SYSTEMS CONTROL

AND FLIGHT DYNAMICS DEPARTMENT

Technical Report

User Manual of the Linear Fractional Representation Toolbox

Version 2.0

J.-F. MAGNI

TR 5/10403.01F DCSD - October, 2005 (revised Feb., 2006)

INFORMATION PROCESSING AND SYSTEMS

ONERA

SYSTEMS CONTROL AND FLIGHT DYNAMICS

DEPARTMENT

Technical Report TR 5/10403.01F DCSD

User Manual of the Linear Fractional Representation Toolbox

Version 2.0

October, 2005 (revised Feb., 2006)

Author

J.-F. MAGNI

Approved by :
P. FABIANI
Interim Head of Systems Control
and Flight Dynamics Department

DCSD-T n° 219/2005

Toulouse Center
2, Avenue E. Belin, B.P. 4025
F-31055 TOULOUSE CEDEX 4
Phone (33) 5.62.25.25.25 - Fax 5.62.25.25.64
http://www.onera.fr/dcsd/
French National Aerospace Research Establishment

IDENTIFICATION SHEET of ONERA document # TR 5/10403.01F DCSD

Issuing department:	Contracting party:	Contract references:
Systems Control and Flight Dynamics	RG	
	Program record # :	Date: October, 2005 (revised Feb., 2006)
Title: *User Manual of the Linear Fractional Representation Toolbox Version 2.0*		
Author: J.-F. MAGNI		

PROTECTION Level:		PROTECTION Release:	
Title	: Not Protected	Title	: not applicable
Sheet	: Not Protected	Sheet	: not applicable
Document	: Not Protected	Document	: not applicable

Author abstract

This document is the user manual of Version 2.0 of the Linear Fractional Representation Toolbox authored by S. Hecker, A. Varga and J.F. Magni.

This toolbox (for use with MATLAB or SCILAB), considers modelling, manipulation, order reduction and approximation of uncertain systems in LFT-form (LFT: Linear Fractional Transformation).

The representation of systems in LFT-form is considered in two complementary ways: the object-oriented (LFR-objects) and the symbolic way.

Version 2.0 is very different from version 1.x, all functions have been re-written, there is no backward compatibility. The main advantage of version 2.0 with respect to version 1.x is that the new LFR-objects can be manipulated like symbolic objects, in particular there is no more restriction on invertibility and uncertainties do not need to be ordered.

Keywords

Modelling, linear fractional transformation, LFT, order reduction, realization, mu-analysis

Toulouse Center
2, Avenue E. Belin, B.P. 4025
F-31055 TOULOUSE CEDEX 4
Phone (33) 5.62.25.25.25 - Fax 5.62.25.25.64
http://www.onera.fr/dcsd/
French National Aerospace Research Establishment

ONERA

DISTRIBUTION LIST of ONERA document # TR 5/10403.01F DCSD

Document only

- *Outside ONERA*
  Web publication
- *Inside ONERA*

  CID Toulouse ..... 1 ex.
  C. BARROUIL (DSB/TIS) ..... 1 ex.
  P. FABIANI (Interim Head DCSD) ..... 2 ex.

  DCSD-Toulouse: J.-F. MAGNI ..... 5 ex.

Identification sheet only

- *Outside ONERA*
  CEDOCAR
- *Inside ONERA*
  ONERA/ISP
  (DCSD-Lille) - (DCSD-Salon de Provence) - (DCSD-Toulouse)
  SGA Toulouse - DEMR - DESP - DMAE - DOTA - DTIM
- *Systematic distribution*
  D - DSG - DTG - DAJ - DSB - DCV
- *Electronic distribution*
  Intranet DCSD : `/ARCHIVES/FICHIDENT/T_R-219_05.html`

Toulouse Center
2, Avenue E. Belin, B.P. 4025
F-31055 TOULOUSE CEDEX 4
Phone (33) 5.62.25.25.25 - Fax 5.62.25.25.64
http://www.onera.fr/dcsd/
French National Aerospace Research Establishment

ONERA

Contents

Chapter 1

Introduction

This document is the user manual of Version 2.0 of the Linear Fractional Representation Toolbox authored by S. Hecker, A. Varga and myself.

This toolbox, for use with MATLAB (or SCILAB), considers modelling, manipulation, order reduction and approximation of uncertain systems in LFT-form (LFT: Linear Fractional Transformation). Such systems will be called Linear Fractional Representations (LFR). Briefly, systems in LFT-form are such that all varying or uncertain parameters are "pulled out" so that the system appears as a nominal system subject to an artificial feedback. This artificial feedback is expected to capture all variability and uncertainties. More generally, nonlinear components and the Laplace variable might also be represented in an artificial feedback form. Such models are essential in modern control theory for robustness analysis (μ-analysis) and for synthesis (robust and/or scheduled control design). "Pulling out" the parameters can be viewed as a realization technique similar to the realization of transfer function matrices leading to state-space representations:

Transfer function matrix	Realization $\longrightarrow$	State-space representation: Figure 2.5
Rational multivariate matrix	Realization $\longrightarrow$	Linear fractional representation: Figure 2.2

A more synthetic presentation of the techniques and tools discussed in this manual can be found in [39, 40, 41, 31, 32]

1.1 Organization of the manual

Chapter 2 gives some basic definitions such as "LFT realization" and presents basic manipulations like the "star product". The objective of this chapter is to give general guidelines for low order LFT generation. For this purpose, an "object-oriented" approach to LFT generation is proposed. From the discussions given in this chapter, it turns out that two main steps are to be considered. First, realization must be done carefully, especially taking advantage of parameter commutativity (details in Chapter 3). Then, some techniques reminiscent of "minimal realization" for standard dynamic systems can be applied for further order reduction (details in Chapter 4).

Chapter 3. This chapter is devoted to LFT realization. First, are considered the conversions from coprime factorizations and from state-space realizations to input/output LFTs. After this discussion, only the state-space form is considered. The realization techniques that are treated are: Morton's technique ([46]), Horner factorization ([52]) and the structured tree decomposition ([21, 22, 23]). We also mention without details a graph-based ([26, 30]) and a matrix-based approach ([12, 13]). It is recommended to use the tree decomposition that is the most natural and the most efficient one. The object-oriented realization technique described in Chapter 2 is also an alternative way for low order realization, but in this case the factorizations that reduce the order must be performed manually.

Chapter 4. This chapter treats the generalization to LFTs of "minimal realization" of standard dynamic systems. Considering LFTs, the size of what is improperly called a "minimal realization" depends on the initial realization considered for oder reduction. The reason for that is that parameter commutativity is not taken into account, and in fact, minimality is truly attained only if parameters do not commute. Before presenting specific LFT order reduction techniques, it is shown how standard system "minimal realization" techniques can be applied to LFTs ($1-D$ technique [36]). Then the generalized Kalman decomposition ([25, 6, 9]) and the generalized Gramians approaches ([5, 8, 10, 11, 54]) are briefly presented ($n-D$ techniques). These two techniques lead to the so-called "minimal realization", the second one offers the possibility of further reduction by approximation. We conclude this chapter by describing a technique that permits us to evaluate precisely LFT approximation errors, and, by the way to model approximation errors.

Chapter 5. This chapter considers complex uncertainties. Such uncertainties are usually used for modelling neglected dynamics. In order to use μ-analysis for performance analysis or for performance robustness analysis, it is also convenient to consider artificial complex uncertainties (Main Loop Theorem).

Chapter 6. This chapter describes the use of LFTs for modelling the *continuum* of linearized models of a nonlinear system. The dependency of parameters on the equilibrium surface is also

taken into account: A technique for finding an explicit form of the equilibrium surface equation is proposed, this technique works for a large class of systems, see §6.3.2, page 134. However, we have to keep in mind that modelling in that way leads to very high order LFTs. Finally is considered the problem of transforming tables of numerical data to LFTs.

1.2 Getting started with the toolbox

This toolbox considers modelling, manipulation, order reduction and approximation of uncertain systems in LFT-form (LFT: Linear Fractional Transformation). Such models consists of a constant interconnection matrix subject to several feedback connections which represent dynamics, standard feedback, uncertainties (real parameters or neglected dynamics) and to some extent, non-linearities. We shall denote these representations as LFR (Linear Fractional Representations). LFR models are required for worst case analysis (using μ-analysis) and, to some extent, for robust or scheduled control design.

