`linorm_subsp` `v1.2` Users' Guide

Paul Schwerdtner Matthias Voigt[*]

June 27, 2018

Abstract

`linorm_subsp` is a MATLAB software package for the computation of the $\mathcal{L}_\infty$-norm of possibly nonrational $\mathcal{L}_\infty$-functions. In particular, transfer functions of large-scale descriptor and delay systems can be handled. In this users' guide we give an overview of this software, provide information on its installation, and illustrate its usage by means of a few examples.

1 Introduction

`linorm_subsp` is a MATLAB package for the computation of the $\mathcal{L}_\infty$-norm of $\mathcal{L}_\infty$-functions. The function $G : \Omega \to \mathbb{C}^{p\times m}$, where $\Omega \subseteq \mathbb{C}$ is open and encloses the imaginary axis, may be given in two different ways.

1. The function can be given in the general form

$$G(s) := C(s)(sE(s) - A(s))^{-1}B(s) + D(s). \tag{1}$$

It is assumed that the matrix-valued functions E, $A : \Omega \to \mathbb{C}^{n\times n}$, $B : \Omega \to \mathbb{C}^{n\times m}$, $C : \Omega \to \mathbb{C}^{p\times n}$, and $D : \Omega \to \mathbb{C}^{p\times m}$ are defined by

$$\begin{aligned} A(s) &:= f_1(s)A_1 + \cdots + f_{\kappa_A}(s)A_{\kappa_A}, \\ B(s) &:= g_1(s)B_1 + \cdots + g_{\kappa_B}(s)B_{\kappa_B}, \\ C(s) &:= h_1(s)C_1 + \cdots + h_{\kappa_C}(s)C_{\kappa_C}, \\ D(s) &:= k_1(s)D_1 + \cdots + k_{\kappa_D}(s)D_{\kappa_D}, \\ E(s) &:= \ell_1(s)E_1 + \cdots + \ell_{\kappa_E}(s)E_{\kappa_E} \end{aligned} \tag{2}$$

for given matrices $A_1, \ldots, A_{\kappa_A}, E_1, \ldots, E_{\kappa_E} \in \mathbb{R}^{n\times n}$, $B_1, \ldots, B_{\kappa_B} \in \mathbb{R}^{n\times m}$, $C_1, \ldots, C_{\kappa_C} \in \mathbb{R}^{p\times n}$, $D_1, \ldots, D_{\kappa_D} \in \mathbb{R}^{p\times m}$, and given meromorphic functions $f_1, \ldots, f_{\kappa_A}, g_1, \ldots, g_{\kappa_B}, h_1, \ldots, h_{\kappa_C}, k_1, \ldots, k_{\kappa_D}, \ell_1, \ldots, \ell_{\kappa_E} : \Omega \to \mathbb{C}$.

2. The function G may also be given as a function handle. Then the structural constraints in (1) and (2) can be circumvented.

[*]Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany,

E-Mails: `p.schwerdtner@campus.tu-berlin.de, mvoigt@math.tu-berlin.de`.

This work is supported by the DFG priority program 1897: "Calm, Smooth and Smart – Novel Approaches for Influencing Vibrations by Means of Deliberately Introduced Dissipation".

We define the space

$$\mathcal{L}_\infty^{p\times m} := \left\{ G|_{\mathrm{i}\mathbb{R}} \;\middle|\; H : \Omega \to \mathbb{C}^{p\times m} \text{ is analytic for an open domain } \Omega \subseteq \mathbb{C} \right.$$
$$\left. \text{with } \mathrm{i}\mathbb{R} \subset \Omega \text{ and } \sup_{\omega\in\mathbb{R}} \|G(\mathrm{i}\omega)\|_2 < \infty \right\},$$

where $\mathbb{C}^+ := \{s \in \mathbb{C} \mid \operatorname{Re}(s) > 0\}$ denotes the open right complex half-plane. For a function $G \in \mathcal{L}_\infty^{p\times m}$, the $\mathcal{L}_\infty$-norm is defined by

$$\|G\|_{\mathcal{L}_\infty} := \sup_{\omega\in\mathbb{R}} \|G(\mathrm{i}\omega)\|_2 = \sup_{\omega\in\mathbb{R}} \sigma_{\max}(G(\mathrm{i}\omega)),$$

where $\sigma_{\max}(\cdot)$ denotes the largest singular value of its matrix argument.

