPACK routine `lp_lrnm`, in which both methods are implemented, offers the following five criteria:

- maximal number of iteration steps: used in LRCF-NM and LRCF-NM-I;
- tolerance for the normalized residual norm: used in LRCF-NM only;
- stagnation of the normalized residual norm (most likely caused by round-off errors): used in LRCF-NM only;
- smallness of the *relative change of the feedback matrix* (RCF): Used in LRCF-NM and LRCF-NM-I;
- stagnation of the relative change of the feedback matrix: used in LRCF-NM and LRCF-NM-I.

Here, the normalized residual norm corresponding to the low rank Cholesky factor $Z^{(k)}$ is defined as

$$\mathrm{NRN}(Z^{(k)}) = \frac{\|C^T Q C + A^T Z^{(k)} Z^{(k)^H} + Z^{(k)} Z^{(k)^H} A - Z^{(k)} Z^{(k)^H} B R^{-1} B^T Z^{(k)} Z^{(k)^H}\|_F}{\|C^T Q C\|_F}, \tag{30}$$

whereas the relative change of the feedback matrix related to the matrices $K^{(k-1)}$ and $K^{(k)}$ is

$$\mathrm{RCF}(K^{(k-1)}, K^{(k)}) = \frac{\|K^{(k)} - K^{(k-1)}\|_F}{\|K^{(k)}\|_F}. \tag{31}$$

Many of the remarks on stopping criteria for the LRCF-ADI iteration made in §3.1.2 also apply to stopping criteria for LRCF-NM or LRCF-NM-I. In particular, the application of stopping criteria, which require the computation of normalized residual norms is numerically expensive. Although the applied computational method [6] exploits the low rank structure of the approximate solutions, it can be more expensive than the iteration itself. Moreover, it is not possible to use residual based stopping criteria for LRCF-NM-I, because there the low rank Cholesky factors $Z^{(k)}$ are not formed at all, which is the only reason why one would apply LRCF-NM-I instead of LRCF-NM.

The consideration of (31) for the construction of heuristic stopping criteria is related to the fact that in some sense the matrices $K^{(k)}$ rather than the low rank Cholesky factors $Z^{(k)}$ or their products are the quantities of interest when the optimal control problem should be solved. However, stopping criteria related to $K^{(k)}$ are somewhat dubious when the optimal feedback K or, more precisely, the product BK^T is very small compared to A, because then small relative changes in K hardly change the closed-loop matrix $A - BK^T$. On the other hand, the accuracy of K does not play a crucial role in such situations, which means that a possibly premature termination of the Newton iteration would not be very harmful.

We will now discuss the five stopping criteria. Convergence plots generated for an example problem illustrate their effects. Note that, similar to the criteria for the LRCF-ADI iteration described in §3.1.2, the following stopping criteria can be "activated" or "avoided".

- **Stopping criterion: maximal number of iteration steps.** This criterion is represented by the input parameter `max_it_r` in the routine `lp_lrnm`. The iteration is stopped by this criterion after `max_it_r` iterations steps. This criterion can be avoided by setting `max_it_r = +Inf` (i.e., `max_it_r` $= \infty$). Obviously, no additional computations need to be performed to evaluate it. The drawback of this stopping criterion is, that it is not directly related to the attainable accuracy. This is illustrated by Figure 7.

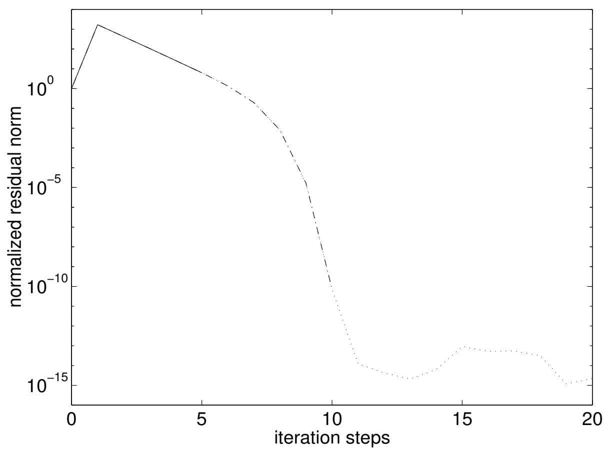

Figure 7: Stopping criterion: maximal number of iteration steps. Solid line: `max_it_r` $= 5$; dashdotted line: `max_it_r` $= 10$; dotted line: `max_it_r` $= 20$. The other four criteria are avoided.

- **Stopping criterion: tolerance for the normalized residual norm.** This criterion is represented by the input parameter `min_res_r` in the routine `lp_lrnm`. The iteration is stopped by this criterion as soon as

$$\mathrm{NRN}(Z^{(k)}) \leq \texttt{min\_res\_r}.$$

This criterion can be avoided by setting `min_res_r` $= 0$. (Because of round-off errors it is practically impossible to attain $\mathrm{NRN}(Z^{(k)}) = 0$.) It requires the computation of normalized residual norms and is computationally expensive. A further drawback of this criterion is that it will either stop the iteration before the maximal accuracy is attained (see `min_res_r` $= 10^{-5}$, 10^{-10} in Figure 8) or it will not stop the iteration at all (see `min_res_r` $= 10^{-15}$ in Figure 8). If you want to avoid this criterion, but compute the convergence history provided by the output vector `res_r`, set `min_res_r` to a value much smaller than the machine precision (say, `min_res_r` $= 10^{-100}$).

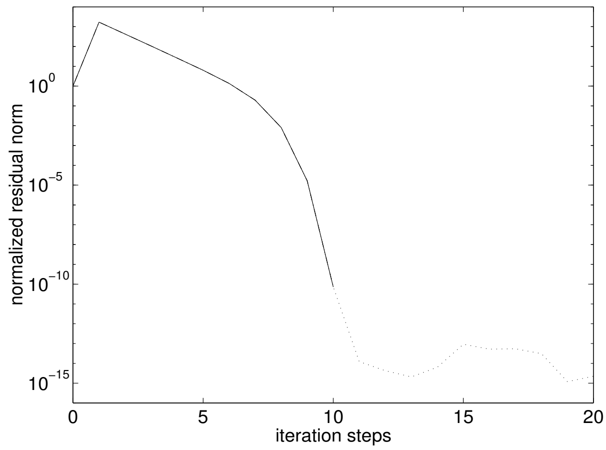

Figure 8: Stopping criterion: tolerance for the normalized residual norm. Solid line: `min_res_r` $= 10^{-5}$; dashdotted line: `min_res_r` $= 10^{-10}$; dotted line: `min_res_r` $= 10^{-15}$. Here, the dashdotted and the solid line are identical. The other four criteria are avoided.

- **Stopping criterion: stagnation of the normalized residual norm.** This criterion is represented by the input parameter `with_rs_r` in the routine `lp_lrnm`. It is activated if `with_rs_r` = `'S'` and avoided if `with_rs_r` = `'N'`. The iteration is stopped by this criterion when a stagnation of the normalized residual norm curve is detected. In contrast to the corresponding criterion for the LRCF-ADI iteration, this criterion stops the iteration, when the stagnation of the normalized residual norm is detected for a single iteration step. Of course, this is a slightly heuristic criterion but it works very well in practice. It requires the computation of the normalized residual norm and is computationally expensive. See Figure 9.

- **Stopping criterion: smallness of the the relative change of the feedback matrix.** This criterion is represented by the input parameter `min_ck_r` in the routine `lp_lrnm`. The iteration is stopped by this criterion as soon as

$$\mathrm{RCF}(K^{(k-1)}, K^{(k)}) \leq \texttt{min\_ck\_r}.$$

This criterion can be avoided by setting `min_ck_r` $= 0$. It is numerically very inexpensive. On the other hand it is heuristic and not directly related to the accuracy in $Z^{(k)}$. See Figure 10.

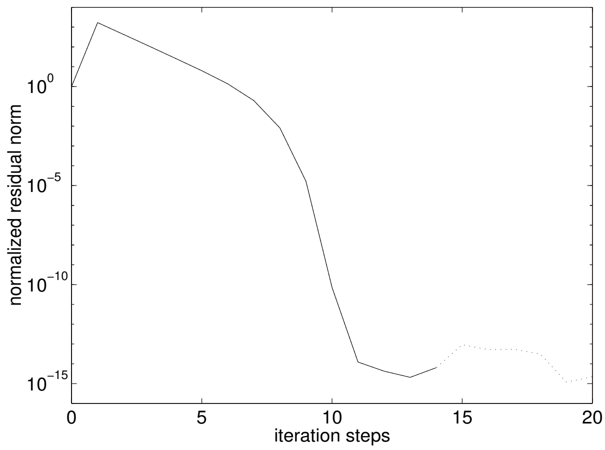

Figure 9: Stopping criterion: Stagnation of the normalized residual norms. Solid line: with_rs_r = 'S'; dotted line: with_rs_r = 'N'. The other four criteria are avoided.

- **Stopping criterion: stagnation of the relative change of the feedback matrix.** This criterion is represented by the input parameter with_ks_r in the routine lp_lrnm. It is activated if with_ks_r = 'L' and avoided if with_ks_r = 'N'. The iteration is stopped by this criterion when a stagnation of the relative change of the feedback matrix is detected. Similar to the last criterion, this is a inexpensive, but heuristic stopping criterion. See Figure 11.

We recommend to use (only) the stopping criterion related to with_rs_r if algebraic Riccati equations solutions of high accuracy should be computed and if it is affordable to compute normalized residual norms. If the computation of the normalized residual norms must be avoided, the combination of the criteria related to min_ck_r and with_rs_r is probably the best choice. Experience shows that often only one of them will stop the iteration after a reasonable number of steps. See, for example, Figure 11, where the criterion related to with_rs_r failed.

5.5 The routine lp_lrnm

Both LRCF-NM and LRCF-NM-I are implemented in the LYAPACK routine lp_lrnm. We provide a brief description of this routine. For more details see the inline documentation which is displayed by typing the MATLAB command >> help lp_lrnm.

The approximate solution of the algebraic Riccati equations (22) is given by the low rank Cholesky factor Z, such that $ZZ^H \approx X$. Z has typically fewer columns than rows. Otherwise, LRCF-NM is useless! In general, Z can be a complex matrix, but the product ZZ^H is real. In the explicit mode of lp_lrnm (i.e., the one for LRCF-NM) an optional internal postprocessing step can be performed, which guarantees that the delivered low rank Cholesky factor Z is real. This requires additional computation. This postprocessing is only done for the low rank Cholesky factor computed in the last Newton step. This means, that its relative contribution to the overall cost is smaller than in the LRCF-ADI iteration.

Calling sequences:

Depending on the choice of the mode parameter zk, the following two calling sequences exist. For zk = 'Z', the low rank Cholesky factor Z is computed by LRCF-NM, whereas

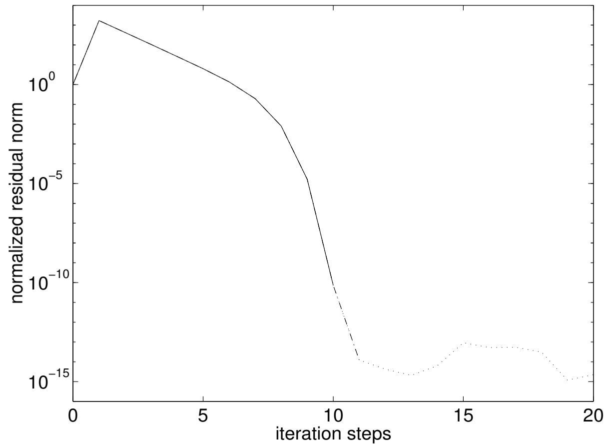

Figure 10: Stopping criterion: Smallness of the the relative change of the feedback matrix. Solid line: min_ck_r $= 10^{-4}$; dashdotted line: min_ck_r $= 10^{-8}$; dotted line: min_ck_r $= 10^{-16}$. The other four criteria are avoided. Note that it is mere coincidence, that min_ck_r $= 10^{-8}$ (dashdotted line) leads to the termination after the "optimal" number of steps.

for zk = 'K', the feedback matrix K is computed by LRCF-NM-I.