The LFR-object of version 2.0 is very different from the previous one (version 1.x), therefore, both version are not compatible, see page §7.3 page 161 for details.

The best way to get started with the toolbox is to read the first part of the manual and to run the numerous illustrative examples therein. See also the HTML tutorial shipped with the toolbox.

First, have a look at the list of functions given on pages 163 to 164. For almost all functions, the on-line `help` message contains an illustrative example that can be run by "copy and paste".

LFR-objects generation. Two kinds of objects are handled by this toolbox: LFR-objects and symbolic objects. The former can be viewed as realizations of the latter (see the figure of page 1) if only scalar uncertainties are considered.

- Getting started with the object-oriented approach: see Example 2.12, page 53.
- Getting started with the symbolic approach: see Example 2.13, page 54.
- Adding complex full blocks, see Example 5.1, page 118 and scalar complex blocks, see Example 5.2, page 120.
- Interpolation tables can easily be transformed to LFR-objects, see Example 6.5, page 142.
- LFR-object generation form minimum and maximum values of matrix entries: see the help file of `bnds2lfr`.

Some transformations. There are several transformations that are straightforward, as for example feedback (see the help files of `lfr/feedback`, `uplft`). We shall mention here:

- Transformation for treating rational representations as polynomial ones, and back transformation to be applied to the so obtained LFR-objects: see Example 3.1, page 69.

- Transformation from the system matrix `[A,B;C,D]` LFR-form to an input/output representation: see Example 3.2, page 73.
- For normalization of real uncertainties see Example 2.8 page 41.
- It is possible to replace parameters of a given object by a functions of other parameters, see Example 6.4, page 138.

Modelling of non-linear systems. Modelling of a continuum of linearized models induces differentiation of some expressions. Two possibilities are treated in details.

- Symbolic approach: see Example 6.1, page 130 plus Example 6.3, page 137.
- Object-oriented approach, see Example 6.2, page 132 plus Example 6.4, page 138.

See 6.3.2, page 134 for the derivation of the equilibrium surface. Note that LFR-objects can also be used for quasi-LPV modelling rather than linearizing ([49]).

Order reduction and approximations. For order reduction, read first page 63. On this page, the fact that we distinguish between the following points is justified.

- For low order realization, the tree decomposition is the most efficient one (however polynomial dependency with respect to uncertain parameters is required). For getting started with the tree decomposition, see Example 3.6 page 86.
- For order reduction after realization, the most efficient technique is the n-D approach. See Example 4.3, page 102.
- For more heuristic approximation techniques it is suggested to read §4.4.1, page 106. The following illustrative examples will be helpful: Examples 4.4, 4.5, page 111 and Example 6.6, page 143.

μ-analysis. This toolbox proposes interfaces with commercial toolboxes.

- Interface with the μ-Analysis and Synthesis Toolbox: see `help lfr2mu`.
- Interface with the LMI Control Toolbox: see of `help lfr2mustab` and `help lfr2mubnd`.

- Interface with the Robust Control Toolbox version 3: see the help file of `lfr2mussv` (the functions `lfr2mu`, `lfr2mustab` and `lfr2mubnd` can also be used with this new toolbox). Note that the function `lfr2rob` can be used to convert LFR-objects to UMAT or USS-objects and, conversely, the function `lfr` can be sused to convert UREAL, UCOMPLEX, UMAT or USS-objects to LFR-objects.

Steps for LFR-modelling. For moderately complex systems with **polynomial** dependency, use `symtreed` and then `reduclfr`. In case of **rational** dependency, read first §3.1, page 67, alternatively you can use the function `sym2lfr` rather than `symtreed` as this function doesn't require polynomial dependency.

1.3 Acknowledgments

1.3.1 Version 1.x

Many people helped me during the development of Version 1 this toolbox. I would like to acknowledge the help of:

Stephane Delanoy	For having initialized the LFR-object definition and manipulation.
Carsten Döll	For having developed the tools that permit Simulink to support LFR-objects. This work is not yet included in the toolbox.
Jean-Paul Dijkgraaf	For having intensively used the toolbox. This work permitted us to discover and fix several bugs. This work will be available when the aircraft models he used will be released in the public domain.
Gilles Ferreres	For giving me some ideas such as the differentiation of LFR-objects.
Christelle Cumer	For participating in the earlier development of the toolbox and for helpful suggestions concerning Gramian-based techniques.
Andras Varga	For helpful discussions relative to order reduction and approximation.
Carolyn Beck	For permitting me to use her MATLAB code for Gramian-based order reduction.

1.3.2 Version 2.0

A large part of the functions of version 2 have been rewritten from version 1 or from scratch by Simon Hecker and Adras Varga.

EMPTY PAGE

Chapter 2

Notations and generalities

In this chapter the notion of LFT Representation (LFR) is defined. Most of notations used later are described. Our objective is not to present in detail the construction of LFR-objects, we just describe the "object-oriented" technique (which is an elementary realization technique) to be able to put in evidence the importance of commutativity of uncertain parameters.

This discussion will allow to distinguish between the (absolute) minimality and the "relative-minimality". The conclusions of this chapter justify the structure adopted for the continuation of this part: At first it is necessary to take advantage of parameter commutativity (realization, see Chapter 3), then one can use the reduction techniques which concern "relative minimality" (see Chapter 4).

In fact, there will be nothing concerning "absolute minimality".

2.1 Generalities

The systems considered in control engineering depend usually on uncertain parameters and on measurable varying parameters. Such parameters will be denoted δ_i in this report. This section introduces Linear Fractional Representations that are representation for modelling such systems

2.1.1 Definition of Linear Fractional Representations

In order to introduce the definition of a Linear Fractional Representation (LFR), let us consider two symbolic objects:

$$K_1(\delta_1) = I_{n_1} \frac{b + \delta_1(bc - ad)}{1 - a\delta_1} \tag{2.1}$$

$$K_2(\delta_2) = I_{n_2}(a\delta_2^2 + b\delta_2 + c) \tag{2.2}$$

in which δ_1 and δ_2 are 1×1 symbolic objects (a, b, c, d are fixed parameters). It is straightforward to check that the feedback loops of Figure 2.1 represent respectively $y = K_1(\delta_1)\ u$ and $y = K_2(\delta_2)\ u$

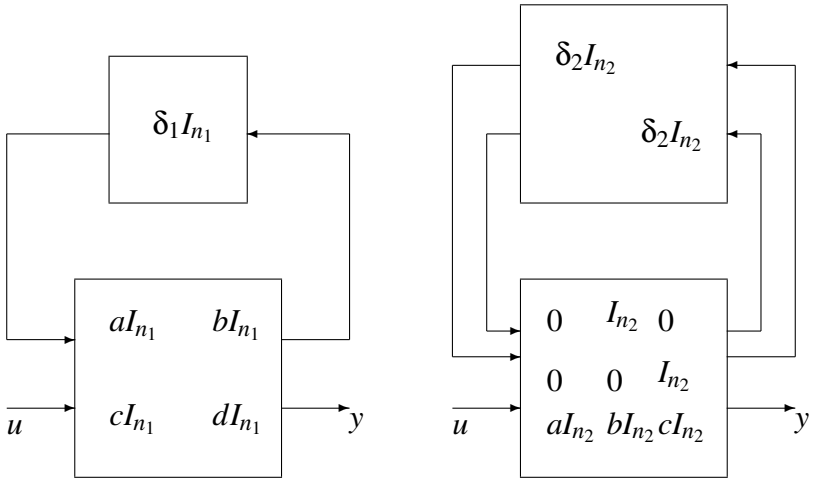

Figure 2.1: Representation of $K_1(\delta_1) : y = K_1(\delta_1)u$ and of $K_2(\delta_2) : y = K_2(\delta_2)u$

The feedback structures of Figure 2.1 are reminiscent of state-space representations. State-space representations are *realizations of transfer function matrices*, this remark suggests the following definition:

Definition 2.1.1 *Let us consider a rational symbolic object*

$$K(1/s, \delta_1, \ldots, \delta_q)$$

A Linear Fractional Representation *(LFR) is a realization as depicted in Figure 2.2 of such a symbolic object. In other words, realizing $K(1/s, \delta_1, \ldots, \delta_q)$ consists of finding the matrices $A, B_1, B_2, C_1, C_2, D_{11}, D_{12}, D_{21}, D_{22}$ such that*

$$y = K(1/s, \delta_1, \ldots, \delta_q)\, u$$

can be represented by

$$\begin{array}{lllll} \dot{x} & = Ax & +B_1 w & +B_2 u \\ z & = C_1 x & +D_{11} w & +D_{12} u \\ y & = C_2 x & +D_{21} w & +D_{22} u \end{array} \tag{2.3}$$

in which

$$w = Diag\{\delta_1 I_{n_1}, \ldots, \delta_q I_{n_q}\} z \tag{2.4}$$

A similar definition holds for discrete time systems ($K(1/z, \delta_1, \ldots, \delta_q)$). The continuous time approach is preferred in this document because in this case, the physical parameters appear more naturally. However, it is always possible to convert a continuous time LFR-object to an equivalent discrete one using the Tustin's transformation (see Example 2.6).