The package `linorm_subsp` is designed to compute this norm efficiently, particularly for the case where n is very large and further $n \gg m, p$. Details on the algorithm and numerical results are described in the two references [1, 3].

2 Overview and Usage

2.1 Content

The package `linorm_subsp` is distributed via the archive `linorm_subsp.tar.gz`. The directory tree in Figure 1 lists all files that are obtained after the extraction of the archive.

2.2 Call of `linorm_subsp`

Within Matlab the call of the function `linorm_subsp` is

```
[ f, z, info ] = linorm_subsp( sys, opt ).
```

If the function G has the format in (1) and (2), then `sys` is a struct containing the information of the matrix-valued functions A, B, C, D, E. Each of these functions is given by the decomposition into its summands by cell arrays

```
sys.X = { X_1, ..., X_kX },
```

where `X_i` represents the i-th matrix for one of the functions under consideration. Moreover, the functions f_i, g_i, h_i, k_i, ℓ_i must in general be supplied as function handles in a row vector

```
sys.fct.x = @(s)[ x_1(s), ..., x_kX(s) ],
```

where `x` represents one of the above function families. In this case, the switch `sys.fct.type = 'l'` must be set.

For the most common function types, several simplifications have been implemented. For the case of the transfer function of a delay system, one can use the switch `sys.fct.type = 'd'`. Then the functions f_i, g_i, h_i, k_i, ℓ_i all attain the form $x_i(s) = \mathrm{e}^{-s\tau_{x_i}}$. Then it is only necessary to give the delays τ_{x_i} within a row vector as

```
sys.fct.x = [ tau_x_1, ..., tau_x_kX ].
```

```
linorm_subsp v1.1
├── AB13HD
│   └── AB13HD.f, linorm_h.F, makefile, make.inc, makemex.m
├── eigopt
│   ├── private
│   │   └── bfgs1run.m, bfgs.m, isnaninf.m, isnonnegint.m,
│   │       isposreal.m, linesch_sw.m, linesch_ww.m, plotcircle.m,
│   │       plotcontours.m, setdefaults.m, setwolfedefaults.m
│   └── eigopt.m, evalq.m, heapinsert.m, heapremove.m,
│       heapsort.m, heapupdate.m, H_infinity.m, isboundary.m,
│       plot_dead.m, plot_graph.m
├── Loewner
│   └── evalFun.m, getLoewnerSystem.m, sampleFun.m, sampleG.m
├── tools
│   └── makeSysStruct.m, plotFun.m
├── userguide
│   └── userguide.bib, userguide.pdf, userguide.tex
├── biorthnorm.m
├── diary.dia
├── getSubspace.m
├── Linf.m
├── linorm_h.mexa64
├── linorm_subsp.m
├── orthnorm.m
├── reduceSystem.m
└── testrun.m
```

Figure 1: Directory tree for the `linorm_subsp` package.

Moreover, if G is the transfer function of a linear system, then A, B, C, D, E are all constant and therefore, `sys.X` only needs to contain the respective matrix $X \in \{A, B, C, D, E\}$.

The variable `opt` is a struct that contains several options that can be specified to customize the $\mathcal{L}_\infty$-norm computation. Possible options and their respective default values are given in Table 1.

Table 1: Possible options in `opt` and their default values.

Option	**Description**
`tolz`	relative tolerance on the change of the optimal frequencies between two consecutive iterations. If the computed optimal frequencies between two consecutive iterations have relative distance less than `opt.tolz`, then the algorithm is assumed to be converged (`default = 1e-6`).
`tolf`	relative tolerance on the change of the computed $\mathcal{L}_\infty$-norms between two consecutive iterations. If the computed $\mathcal{L}_\infty$-norms between two consecutive iterations have relative distance less than `opt.tolf`, then the algorithm is assumed to be converged (`default = 1e-6`).

