SLICOT Working Note 1998-5

Benchmark collections in SLICOT[1]

Volker Mehrmann[2], Thilo Penzl[3]

June 1998

[1]This paper presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001-Leuven-Heverlee, BELGIUM. This report is also available by anonymous ftp from *wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN1998-5.ps.Z*

[2]Fakultät für Mathematik, TU Chemnitz, D–09107 Chemnitz, Germany. Email: `volker.mehrmann@mathematik.tu-chemnitz.de`

[3]Fakultät für Mathematik, TU Chemnitz, D–09107 Chemnitz, Germany. Email: `thilo.penzl@mathematik.tu-chemnitz.de`

Contents

1 Introduction

Benchmark collections provide data sets which represent test problems for certain classes of mathematical problems. They play an important role in the development of numerical state-of-the-art software. Benchmark collections are essential for validating new numerical methods and their software implementations. Moreover, the comparison of different numerical solution methods for a mathematical problem requires the availability of a thoroughly selected set of test examples. Furthermore, benchmark collections can be used as a reference for developers of new numerical software.

On the one hand, benchmark collections should contain data sets which reflect "real world" problems that typically arise in practical applications. On the other hand, they have to deliver test examples which usually cause problems for numerical methods (due to, e.g., ill-conditioning, near-singularity). The latter class of examples gives insight in the limits of applicability of numerical software. The purpose of the SLICOT benchmark collections is to establish an environment for testing the subroutines within the SLICOT library and those subroutines which are intended to be included in the library. Moreover, they should enable to compare the performance of SLICOT routines with other numerical software.

This document contains guidelines for setting up benchmark collections for SLICOT and deals with related questions. A considerable part of it is adopted from the benchmark collections

CAREX [1] and DAREX [2]. In Section 2 the basic structure of the SLICOT benchmark library is explained. Contributions to this library are expected by several NICONET partners as well as by persons or institutions outside NICONET. This interaction results in some organizational issues which are discussed in Section 3. Section 4 containes a few guidelines to be followed by contributors of benchmark examples outside NICONET. The implementation of benchmark routines by NICONET partners is addressed in Section 5.

Basically, SLICOT benchmark routines have to conform to the SLICOT standards [4]. In this document we do not present further narrow standards. It should rather be understood as a collection of guidelines to be followed as far as possible. Due to the great variety of topics covered by the SLICOT benchmark library and of test problems contained in this library deviations from these guidelines may be appropriate or even necessary in some situations.

Finally, it should be noted that notations of the type "G1" or "I3" are used in the following sections. These notations are only valid within this document. In particular, they do **not** correspond to the nomenclature of SLICOT routines.

2 Topics and Subtopics

The SLICOT benchmark library is intended to cover several *topics* such as "continuous-time Riccati equations" or "continuous-time Lyapunov equations". A complete list of the topics and the responsible NICONET partners can be found in Appendix A.

Topics may consist of several *subtopics*. Taking the Topic "continuous-time Lyapunov equations" as an example, it is sensible to distinguish between two subtopics:

- Lyapunov equations of the type
$$A^T X + XA = -Y;$$
- Lyapunov equations of the type
$$A^T X + XA = -BB^T,$$
where B is a rectangular matrix.

3 Organizational issues

Each topic of the SLICOT benchmark library is covered by a benchmark routine. These routines are implemented and maintained by a *responsible partner* (RP) of the NICONET project. Duties of these responsible partners are the following:

- The RP has to look for benchmark examples. Benchmark examples can be found in publications, in public data bases (e.g., [3]), etc.
- The RP collects the detected examples and those examples submitted by contributors outside NICONET.

- The RP has to decide which of the collected benchmark examples will be included in the respective benchmark collection.
- The RP implements the benchmark collection in the form of a benchmark routine. Moreover, he/she provides a documentation of this routine.
- The RP updates the benchmark collection when new interesting test problems are found. He/she provides the routine with release numbers.