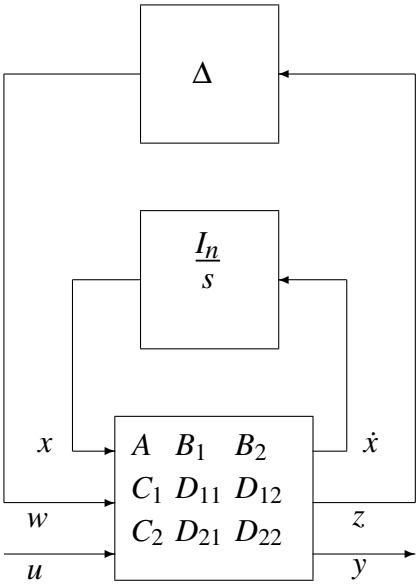

Figure 2.2: LFR: realization of $K(1/s, \delta_1, \ldots, \delta_q)$

Comment 2.1.2 *In § 2.5, it will be shown how to build realizations corresponding to the above definition. In fact, this is very easy because we just need to realize the elementary objects* $1/s$ *and* δ_i*'s and then to combine these realizations (object-oriented realization).*

Comment 2.1.3 *In the above definition,* Δ *stands for the matrix*

$$\Delta = Diag\{\delta_1 I_{n_1}, \ldots, \delta_q I_{n_q}\} \tag{2.5}$$

For simplifying notations, we shall also use an alternative notation (which one is used is expected to be clear from the context)

$$\Delta = Diag\{I_n/s, \delta_1 I_{n_1}, \ldots, \delta_q I_{n_q}\} \tag{2.6}$$

in which case, the realization becomes like in Figure 2.3.

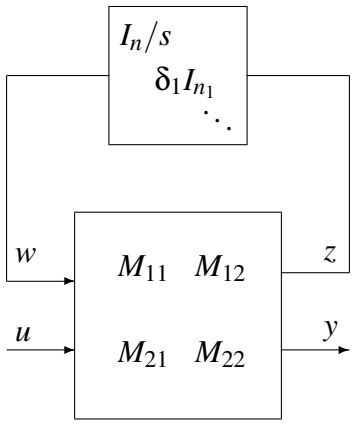

Figure 2.3: Simplified notation for an LFR realization

Correspondences Figures 2.3 $\leftrightarrow$ 2.2

$$M_{11} \leftrightarrow \begin{bmatrix} A & B_1 \\ C_1 & D_{11} \end{bmatrix}; M_{12} \leftrightarrow \begin{bmatrix} B_2 \\ D_{12} \end{bmatrix}; M_{21} \leftrightarrow \begin{bmatrix} C_2 & D_{21} \end{bmatrix}; M_{22} \leftrightarrow D_{22}$$

2.1.2 Special cases

Using the original notation of Definition 2.1.1 (not the simplified notation of Comment 2.1.3), we have the following special cases:

- *Uncertain non-dynamic matrices*, there is no feedback *via* I/s: see Figure 2.4.
- *A standard linear system*, there is no feedback *via* Δ see Figure 2.5. (A, B_2, C_2, D_{22}) is a realization of a transfer matrix $M(s)$, it means that $M(s) = C_2(sI - A)^{-1}B_2 + D_{22}$.
- *Constant matrices*, there is no feedback *via* Δ and I/s: $y = D_{22}u$.

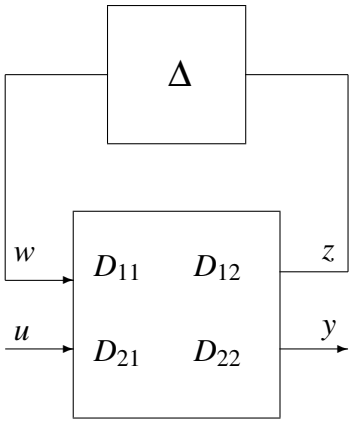

Figure 2.4: LFR representation of an uncertain matrix

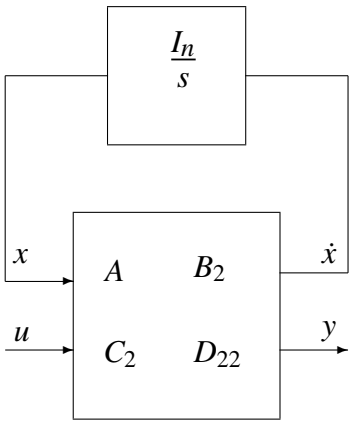

Figure 2.5: LFR representation of a linear system

Software relative to this section

Illustrated functions:

- `lfr` for LFR-object generation and for conversion to LFR-objects.
  - Example 2.1 illustrates how the function `lfr` can be used for generating LFR-objects from the sub-matrices appearing in Figure 2.2.
  - Example 2.2 illustrates conversion of objects belonging to other classes (UMAT, SS, PCK, DOUBLE) into LFR-objects.
  - Example 2.3 shows other possibilities offered by the function `lfr` for generating elementary LFR-objects.
- `lfrdata` recovers numerical data from LFR-objects,
- `uplft` computes $\mathcal{F}_u(M,\Delta)$ (see §2.2),
- `size` displays size information.

Example 2.1 LFR-object generation using the function `lfr`: For example let us build the representation of (2.2) in which `a = 1; b = 2; c = 3`, the transfer gain is 2×2.

$$K_2(\delta_2) = I_2(1\delta_2^2 + 2\delta_2 + 3)$$

the matrices $D_{11}, D_{12}, D_{21}, D_{22}$ can be identified from Figure 2.1 (right hand side).

```
>> D11 = [0 0 1 0;0 0 0 1;0 0 0 0;0 0 0 0];
>> D12 = [0 0;0 0;1 0;0 1];
>> D21 = [1 0 2 0;0 1 0 2];
>> D22 = [3 0;0 3];

>> blk = struct('names',{{'delta_2'}},'desc',[4;4;1;1;1;1;1;2;-1;1;0]);

>> K2 = lfr(D11,D12,D21,D22,blk);
```

The input argument `blk` requires some explanation, see page 165 for details. Briefly, the two first entries of `blk.desc` correspond to the size of the block corresponding to parameter `delta_2`. Other entries are standard for scalar real blocks with non specified variation bounds.

Having such an LFR-object, it is possible to recover the matrices `D11`, `D12`, `D21`, `D22` and `blk` by typing respectively:

```
>> K2a = K2.a;
>> K2b = K2.b;
>> K2c = K2.c;
>> K2d = K2.d;
>> K2blk = K2.blk;
```

For the same purpose we can use `lfrdata`:

```
>> [K2a,K2b,K2c,K2d,K2blk] = lfrdata(K2);
```

The function `size` applied to an LFR-object give the following information

```
>> size(K2)

LFR-object with 2 output(s), 2 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name    Dims  Type   Real/Cplx   Full/Scal   Bounds
 delta_2 4x4   LTI       r           s        [-1,1]
```

In order to conclude this example we can check the result by closing the Δ-loop. For that we shall invoke `uplft`. Let us assume that $\delta_2 = 10$

```
>> valK2 = uplft(K2,{'delta_2'},10);
>> valK2.d

ans =

   123     0
     0   123
```

Computing manually we obtain $K_2(\delta_2 = 10) = 123I_2$ as above. ■

■■■■■■

Example 2.2 Conversion of standard linear systems as SS-objects, PCK-objects and constant matrices can be converted to LFR-objects by invoking the function `lfr`.

```
>> sys0 = rss(5,2,3);
>> [a,b,c,d] = ssdata(sys0);

>> sys1 = lfr(a,b,c,d,'c');
>> sys2 = lfr(sys0);
>> sys3 = lfr(pck(a,b,c,d));
```

Note that `'c'` must be replaced by `'d'` for generating a discrete time system.