- zk = 'Z':

```
[Z, flag_r, res_r, flp_r, flag_l, its_l, res_l, flp_l] = ...
lp_lrnm( zk, rc, name, B, C, Q0, R0, K_in, max_it_r, ...
min_res_r, with_rs_r, min_ck_r, with_ks_r, info_r, kp, km, ...
l0, max_it_l, min_res_l, with_rs_l, min_in_l, info_l )
```

- zk = 'K':

```
[K_out, flag_r, flp_r, flag_l, its_l, flp_l] = lp_lrnm(...
zk, name, B, C, Q0, R0, K_in, max_it_r, min_ck_r, ...
with_ks_r, info_r, kp, km, l0, max_it_l, min_in_l, info_l )
```

Input parameters:

zk: Mode parameter, which is either 'Z' or 'K'. If zk = 'Z', the low rank Cholesky factor $Z = Z^{(k_{max})}$ is computed by LRCF-NM. Otherwise, $K^{(k_{max})}$ is directly computed by LRCF-ADI-I.

rc: Mode parameter, which is either 'R' or 'C'. If rc = 'C', the routine delivers a low rank Cholesky factor, which is not real when non-real shift parameters are used in the last Newton step. Otherwise, this possibly complex low rank Cholesky factor is transformed into a real low rank Cholesky factor $\tilde{Z}$, which describes the identical approximate solution $\tilde{Z}\tilde{Z}^T$. $\tilde{Z}$ is returned instead of Z. The parameter rc is not needed in the mode for LRCF-NM-I, because the returned feedback (parameter K_out) is always real, provided that $K^{(0)}$ (parameter K_in) is real.

name: The basis name of the user supplied functions that realize basic matrix operations with A.

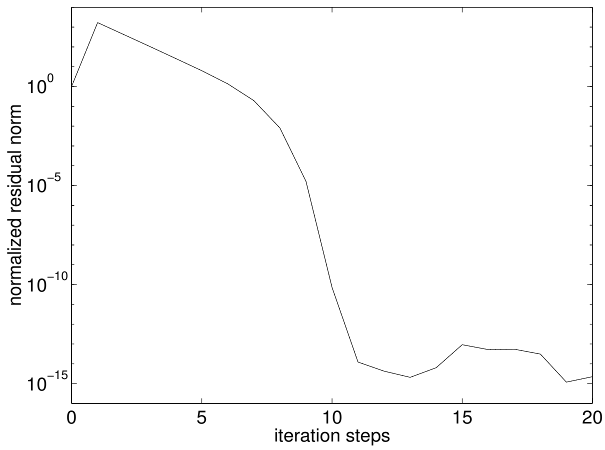

Figure 11: Stopping criterion: Stagnation of the relative change of the feedback matrix. Solid line: `with_rs_r = 'L'`; dotted line: `with_rs_r = 'N'`. The other four criteria are avoided. For this particular example, no stagnation in the relative change of the feedback matrix is observed within 20 iteration steps.

`B`: System matrix B.

`C`: System matrix C.

`Q0`: The Cholesky factor $\tilde{Q}$ defined in (28).

`R0`: The Cholesky factor $\tilde{R}$ defined in (28).

`K_in`: The stabilizing initial state feedback $K^{(0)}$. If A is stable, $K^{(0)} = 0$ can be used, for example.

`max_it_r`: Stopping parameter for (outer) Newton iteration. See §5.4.

`min_res_r`: Stopping parameter for (outer) Newton iteration. See §5.4.

`with_rs_r`: Stopping parameter for (outer) Newton iteration. See §5.4.

`min_ck_r`: Stopping parameter for (outer) Newton iteration. See §5.4.

`with_ks_r`: Stopping parameter for (outer) Newton iteration. See §5.4.

`info_r`: Parameter, which determines the "amount" of information on the (outer) Newton iteration that is provided as text and/or residual history plot. The following values are possible: `info_r` = `0` (no information), `1`, `2`, and `3` (most possible information).

`l0`: Parameter l_0 for the ADI parameter routine `lp_para`, which is invoked in each Newton step. Note that $k_+ + k_- > 2l_0$ is required.

`kp`: Parameter k_+ for the ADI parameter routine `lp_para`.

`km`: Parameter k_- for the ADI parameter routine `lp_para`.

`max_it_l`: Stopping parameter for the (inner) LRCF-ADI iterations. See §3.1.2.

`min_res_l`: Stopping parameter for the (inner) LRCF-ADI iterations. See §3.1.2.

with_rs_l: Stopping parameter for the (inner) LRCF-ADI iterations. See §3.1.2.

min_in_l: Stopping parameter for the (inner) LRCF-ADI iterations. See §3.1.2.

info_l: Parameter, which determines the "amount" of information on the (inner) LRCF-ADI iterations that is provided as text and/or residual history plot. The following values are possible: info_l = 0 (no information), 1, 2, and 3 (most possible information).

Output parameters:

Z: The low rank Cholesky factor Z, which is the result of LRCF-NM. It can be complex under certain circumstances.

K_out: The matrix $K^{(k_{max})}$, which is the result of LRCF-NM-I.

flag_r: A flag, that shows by which stopping criterion (or stopping parameter) the (outer) Newton iteration has been stopped. Possible values are 'I' (for max_it_r), 'R' (for min_res_r), 'S' (for with_rs_r), 'K' (for min_ck_r), and 'L' (for with_ks_r).

res_r: A vector containing the history of the algebraic Riccati equations normalized residual norms (30). res_r(1) = 1 and res_r($i+1$) is the normalized residual norm w.r.t. the Newton step i. If the stopping criteria are chosen, so that the normalized residual norms need not be computed, res_r = [] is returned.

flp_r: A vector containing the history of the flops needed for the algorithm. flp_r(1) = 0 and flp_r($i+1$) is the number of flops required for the Newton steps 1 to i. flp_r displays the number of flops required for the actual iteration. It also contains the numerical costs for all user supplied functions invoked within lp_lrnm as well as the computation of the sets of ADI shift parameters. However, the costs for the computation of Riccati equations or Lyapunov equation normalized residual norms are not included.

flag_l: Vector containing the values flag returned by the LRCF-ADI routine lp_lradi, which is called in each Newton step.

its_l: Vector containing the number of iteration steps of the (inner) LRCF-ADI iterations.

res_l: Matrix whose columns contain the normalized residual norm history vectors res returned by the LRCF-ADI routine lp_lradi. Here, normalized residual norms in the sense of (13) are considered.

flp_l: Matrix whose columns contain the flop history vectors flp returned by the LRCF-ADI routine lp_lradi in each Newton step.

6 Supplementary routines and data files

Supplementary routines are routines which do not play a central role in LYAPACK but can be used to generate test problems in order to validate the results delivered by LYAPACK main routines. There are also test examples in form of data files provided.

6.1 Computation of residual norms for Lyapunov and Riccati equations

The accuracy of the approximate solution ZZ^H of the Lyapunov equation (2) or the Riccati equation (8) can be assessed by the residual norm of the Lyapunov equation

$$\|FZZ^H + ZZ^H F^T + GG^T\|_F \tag{32}$$

or the residual norm of the Riccati equation

$$\|C^T QC + A^T ZZ^H + ZZ^H A - ZZ^H BR^{-1} B^T ZZ^H\|_F \,. \tag{33}$$

The following two LYAPACK routines can be used to compute such norms.

`lp_nrm`: Computes the Lyapunov equation residual norm (32) by the technique described in [39].

`lp_rcnrm`: Computes the Riccati equation residual norm (33) by the technique described in [6].

Note, that these routines do not evaluate the residual matrices, i.e., the terms inside the norms. They rather make use of the low rank structure of ZZ^H, which is often much more efficient w.r.t. both memory and computation. However, both routines are not efficient if the number of columns in Z is almost n or even larger than n.

6.2 Evaluation of model reduction error

The accuracy of the reduced system (3), which approximates the system (1), is usually evaluated by comparing their transfer functions $\hat{G}(s)$ and $G(s)$ on the imaginary axis, which show the frequency responses of both systems.

If the system is a single-input single-output (SISO) system (i.e., $m = q = 1$) and the reduced system is not very accurate, *simultaneous magnitude Bode plots* can be used to compare both frequency responses. To do this, one plots the functions $|G(\jmath\omega)|$ and $|\hat{G}(\jmath\omega)|$ simultaneously for a certain "frequency range" $\omega \in [\omega_{min}, \omega_{max}]$. There, exist also Bode phase plots, where the phase angles of the complex functions $G(\jmath\omega)$ and $\hat{G}(\jmath\omega)$ are compared, but these are usually less important. If the system is not a SISO system, mq plots w.r.t. the single components of the transfer function can be used for the comparison.

If the system has multiple inputs or multiple outputs, or when the approximation error of the reduced system is very small, error plots, which show the function $\|G(\jmath\omega) - \hat{G}(\jmath\omega)\|$ for an interval $\omega \in [\omega_{min}, \omega_{max}]$ are more meaningful.

To generate either type of plot, the following LYAPACK functions can be used.

`lp_lgfrq`: Generates a set of logarithmically distributed "frequency sampling points" ω_i ($i = 1, \dots, i_{max}$) in the interval $[\omega_{min}, \omega_{max}]$, i.e, $\omega_1 = \omega_{min}$, $\omega_{i_{max}} = \omega_{max}$, and $\omega_{i+1}/\omega_i = \text{const.}$

`lp_trfia`: Generates the matrices $G(\jmath\omega_i)$ ($i = 1, \dots, i_{max}$). Their stacked columns are stored in an $mq \times i_{max}$ "transfer function sample" matrix G_s.

`lp_gnorm`: Computes $\|G(\jmath\omega_i)\|$ ($i = 1, \dots, i_{max}$), where the matrices $\|G(\jmath\omega_i)\|$ are retrieved from the matrix G_s generated by `lp_trfia`.

Finally, a few comments on the usage of these functions should be made.

Unlike the other LYAPACK routines, which have access to the system matrix A, lp_trfia does not make use of user supplied functions. On the other hand, this routine can be applied to the more general form of a dynamical system (which is slightly more general than (9))

$$\begin{aligned} E\dot{x}(\tau) &= Ax(\tau) + Bu(\tau) \\ y(\tau) &= Cx(\tau) + Du(\tau) \end{aligned} \tag{34}$$

to generate its transfer function $G(s) = C(sE-A)^{-1}B+D$ on the imaginary axis. However, it is required that all matrices are given explicitely. A and E should be preferably sparse.

Typically, lp_trfia and lp_gnorm will be used subsequently. It is important that the same set of frequency sampling points ω_i is used in both routines. If the mq Bode magnitude plots of a system with multiple inputs or multiple outputs should be generated, then lp_gnorm must be applied mq times to the single rows of the matrix G_s generated by lp_trfia. The approximation error function $\|G(\jmath\omega) - \hat{G}(\jmath\omega)\|$ can be evaluated easily. First, lp_trfia is applied to both the original and the reduced system, which results in the transfer function samples G_s and $\hat{G}_s$. Then, lp_gnorm is applied to the difference $G_s - \hat{G}_s$, which delivers the desired result.

6.2.1 Generation of test examples

The following two routines can generate very simple test examples of systems (1).

fdm_2d_matrix: Generates the negative stiffness matrix for a 2D parabolic differential equation, which is semidiscretized by the *finite difference method* (FDM). This stiffness matrix can be used as system matrix A.

fdm_2d_vector: Generates the corresponding load vectors, which can be used as system matrices B and C^T.