`maxit`	maximum number of iterations allowed until termination of the algorithm (`default = 30`).
`initialPoints`	initial frequencies of the initial interpolation points on the imaginary axis.
`prtlevel`	specifies the print level as follows: = 0: return no information; = 1: return minimal information; = 2: return full information (default = 0).
`boydbalak`	specifies which implementation of the Boyd-Balakrishnan algorithm is used for the computation of the $\mathcal{L}_\infty$-norm of the reduced functions as follows: = 0: use the FORTRAN routine `AB13HD.f` which is called by the gateway function generated by `linorm_h.F`; = 1: use the function 'norm' of the MATLAB Control System Toolbox (WARNING: This may not work, if the reduced descriptor matrix pars.E is singular.); (default = 0).
`eigopt.bounds`	a vector `[ lowerBound, upperBound ]` specifying the interval in which `eigopt` shall optimize the maximum singular value of the function $G(s)$ in (1). This option is only used if `opt.doLoewner == 0`.
`eigopt.gamma`	a lower bound for the second derivative of the the function $-\sigma_{\max}(G(\mathrm{i}\omega))$ where ω is within `opt.eigopt.bounds`. This option is only used if `opt.doLoewner == 0`.
`keepSubspaces`	specifies whether the intermediate subspaces obtained during the iteration are kept as follows: = 0: only the subspaces of the initial reduced function and the past two subspaces obtained during the iteration are kept; = 1: all the interpolation subspaces are kept during the iteration. This will let the dimensions of projection spaces grow in every iteration but is more robust in many cases (`default = 1`).
`biorth`	specifies which orthonormalization scheme is used as follows: = 0: the intermediate projection spaces contained in U and V are orthonormalized separately, i. e., $U^{\mathsf{H}}U = V^{\mathsf{H}}V = I_k$ for some k; = 1: the intermediate projection spaces are bi-orthonormalized, i. e., $U^{\mathsf{H}}V = I_k$ for some k (`default = 0`).

`orthtol`	relative truncation tolerance in `svd` for the determination of an orthonormal basis of a subspace or bi-orthonormal bases of two subspaces (`default = 1e-12`).
`maxSing`	specifies how the projection spaces are constructed as follows: $= 0$: all the singular vectors of $G(\mathrm{i}\omega)$ at an interpolation point $\mathrm{i}\omega$ are included in the updated projection spaces; $= 1$: only the singular vectors corresponding to the largest singular value of $G(\mathrm{i}\omega)$ at an interpolation point $\mathrm{i}\omega$ are included in the updated projection spaces (`default = 0`).
`doLoewner`	specifies whether the Loewner approach is used to calculate the $\mathcal{L}_\infty$-norm as follows: $= 0$: the Loewner framework is not used; $= 1$: the Loewner framework is used (`default = 0`).
`fnHandle`	specifies whether a function handle is passed to linorm_subsp. In this case, the Loewner approach is used to construct a rational function that matches the function evaluations. $= 0$: no function handle, but a function of the previously defined format is passed; $= 1$: a function handle is passed (`default = 0`).

Initial frequencies must always be given in order to start the computation of the $\mathcal{L}_\infty$-norm with `linorm_subsp`. Further, `eigopt.bounds` and `eigopt.gamma` must be provided if `eigopt` is used for computing the norm. The routine `linorm_subsp` has three output arguments, where `f` denotes the computed value of the $\mathcal{L}_\infty$-norm, `z` is the corresponding optimal frequency, and `info` is a struct containing information about the computation such as an error indicator. A list of the information gained throughout the computation is contained in Table 2.

Table 2: Information contained in `info` on exit.

Variable	**Description**
`time`	time needed to compute the result.
`iterations`	number of iterations at termination of the algorithm.
`finaltolz`	relative distance of the variable `z` between the last two iterations before termination.
`finaltolf`	relative distance of the variable `f` between the last two iterations before termination.
`termcrit`	specifies which termination criteria are satisfied as follows: $= 0$: both the optimal frequency `z` and the $\mathcal{L}_\infty$-norm `f` have converged; $= 1$: only the optimal frequency `z` has converged, but the $\mathcal{L}_\infty$-norm may be inaccurate;

	= 2: only the $\mathcal{L}_\infty$-norm `f` has converged, but the optimal frequency may be inaccurate; = 3: the algorithm has terminated by reaching the maximum number of iterations.
`error`	contains an error indicator as follows: = 0: return without an error; = 1: the maximum number of iterations specified in `opt.maxit` has been exceeded.