Now, let us illustrate the conversion between LFR-objects and UMAT-objects.

```
>> x=ureal('a',2.2,'Range',[1 3]);
>> y=ureal('b',3.3,'Range',[2 4]);
>> z=ureal('c',5.5,'Range',[3 7]);
>> s=ucomplex('d',1,'Radius',2);
>> t=ucomplex('e',i,'Radius',3);
>> A=[x*y+2 z*y;s 1+t*z]

UMAT: 2 Rows, 2 Columns
  a: real, nominal = 2.2, range = [1  3], 1 occurrence
  b: real, nominal = 3.3, range = [2  4], 2 occurrences
  c: real, nominal = 5.5, range = [3  7], 2 occurrences
  d: complex, nominal = 1, radius = 2, 1 occurrence
  e: complex, nominal = 0+1i, radius = 3, 1 occurrence

>> B = lfr(A);
>> size(B)

LFR-object with 2 output(s), 2 input(s) and 0 state(s).
Uncertainty blocks (globally (7 x 7)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
 a     1x1   LTI       r           s       [1,3], nominal=2.2
 b     2x2   LTI       r           s       [2,4], nominal=3.3
 c     2x2   LTI       r           s       [3,7], nominal=5.5
 d     1x1   LTI       c           s       |1 - d| < 2
 e     1x1   LTI       c           s       |0+1i - e| < 3
```

Note that the nominal values are not centered in the ranges of variations (see functions `lfrs` and `normalizelfr`). The converse of the above conversion is `A = lfr2rob(B)`.

■■■■■■

Example 2.3 The function `lfr` permits us to define directly elementary LFR-objects with various features, the syntax is `syslfr = lfr(NAME,TYPE,DIMS,BOUND,BTYPE)` where

- `NAME`: string with name of the uncertainty block
- `TYPE`: composite string defining the uncertainty properties:
  - **nature**:
    - `'lti'` → linear time-invariant (Default)
    - `'ltv'` → linear time-varying
    - `'nl'` → arbitrary nonlinear
    - `'nlm'` → nonlinear memoryless
  - **structure**:
    - `'s'` → scalar block (Default)
    - `'f'` → full block
  - **value**:
    - `'r'` → real-valued (Default)
    - `'c'` → complex-valued

  (*e.g.*, `'ltifc'` means LTI full complex block)
- `DIMS`: dimension of uncertainty block, *e.g.*, `DIMS = [1,2]` or `DIMS = 2` (for 2×2).
- `BOUND`: quantitative information about the uncertainty:
  - **minmax-bounded** uncertainty (for real scalar uncertainties): set `BOUND = [min,max]` or `BOUND = max` (if `min = - max`).
  - **frequency weighted** bound: `BOUND` is the SISO system $W(s)$ for frequency-weighted bounds: $|\Delta(j\omega)| < |W(jw)|$ (*e.g.*, `BOUND = ltisys(-1,1,1,0)`)
  - **sector-bounded** uncertainty: set `BOUND = [a,b]` for uncertainties valued in the sector $\{a,b\}$
  - **disc-bounded** uncertainty (for complex scalar uncertainties): set `BOUND = [a,b]` for uncertainties valued in the disc of center a (possibly complex) and radius b.
- `BTYPE`:
  - `'minmax'` → min/max-values bound

- `'freq'` → frequency dependent bound
- `'sec'` → sector bounded
- `'disc'` → disc bounded

Let us consider three examples. First, generation of a real scalar uncertainty repeated 4 times, the uncertain parameter `x` varies in the interval `[2 8]`.

```
>> X = lfr('x','ltisr',4,[2 8],'minmax');
>> size(X)

LFR-object with 4 output(s), 4 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
 x     4x4   LTI       r           s        [2,8]
```

Then, a complex scalar uncertainty `y` in the disc of center `1+i` and radius `2.2`.

```
>> Y = lfr('y','ltisc',1,[1+i 2.2],'disc');
>> size(Y)

LFR-object with 1 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (1 x 1)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
 y     1x1   LTI       c           s        |1+1i - y| < 2.2
```

Finally, generation of a full complex block of size 2×4 bounded by the filter $W(s) = \frac{1}{1+0.1s}$.

```
>> Z = lfr('z','ltifc',[2 4],ltisys(-10,-10,1,0),'freq');
>> size(Z)

LFR-object with 2 output(s), 4 input(s) and 0 state(s).
Uncertainty blocks (globally (2 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
 z     2x4   LTI       c           f        freq. dependent
```

2.2 Feedback loops

The general case of Figure 2.2 is often represented in a simplified way, the meaning of notations has to be understood according to the context. Often, Δ represents both parameter variations and integrations, see Comment 2.1.3 and Figure 2.3 on page 20. We shall also consider parameter dependent state-space representations (see §2.2.2) or parameter dependent transfer function matrices (see §2.2.3).

2.2.1 Star product and fractional transformations

Definition of $\mathcal{F}_u(M,\Delta)$. Let us consider the $M-\Delta$ form of Figure 2.3. The transfer between u and y that is obtained after closing the loop Δ is denoted $\mathcal{F}_u(M,\Delta)$:

$$\mathcal{F}_u(M,\Delta) = M_{21}\Delta(I-M_{11}\Delta)^{-1}M_{12}+M_{22} \tag{2.7}$$

where

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

$\mathcal{F}_u()$: upper Linear Fractional Transformation

Definition of $\mathcal{F}_l(M,K)$. Let us consider the feedback loop of Figure 2.3. After closing the loop $y = Ku$, the transfer "seen from Δ" is denoted $\mathcal{F}_l(M,K)$:

$$\mathcal{F}_l(M,K) = M_{12}K(I-M_{22}K)^{-1}M_{21}+M_{11} \tag{2.8}$$

$\mathcal{F}_l()$: lower Linear Fractional Transformation

The star product. This product consists of replacing the Δ block by an LFR-object. Examples:

- It can be useful when an LFR-object is generated without taking parameter normalization into account. Assume that the admissible variation of a real uncertain parameter δ_1 are $\delta_1^- \le \delta_1 \le \delta_1^+$. Therefore, if we want variation between -1 and $+1$ (as recommended for using μ-analysis) δ_1 must be replaced by

$$\delta_1 \to \frac{\delta_1^+ + \delta_1^1}{2} + \frac{\delta_1^+ - \delta_1^1}{2}\delta_1' \tag{2.9}$$

  A similar transformation must also be applied to the other real uncertain parameters. Clearly, that means that the δ_i's of the original Δ-matrix must be replaced by an LFR-object as depicted in Figure 2.6.

- An other case where such a transformation is useful will be considered in Chapter 6: *continuum of the linearized models of a non-linear system modelled as an LFR-object*. Briefly, such a model can be computed ignoring first the dependency of the parameters on the equilibrium surface. In a second step, a subset of parameters is expressed as a function of the other parameters in order to take into account the equilibrium surface constraint. The parameters of the first subset become in that way LFR-objects depending on the other ones. Here again the star product can be used.