The matrices of a generalized system (9), which arises from the semidiscretization of a steel rail cooling problem (see, e.g., [39]) by the *finite element method* (FEM), are provided in two MATLAB data files.

rail821.mat: Data for a coarse discretization: $n = 821$, $m = q = 6$.

rail3113.mat: Data for a finer discretization: $n = 3113$, $m = q = 6$.

6.3 Case studies

The usage of the routines for computing Lyapunov equation or Riccati equation residual norms is demonstrated in §C.2.1 and §C.4.1, respectively. The application of lp_lgfrq, lp_trfia, and lp_gnorm is demonstrated in §§C.3.1 and C.3.3. Routines for the generation of test examples and data files are used in all demo programs in §C.

7 Alternative methods

Under certain conditions LYAPACK works very well for the types of problems described in §1.1. However, we are far from claiming that the methods implemented in this package are the ultimate solution techniques for the respective problems. In this section, we want to give a brief and by far not complete survey on alternative methods. In many cases, no comparative studies of these methods have been done. LYAPACK is one step in this direction.

- **Lyapunov equations.** Standard techniques for small dense Lyapunov equations are the Bartels-Stewart method [3] or Hammarling method [20]. Extensions of these methods to generalized Lyapunov equations are described in [38]. Large dense Lyapunov equations can be solved by sign function based techniques [42, 1, 5, 8] (see also references to Riccati equations), which perform well on parallel computers. This also applies to the squared Smith method [45]. Relatively large sparse Lyapunov equations can be solved by (standard) ADI, e.g., [36, 50]. Several approaches for the iterative solution of large sparse Lyapunov equations exist. In LYAPACK low rank versions of the ADI method, which is related to rational matrix functions, are used [31, 37, 6, 33]. Krylov subspace methods, which are related to matrix polynomials have been proposed in [43, 22, 24], for example.

- **Model reduction.** Model reduction methods for small, possibly dense systems are abundant. The perhaps most popular technique for reducing stable systems is balanced truncation [35] and all-optimal Hankel norm approximation [18]. Numerically elaborate implementations of the balanced truncation technique are proposed in [47, 44, 48]. Algorithms for solving large dense model reduction problems on parallel computers can be found in [9]. The majority of model reduction methods for large sparse problems is related to Padé approximations of the underlying transfer function, e.g., [41, 15, 12, 16]. A quite detailed survey on this topic can be found in [13]. Methods that are (directly) based on Krylov subspace techniques have been proposed in [25, 23, 26]. The algorithms implemented in LYAPACK are described in [32, 39, 33] at length.

- **Riccati equations and optimal control problems.** In LYAPACK, only the solution of large optimal control problems by solving Riccati equations is considered [6]. However, "Riccati equation-free" solution techniques for optimal control problems surely exist. Standard techniques for small, possibly dense Riccati equations are the Schur method [30], (standard) Newton method and modifications [28, 34, 29, 4], and the sign function method, e.g., [42, 10, 17, 27].

Numerically reliable and versatile codes for dense problems of moderate size are can be found in the freeware subroutine library SLICOT (Subroutine Library in Control Theory) [7].

A Acronyms and symbols

ADI	alternating direction implicit (algorithm)
BMO	basic matrix operation
CALE	continuous-time algebraic Lyapunov equation
CARE	continuous-time algebraic Riccati equation
DSPMR	dominant subspaces projection model reduction (algorithm)
FDM	finite difference method
FEM	finite element method
flop	floating point operation
NRN	normalized residual norm
LQOCP	linear-quadratic optimal control problem
LRCF	low rank Cholesky factor
LRCF-ADI	low rank Cholesky factor ADI (algorithm)
LRCF-NM	low rank Cholesky factor Newton method (algorithm)
LRCF-NM-I	low rank Cholesky factor Newton method – implicit version (algorithm)
LRCFP	low rank Cholesky factor product
LRSRM	low rank square root method (algorithm)
LYAPACK	Lyapunov package
PDE	partial differential equation
RCF	relative change of the feedback matrix
SISO	single-input single-output
SLE	system of linear equations
SVD	singular value decomposition
USF	user-supplied function

A^H	conjugate transposed of the matrix A
A^T	transposed of the matrix A
$\mathbb{C}$, $\mathbb{C}^n$, $\mathbb{C}^{n,m}$	complex numbers, vectors, matrices
$\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^{n,m}$	real numbers, vectors, matrices
$\|A\|$	spectral norm of the matrix A
$\|A\|_F$	Frobenius norm of the matrix A
$\|G\|_{L_\infty}$	L_∞ norm of a dynamical system
$\sigma(A)$	spectrum of the matrix A
$\jmath$	$\sqrt{-1}$
$*$	"wild card"

B List of LYAPACK routines

B.1 Main routines

These are the essential computational routines, which are called within the main programs written by users themselves.

`lp_dspmr`: Model reduction algorithm DSPMR.

`lp_lradi`: LRCF-ADI iteration for solving Lyapunov equations.

`lp_lrnm`: Both versions of Newton method (LRCF-NM and LRCF-NM-I) for solving Riccati equations and optimal control problems.

`lp_lrsrm`: Model reduction algorithm LRSRM.

`lp_para`: Computation of ADI shift parameters.

B.2 Supplementary routines and data files

The following routines can be used for a verification of the results delivered by LYAPACK main routines.

`lp_gnorm`: Computation of norms of transfer function sample.

`lp_lgfrq`: Computation of logarithmically distributed frequency sampling points in a certain frequency range.

`lp_nrm`: Efficient computation of the Lyapunov equation residual norm.

`lp_rcnrm`: Efficient computation of the Riccati equation residual norm.

`lp_trfia`: Computation of transfer function sample.

The following routines and data files are used for generating test examples.

`fdm_2d_matrix`: Generates negative stiffness matrix for 2D PDE problem.

`fdm_2d_vector`: Generates load vector for 2D PDE problem.

`rail821.mat`: Data file for steel rail cooling problem (order $n = 821$).

`rail3113.mat`: Data file for steel rail cooling problem (order $n = 3113$).

B.3 Auxiliary routines

These are routines for internal use. They are not intended for explicit use in main programs.

`lp_arn_m`: Arnoldi process w.r.t. F^{-1}.

`lp_arn_p`: Arnoldi process w.r.t. F.

`lp_e`: Evaluation of certain strings.

`lp_mnmx`: Suboptimal solution of ADI minimax problem.

`lp_nrmu`: Efficient computation of the Lyapunov equation residual norm based on updated QR factorizations.

`lp_prm`: Bandwidth reduction by reordering the rows and columns of a matrix or a matrix pair.

`lp_s`: Auxiliary routine for `lp_mnmx`.

B.4 User-supplied functions

This class of user supplied functions comprises a relatively large number of routines. The routine name and the purpose of the single routines arises from the respective combination of the basis name and the extension(s)

$$[\text{USF name}] \quad = \quad [\text{basis name}]\_[\text{extension(s)}],$$

which has been discussed quite detailed in §2.2.

- [basis name]:

  `as`: Standard system (1); sparse matrix A is symmetric and shift parameters are real; (shifted) linear systems are solved directly.

  `au`: Standard system (1); sparse matrix A is (possibly) unsymmetric or shift parameters are not necessarily real; (shifted) linear systems are solved directly.

  `au_qmr_ilu`: Standard system (1); sparse matrix A is (possibly) unsymmetric or shift parameters are not necessarily real; (shifted) linear systems are solved iteratively by QMR with ILU preconditioning.

  `msns`: Generalized system (9); sparse (definite) matrices M and N are symmetric and shift parameters are real; (shifted) linear systems are solved directly.

  `munu`: generalized system (9); sparse matrices M and N are (possibly) unsymmetric or shift parameters are not necessarily real; (shifted) linear systems are solved directly.

- [extension(s)]:

  `m_i`: Initialization or generation of data needed for multiplications with A.

  `m`: Perform multiplication.

  `m_d`: Delete data that has been needed for multiplications.

  `l_i`: Initialization or generation of data needed for solving linear systems with A.

  `l`: Solve linear system.

  `l_d`: Delete data that has been needed for solving linear systems.

  `s_i`: Initialization or generation of data needed for solving shifted linear systems.

  `s`: Solve shifted linear system.

  `s_d`: Delete data that has been needed for solving shifted linear systems.

  `pre`: Preprocessing (not for `au_qmr_ilu`).

  `pst`: Postprocessing (not for `au_qmr_ilu`).

B.5 Demo programs

`demo_l1`: Demo program for LRCF-ADI iteration and computation of ADI parameters.

`demo_m1`, `demo_m2`: Demo programs for model reduction.

`demo_u1`, `demo_u2`, `demo_u3`: Demo programs for user supplied functions.

`demo_r1`: Demo program for Riccati equations and optimal control problems.

C Case studies

In this section we provide listings of the demo programs which are included in LYAPACK. In these programs, we usually provide matrices that correspond to transformed (preprocessed) problems with a zero subscript (e.g., A0 or A_0) to distinguish them from data related to the original problem (e.g., A or A).

C.1 Demo programs for user-supplied functions

C.1.1 Demo program demo_u1:

```
%
%  REALIZATION OF BASIC MATRIX OPERATIONS BY USER-SUPPLIED
%  FUNCTIONS 'au_*'
%
%
%  This demo program shows how the user-supplied functions 'au_*' work.
%  This means that we consider (possibly) unsymmetric matrices and
%  (possibly) non-real shift parameters.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example we use a simple FDM-semidiscretized PDE problem
% (an instationary convection-diffusion heat equation on the unit square
% with homogeneous 1st kind boundary conditions).
% We reorder the columns and rows of the resulting stiffness matrix by
% a random permutation, to generate a "bad" nonzero pattern.

n0 = 20;    % n0 = number of grid points in either space direction;
            % n = n0^2 is the problem dimension!
            % (Change n0 to generate problems of different size.)

A = fdm_2d_matrix(n0,'10*x','100*y','0');    % Note: A is unsymmetric.
[dummy,pm] = sort(randn(n0^2,1));   % generate a random permutation
A = A(pm,pm);

disp('Problem dimensions:')

n = size(A,1)   % problem order

t = 3; j = sqrt(-1);            % generate complex matrix X0, which
X0 = randn(n,t)+j*randn(n,t);   % contains much fewer columns than rows


% -----------------------------------------------------------------------
% Preprocessing
% -----------------------------------------------------------------------

[A0,dummy,dummy,prm,iprm] = au_pre(A,[],[]);
```

```
                        % au_pre realizes a preprocessing:
                        % - The columns and rows of A are simultaneously
                        %   reordered to reduce the bandwidth. The result
                        %   is A0. prm and iprm are the corresponding
                        %   permutation and inverse permutation.
                        % - Since we consider only the matrix A but not
                        %   a dynamical system, we use [] as 2nd and 3rd
                        %   input parameter. dummy = [] is returned.

figure(1), hold off, clf
spy(A)
title('Before preprocessing: nonzero pattern of A.')

figure(2), hold off, clf
spy(A0)
title('After preprocessing: nonzero pattern of A_0.')

disp('Verification (test_1, test_2, ... should be small):')


% ----------------------------------------------------------------------
% Multiplication of matrix A0 with X0
% ----------------------------------------------------------------------

au_m_i(A0);   % initialization and generation of data needed for matrix
              % multiplications with A0 and A0'

Y0 = au_m('N',X0);   % compute Y0 = A0*X0
T0 = A0*X0;
test_1 = norm(Y0-T0,'fro')


% ----------------------------------------------------------------------
% Multiplication of (transposed) matrix A0' with X0
% ----------------------------------------------------------------------

Y0 = au_m('T',X0);   % compute Y0 = A0'*X0
T0 = A0'*X0;
test_2 = norm(Y0-T0,'fro')


% ----------------------------------------------------------------------
% Solution of system of linear equations with A0
% ----------------------------------------------------------------------

au_l_i;   % initialization for solving systems with A0 and A0'

Y0 = au_l('N',X0);   % solve A0*Y0 = X0
test_3 = norm(A0*Y0-X0,'fro')
```

```
% -----------------------------------------------------------------------
% Solution of (transposed) system of linear equations with A0'
% -----------------------------------------------------------------------

Y0 = au_l('T',X0);   % solve A0'*Y0 = X0
test_4 = norm(A0'*Y0-X0,'fro')


% -----------------------------------------------------------------------
% Solve shifted systems of linear equations, i.e.
% solve (A0+p(i)*I)*Y0 = X0.
% -----------------------------------------------------------------------

disp('Shift parameters:')
p = [ -1; -2+3*j; -2-3*j ]

au_s_i(p)   % initialization for solution of shifted systems of linear
            % equations with system matrix A0+p(i)*I and A0'+p(i)*I
            % (i = 1,...,3)

Y0 = au_s('N',X0,1);
test_5 = norm(A0*Y0+p(1)*Y0-X0,'fro')

Y0 = au_s('N',X0,2);
test_6 = norm(A0*Y0+p(2)*Y0-X0,'fro')

Y0 = au_s('N',X0,3);
test_7 = norm(A0*Y0+p(3)*Y0-X0,'fro')


% -----------------------------------------------------------------------
% Solve (transposed) shifted systems of linear equations, i.e.
% solve (A0'+p(i)*I)*Y0 = X0.
% -----------------------------------------------------------------------

Y0 = au_s('T',X0,1);
test_8 = norm(A0'*Y0+p(1)*Y0-X0,'fro')

Y0 = au_s('T',X0,2);
test_9 = norm(A0'*Y0+p(2)*Y0-X0,'fro')

Y0 = au_s('T',X0,3);
test_10 = norm(A0'*Y0+p(3)*Y0-X0,'fro')


% -----------------------------------------------------------------------
% Postprocessing
% -----------------------------------------------------------------------
```

```
%
% There is no postprocessing.


% -----------------------------------------------------------------------
% Destroy global data structures (clear "hidden" global variables)
% -----------------------------------------------------------------------

au_m_d;   % clear global variables initialized by au_m_i
au_l_d;   % clear global variables initialized by au_l_i
au_s_d(p);   % clear global variables initialized by au_s_i
```