Moreover, announcements (e.g., in newsletters, in newsgroups, at conferences) can help to find contributors of benchmark examples outside NICONET. Such announcements can be done by all NICONET partners. For simplicity, contributors outside NICONET should send benchmark examples to the NICONET partner *TU Chemnitz*. This partner will forward these examples to one or more responsible partners. The latter is the case if the respective benchmark example is suited for several benchmark collections.

4 Guidelines for the submission of benchmark examples by external contributors

It is important to get benchmark examples reflecting current "real world" problems from persons outside NICONET. In order to encourage potential contributors to a submission of benchmark examples it is sensible to keep the requirements as low as possible. As a consequence, a certain amount of postprocessing of the submitted examples will generally be necessary.

In any case, contributors have to declare the topic and the subtopic (if necessary), to which the example corresponds. If the data of the submitted example is contained in a file in a particular format, the contributor has to ensure that the responsible partner is able to retrieve the data. Misinterpretations of the data must be avoided. For convenience and safety, data should be provided in MATLAB or ASCII format if this is possible. Moreover, contributors should provide a short description (preferably in LaTeX or ASCII) of the example or a reference to such a description.

If contributors want to submit benchmark examples depending on parameters, it is strongly recommended to provide them in form of a MATLAB or FORTRAN routine. Contributors have to declare which values these parameters can attain (e.g., a certain interval for real parameters). Moreover, they must provide a default value for each parameter. These values should reflect standard conditions of the underlying problem.

5 Guidelines for the implementation of benchmark routines

For two reasons there is no need for further strict standards for benchmark routines besides the SLICOT implementation and documentation standards [4]. Firstly, benchmark routines are not expected to be integrated by SLICOT users in their codes, except for the purpose of testing. Secondly, the number of the different benchmark routines will be relatively small. It is more appropriate to set up a couple of guidelines for their implementation. These guidelines leave a certain amount of freedom to the responsible partner when implementing a benchmark routine. A pair of classifications of benchmark examples, which are needed in the remainder of this document, are introduced in the following subsection.

5.1 Classifications

5.1.1 Classification with respect to scalability and dependence on parameters

Benchmark examples are subdivided into the following four groups with respect to scalability and dependence on parameters:

G1: parameter-free problems of fixed size;

G2: parameter-dependent problems of fixed size;

G3: examples of scalable size without parameters;

G4: parameter-dependent examples of scalable size.

This classification is visible to the users of a benchmark routine. It is reflected by the documentation of the routine (cf. [1] and [2]) and by its calling sequence. A single benchmark example within a benchmark routine is addressed by a double index "g.n" where "g" is the group index (1, ..., 4 for G1, ..., G4, respectively) and "n" is the number within the group (1, 2, 3, ...).

In some cases it is sensible to subdivide a benchmark example into *subexamples*. For instance, the models of six inverted pendulums with 2, 3, 4, 5, 6, and 10 masses can be considered as one benchmark example. The subexamples are addressed by a *subindex*, which is considered as a regular integer parameter of the benchmark example within the array `IPAR` (cf. Subsection 5.3). For the example above, two definitions of the subindex are possible. It represents either the number of the model (i.e., 1, 2, 3, 4, 5, or 6) or the number of different masses (i.e., 2, 3, 4, 5, 6, or 10).

5.1.2 Classification with respect to the implementational type

Basically, there are five possibilities to implement examples. These differ in the way parameters and data are handled. In the following list, the term "subroutine" should be understood in a wider sense. Here, "subroutine" means a part of the benchmark routine (i.e., a few lines of code within the routine). The different implementational types can be characterized as follows:

I1: The example is provided as a "subroutine" which depends neither on parameters nor on extensive data. Consequently, no data has to be stored in a separate data file.

I2: All the data of the example is stored "explicitly" in a separate data file which is loaded by the benchmark routine. This means that the routine writes the loaded data to the output parameters without processing these data.

I3: The example is provided as a "subroutine" which depends on a small number of parameters. These are provided as input parameters of the benchmark subroutine. No data are stored in a separate data file.