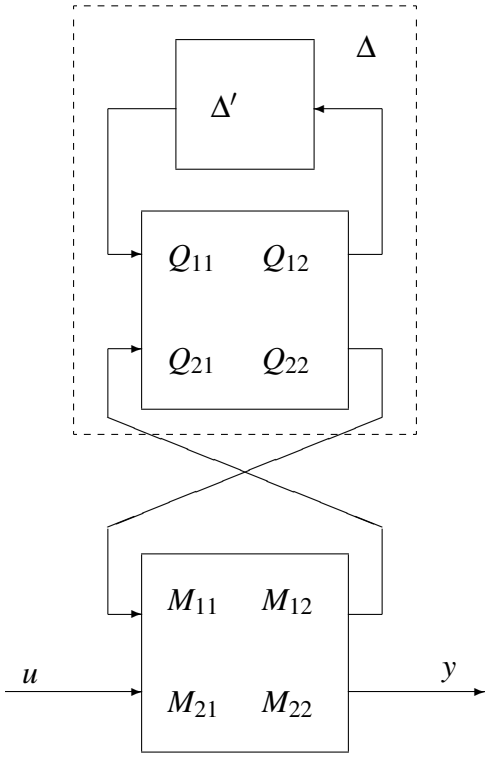

Figure 2.6: The star product

Let us consider two LFR-objects as in Figure 2.3 denoted respectively $\mathcal{F}_u(M,\Delta)$ and $\mathcal{F}_u(Q,\Delta')$ where

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$$

The star product is equal to $\mathcal{F}_u(R,\Delta')$ where the sub-matrices of R in a natural partitioned form are:

$$\begin{aligned} R_{11} &= Q_{11} + Q_{12}M_{11}(I - Q_{22}M_{11})^{-1}Q_{21}; \\ R_{12} &= Q_{12}(I - M_{11}Q_{22})^{-1}M_{12}; \\ R_{21} &= M_{21}(I - Q_{22}M_{11})^{-1}Q_{21}; \\ R_{22} &= M_{22} + M_{21}Q_{22}(I - M_{11}Q_{22})^{-1}M_{12}; \end{aligned}$$

2.2.2 Parameter dependent state-space representations

The loop Δ of Figure 2.2 can be considered as being closed. In that case we obtain the parameter dependent matrices $(A(\Delta), B(\Delta), C(\Delta), D(\Delta))$ of the system. These objects are used in the context of multi-model synthesis.

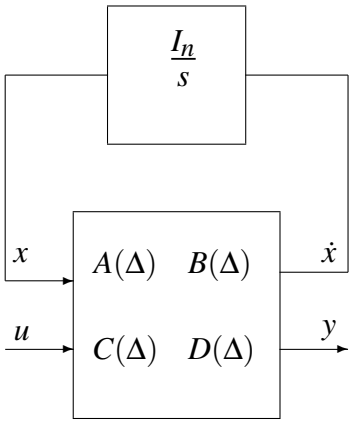

Figure 2.7: Parameter dependent state-space representation

After closing the Δ loop of Figure 2.2, the uncertain matrices of figure 2.7 can be written:

$$\begin{aligned} A(\Delta) &= A + B_1\Delta(I - D_{11}\Delta)^{-1}C_1 \\ B(\Delta) &= B_2 + B_1\Delta(I - D_{11}\Delta)^{-1}D_{12} \\ C(\Delta) &= C_2 + D_{21}\Delta(I - D_{11}\Delta)^{-1}C_1 \\ D(\Delta) &= D_{22} + D_{21}\Delta(I - D_{11}\Delta)^{-1}D_{12} \end{aligned} \tag{2.10}$$

However, it is generally more convenient to use the parameter dependent form of

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}(\Delta) \tag{2.11}$$

as in (3.5) on page 72, because in this case, the complexity of Δ is not repeated in A, B, C and D as above but is repeated only once.

2.2.3 Transfer function matrices

In the context of the μ-analysis (see also page 57), the I/s loop is considered as being closed. After closing the I/s loop of Figure 2.2, the transfers M_{ij} of Figure 2.8 can be written as:

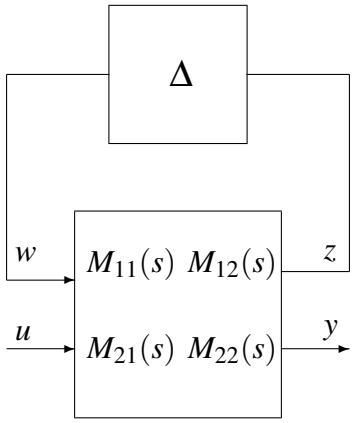

Figure 2.8: $M-\Delta$ form

$$\begin{aligned} M_{11}(s) &= C_1(sI-A)^{-1}B_1+D_{11} \\ M_{12}(s) &= C_1(sI-A)^{-1}B_2+D_{12} \\ M_{21}(s) &= C_2(sI-A)^{-1}B_1+D_{21} \\ M_{22}(s) &= C_2(sI-A)^{-1}B_2+D_{22} \end{aligned} \tag{2.12}$$

For μ-analysis, the feedback loop is closed and s is fixed to $s = j\omega_i$.

Software relative to this section

Illustrated functions:

- `uplft` computes $\mathcal{F}_u(M,\Delta)$ (see Example 2.4), note that there is no function available for computing $\mathcal{F}_l(M,\Delta)$ as this operation is almost treated by the function `feedback` (see Example 2.5).
- `eval` might be used instead of `uplft` (see the end of Example 2.4) but it is much more powerful:
  - it is also an advanced star product, for example useful for applying the Tustin's transformation (see Example 2.6),
  - it permits us to manipulate LFR-objects in which sub-matrices are themselves LFR-objects (see Example 2.7).
- `rlfr` random LFR-object generation (see Example 2.4).

Example 2.4 We have already briefly illustrated the use of the function `uplft` in Example 2.1. This function corresponds to $\mathcal{F}_u(M,\Delta)$ as defined in Equation (2.7). In fact it is more general as it permits us to fix some of the δ_i's in Δ, while other parameters remain uncertainties, leading to a new matrix Δ of lower size containing only the non fixed parameters on its diagonal. For an illustrative purpose let us generate a random LFR-object using the function `rlfr`. This random object has 2 inputs, 3 outputs, 4 states, a first parameter δ_1 repeated 5 times (name `'d_1'`), a second parameter δ_2 repeated 6 times (name `'d_2'`)

```
>> G = rlfr(4,2,3,5,6,'d_');
```

LFR-object corresponding to $s = 10j$ and `'d_2'` $= 20$:

```
>> Gred1 = uplft(G,{'1/s','d_2'},[1/(10*j),20]);
```

Resulting in

```
>> size(G)

LFR-object with 2 output(s), 3 input(s) and 4 state(s).
```

```
Uncertainty blocks (globally (11 x 11)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds
 d_1    5x5    LTI        r            s          [-1,1]
 d_2    6x6    LTI        r            s          [-1,1]

>> size(Gred)

LFR-object with 2 output(s), 3 input(s) and 0 state(s).
Uncertainty blocks (globally (5 x 5)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds
 d_1    5x5    LTI        r            s          [-1,1]
```

Instead of using the function `uplft`, it is also possible to use the function `eval` that evaluates and LFR-object from values in the workspace. For example

```
>> Int = 1/(10*j);
>> d_2 = 20;
>> Gred2 = eval(G);
```

`Gred2` is the same object as `Gred1`.

■■■■■■

Example 2.5 Now let us illustrate the transformation that corresponds to Equation (2.8).

```
>> K = rand(3,2);
>> Gfb = zeros(0,2)*feedback(G,K,1)*zeros(3,0);
```

Resulting in

```
>> size(Gfb)

LFR-object with 0 output(s), 0 input(s) and 4 state(s).
Uncertainty blocks (globally (11 x 11)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds
 d_1    5x5    LTI        r            s          [-1,1]
 d_2    6x6    LTI        r            s          [-1,1]
```

The matrix K plays here the role of a proportional feedback. K could also be a dynamic system (dynamic feedback) or an LFR-object (scheduled possibly dynamic feedback gain). It is worth noting that the new system has no longer inputs and outputs. In other words, this system is ready for μ-analysis (this transformation is already performed inside the functions for μ-analysis: `lfr2mu`, `lfr2mubnd`, `lfr2mustab` and `lfr2mussv`).