C.1.2 Demo program demo_u2:

```
%
%  REALIZATION OF BASIC MATRIX OPERATIONS BY USER-SUPPLIED
%  FUNCTIONS 'au_qmr_ilu_*'
%
%
%  This demo program shows how the user-supplied functions 'au_qmr_ilu_*'
%  work.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example, we use a simple FDM-semidiscretized PDE problem
% (an instationary convection-diffusion heat equation on the unit square
% with homogeneous 1st kind boundary conditions).
% We reorder the columns and rows of the resulting stiffness matrix by
% a random permutation to generate a "bad" nonzero pattern.

n0 = 30;   % n0 = number of grid points in either space direction;
           % n = n0^2 is the problem dimension!
           % (Change n0 to generate problems of different size.)

A = fdm_2d_matrix(n0,'10*x','100*y','0');   % Note: A is unsymmetric.
[dummy,pm] = sort(randn(n0^2,1));   % generate a random permutation
A = A(pm,pm);

disp('Problem dimensions:')

n = size(A,1)   % problem order

t = 3; j = sqrt(-1);            % generate complex matrix X, which
X = randn(n,t)+j*randn(n,t);    % contains much fewer columns than rows
```

```
% ----------------------------------------------------------------------
% Preprocessing
% ----------------------------------------------------------------------
%
% There is no preprocessing.

figure(1), hold off, clf
spy(A)
title('Nonzero pattern of A.')

disp('Verification (test_1, test_2, ... should be small):')


% ----------------------------------------------------------------------
% Multiplication of matrix A with X
% ----------------------------------------------------------------------

mc = 'M',    % optimize for memory, i.e., preconditioners will be
             % generated right before any QMR run
max_it_qmr = 50,   % maximal number of QMR iteration steps
tol_qmr = 1e-15,   % normalized residual norm for stopping the QMR
                   % iterations
tol_ilu = 1e-2,   % dropping tolerance for generating ILU preconditioners
info_qmr = 2,   % amount of displayed information on performance of
                % ILU-QMR iteration

disp('NOTE: The USFs will return a warning message, when they fail to')
disp('      fulfill the stopping criteria for the ILU-QMR iteration.')
disp('      Also, the attained accuracy is displayed, which allows the')
disp('      user to judge, whether the results are still acceptable or')
disp('      not.')
pause(5)

au_qmr_ilu_m_i(A,mc,max_it_qmr,tol_qmr,tol_ilu,info_qmr);
               % initialization and generation of data needed for matrix
               % multiplications with A

Y = au_qmr_ilu_m('N',X);   % compute Y = A*X (here, of course, QMR is
                           % not involved)
T = A*X;
test_1 = norm(Y-T,'fro')


% ----------------------------------------------------------------------
% Multiplication of (transposed) matrix A' with X
% ----------------------------------------------------------------------

Y = au_qmr_ilu_m('T',X);   % compute Y = A'*X (here, of course, QMR is
                           % not involved)
T = A'*X;
```

```
test_2 = norm(Y-T,'fro')


% -----------------------------------------------------------------------
% Solution of system of linear equations with A
% -----------------------------------------------------------------------

au_l_i;   % initialization for solving systems with A and A'

Y = au_qmr_ilu_l('N',X);   % solve A*Y = X
test_3 = norm(A*Y-X,'fro')


% -----------------------------------------------------------------------
% Solution of (transposed) system of linear equations with A'
% -----------------------------------------------------------------------

Y = au_qmr_ilu_l('T',X);   % solve A'*Y = X
test_4 = norm(A'*Y-X,'fro')


% -----------------------------------------------------------------------
% Solve shifted systems of linear equations, i.e.
% solve (A+p(i)*I)*Y = X.
% -----------------------------------------------------------------------

disp('Shift parameters:')
p = [ -1; -2+3*j; -2-3*j ]

au_qmr_ilu_s_i(p)   % initialization for solution of shifted systems of
                    % linear equations with system matrix A+p(i)*I and
                    % A'+p(i)*I (i = 1,...,3)

Y = au_qmr_ilu_s('N',X,1);
test_5 = norm(A*Y+p(1)*Y-X,'fro')

Y = au_qmr_ilu_s('N',X,2);
test_6 = norm(A*Y+p(2)*Y-X,'fro')

Y = au_qmr_ilu_s('N',X,3);
test_7 = norm(A*Y+p(3)*Y-X,'fro')


% -----------------------------------------------------------------------
% Solve (transposed) shifted systems of linear equations, i.e.
% solve (A'+p(i)*I)*Y = X.
% -----------------------------------------------------------------------

Y = au_qmr_ilu_s('T',X,1);
test_8 = norm(A'*Y+p(1)*Y-X,'fro')
```

```
Y = au_qmr_ilu_s('T',X,2);
test_9 = norm(A'*Y+p(2)*Y-X,'fro')

Y = au_qmr_ilu_s('T',X,3);
test_10 = norm(A'*Y+p(3)*Y-X,'fro')


% -----------------------------------------------------------------------
% Postprocessing
% -----------------------------------------------------------------------
%
% There is no postprocessing.


% -----------------------------------------------------------------------
% Destroy global data structures (clear "hidden" global variables)
% -----------------------------------------------------------------------

au_qmr_ilu_m_d;   % clear global variables initialized by au_qmr_ilu_m_i
au_qmr_ilu_l_d;   % clear global variables initialized by au_qmr_ilu_l_i
au_qmr_ilu_s_d(p);   % clear global variables initialized by
                     % au_qmr_ilu_s_i
```

C.1.3 Demo program demo_u3:

```
%
%  REALIZATION OF BASIC MATRIX OPERATIONS BY USER-SUPPLIED
%  FUNCTIONS 'munu_*'
%
%
%  This demo program shows how the user-supplied functions 'munu_*' work.
%  In this particular case, we consider a generalized dynamical system
%  with symmetric matrices M and N, but we will use non-real shift
%  parameters. For this reason, 'munu_*' is used instead of 'msns_*'.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example, we use an FEM-semidiscretized problem, which leads to
% a generalized system where M (the mass matrix) and N (the negative
% stiffness matrix) are sparse, symmetric, and definite.

load rail821   % load the matrices M N

disp('Problem dimensions:')
```

```
n = size(M,1)   % problem order

t = 3; j = sqrt(-1);            % generate complex matrix X0, which
X0 = randn(n,t)+j*randn(n,t);   % contains much fewer columns than rows


% ----------------------------------------------------------------------
% Preprocessing
% ----------------------------------------------------------------------

[M0,ML,MU,N0,dummy,dummy,prm,iprm] = munu_pre(M,N,[],[]);

                        % munu_pre realizes a preprocessing:
                        % - The columns and rows of M and N are
                        %   simultaneously reordered to reduce the
                        %   bandwidth. The result is M0 and N0. prm and
                        %   iprm are the corresponding permutation and
                        %   inverse permutation.
                        % - A Cholesky factorization of M is computed,
                        %   so that the implicit system matrix A0 is
                        %   A0 = inv(ML)*N0*inv(MU).
                        % - Since we consider only the matrix A0 but not
                        %   a dynamical system, we use [] as 2nd and 3rd
                        %   input parameter. dummy = [] is returned.

figure(1), hold off, clf
spy(M)
title('Before preprocessing: nonzero pattern of M. That of N is the same.')

figure(2), hold off, clf
spy(M0)
title('After preprocessing: nonzero pattern of M_0. That of N_0 is the same.')

disp('Verification (test_1, test_2, ... should be small):')


% ----------------------------------------------------------------------
% Multiplication of matrix A0 with X0
% ----------------------------------------------------------------------

munu_m_i(M0,ML,MU,N0)   % initialization and generation of data needed
                        % for matrix multiplications with A0

Y0 = munu_m('N',X0);   % compute Y0 = A0*X0
T0 = ML\(N0*(MU\X0));
test_1 = norm(Y0-T0,'fro')


% ----------------------------------------------------------------------
% Solution of system of linear equations with A0
```

```
% ----------------------------------------------------------------------

munu_l_i;   % initialization for solving systems solve with A0

Y0 = munu_l('N',X0);   % solve A0*Y0 = X0
test_2 = norm(ML\(N0*(MU\Y0))-X0,'fro')


% ----------------------------------------------------------------------
% Solve shifted systems of linear equations, i.e.
% solve (A0+p(i)*I)*Y0 = X0.
% ----------------------------------------------------------------------

disp('Shift parameters:')
p = [ -1; -2+3*j; -2-3*j ]

munu_s_i(p)   % initialization for solution of shifted systems of linear
              % equations with system matrix A0+p(i)*I  (i = 1,...,3)

Y0 = munu_s('N',X0,1);
test_3 = norm(ML\(N0*(MU\Y0))+p(1)*Y0-X0,'fro')

Y0 = munu_s('N',X0,2);
test_4 = norm(ML\(N0*(MU\Y0))+p(2)*Y0-X0,'fro')

Y0 = munu_s('N',X0,3);
test_5 = norm(ML\(N0*(MU\Y0))+p(3)*Y0-X0,'fro')


% ----------------------------------------------------------------------
% Postprocessing
% ----------------------------------------------------------------------
%
% There is no postprocessing.


% ----------------------------------------------------------------------
% Destroy global data structures (clear "hidden" global variables)
% ----------------------------------------------------------------------

munu_m_d;   % clear global variables initialized by munu_m_i
munu_l_d;   % clear global variables initialized by munu_l_i
munu_s_d(p);   % clear global variables initialized by munu_s_i
```

C.2 Demo program for LRCF-ADI iteration and algorithm for computing ADI parameters