3 Installation

Almost all routines used by `linorm_subsp` are written in MATLAB. Only the routine `AB13HD` in the subdirectory `AB13HD` has to be compiled before usage. It is the Fortran implementation of the Boyd-Balakrishnan algorithm for descriptor systems from [2]. We have provided a precompiled gateway function `linorm_h.mexa64` that can be used as is on 64bit Linux machines. This is the case if the call

```
computer
```

in MATLAB returns the result `'GLNX64'`. Otherwise, the gateway function has to be generated manually. We describe this process for Linux machines within the next subsections. On other architectures, these steps must be adapted.

From version 1.2 of `linorm_subsp`, the option `boydbalak = 1` can be used to choose MATLAB's built-in implementation of the Boyd-Balakrishnan algorithm. It uses the SLICOT implementation of the algorithm for standard and generalized state-space systems. This avoids the need of recompiling `AB13HD.f` and `linorm_h.F` on certain architectures and the following steps of the installation can be skipped.

WARNING: Using the option `boydbalak = 1` may result in false results or an error, if one of the reduced descriptor system realizations constructed in the algorithm has a singular descriptor matrix contained in `pars.E`.

3.1 Before Installation

The routine `AB13HD` and its gateway function make calls to subprograms from the software packages SLICOT (Subroutine Library in Control Theory), LAPACK (Linear Algebra Package) and BLAS (Basic Linear Algebra Subprograms). Thus it is necessary to download and install these libraries before compilation.

SLICOT source code and the prebuilt library are freely available for academic users after registration from the SLICOT website[1]. The LAPACK and BLAS libraries are freely downloadable from netlib [2]. However, for maximum efficiency it is recommended to use machine-specific, optimized versions whenever possible. The library versions that are provided by MATLAB should be sufficiently efficient.

[1] available from `http://slicot.org`.

[2] available from `http://www.netlib.org/`.

3.2 Building `AB13HD.o`

To compile the Fortran source code `AB13HD.f` to obtain the object file `AB13HD.o`, a make file `makefile` and the associated file `make.inc` have been provided within the subfolder `AB13HD`. In order to use this make file on a specific Unix platform, some changes may be necessary in these files.

The changes in `make.inc` might define the specific the compiler, linker, and compiler options, as well as the location and names of the SLICOT, LAPACK, and BLAS libraries, which the program files should be linked to. Details are given in the file `make.inc`.

IMPORTANT: On 64bit platforms the code must be compiled with the options `-fPIC` and `-fdefault-integer-8`, for instance by setting

```
OPTS = -O2 -fPIC -fdefault-integer-8
```

in `make.inc`.

After performing the necessary changes, as suggested in the comments of `makefile` and `make.inc`, the object file `AB13HD.o` can be obtained by calling

```
make
```

in a shell from the subdirectory `AB13HD`.

3.3 Creating a Gateway Function for Running `AB13HD` in Matlab

For calling `AB13HD.o` from MATLAB, the MEX-file `linorm_h.F` has been written to generate a gateway function. To do this, one must first modify the file `makemex.m` in the subfolder `AB13HD` and set the location of the compiled SLICOT library in the variable `libslicot`. After that, the MATLAB call

```
makemex
```

from the subdirectory `AB13HD` should generate the gateway function which can then be called by `linorm_subsp`.

4 Examples

Here we explain four examples to the illustrate the usage of `linorm_subsp`. The output for all of these and further examples can be found in the file `diary.dia`.

4.1 Minimal Example – `build`

`build` is a SLICOT benchmark model[3] with system matrices of small size. The dynamical system is given by

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{3}$$

with E, $A \in \mathbb{R}^{48\times 48}$, B, $C^{\mathsf{T}} \in \mathbb{R}^{48\times 1}$, and $D = 0$. First, the system matrices need to be stored in a struct that fits the convention used in `linorm_subsp`. For

[3] see `http://slicot.org/20-site/126-benchmark-examples-for-model-reduction`.

convenience, the function `makeSysStruct` from the subdirectory `tools` can be called. This function takes the matrices E, A, B, C, and D as arguments and saves them within a struct that is the output of the `makeSysStruct`. Afterward, initial frequencies for the interpolation need to be defined since this is a mandatory input argument of `linorm_subsp`. To compute the $\mathcal{L}_\infty$-norm of the system's transfer function, `linorm_subsp` can be called as follows:

```
%
% sys is constructed.
%
sys = makeSysStruct( E, A, B, C, 0 );
%
% Initial frequencies are defined.
%
opt.initialPoints = linspace( 0, 100, 10 );
%
% Main call.
%
[ f, z, info ] = linorm_subsp( sys, opt );
```