■■■■■■

Example 2.6 Illustration of `eval` used as an elaborated star product. Instead of replacing globally the matrix Δ as in Figure 2.6, it is possible to replace selected entries of Δ. In this example, an LFR-object `M1` depending on `d1` and `d2` is defined, then, `d2` is replaced by a function of `d1`.

```
>> M1 = rlfr(4,2,3,4,4,'d');
>> size(M1)

LFR-object with 2 output(s), 3 input(s) and 4 state(s).
Uncertainty blocks (globally (8 x 8)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
 d1    4x4   LTI       r           s          [-1,1]
 d2    4x4   LTI       r           s          [-1,1]
```

In order to define `d2` as a function of `d1`, these elementary LFR-object must be defined in the workspace

```
>> d1 = lfr('d1','ltisr');
>> d2 = lfr('d2','ltisr');
>> d2 = d1^2;
>> M2 = eval(M1);
>> M2.blk.names

ans =

    '1/s'    'd1'
```

Next step the Tustin's transformation is applied to `M2`. The operators $1/s$ and $1/z$ are denoted by default `Int` and `Delay`. First, $1/z$ is defined in the workspace and then, the Tustin's transformation is applied:

$$s^{-1} = 0.1\,\frac{1+z^{-1}}{1-z^{-1}}$$

```
>> Delay = lfr(0,1,1,0,'d');
>> Int =0.1 * (1+Delay) / (1-Delay);
>> M3  = eval(M2);
>> M3.blk.names

ans =

    '1/z'    'd1'
```

■■■■■■

Example 2.7 Illustration of `eval` used in case sub-matrices of an LFR-object are themselves LFR-object's

```
>> A = rlfr(0,4,4,2,2,'a');
>> B = rlfr(0,4,2,2,2,'b');
>> C = rlfr(0,2,4,2,2,'c');
>> D = zeros(2,2);
>> blk = struct('names',{{'1/s'}},'desc',[4;4;0;1;1;1;0;0;0;0]);
>> sys = lfr(A,B,C,D,blk);
>> size(sys)

LFR-object with 2 output(s), 2 input(s) and 4 state(s).
Uncertainty blocks (globally (0 x 0)):
```

In order to pull out the matrices Δ of the sub-matrices, it suffices to invoke the function `eval`:

```
>> sys = eval(sys);
>> size(sys)

LFR-object with 2 output(s), 2 input(s) and 8 state(s).
Uncertainty blocks (globally (12 x 12)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
 a1    2x2   LTI       r           s          [-1,1]
 a2    2x2   LTI       r           s          [-1,1]
 b1    2x2   LTI       r           s          [-1,1]
 b2    2x2   LTI       r           s          [-1,1]
 c1    2x2   LTI       r           s          [-1,1]
 c2    2x2   LTI       r           s          [-1,1]
```

As expected, the uncertainties of the sub-matrices are now in the global matrix Δ. Note that the function `abcd2lfr` can also be used: `sys = abcd2lfr([A B;C D],4);`.

2.3 Normalization

For μ-analysis, it is generally better to normalize the parameter variations between -1 and $+1$. This problem is considered in § 2.3.1 as a special case of a more general transformation applied to the matrix Δ. In § 2.3.2 is considered the problem of normalizing when the nominal value is not centered in the variation range. The problem of "inverting" a non-invertible LFR-object is also somewhat related to normalization, *i.e.*, to nominal values, it is treated in § 2.3.3.

2.3.1 Standard normalization

For μ-analysis, it is generally better to normalize the parameter variations between -1 and $+1$. For example if an LFR-object depends on two parameters δ_1 and δ_2 with

$$\begin{array}{l}\delta_1 \in [\delta_1^- \ \ \delta_1^+] \\ \delta_2 \in [\delta_2^- \ \ \delta_2^+]\end{array}$$

normalizing consists of replacing δ_1, δ_2 by δ'_1, δ'_2 where

$$\begin{array}{l}\delta_1 = \frac{\delta_1^+ + \delta_1^-}{2} + \frac{\delta_1^+ - \delta_1^-}{2}\delta'_1 \\ \delta_2 = \frac{\delta_2^+ + \delta_2^-}{2} + \frac{\delta_2^+ - \delta_2^-}{2}\delta'_2\end{array}$$

because in this case, the variations of $\delta_1 \in [\delta_1^- \ \ \delta_1^+]$ and $\delta_2 \in [\delta_2^- \ \ \delta_2^+]$ are equivalent to

$$\begin{array}{l}\delta'_1 \in [-1 \ \ +1] \\ \delta'_2 \in [-1 \ \ +1]\end{array}$$

More generally, normalization consists of replacing Δ by $P\Delta' Q + R$. The expression of P, Q and R in the above example is

$$P = \begin{bmatrix} I_{n_1} & 0 \\ 0 & I_{n_2} \end{bmatrix}; Q = \begin{bmatrix} I_{n_1}\frac{\delta_1^+ - \delta_1^-}{2} & 0 \\ 0 & I_{n_2}\frac{\delta_2^+ - \delta_2^-}{2} \end{bmatrix}; R = \begin{bmatrix} I_{n_1}\frac{\delta_1^+ + \delta_1^-}{2} & 0 \\ 0 & I_{n_2}\frac{\delta_2^+ + \delta_2^-}{2} \end{bmatrix}$$

which has a trivial extension in the general case.

The following lemma computes the new equivalent LFR realization in which Δ is replaced by Δ'.

Lemma 2.3.1 *If the matrix Δ is replaced by $P\Delta' Q + R$, we have*

$$\mathcal{F}_u\left(\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \Delta\right) = \mathcal{F}_u\left(\begin{bmatrix} M'_{11} & M'_{12} \\ M'_{21} & M'_{22} \end{bmatrix}, \Delta'\right)$$

provided that

$$\begin{aligned} M'_{11} &= Q(I-M_{11}R)^{-1}M_{11}P \\ M'_{12} &= Q(I-M_{11}R)^{-1}M_{12} \\ M'_{21} &= M_{21}P+M_{21}R(I-M_{11}R)^{-1}M_{11}P \\ M'_{22} &= M_{22}+M_{21}R(I-M_{11}R)^{-1}M_{12} \end{aligned}$$

Proof. The matrix Δ is replaced by $P\Delta'Q+R$. Let us consider $\mathcal{F}_u(M,\Delta)$ as in (2.7):

$$y = (M_{22}+M_{21}\Delta(I-M_{11}\Delta)^{-1}M_{12})u$$

with z and w defined as input and output of Δ (see Figure 2.8), we have

$$\begin{aligned} z &= M_{11}w+M_{12}u \\ y &= M_{21}w+M_{22}u \\ w &= \Delta z \end{aligned} \tag{2.13}$$

We shall split $w=\Delta z$ considering $\Delta = P\Delta'Q+R$ as follows

$$\begin{aligned} w &= Pw'+Rz \\ z' &= Qz \\ w' &= \Delta' z' \end{aligned}$$

Therefore, after substitution of w

$$z = (I-M_{11}R)^{-1}M_{11}Pw'+(I-M_{11}R)^{-1}M_{12}u$$

it follows that

$$z' = Q(I-M_{11}R)^{-1}M_{11}Pw'+Q(I-M_{11}R)^{-1}M_{12}u$$

and

$$y = (M_{21}P+M_{21}R(I-M_{11}R)^{-1}M_{11}P)w'+(M_{22}+M_{21}R(I-M_{11}R)^{-1}M_{12})u$$

Finally, Equation (2.13) is replaced by:

$$\begin{aligned} z' &= Q(I-M_{11}R)^{-1}M_{11}Pw'+Q(I-M_{11}R)^{-1}M_{12}u \\ y &= (M_{21}P+M_{21}R(I-M_{11}R)^{-1}M_{11}P)w'+(M_{22}+M_{21}R(I-M_{11}R)^{-1}M_{12})u \\ w' &= \Delta' z' \end{aligned} \tag{2.14}$$

comparing term by term (2.13) and (2.14) we obtain

$$\begin{aligned} M'_{11} &= Q(I-M_{11}R)^{-1}M_{11}P \\ M'_{12} &= Q(I-M_{11}R)^{-1}M_{12} \\ M'_{21} &= M_{21}P+M_{21}R(I-M_{11}R)^{-1}M_{11}P \\ M'_{22} &= M_{22}+M_{21}R(I-M_{11}R)^{-1}M_{12} \end{aligned}$$

as stated in the lemma. ■

Note that Lemma 2.3.1 can also be used in several other cases, for example when the order of the parameters is modified.

2.3.2 Normalization with non-centered nominal value

It is often necessary to consider parameters for which the nominal value is not centered in the range of variations. For example consider $V \in [V_{\min}, V_{\max}]$, with nominal value $V_{\text{nom}} = 1/2(V_{\min} + V_{\max})$. If V is replaced by $1/V$, the nominal value of this new parameter is $1/V_{\text{nom}}$, it is not in the middle of the new variation range $[1/V_{\max}, 1/V_{\min}]$.