C.2.1 Demo program demo_l1

```
%
%  SOLUTION OF LYAPUNOV EQUATION BY THE LRCF-ADI METHOD (AND GENERATION
%  OF ADI PARAMETERS)
%
%  This demo program shows how the routines 'lp_para' (computation of
%  ADI shift parameters) and 'lp_lradi' (LRCF-ADI iteration for the
%  solution of the Lyapunov equation F*X+X*F'=-G*G') work.  Also, the
%  use of user-supplied functions is demonstrated.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example, we use a simple FDM-semidiscretized PDE problem
% (an instationary convection-diffusion heat equation on the unit square
% with homogeneous 1st kind boundary conditions).

n0 = 20;   % n0 = number of grid points in either space direction;
           % n = n0^2 is the problem dimension!
           % (Change n0 to generate problems of different size.)

F = fdm_2d_matrix(n0,'10*x','100*y','0');
G = fdm_2d_vector(n0,'.1<x<=.3');

disp('Problem dimensions:')

n = size(G,1)   % problem order
m = size(G,2)   % number of columns in factor of r.h.s. (mostly, the rank
                % of the r.h.s.)


% -----------------------------------------------------------------------
% Initialization/generation of data structures used in user-supplied
% functions and computation of ADI shift parameters
% -----------------------------------------------------------------------
%
% Note that the routines 'au_m_i', 'au_l_i', and 'au_s_i' create global
% variables, which contain the data that is needed for the efficient
% realization of basic matrix operations with F (multiplications,
% solution of systems of linear equations, solution of shifted systems
% of linear equations).

name = 'au';   % basis name of user-supplied functions applied to the
               % problem with nonsymmetric F. Note: in this class of
               % user-supplied functions, sparse LU factorizations are
               % applied to solve (shifted) systems of linear equations.
```

```
f = flops;
[F0,G0,dummy,prm,iprm] = au_pre(F,G,[]);   % preprocessing (reordering
                                           % for bandwidth reduction)
                                           % Note the dummy parameter,
                                           % which will be set to [] on
                                           % exit.

au_m_i(F0);   % initialization for matrix multiplications with F0

au_l_i;   % initialization for solving systems with F0 (This is needed in
          % the Arnoldi algorithm w.r.t. inv(F0). The Arnoldi algorithm
          % is part of the algorithm in 'lp_para'.)

disp('Parameters for heuristic algorithm which computes ADI parameters:')
l0 = 15   % desired number of distinct shift parameters
kp = 50   % number of steps of Arnoldi process w.r.t. F0
km = 25   % number of steps of Arnoldi process w.r.t. inv(F0)

b0 = ones(n,1);   % This is just one way to choose the Arnoldi start
                  % vector.

p = lp_para(name,[],[],l0,kp,km,b0);   % computation of ADI shift
                                       % parameters

disp('Actual number of ADI shift parameters:');
l = length(p)

disp('ADI shift parameters:');
p

au_s_i(p)   % initialization for shifted systems of linear equations with
            % F0+p(i)*I  (i = 1,...,l)

disp('Flops required for a-priori computations:')
a_priori_flops = flops-f

% -----------------------------------------------------------------------
% Solution of Lyapunov equation F*X+X*F' = -G*G' (or, more precisely,
% the transformed equation F0*X0+X0*F0' = -G0*G0') by LRCF-ADI iteration
% -----------------------------------------------------------------------
%
% The approximate solution is given by the low rank Cholesky factor Z0,
% i.e., Z0*Z0' is approximately X0
%
% The stopping criteria are chosen, such that the iteration is stopped
% shortly after the residual curve stagnates. This requires the sometimes
% expensive computation of the residual norms. (If you want to avoid
% this, you might choose max_it = 500 (large value), min_res = 0
% ("avoided"), with_rs = 'N' ("avoided"), min_in = 1e-12 ("activated").)
```

```
disp('Parameters for stopping criteria in LRCF-ADI iteration:')
max_it = 500   % max. number of iteration steps (here, a very large
               % value, which will probably not stop the iteration)
min_res = 0   % tolerance for normalized residual norm (criterion
              % is "avoided")
with_rs = 'S'   % stopping criterion "stagnation of the normalized
                % residual norms" activated
min_in = 0   % threshold for smallness of values ||V_i||_F (criterion
             % is "avoided")

disp('Further input parameters of the routine ''lp_lradi'':');
tp = 'B'   % type of Lyapunov equation to be solved
           % (here, F0*X0+X0*F0'=-G0*G0')
zk = 'Z'   % compute Z0 or generate Z0*Z0'*K0 (here, Z0)
rc = 'C'   % compute possibly complex Z0 or demand for real Z0 (here,
           % a complex matrix Z0 may be returned)
Kf = [], Bf = []   % feedback matrices (these parameters are only used
                   % in the Newton iteration)
info = 3    % information level (here, maximal amount of information is
            % provided during the LRCF-ADI iteration)

figure(1), hold off; clf;   % (lp_lradi will plot residual history.)

[Z0,flag,res,flp] = lp_lradi(tp,zk,rc,name,Bf,Kf,G0,p,max_it,min_res,...
                             with_rs,min_in,info);

                    % Note that in lp_lradi the transformed r.h.s.
                    % matrix G0 must be used.

disp('Termination flag of the routine ''lp_lradi'':')
flag
disp('Internally computed normalized residual norm (of final iterate):');
final_nrn = res(end)
disp('Number of flops required for the whole iteration');
disp('(without a-priori computation and computation of residual norm):');
lrcf_adi_flops = flp(end)


% -----------------------------------------------------------------------
% Postprocessing, destroy global data structures
% -----------------------------------------------------------------------
%
% NOTE: The matrices F and G have been reordered in the preprocessing
% step (''au_pre'') resulting in F0 and G0. This means that the rows of the
% matrix Z0 must be re-reordered in a postprocessing step to obtain the
% solution to the original Lyapunov equation!

Z = au_pst(Z0,iprm);
```

```
au_m_d;   % clear global variables initialized by au_m_i
au_l_d;   % clear global variables initialized by au_l_i
au_s_d(p);   % clear global variables initialized by au_s_i

disp('Size of Z:');
size_Z = size(Z)
disp('Is Z real ( 0 = no, 1 = yes )?')
is_real = ~any(any(imag(Z)))


% -----------------------------------------------------------------------
% Verify the result
% -----------------------------------------------------------------------
%
% Note that this is only an "illustrative" way of verifying the accuracy
% by computing the (normalized) residual norm. A more practical (because
% less expensive) way is evaluating the residual norm by means of the
% routine 'lp_nrm' (Must be applied before postprocessing!), if the
% residual norms have not been generated during the iteration.

disp('The attained residual norm:')
res_norm = norm(F*Z*Z'+Z*Z'*F'+G*G','fro')
disp('The attained normalized residual norm:')
normal_res_norm = res_norm/norm(G*G','fro')
```

C.2.2 Results and remarks

In demo_l1 the LRCF-ADI iteration is stopped by the stopping criterion related to the parameter with_rs (stagnation of the residual norm). The number of iteration steps is 43. Hence, the low rank Cholesky factor Z is a 400×43 matrix. It is not real. The attained normalized residual norm is approximatelly $1.4 \cdot 10^{-15}$. About $4 \cdot 10^{6}$ flops were needed for the computations (without computing the residual norms). Figure 12 shows the residual history.

C.3 Demo programs for model reduction algorithms

C.3.1 Demo program demo_m1

```
%
%  MODEL REDUCTION BY THE ALGORITHMS LRSRM AND DSPMR. THE GOAL IS TO
%  GENERATE A REDUCED SYSTEM OF VERY SMALL ORDER.
%
%  This demo program shows how the model reduction routines 'lp_lrsrm'
%  and 'lp_dspmr' work. Also, the use of 'lp_lradi', supplementary
%  routines, and user-supplied functions is demonstrated.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
```

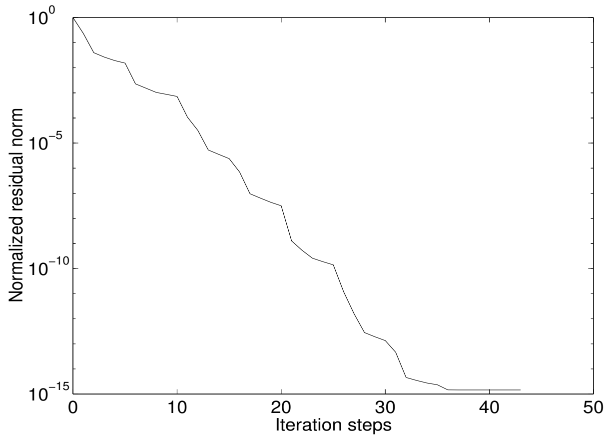

Figure 12: Residual history for the LRCF-ADI iteration in `demo_l1`.

```
%
% This is an artificial test problem of a system, whose Bode plot shows
% "spires".

A = sparse(408,408); B = ones(408,1); C = ones(1,408);
A(1:2,1:2) = [-.01 -200; 200 .001];
A(3:4,3:4) = [-.2 -300; 300 -.1];
A(5:6,5:6) = [-.02 -500; 500 0];
A(7:8,7:8) = [-.01 -520; 520 -.01];
A(9:408,9:408) = spdiags(-(1:400)',0,400,400);

disp('Problem dimensions:')

n = size(A,1)   % problem order (number of states)
m = size(B,2)   % number of inputs
q = size(C,1)   % number of outputs

% ----------------------------------------------------------------------
% Initialization/generation of data structures used in user-supplied
% functions and computation of ADI shift parameters
% ----------------------------------------------------------------------
%
% See 'demo_u1', 'demo_u2', 'demo_u3', and 'demo_l1' for more detailed
% comments.
%
% Note that A is a tridiagonal matrix. No preprocessing needs to be done.

name = 'au';
au_m_i(A);   % initialization for multiplication with A
au_l_i;   % initialization for solving systems with A

disp('Parameters for heuristic algorithm which computes ADI parameters:')
l0 = 10   % desired number of distinct shift parameters
```

```
kp = 30   % number of steps of Arnoldi process w.r.t. A
km = 15   % number of steps of Arnoldi process w.r.t. inv(A)

b0 = ones(n,1);   % This is just one way to choose the Arnoldi start
                  % vector.

p = lp_para(name,[],[],l0,kp,km,b0);   % computation of ADI shift
                                       % parameters

disp('Actual number of ADI shift parameters:');
l = length(p)

disp('ADI shift parameters:');
p

au_s_i(p)   % initialization for shifted systems of linear equations
            % with A+p(i)*I  (i = 1,...,l)


% -----------------------------------------------------------------------
% Solution of Lyapunov equations A*X+X*A' = -B*B' and
% A'*X+X*A = -C'*C
% -----------------------------------------------------------------------

disp('Parameters for stopping criteria in LRCF-ADI iteration:')
max_it = 20   % (will stop the iteration)
min_res = 1e-100   % (avoided, but the residual history is shown)
with_rs = 'N'   % (avoided)
min_in = 0   % (avoided)

zk = 'Z';
rc = 'C';
Bf = [];
Kf = [];
info = 3;

disp('... solving A*XB+XB*A'' = - B*B''...');
tp = 'B';

figure(1), hold off; clf;    % (lp_lradi will plot residual history.)

[ZB,flag_B] = lp_lradi(tp,zk,rc,name,Bf,Kf,B,p,max_it,min_res,...
              with_rs,min_in,info);
                                       % compute ZB

title('LRCF-ADI for CALE  AX_{B}+X_{B}A^T = -BB^T')
disp('Termination flag:')
flag_B
disp('Size of ZB:');
size_ZB = size(ZB)
```

```
disp('... solving A''*XC+XC*A = - C''*C...');
tp = 'C';

figure(2), hold off; clf;    % (lp_lradi will plot residual history.)

[ZC,flag_C] = lp_lradi(tp,zk,rc,name,Bf,Kf,C,p,max_it,min_res,...
              with_rs,min_in,info);
                                   % compute ZC

title('LRCF-ADI for CALE  A^T X_{C} + X_{C} A_ = -C^TC')
disp('Termination flag:')
flag_C
disp('Size of ZC:');
size_ZC = size(ZC)


% -----------------------------------------------------------------------
% Plot the transfer function of the system for a certain frequency range
% -----------------------------------------------------------------------

disp('... computing transfer function of original system ...');

freq = lp_lgfrq(100,1000,200);    % generate a set of 200 "frequency
                                  % sampling points" in the interval
                                  % [100,1000].
G = lp_trfia(freq,A,B,C,[],[]);   % compute "transfer function sample"
                                  % for these frequency points
nrm_G = lp_gnorm(G,m,q);   % compute norms of the "transfer function
                           % sample" for these frequency points

figure(3); hold off; clf;
loglog(freq,nrm_G,'k:');
xlabel('\omega');
ylabel('Magnitude');
t_text = 'Bode plots:  dotted: ||G||';
title(t_text);
pause(1)


% -----------------------------------------------------------------------
% Generate reduced systems
% -----------------------------------------------------------------------

disp('Parameters for model reduction:')
max_ord = 10   % (This parameter determines the reduced order.)
tol = 0   % (avoided)


disp('... computing reduced system by LRSRM ...');
```

```
[Ars,Brs,Crs] = lp_lrsrm(name,B,C,ZB,ZC,max_ord,tol);   % run LRSRM

disp('Reduced order:')
disp(length(Ars))

Grs = lp_trfia(freq,Ars,Brs,Crs,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_Grs = lp_gnorm(Grs,m,q);   % compute norm transfer function samples
                               % of reduced system
figure(3); hold on
loglog(freq,nrm_Grs,'r-');
t_text = [t_text, ',    solid: ||G_{LRSRM}||'];
title(t_text); pause(1)


disp('... computing reduced system by DSPMR ...');
[Ard,Brd,Crd] = lp_dspmr(name,B,C,ZB,ZC,max_ord,tol);   % run DSPMR

disp('Reduced order:')
disp(length(Ard))

Grd = lp_trfia(freq,Ard,Brd,Crd,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_Grd = lp_gnorm(Grd,m,q);   % compute norm transfer function samples
                               % of reduced system
figure(3); hold on
loglog(freq,nrm_Grd,'b--');
t_text = [t_text, ',    dashed: ||G_{DSPMR}||'];
title(t_text); pause(1)


% -----------------------------------------------------------------------
% Destroy global data structures
% -----------------------------------------------------------------------

au_m_d;
au_l_d;
au_s_d(p);
```