I4: The example is provided as a "subroutine" depending on many "fixed" parameters (e.g., the parameters of a large Toeplitz matrix). These are provided in a separate data file. However, a part of the data may be stored "explicitly" in the data file.

I5: This type is a combination of I3 and I4. There are parameters in a data file as well as in the calling sequence. However, "explicit" data and parameters are generally stored in the same data file.

5.2 Some general guidelines

Each topic of the SLICOT benchmark library has to be covered by a certain benchmark routine which is flexible enough to handle each of the subtopics. In general, the generation of the data of a test example within a benchmark routine will not require a large amount of computation. Therefore, the performance of benchmark routines is less important than that of the computational routines in SLICOT. When setting up a benchmark routine the responsible partner should pay particular attention to a simple and clearly arranged implementation. Subsection 5.3 contains guidelines for structuring the calling sequence of a benchmark routine whereas the structure of external data files is discussed in Subsection 5.4. These guidelines should be considered as a compromise between the desired flexibility and simplicity of the routines.

5.3 The calling sequence of benchmark routines

This subsection contains guidelines for the interface of benchmark routines. These guidelines are illustrated by the following sample subroutine, which corresponds to the topic "continuous-time Lyapunov equations". Here, this routine is named `BX00XD`.

```
SUBROUTINE BX00XD(DEF,NR,DPAR,IPAR,VEC,N,M,A,LDA,B,LDB,Y,LDY,X,LDX,INFO)

CHARACTER         DEF
DOUBLE PRECISION  DPAR(*), N, M, A(LDA,*), B(LDB,*), Y(LDY,*), X(LDX,*)
INTEGER           NR(2), IPAR(*), LDA, LDB, LDY, LDX, INFO
LOGICAL           VEC(6)
```

The structures of the calling sequences of benchmark routines have to conform to the following regulations.

- The calling sequences of benchmark routines should contain the following parameters (The remarks in parentheses correspond to the sample routine `BX00XD`.):
  - a mode parameter (`DEF`) which determines whether default values are to be used in parameter arrays (such as `DPAR`) or not;
  - an input parameter which selects the desired benchmark example (`NR`);

- arrays containing parameters the benchmark example depends on (e.g., `DPAR`, `IPAR`);
- a flag vector which displays the availability of the output data for the desired benchmark example (`VEC`);
- a set of parameters, which represent the output data of the benchmark example (`N`, `M`, `A`, `B`, `Y`, `X`);
- leading dimensions of two-dimensional arrays (`LDA`, `LDB`, `LDY`, `LDX`);
- an error indicator (`INFO`).

- The number of the desired benchmark example has to be provided as an integer array consisting of two elements. Here, `NR(1)` describes the group (G1,...,G4) to which the benchmark example corresponds (i.e., `NR(1)` $\in \{1, 2, 3, 4\}$). `NR(2)` contains the number within this group (i.e., `NR(2)` $\in \{1, 2, 3, \ldots\}$). The use of such a double index enables to add new examples in updated version without renumbering those examples already contained in the collection.

- Benchmark examples of the Group G2, G3, or G4 depend on parameters. In this context, the scaling parameter in G3 is considered as a regular integer parameter. If `DEF .EQ. 'N'` on entry, then the parameters of the desired benchmark example have to be provided by the one-dimensional parameter arrays

  `IPAR` integer parameters;

  `SPAR` single precision parameters;

  `DPAR` double precision parameters;

  `CPAR` complex single precision parameters;

  `ZPAR` complex double precision parameters;

  `BPAR` logical parameters.

  If `DEF .EQ. 'D'`, these parameter arrays are not referenced on entry. Instead, default values are used. These values must be defined within in the benchmark routine.

  On exit, the parameter arrays can be overwritten with accompanying information about the benchmark example (e.g., condition numbers).