So, for recentering the nominal value when an LFR-object is being normalized, we shall use a monotonous function $f(\delta)$ such that

$$\begin{array}{rcl} f(\delta_{\min}) & = & -1 \\ f(\delta_{\max}) & = & 1 \\ f(\delta_{\text{nom}}) & = & 0 \end{array}$$

Such a function f was reported by J. Cockburn in the comments concerning Reference [31]:

$$\begin{array}{rcl} \alpha & = & (\delta_{\min}\,\delta_{\text{nom}} + \delta_{\max}\,\delta_{\text{nom}} - 2\,\delta_{\min}\,\delta_{\max})/(\delta_{\max} - \delta_{\min}) \\ \beta & = & (2\,\delta_{\text{nom}} - \delta_{\min} - \delta_{\max})/(\delta_{\max} - \delta_{\min}) \\ \gamma & = & \delta_{\text{nom}} \end{array}$$

$$f(\delta) = \frac{\alpha + \beta x}{1 + \gamma\delta} \tag{2.15}$$

This function can be rewritten in such a way that δ appears only once so that replacing δ by $f(\delta)$ will not encrease the size of the considered LFR-object.

It is more difficult than in the previous section to identify the transformations to apply to the matrices M_{ij} for such a normalization. However, a very simple trick can be used: it suffices to define the new form of the parameters in the workspace ($\delta = f(\delta)$) and then to invoke the function `eval`.

2.3.3 Use of a dummy parameter for inversion

In this section introduces a trick proposed by Hecker and Varga in [31] for inversion of LFR-objects when the nominal value is zero or singular (MIMO case). Let us consider an LFT

$$M(\Delta) = \mathcal{F}_u\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}, \Delta\right) \tag{2.16}$$

the inverse is given by (see (7.7))

$$M(\Delta)^{-1} = \mathcal{F}_u\left(\begin{bmatrix} A - BD^{-1}C & BD^{-1} \\ -D^{-1}C & D^{-1} \end{bmatrix}, \Delta\right) \tag{2.17}$$

so, for inverting (2.16) with the above classical formula it is required that D is invertible. In order to introduce a trick that will permit us to avoid this difficulty, let us consider a very simple example (generalization straightforward).

Let a be a real parameter. We can write a in LFT form as

$$a = \mathcal{F}_u\left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, a\right)$$

but $1/a$ cannot be written directly in LFT form because the corresponding "D" matrix is zero, so, "D" cannot be inverted. The normalization of §2.3.1 can be used to move the nominal value of a around some non-zero value a_0. It means that a is replaced by the user by $a_0 + \delta_a$ before inversion. Then,

$$a = \mathcal{F}_u\left(\begin{bmatrix} 0 & 1 \\ 1 & a_0 \end{bmatrix}, \delta_a\right)$$

this LFT form of a can be inverted using (2.17).

It is suggested here to proceed in a different way by introducing a dummy parameter (that will be denoted x): a is replaced by

$$a' = (a+1)x - 1 \tag{2.18}$$

Clearly, a' has the following properties

- For $x = 1$, we have $a' = a$.
- a' has a simple LFT form:

$$a' = \mathcal{F}_u\left(\left[\begin{array}{cc|c} 0 & 0 & 1 \\ 1 & 0 & 0 \\ \hline 1 & 1 & -1 \end{array}\right], \begin{bmatrix} x & 0 \\ 0 & a \end{bmatrix}\right) \tag{2.19}$$

- For $a = x = 0$, $a' = -1$ is non-zero so, can be inverted (or equivalently, the "D" matrix in the above LFT is invertible)..

These remarks justify the following strategy for inverting an LFR object: use formula (2.17) if the "D" matrix is invertible, otherwise, introduce the dummy parameter x as in (2.18), transform the result to an LFT form like in (2.19)[1] and then, invert using (2.17). Evaluation at $x = 1$ will eliminate the dummy parameter.

[1] In the matrix case, the 1's of (2.19) become identity matrices, a is the square LFR-object to be inverted and so on.

When an LFR-object is (partly) evaluated (functions `uplft` or `eval`) or normalized (function `normalizelfr`), if there is no more division by zero, it becomes possible to close the loop through xI with $x = 1$. In that way all the block corresponding to the dummy parameter will vanish. In fact, the toolbox reduces this block to its minimum size if it cannot vanish completely.

Software relative to this section

Illustrated functions:

- `normalizelfr`: normalization of parameter variations see Example 2.8.
- `set`: for modifying the block structure of an LFR-object, here. In Examples 2.8 and 2.9 the parameter variation bounds are modified.
- `unormalize` and `actualval`: see Example 2.9.

Example 2.9 shows how the "constant block" may vanishes using `normalizelfr` after modifying the parameter variation bounds (using `set`) so that the nominal value becomes non-zero.

Example 2.8 An LFR-object is generated first without lower and an upper bound for parameter variation (using `rlfr`):

```
>> S = rlfr(4,2,2,4,1,1,1,'d');
>> size(S)

LFR-object with 2 output(s), 2 input(s) and 4 state(s).
Uncertainty blocks (globally (7 x 7)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds
 d1     4x4    LTI        r            s          [-1,1]
 d2     1x1    LTI        r            s          [-1,1]
 d3     1x1    LTI        r            s          [-1,1]
 d4     1x1    LTI        r            s          [-1,1]
```

Bounds are defined for the four parameters using the function `set`

```
>> set(S,{'d1'},{[-2,6]},{'minmax'});
>> set(S,{'d2'},{[-6,2]},{'minmax'});
>> set(S,{'d3'},{[-2,2]},{'minmax'});
>> set(S,{'d4'},{[-1,1]},{'minmax'});

>> size(S)

LFR-object with 2 output(s), 2 input(s) and 4 state(s).
```

```
Uncertainty blocks (globally (7 x 7)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
 d1    4x4   LTI       r           s          [-2,6]
 d2    1x1   LTI       r           s          [-6,2]
 d3    1x1   LTI       r           s          [-2,2]
 d4    1x1   LTI       r           s          [-1,1]
```

This object has four uncertain parameters with admissible variations

$$\begin{aligned} -2 \leq \texttt{d1} \leq +6 \\ -6 \leq \texttt{d2} \leq +2 \\ -2 \leq \texttt{d3} \leq +2 \\ -1 \leq \texttt{d4} \leq +1 \end{aligned}$$

Normalization means that we replace `d1` by an expression of the form `(d1_max + d1_min)/2 + d1*(d1_max - d1_min)/2` (similar for the other parameters) in such a way that admissible variations become between -1 and $+1$. For that purpose, it remains to invoke the function `normalizelfr`:

```
>> Snorm = normalizelfr(S,{'d1','d2','d3','d4'});
>> size(Snorm)

LFR-object with 2 output(s), 2 input(s) and 4 state(s).
Uncertainty blocks (globally (7 x 7)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
 d1    4x4   LTI       r           s          [-1,1]
 d2    1x1   LTI       r           s          [-1,1]
 d3    1x1   LTI       r           s          [-1,1]
 d4    1x1   LTI       r           s          [-1,1]
```

It can be checked that `uplft(S,'d1','d2','d3','d4',[6,-6,2,-1])` and `uplft(Snorm,'d1','d2','d3','d4',[1,-1,1,-1])` give the same result.

■■■■■■

Example 2.9 This example illustrates how non-invertible LFR-objects can be "inverted" by using the "constant block" trick. It is also shown how normalization removes the "constant block" and how to retrieve the original LFR-object after normalization. Let us consider a random LFR-object with "*D*" matrix set to zero.

```
>> S = rlfr(5,3,3,2,2,'d');
>> S.d = zeros(3,3);

>> invS = inv(S);
>> size(invS)

LFR-object with 3 output(s), 3 input(s) and 5 state(s).
Dimension of constant block in uncertainty matrix: 3
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
 d1    2x2   LTI       r           s          [-1,1]
 d2    2x2   LTI       r           s          [-1,1]
```

The second line displayed by the function size shows that the dummy parameter is repeated 3 times in a block named "constant block". This block vanishes generally when functions like `uplft`, `eval`, `normalizelfr` are invoked (of course, if there is no more division by zero), for example if we change the nominal value to 4 (not in the middle of the interval $[2\ \ 8]$) and normalize:

```
>> set(invS,{'d1','d2'},{[2 8 4],[2 8 4]},{'minmax','minmax'});
>> size(invS)

LFR-object with 3 output(s), 3 input(s) and 5 state(s).
Dimension of constant block in uncertainty matrix: 3
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
 d1    2x2   LTI       r           s       [2,8], nominal=4
 d2    2x2   LTI       r           s       [2,8], nominal=4

>> invS = normalizelfr(invS);
>> size(invS)

LFR-object with 3 output(s), 3 input(s) and 5 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal    Bounds
 d1    2x2   LTI       r           s        [-1,1]
 d2    2x2   LTI       r           s        [-1,1]
```

as expected the "constant block" block has disappeared because the inversion is feasible at the new nominal values of the parameters.