C.3.2 Results and remarks

In the demo program `demo_m1` we use very inaccurate Gramians. The normalized residual norms are only $\approx 7.8 \cdot 10^{-2}$. The reduced order of the systems delivered by LRSRM and DSPMR is as low as 10. The result of the demo program is shown in Figure 13. There simultaneous Bode magnitude plots of the original system and and both reduced systems are shown.

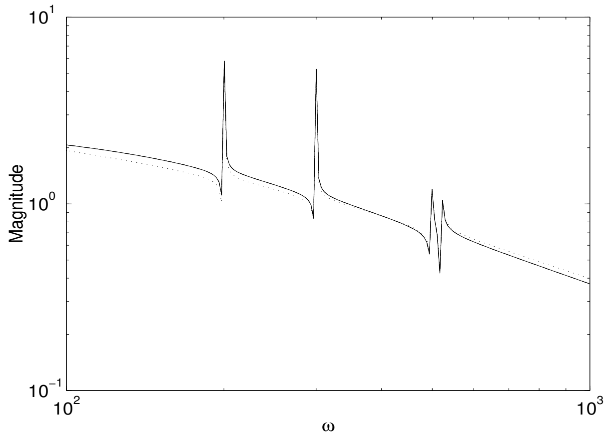

Figure 13: Simultaneous Bode magnitude plots of original system (dotted), reduced system by LRSRM, and reduced system by DSPMR generated by the demo program demo_m1. Both Bode plots of both reduced systems are almost identical and shown as solid line.

C.3.3 Demo program demo_m2

```
%
%  MODEL REDUCTION BY THE ALGORITHMS LRSRM AND DSPMR. THE GOAL IS TO
%  GENERATE A "NUMERICALLY MINIMAL REALIZATION" OF THE GIVEN SYSTEM
%  AS WELL AS A REDUCED SYSTEM OF RELATIVELY SMALL ORDER.
%
%  This demo program shows how the model reduction routines 'lp_lrsrm'
%  and 'lp_dspmr' work. Also, the use of 'lp_lradi', supplementary
%  routines, and user-supplied functions is demonstrated.

% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example, we use an FEM-semidiscretized problem, which leads to
% a generalized system where M (the mass matrix) and N (the negative
% stiffness matrix) are sparse, symmetric, and definite.

load rail821   % load the matrices M N Btilde Ctilde of the generalized
               % system

%load rail3113   % Uncomment this to get an example of larger order.

disp('Problem dimensions:')

n = size(M,1)   % problem order (number of states)
m = size(Btilde,2)   % number of inputs
q = size(Ctilde,1)   % number of outputs

% -----------------------------------------------------------------------
```

```
% Initialization/generation of data structures used in user-supplied
% functions and computation of ADI shift parameters
% -----------------------------------------------------------------------
%
% See 'demo_u1', 'demo_u2', 'demo_u3', and 'demo_l1' for more detailed
% comments.

name = 'msns';
[M0,MU,N0,B0,C0,prm,iprm] = msns_pre(M,N,Btilde,Ctilde);  % preprocessing
msns_m_i(M0,MU,N0);   % initialization for multiplication with A0
msns_l_i;   % initialization for solving systems with A0

disp('Parameters for heuristic algorithm which computes ADI parameters:')
l0 = 20   % desired number of distinct shift parameters
kp = 50   % number of steps of Arnoldi process w.r.t. A0
km = 25   % number of steps of Arnoldi process w.r.t. inv(A0)

b0 = ones(n,1);   % This is just one way to choose the Arnoldi start
                  % vector.

p = lp_para(name,[],[],l0,kp,km,b0);   % computation of ADI shift
                                       % parameters

disp('Actual number of ADI shift parameters:');
l = length(p)

disp('ADI shift parameters:');
p

msns_s_i(p)   % initialization for shifted systems of linear equations
              % with A0+p(i)*I  (i = 1,...,l)


% -----------------------------------------------------------------------
% Solution of Lyapunov equations A0*XB0+XB0*A0' = -B0*B0' and
% A0'*XC0+XC0*A0 = -C0'*C0
% -----------------------------------------------------------------------

disp('Parameters for stopping criteria in LRCF-ADI iteration:')
max_it = 200   % (large value)
min_res = 0   % (avoided)
with_rs = 'S'   % ("activated")
min_in = 0   % (avoided)

zk = 'Z';
rc = 'C';
Bf = [];
Kf = [];
info = 3;
```

```
disp('... solving A0*XB0+XB0*A0'' = - B0*B0''...');
tp = 'B';
figure(1), hold off; clf;
[ZB0,flag_B] = lp_lradi(tp,zk,rc,name,Bf,Kf,B0,p,max_it,min_res,...
               with_rs,min_in,info);
                                       % compute ZB0

title('LRCF-ADI for CALE  A_0X_{B0}+X_{B0}A_0^T = -B_0B_0^T')
disp('Termination flag:')
flag_B
disp('Size of ZB0:');
size_ZB0 = size(ZB0)

disp('... solving A0''*XC0+XC0*A0 = - C0''*C0...');
tp = 'C';
figure(2), hold off; clf;
[ZC0,flag_C] = lp_lradi(tp,zk,rc,name,Bf,Kf,C0,p,max_it,min_res,...
               with_rs,min_in,info);
                                       % compute ZC0

title('LRCF-ADI for CALE  A_0^T X_{C0} + X_{C0} A_0 = -C_0^TC_0')
disp('Termination flag:')
flag_C
disp('Size of ZC0:');
size_ZC0 = size(ZC0)


% ---------------------------------------------------------------------
% Plot the transfer function of the system for a certain frequency range
% ---------------------------------------------------------------------

disp('... computing transfer function of original system ...');

freq = lp_lgfrq(1e-10,1e10,200);   % generate a set of 200 "frequency
                                   % sampling points" in the interval
                                   % [10^-10,10^+10].
G = lp_trfia(freq,N,Btilde,Ctilde,[],M);   % compute "transfer function
                                           % sample" for these frequency
                                           % points
nrm_G = lp_gnorm(G,m,q);   % compute norms of the "transfer function
                           % sample" for these frequency points

figure(3); hold off; clf;
loglog(freq,nrm_G,'k:');
xlabel('\omega');
ylabel('Magnitude');
t_text = 'dotted: ||G||';
title(t_text);
pause(1)
```

```
% ----------------------------------------------------------------------
% Generate reduced systems of high accuracy and possibly high order
% ----------------------------------------------------------------------

disp(' ')
disp('Generate reduced systems of high accuracy and possibly high order')
disp('-----------------------------------------------------------------')

disp('Parameters for model reduction:')
max_ord = []   % (avoided)
tol = 1e-14   % (This criterion determines the reduced order. The very
              % small value is chosen to generate a "numerically minimal
              % realization".)


disp('... computing reduced system by LRSRM ...');
[Ars,Brs,Crs] = lp_lrsrm(name,B0,C0,ZB0,ZC0,max_ord,tol);   % run LRSRM

disp('Reduced order:')
disp(length(Ars))

Grs = lp_trfia(freq,Ars,Brs,Crs,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_dGrs = lp_gnorm(G-Grs,m,q);   % compute norm of DIFFERENCE of
                                  % transfer function samples of original
                                  % and reduced system.
figure(3); hold on
loglog(freq,nrm_dGrs,'r-');
t_text = [t_text, ',    solid: ||G-G_{DSPMR}||'];
title(t_text); pause(1)


disp('... computing reduced system by DSPMR ...');
[Ard,Brd,Crd] = lp_dspmr(name,B0,C0,ZB0,ZC0,max_ord,tol);   % run DSPMR

disp('Reduced order:')
disp(length(Ard))

Grd = lp_trfia(freq,Ard,Brd,Crd,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_dGrd = lp_gnorm(G-Grd,m,q);   % compute norm of DIFFERENCE of
                                  % transfer function samples of original
                                  % and reduced system.
figure(3); hold on
loglog(freq,nrm_dGrd,'b--'); pause(1)
t_text = [t_text, ',    solid: ||G-G_{LRSRM}||'];
title(t_text); pause(1)
```

```
% ----------------------------------------------------------------------
% Generate reduced systems of low order
% ----------------------------------------------------------------------

disp(' ')
disp('Generate reduced systems of low order')
disp('-------------------------------------')

disp('Parameters for model reduction:')
max_ord = 25   % (This criterion determines the reduced order.)
tol = 0    % (avoided)


disp('... computing reduced system by LRSRM ...');
[Ars,Brs,Crs] = lp_lrsrm(name,B0,C0,ZB0,ZC0,max_ord,tol);   % run LRSRM

disp('Reduced order:')
disp(length(Ars))

Grs = lp_trfia(freq,Ars,Brs,Crs,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_dGrs = lp_gnorm(G-Grs,m,q);   % compute norm of DIFFERENCE of
                                  % transfer function samples of original
                                  % and reduced system.
figure(3); hold on
loglog(freq,nrm_dGrs,'r-');


disp('... computing reduced system by DSPMR ...');
[Ard,Brd,Crd] = lp_dspmr(name,B0,C0,ZB0,ZC0,max_ord,tol);   % run DSPMR

disp('Reduced order:')
disp(length(Ard))

Grd = lp_trfia(freq,Ard,Brd,Crd,[],[]);   % compute "transfer function
                                          % sample" for reduced system
nrm_dGrd = lp_gnorm(G-Grd,m,q);   % compute norm of DIFFERENCE of
                                  % transfer function samples of original
                                  % and reduced system.
figure(3); hold on
loglog(freq,nrm_dGrd,'b--');


% ----------------------------------------------------------------------
% Destroy global data structures
% ----------------------------------------------------------------------

msns_m_d;
msns_l_d;
msns_s_d(p);
```