4.2 MIMO Example – `mimo46x46_system`

In this example, a more customized use of `linorm_subsp` is shown. The model `mimo46x46_system`[4] is a linear descriptor system as in (3) defined by matrices of much larger dimension, i. e., E, $A \in \mathbb{R}^{13250\times13250}$ and B, $C^\mathsf{T} \in \mathbb{R}^{13250\times46}$. Because of the considerable number of columns in B and C^T it is advisable to only take the singular vectors corresponding to the maximum singular values of the reduced functions into the interpolation subspaces by using the option `opt.maxSing = 1`. the function call is given by the following code:

```
%
% sys is constructed.
%
sys = makeSysStruct( E, A, B, C, 0 );
%
% Set options.
%
opt.initialPoints = linspace( 0, 10, 10 );
opt.maxSing = 1;
%
% Main call.
%
[ f, z, info ] = linorm_subsp( sys, opt );
```

[4]available from `https://sites.google.com/site/rommes/software`.

4.3 Delay Example Using `eigopt`

In this example, we aim to compute the $\mathcal{L}_\infty$-norm of the transfer function of a delay system. This system is given by

$$\begin{aligned} E\dot{x}(t) &= A_1x(t) + A_2x(t-\tau) + Bu(t), \\ y(t) &= Cx(t). \end{aligned}$$

The input struct `sys` is constructed using a cell array to store A_1 and A_2 instead of calling `makeSysStruct`. Moreover, the flag `sys.fct.type = 'd'` must be set to indicate that the system is a delay system and the delays for A_1 and A_2 are defined within `sys.fct.a`. The following code determines the struct `sys`:

```
sys.A = { A0, A1 };
sys.B = B;
sys.C = C;
sys.D = D;
sys.E = E;
sys.fct.type = 'd';
sys.fct.a = [ 0, tau ];
```

In the script below this is done within the function `delay_model` (which also takes inputs for the parameters of the model).

Finally, the options for `eigopt` are set and the folder `eigopt` is added to MATLAB's path. The complete call looks as follows:

```
%
% Add path for eigopt.
%
addpath( 'eigopt' );
%
% sys is constructed.
%
sys = delay_model( 500, 5, 0.01, 1 );
%
% eigopt options are set.
%
opt.eigopt.bounds = [ 0, 50 ];
opt.eigopt.gamma = -100;
opt.initialPoints = linspace( 0, 50, 10 );
%
% Main call.
%
[ f, z, info ] = linorm_subsp( sys, opt );
```

4.4 Delay Example Using the Loewner Approach

Now, the $\mathcal{L}_\infty$-norm of the same system is computed using the Loewner approach. The main difference is that the directory containing the functions necessary to compute the Loewner matrices must be included to the MATLAB path. Also the option `opt.doLoewner = 1` must be set. The complete call looks as follows:

```
%
% Add path for Loewner.
%
addpath( 'Loewner' );
%
% sys is constructed.
%
sys = delay_model( 500, 5, 0.01, 1 );
%
% Choose the Loewner framework.
%
opt.doLoewner = 1;
opt.initialPoints = linspace( 0, 50, 10 );
%
% Main call.
%
[ f, z, info ] = linorm_subsp( sys, opt );
```

References

[1] N. Aliyev, P. Benner, E. Mengi, P. Schwerdtner, and M. Voigt. Large-scale computation of $\mathcal{L}_\infty$-norms by a greedy subspace method. *SIAM J. Matrix Anal. Appl.*, 38(4):1496–1516, 2017.

[2] P. Benner, V. Sima, and M. Voigt. $\mathcal{L}_\infty$-norm computation for continuous-time descriptor systems using structured matrix pencils. *IEEE Trans. Automat. Control*, 57(1):233–238, 2012.

[3] P. Schwerdtner and M. Voigt. Computation of the $\mathcal{L}_\infty$-norm using rational interpolation. *IFAC-PapersOnLine*, 2018. Joint 9th IFAC Symposium on Robust Control Design and 2nd IFAC Workshop on Linear Parameter Varying Systems, Florianópolis, Brasilien, 2018. Accepted for publication, also available from `http://www.math.tu-berlin.de/fileadmin/i26/Bilder_Webseite/AG_ModNumDiff/FG_NumMath/mvoigt/SchV18.pdf`.