- The set of all quantities describing benchmark examples of the respective topic is referred to as *output data* of the benchmark routine. For instance, the output data of the routine `BX00XD` consists of the dimensions n (the order of the equation) and m (the number of columns in the matrix B) and of the matrices A, B, Y, and X. Depending on several circumstances (e.g., the subtopic to which the benchmark example corresponds, the availability of the exact solution X) only a subset of the output data can generally be provided by the benchmark routine. The available output data of the desired benchmark example is determined by the logical output vector `VEC`. The i-th quantity (with respect to the order of appearance in the calling sequence) within the set of parameters providing output data is available if and only if `VEC(`i`) .EQ. .TRUE.`. For instance, if the benchmark routine `BX00XD` returns

$$\mathtt{VEC}(i) = \begin{cases} \mathtt{.TRUE.} & : \quad i = 1, 3, 5 \\ \mathtt{.FALSE.} & : \quad i = 2, 4, 6 \end{cases}$$

then the quantities n, A, and Y are returned via the parameters `N`, `A`, and `Y`, respectively, whereas the parameters `M`, `B`, and `X` contain no output data.

5.4 Data files loaded by benchmark routines

Data for benchmark examples of the implementational types I2, I4, and I5 are partially or entirely stored in separate data files which are loaded by the corresponding benchmark routine. In context with these data files the following guidelines should be taken into account.

- The data of a benchmark example are usually stored in a single data file. An exception are examples which are subdivided into "subexamples". Here, it is allowed to store the data of each subexample in a separate data file. For instance, the data for the example of the inverted pendulums (see Subsubsection 5.1.1) should be stored in 6 separate data files.
- For benchmark examples of the implementational types I4 and I5, the data file strongly depends on the structure of the example. For this reason, there are no general rules or recommendations for the structure of the data file.
- For benchmark examples of the implementational type I2, it is recommended to provide the data in files which have the structure of MATLAB data files.
- The names of the data files consist of 7 or 8 characters and the extension `.dat`. They have the structure

  `AAAABCCD.dat`

  where the capitals have the following meaning:

`AAAA`	The name of benchmark routine, e.g., 'BX00';
`B`	One of the digits '1', '2', '3', or '4'. It indicates that the example belongs to the groups G1, G2, G3, or G4, respectively;
`CC`	The number of the example within the group ('01', '02', ...);
`D`	The number of the subexample ('1', '2', ...,'A', 'B', ...) within the example. This character is omitted, if the example does not contain subexamples.

References

[1] P. Benner, A.J. Laub, and V. Mehrmann. A collection of benchmark examples for the numerical solution of algebraic Riccati equations I: Continuous-time case. Preprint SPC95-22, Technische Universität Chemnitz, Chemnitz, Germany, 1995.

[2] P. Benner, A.J. Laub, and V. Mehrmann. A collection of benchmark examples for the numerical solution of algebraic Riccati equations II: Discrete-time case. Preprint SPC95-23, Technische Universität Chemnitz, Chemnitz, Germany, 1995.

[3] DAISY (Database for the Identification of Systems). http://www.esat.kuleuven.ac.be/sista/daisy/.

[4] SLICOT Implementation and Documentation Standards. http://www.win.tue.nl/wgs/stand.html.

A List of topics in the SLICOT benchmark library

The following list contains the topics of the SLICOT benchmark library and the related responsible partners (RP) within the NICONET project.

- Continuous-time Lyapunov equations (RP: V. Mehrmann, T. Penzl – *TU Chemnitz*)
- Continuous-time Riccati equations (RP: P. Benner – *Universität Bremen*)
- Continuous-time state-space systems (RP: V. Mehrmann, T. Penzl – *TU Chemnitz*)
- Discrete-time Lyapunov equations (RP: V. Mehrmann, T. Penzl – *TU Chemnitz*)
- Discrete-time Riccati equations (RP: P. Benner – *Universität Bremen*)
- Discrete-time state-space systems (RP: V. Mehrmann, T. Penzl – *TU Chemnitz*)
- H_∞ control (RP: D.W. Gu – *Leicester University*)
- Matrix exponentials (RP: P. van Dooren – *Université Catholique de Louvain*)
- Periodic systems (RP: P. van Dooren – *Université Catholique de Louvain*)
- Identification (RP: M. Verhaegen – *TU Delft*)