C.3.4 Results and remarks

In contrast to `demo_m1`, we use very accurate Gramians in the program `demo_m2`. The normalized residual norms for Z_{B0} and Z_{C0} are $\approx 7.0 \cdot 10^{-14}$ and $\approx 1.7 \cdot 10^{-14}$, respectively. Using these low rank Cholesky factors of the Gramians we generate two pairs of reduced systems by LRSRM and DSPMR. In the first run we attempt to generate a pair of reduced systems, which are very accurate. Indeed, Figure 14 shows that the approximation error $\|G(\jmath\omega) - \hat{G}(\jmath\omega)\|$ for both reduced systems is very small compared to the Bode magnitude function of the original system. We allow the reduced order to be relatively large by choosing a very small value for `tol`. These orders are 118 for LRSRM and 208 for DSPMR. LRSRM and DSPMR deliver almost identical results w.r.t. the approximation error, but LRSRM delivers a system of lower order. In the second run, we use fixed reduced orders $k = 25$. We still obtain relatively small approximation errors; see Figure 14. Here, the result by LRSRM is again better than that by DSPMR. Note that we show approximation errors in Figure 14 as opposed to simultaneous Bode plots in Figure 13. The reduced systems generated by `demo_m2` are so accurate that identical curves would be displayed in simultaneous Bode magnitude plots.

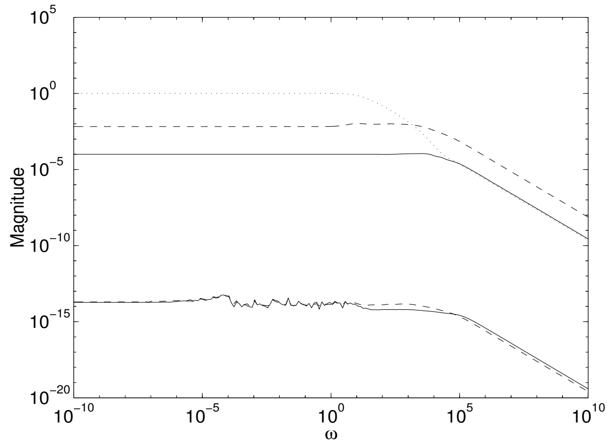

Figure 14: Results of `demo_m2`. The dotted line is the Bode magnitude plot of the original system, i.e, the function $\|G(\jmath\omega)\|$. The solid and dashed lines are the **approximation errors**, i.e., the functions $\|G(\jmath\omega) - \hat{G}(\jmath\omega)\|$, for LRSRM and DSPMR, respectively. The lower two curves correspond to the first run (highly accurate reduced systems, "numerically minimal realization") and the upper two to the second run (low reduced order).

C.4 Demo program for algorithms for Riccati equations and linear-quadratic optimal problems

C.4.1 Demo program `demo_r1`

```
%
% SOLUTION OF RICCATI EQUATION BY LRCF-NM AND SOLUTION OF LINEAR-
% QUADRATIC OPTIMAL CONTROL PROBLEM BY LRCF-NM-I
%
% This demo program shows how both modes (i.e., the one for LRCF-NM and
% the one for LRCF-NM-I) work. Also, the use of user-supplied functions
% is demonstrated in this context.
```

```
% -----------------------------------------------------------------------
% Generate test problem
% -----------------------------------------------------------------------
%
% As test example, we use a simple FDM-semidiscretized PDE problem
% (an instationary heat equation on the unit square with homogeneous 1st
% kind boundary conditions).
%
% Note that the negative stiffness matrix A is symmetric.

n0 = 20;   % n0 = number of grid points in either space direction;
           % n = n0^2 is the problem dimension!
           % (Change n0 to generate problems of different size.)

A = fdm_2d_matrix(n0,'0','0','0');
B = fdm_2d_vector(n0,'.1<x<=.3');
C = (fdm_2d_vector(n0,'.7<x<=.9'))';

Q0 = 10   % Q = Q0*Q0' = 100
R0 = 1   % R = R0*R0' = 1

K_in = [];   % Initial feedback K is zero (Note that A is stable).

disp('Problem dimensions:')

n = size(A,1)   % problem order (number of states)
m = size(B,2)   % number of inputs
q = size(C,1)   % number of outputs


% -----------------------------------------------------------------------
% Initialization/generation of data structures used in user-supplied
% functions
% -----------------------------------------------------------------------
%
% Note that we use routines 'au_*' rather than the routines 'as_*',
% although A is symmetric. This is because ADI shift parameters w.r.t.
% the nonsymmetric closed loop matrix A-B*K' (generated in the routine
% lp_lrnm) might be not real. The routines 'as_*' are restricted to
% problems, where the shift parameters are real.

name = 'au';

[A0,B0,C0,prm,iprm] = au_pre(A,B,C);   % preprocessing (reordering for
                                        % bandwidth reduction)

% Note that K_in is zero. Otherwise it needs not be transformed as well.

au_m_i(A0);   % initialization for matrix multiplications with A0
```

```
au_l_i;   % initialization for solving systems with A0 (This is needed in
          % the Arnoldi algorithm w.r.t. inv(A0). The Arnoldi algorithm
          % is part of the algorithm in 'lp_para', which in turn will
          % be invoked in each Newton step in the routine 'lp_lrnm'.)

% Note that 'au_s_i' will be invoked repeatedly in 'lp_lrnm'.

disp('Parameters for heuristic algorithm which computes ADI parameters:')
l0 = 15   % desired number of distinct shift parameters
kp = 50   % number of steps of Arnoldi process w.r.t. A0-B0*K0'
km = 25   % number of steps of Arnoldi process w.r.t. inv(A0-B0*K0')


% -----------------------------------------------------------------------
% Compute LRCF Z0 by LRCF-NM
% -----------------------------------------------------------------------
%
% The approximate solution is given by the low rank Cholesky factor Z0,
% i.e., Z0*Z0' is approximately X0, where X0 is the solution of the
% transformed Riccati equation
%
%   C0'*Q0*Q0'*C0+A0'*X0+X0*A0-X0*B0*inv(R0*R0')*B0'*X0 = 0.
%
% The stopping criteria for both the (outer) Newton iteration and the
% (inner) LRCF-ADI iteration are chosen, such that the iterations are
% stopped shortly after the residual curves stagnate. This requires
% the sometimes expensive computation of the Lyapunov equation
% and Riccati equation residual norms.

disp('Parameters for stopping the (outer) Newton iteration:')
max_it_r = 20   % max. number of iteration steps (here, a very large
                % value, which will probably not stop the iteration)
min_res_r = 0   % tolerance for normalized residual norm (criterion
                % is "avoided")
with_rs_r = 'S'   % stopping criterion "stagnation of the normalized
                  % residual norms" activated
min_ck_r = 0   % stopping criterion "smallness of the RCF" ("avoided")
               % (RCF = relative change of the feedback matrix)
with_ks_r = 'N'   % stopping criterion "stagnation of the RCF"
                  % (criterion is "avoided")

disp('Parameters for stopping the (inner) LRCF-ADI iterations:')
max_it_l = 500   % max. number of iteration steps (here, a very large
                 % value, which will probably not stop the iteration)
min_res_l = 0   % tolerance for normalized residual norm (criterion
                % is "avoided")
with_rs_l = 'S'   % stopping criterion "stagnation of the normalized
                  % residual norms" activated
min_in_l = 0   % threshold for smallness of values ||V_i||_F
```

```
                % (criterion is "avoided")

disp('Further input parameters of the routine ''lp_lrnm'':');
zk = 'Z'   % compute Z0 by LRCF-NM or generate directly
           % K_out = Z0*Z0'*K_in (here, Z0 is computed)
rc = 'C'   % compute possibly complex Z0 or demand for real Z0 (here,
           % a complex matrix Z0 may be returned)
info_r = 3;   % information level for the Newton iteration (here,
              % maximal amount of information is provided)
info_l = 3;   % information level for LRCF-ADI iterations (here,
              % maximal amount of information is provided)

randn('state',0);   % (This measure is taken to make the test results
                    % repeatable. Note that a random vector is involved
                    % into the computation of ADI parameters inside
                    % 'lp_lrnm'.)

[Z0, flag_r, res_r, flp_r, flag_l, its_l, res_l, flp_l] = lp_lrnm(...
zk, rc, name, B0, C0, Q0, R0, K_in, max_it_r, min_res_r, with_rs_r,...
min_ck_r, with_ks_r, info_r, kp, km, l0, max_it_l, min_res_l,...
with_rs_l, min_in_l, info_l );


disp('Results for (outer) Newton iteration in LRCF-NM:')

disp('Termination flag:')
flag_r

disp('Internally computed normalized residual norm (of final iterate):');
final_nrn_r = res_r(end)


disp('Results for (inner) LRCF-ADI iterations in LRCF-NM:')

disp('Termination flags:')
flag_l

disp('Number of LRCF-ADI iteration steps:')
its_l

disp('Internally computed normalized residual norms (of final iterates):');
final_nrn_l = [];
for i = 1:length(its_l)
  final_nrn_l = [final_nrn_l; res_l(its_l(i)+1,i)];
end
final_nrn_l


% ----------------------------------------------------------------------
% Compute (approximately) optimal feedback K0 by LRCF-NM-I
```

```
% ----------------------------------------------------------------------
%
% Here, the matrix K0 that solves the (transformed) linear-quadratic
% optimal control problem is computed by LRCF-NM-I.
%
% The stopping criteria for both the (outer) Newton iteration and the
% (inner) LRCF-ADI iteration are chosen by inexpensive heuristic
% criteria.

disp('Parameters for stopping the (outer) Newton iteration:')
max_it_r = 20   % max. number of iteration steps (here, a very large
                % value, which will probably not stop the iteration)
min_res_r = 0   % tolerance for normalized residual norm (criterion
                % is "avoided")
with_rs_r = 'N'   % stopping criterion "stagnation of the normalized
                  % residual norms" (criterion is "avoided")
min_ck_r = 1e-12   % stopping criterion "smallness of the RCF"
                   % ("activated")
with_ks_r = 'L'   % stopping criterion "stagnation of the RCF"
                  % ("activated")

disp('Parameters for stopping the (inner) LRCF-ADI iterations:')
max_it_l = 500   % max. number of iteration steps (here, a very large
                 % value, which will probably not stop the iteration)
min_res_l = 0   % tolerance for normalized residual norm (criterion
                % is "avoided")
with_rs_l = 'N'   % stopping criterion "stagnation of the normalized
                  % residual norms" (criterion is "avoided")
min_in_l = 1e-12   % threshold for smallness of values in ||V_i||_F
                   % (criterion is "activated")

disp('Further input parameters of the routine ''lp_lradi'':');
zk = 'K'
rc = 'C'
info_r = 3
info_l = 3

randn('state',0);

[K0, flag_r, flp_r, flag_l, its_l, flp_l] = ...
lp_lrnm( zk, name, B0, C0, Q0, R0, K_in, max_it_r, min_ck_r, ...
with_ks_r, info_r, kp, km, l0, max_it_l, min_in_l, info_l );


disp('Results for (outer) Newton iteration in LRCF-NM-I:')

disp('Termination flag:')
flag_r
```

```
disp('Results for (inner) LRCF-ADI iterations in LRCF-NM-I:')

disp('Termination flags:')
flag_l

disp('Number of LRCF-ADI iteration steps:')
its_l


% -----------------------------------------------------------------------
% Postprocessing, destroy global data structures
% -----------------------------------------------------------------------
%
% Note that both the LRCF Z0 and the state feedback K0 must be
% postprocessed in order to attain the results for the original problems.

Z = au_pst(Z0,iprm);
K = au_pst(K0,iprm);

% Note that 'au_s_d' has already been invoked in 'lp_lrnm'.

au_l_d;   % clear global variables initialized by au_l_i
au_m_d;   % clear global variables initialized by au_m_i

disp('Size of Z:');
size_Z = size(Z)
disp('Is Z real ( 0 = no, 1 = yes )?')
Z_is_real = ~any(any(imag(Z)))
disp('Is K real ( 0 = no, 1 = yes )?')
K_is_real = ~any(any(imag(K)))


% -----------------------------------------------------------------------
% Verify the result
% -----------------------------------------------------------------------
%
% Note that this is only an "illustrative" way of verifying the accuracy
% by computing the (normalized) residual norm of the Riccati equation.
% A more practical (because less expensive) way is evaluating the residual
% norm by means of the routine 'lp_rcnrm' (Must be applied before
% postprocessing!), if the residual norms have not been generated during
% the iteration.
%
% In general the result for LRCF-NM-I cannot be verified. However, we
% will compare the delivered feedback K with the feedback matrix computed
% by use of the LRCF Z.

disp('The attained CARE residual norm:')
res_norm = norm(C'*Q0*Q0'*C+A'*Z*Z'+Z*Z'*A-Z*Z'*B*((R0*R0')\B')*Z*Z',...
'fro')
```

```
disp('The attained normalized CARE residual norm:')
normal_res_norm = res_norm/norm(C'*Q0*Q0'*C,'fro')

disp('The normalized deviation of the feedback matrices computed by')
disp('LRCF-NM and LRCF-NM-I (small value --> high accuracy):');
KE = Z*Z'*(B/(R0*R0'));
norm_dev = norm(K-KE,'fro')/max([norm(K,'fro'),norm(KE,'fro')])
```

C.4.2 Results and remarks

In `demo_r1` both LRCF-NM and LRCF-NM-I are applied to the same problem. In the first run, the low rank Cholesky factor Z for the solution of the Riccati equation is computed. Residual based stopping criteria are used. See Figure 15 for the normalized residual norm history. The (approximate) optimal state feedback $K^{(E)}$ (variable `KE`) for the solution of the optimal control problem is computed "explicitely" by $K^{(E)} = ZZ^HBR^{-1}$. In the second run, the optimal control problem is solved directly by LRCF-NM-I, which delivers the approximate optimal state feedback K. Heuristic stopping criteria are used. The results of both runs are compared by the normalized deviation of the state feedback matrices:

$$\frac{\|K - K^{(E)}\|_F}{\max\{\|K\|_F, \|K^{(E)}\|_F\}} \approx 5.9 \cdot 10^{-16}.$$

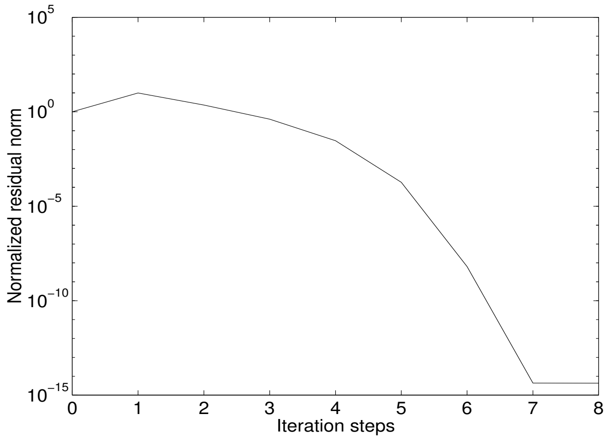

Figure 15: normalized residual norm history of the LRCF-NM in `demo_r1`.

References

[1] F. Aliev and V. Larin, *Construction of square root factor for solution of the Lyapunov matrix equation*, Sys. Control Lett., 20 (1993), pp. 109–112.

[2] A. Antoulas, D. Sorensen, and S.Gucercin, *A survey of model reduction methods for large-scale systems*, preprint, Dept. of Electr. and Comp. Engineering, Rice University, Houston, Texas 77251-1892, USA, 2000.

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

[4] P. Benner and R. Byers, *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Trans. Automat. Control, 43 (1998), pp. 101–107.

[5] P. Benner, J. Claver, and E. Quintana-Ortí, *Parallel distributed solvers for large stable generalized Lyapunov equations*, Parallel Processing Letters, 9 (1999), pp. 147–158.

[6] P. Benner, J. Li, and T. Penzl, *Numerical solution of large lyapunov equations, riccati equations, and linear-quadratic optimal control problems*, in preparation, Zentrum f. Technomathematik, Fb. Mathematik und Informatik, Univ. Bremen, 28334 Bremen, Germany, 2000.

[7] P. Benner, V. Mehrmann, V. Sima, S. V. Huffel, and A. Varga, *SLICOT - a subroutine library in systems and control theory*, Applied and Computational Control, Signals, and Circuits, 1 (1999), pp. 505–546.

[8] P. Benner and E. Quintana-Orti, *Solving stable generalized Lyapunov equations with the matrix sign function*, Numer. Alg., 20 (1999), pp. 75–100.

[9] P. Benner, E. Quintana-Orti, and G. Quintana-Orti, *Balanced truncation model reduction of large-scale dense systems on parallel computers.* Submitted for publication, 1999.

[10] R. Byers, *Solving the algebraic Riccati equation with the matrix sign function*, Linear Algebra Appl., 85 (1987), pp. 267–279.

[11] E. Davison, *A method for simplifying linear dynamic systems*, IEEE Trans. Automat. Control, 11 (1966), pp. 93–101.

[12] P. Feldmann and R. Freund, *Efficient linear circuit analysis by Padé approximation via the Lanczos process*, IEEE Trans. Computer-Aided Design, 14 (1995), pp. 639–649.

[13] R. Freund, *Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation.* Numerical Analysis Manuscript No. 98-3-02, Bell Laboratories, Murray Hill, New Jersey, 1998.

[14] R. Freund and N. Nachtigal, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.

[15] K. Gallivan, E. Grimme, and P. V. Dooren, *Asymptotic waveform evaluation via a Lanczos method*, Appl. Math. Lett., 7 (1994), pp. 75–80.

[16] ——, *A rational Lanczos algorithm for model reduction*, Numer. Alg., 12 (1996), pp. 33–63.

[17] J. Gardiner and A. Laub, *Parallel algorithms for the algebraic Riccati equations*, Internat. J. Control, 54 (1991), pp. 1317–1333.

[18] K. Glover, *All optimal Hankel norm approximations of linear multivariable systems and their L^∞-error bounds*, Internat. J. Control, 39 (1984), pp. 1115–1193.

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

[20] S. Hammarling, *Numerical solution of the stable, non–negative definite Lyapunov equation*, IMA J. Numer. Anal., 2 (1982), pp. 303–323.

[21] C. He and V. Mehrmann, *Stabilization of large linear systems*, in Preprints of the European IEEE Workshop CMP'94, Prague, September 1994, L. Kulhavá, M. Kárný, and K. Warwick, eds., 1994, pp. 91–100.

[22] D. Hu and L. Reichel, *Krylov-subspace methods for the Sylvester equation*, Linear Algebra Appl., 172 (1992), pp. 283–313.

[23] I. Jaimoukha, *A general minimal residual Krylov subspace method for large scale model reduction*, IEEE Trans. Automat. Control, 42 (1997), pp. 1422–1427.

[24] I. Jaimoukha and E. Kasenally, *Krylov subspace methods for solving large Lyapunov equations*, SIAM J. Numer. Anal., 31 (1994), pp. 227–251.

[25] ——, *Oblique projection methods for large scale model reduction*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 602–627.

[26] ——, *Implicitly restarted Krylov subspace methods for stable partial realizations*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 633–652.

[27] C. Kenney and A. Laub, *The matrix sign function*, IEEE Trans. Automat. Control, 40 (1995), pp. 1330–1348.

[28] D. Kleinman, *On an iterative technique for Riccati equation computations*, IEEE Trans. Automat. Control, 13 (1968), pp. 114–115.

[29] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*, Clarendon Press, Oxford, 1995.

[30] A. Laub, *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Control, 24 (1979), pp. 913–921.

[31] J. Li, F. Wang, and J. White, *An efficient Lyapunov equation-based approach for generating reduced-order models of interconnect*, in Proc. 36th IEEE/ACM Design Automation Conference, New Orleans, LA, 1999.

[32] J. Li and J. White, *Efficient model reduction of interconnect via approximate system Grammians*, in Proc. IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, 1999.

[33] P. Li and T. Penzl, *Approximate balanced truncation of large generalized state-space systems*, in preparation, Fak. f. Mathematik, TU Chemnitz, D-09107 Chemnitz, 2000.

[34] V. Mehrmann, *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution*, vol. 163 of Lecture Notes in Control and Information Sciences, Springer-Verlag, Heidelberg, 1991.

[35] B. C. Moore, *Principal component analysis in linear systems: Controllability, observability, and model reduction*, IEEE Trans. Automat. Control, 26 (1981), pp. 17–31.

[36] D. Peaceman and H. Rachford, *The numerical solution of elliptic and parabolic differential equations*, J. Soc. Indust. Appl. Math., 3 (1955), pp. 28–41.

[37] T. Penzl, *A cyclic low rank Smith method for large sparse Lyapunov equations.* to appear in SIAM J. Sci. Comput.

[38] ——, *Numerical solution of generalized Lyapunov equations*, Advances in Comp. Math., 8 (1998), pp. 33–48.

[39] ——, *Algorithms for model reduction of large dynamical systems.* Submitted for publication., 1999.

[40] ——, *Eigenvalue decay bounds for solutions of Lyapunov equations: the symmetric case.* Submitted for publication., 1999.

[41] L. Pillage and R. Rohrer, *Asymptotic waveform evaluation for timing analysis*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 352–366.

[42] J. Roberts, *Linear model reduction and solution of the algebraic Riccati equation by use of the sign function*, Internat. J. Control, 32 (1980), pp. 677–687.

[43] Y. Saad, *Numerical solution of large Lyapunov equations*, in Signal Processing, Scattering, Operator Theory and Numerical Methods, M. Kaashoek, J. V. Schuppen, and A. Ran, eds., Birkhäuser, Boston, MA, 1990, pp. 503–511.

[44] M. Safonov and R. Chiang, *A Schur method for balanced-truncation model reduction*, IEEE Trans. Automat. Control, 34 (1989), pp. 729–733.

[45] R. Smith, *Matrix equation $XA + BX = C$*, SIAM J. Appl. Math., 16 (1968).

[46] G. Starke, *Optimal alternating direction implicit parameters for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 28 (1991), pp. 1431–1445.

[47] M. Tombs and I. Postlethwaite, *Truncated balanced realization of stable, non-minimal state-space systems*, Internat. J. Control, 46 (1987), pp. 1319–1330.

[48] A. Varga, *Efficient minimal realization procedure based on balancing*, in Prepr. of IMACS Symp. on Modelling and Control of Technological Systems, A. E. Moudni, P. Borne, and S. Tzafestas, eds., vol. 2, 1991, pp. 42–49.

[49] E. Wachspress, *Iterative Solution of Elliptic Systems*, Prentice-Hall, 1966.

[50] ——, *Iterative solution of the Lyapunov matrix equation*, Appl. Math. Lett., 1 (1988), pp. 87–90.

[51] ——, *The ADI minimax problem for complex spectra*, in Iterative Methods for Large Linear Systems, D. Kincaid and L. Hayes, eds., Academic Press, San Diego, 1990, pp. 251–271.