Note that it is possible to compute the actual values from normalized ones. For example:

```
par_nom = [-0.5 0.5];
par_act = actualval(invS,{'d1','d2'},par_nom)

par_act =

  2.8571    5.6000
```

These values correspond to $f^{-1}(-0.5)$ and $f^{-1}(0.5)$ where f is defined as in (2.15).

The above normalization can be undone as follows

```
S2 = unnormalize(invS);
S3 = set(inv(S),{'d1','d2'},{[2 8 4],[2 8 4]},{'minmax','minmax'});
distlfr(S2,S3)

ans =

  2.4107e-14
```

2.4 Well-posedness and non-singularity

2.4.1 Well-posedness

LFR-objects are given in a feedback form as in (2.7):

$$\mathcal{F}_u(M,\Delta) = M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} + M_{22}$$

Unless $M_{11} = 0$, the matrix $(I - M_{11}\Delta)$ cannot be inverted for all values of Δ. In fact, it is sufficient that invertibility is feasible for all relevant values of Δ: If for example, parameter variations are normalized between -1 and $+1$, the matrix $(I - M_{11}\Delta)$ must be invertible in the unit ball.

Δ satisfies a constraint of structure (it is block-diagonal), the condition for invertibility is $\mu(M_{11}) < 1$ by definition of μ. This property of invertibility is called "'well-posedness". So we have

Definition 2.4.1 *The LFR-object $\mathcal{F}_u(M,\Delta) = M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} + M_{22}$ is said to be well-posed in the unit ball if and only if*

$$\mu(M_{11}) < 1 \tag{2.20}$$

The μ "measure" corresponds to the inverse of the norm of the smallest value of Δ (with given structure) at which the invertibility of $(I - M_{11}\Delta)$ is not satisfied. Therefore, the radius of the ball in which $(I - M_{11}\Delta)$ can be inverted is $1/\mu(M_{11})$.

Definition 2.4.2 *The well-posedness radius of the LFR-object $\mathcal{F}_u(M,\Delta) = M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} + M_{22}$ is defined by*

$$\textit{Well-posedness radius} = \frac{1}{\mu(M_{11})} \tag{2.21}$$

The standard operations on LFR-objects are defined in § 7.1 (page 147). Keeping in mind that an LFR-object is well-posed if its "$I - M_{11}\Delta$" matrix can be inverted, it is clear from the formulae of page 147 that the following operations

- addition (see (7.2))
- multiplication (see (7.3))
- vertical concatenation (see (7.4))

- horizontal concatenation (see (7.5))
- juxtaposition (see (7.6))

are such that the resulting well-posedness radius is the smallest of the ones of both factors. For

- transposition (see (7.1))
- scalar multiplication
- conjugation
- real part (see (7.11))
- imaginary part (see (7.12))
- and the transformation $[M\ \overline{M}] \rightarrow [\Re(M)\ \Im(M)]$ (see (7.15))

the resulting well-posedness radius is not modified (see Lemma 7.1.3, page 153 for the last three items).

2.4.2 Non-singularity

The resulting well-posedness radius is not related to the original one in the following cases:

- inversion (see (7.7))
- null space computation, see § 7.1.8

In the second case (null space computation) well-posedness is also a problem of inversion. Equation (7.7) of page 149 is:

$$\mathcal{F}_u(M,\Delta)^{-1} = \mathcal{F}_u\left(\left[\begin{array}{c|c} M_{11}-M_{12}M_{22}^{-1}M_{21} & M_{12}M_{22}^{-1} \\ \hline -M_{22}^{-1}M_{21} & M_{22}^{-1} \end{array}\right], \Delta\right)$$

So, the well-posedness radius of $M(\Delta)^{-1}$ is

$$\frac{1}{\mu(M_{11}-M_{12}M_{22}^{-1}M_{21})}$$

in fact it defines a ball in which $M(\Delta)$ is non-singular. Therefore, it is natural to define the non-singularity radius as follows.

Definition 2.4.3 *The non-singularity radius of the square LFR-object (non-singular for $\Delta = 0$) $\mathcal{F}_u(M,\Delta) = M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} + M_{22}$ is defined by*

$$\textit{Non-singularity radius} = \frac{1}{\mu(M_{11} - M_{12}M_{22}^{-1}M_{21})} \tag{2.22}$$

Uniform controllability. This definition can be used for example for checking the uniform controllability of a system $(A(\Delta), B(\Delta))$ in the unit ball. Consider the matrix

$$M(\Delta, \lambda^R, \lambda^I) = \left[\begin{array}{cc} A(\Delta) - (\lambda^R + j\lambda^I)I & B(\Delta) \end{array} \right]$$

in which λ^R and λ^I are bounded (in the area of the complex plane containing the dominant modes of the system), and then normalized between -1 and $+1$. $M(\Delta, \lambda^R, \lambda^I)$ can be modeled as an LFR-object $M(\Delta')$ with a new "Δ" matrix

$$\Delta' = \left[\begin{array}{ccc} \Delta & 0 & 0 \\ 0 & \lambda^R I_n & 0 \\ 0 & 0 & \lambda^I I_n \end{array} \right]$$

All entries of Δ' being real we can apply conjugate transposition to $M(\Delta')$ so that the non-singularity of $M(\Delta')$ (non-square object) becomes equivalent to the non-singularity of $M(\Delta')\overline{M}(\Delta')^T$ (square object compatible with Definition 2.22).

Finally we have a controllability test that is $(A(\Delta), B(\Delta))$ is uniformly controllable in the unit ball if and only if the non-singularity radius of $M(\Delta')\overline{M}(\Delta')^T$ is larger than one.

We can also check the non-singularity of the matrix

$$[B(\Delta)\ A(\Delta)B(\Delta)\ A^2(\Delta)B(\Delta)\ \ldots]$$

(not very reliable on account of the powers of $A(\Delta)$.

Software relative to this section

Illustrated functions:

- `wp_rad` computes the well-posedness radius, see Example 2.10,
- `ns_rad` computes the non-singularity radius, see Example 2.10.

Example 2.10 The function `lfrs` will be presented in the next section, it is used here for generating an LFR-object with *a priori* known well-posedness radius and non-singularity radius: The following matrix `M` is not well-posed for $b+c=2$ and the corresponding worst case is $b=c=1$.

```
lfrs a b c d
M = [(1+a)/(2-b-c) 2;2*a 3+d];
```

Well-posedness analysis:

```
>> [rad_min,rad_max,pertnames,pert] = wp_rad(M)

rad_min =

    1.0000

rad_max =

    1.0000

pertnames =

    'a'    'b'    'c'    'd'

pert =

    [0]    [1.0000]    [1.0000]    [0]
```

As expected, the smallest value of Δ at which well-posedness is not satisfied is for `b` and `c` equal to 1.

The matrix `M` becomes singular for $a=1/3$ and $b=c=d=-1/3$. Let us check this result:

```
>> [rad_min,rad_max,pertnames,pert] = ns_rad(M)

rad_min =

    0.3331

rad_max =

    0.3333

pertnames =

    'a'    'b'    'c'    'd'

pert =

    [0.3333]    [-0.3333]    [-0.3333]    [-0.3333]
```

2.5 Object-oriented realization of Linear Fractional Representations

With a software such as MATLAB, linear systems can be handled as matrices. Considering the similarity between LFR-objects and linear systems (compare figures 2.2 and 2.5), naturally, it will also be possible to treat LFR-objects as matrices. In particular, it is possible to add (put in parallel), subtract, multiply (put in series), transpose, invert, compute a feedback, and concatenate LFR-objects (see §7.1 for details).

The first contribution in this direction is in the toolbox for MATLAB described in Terlouw and Lambrechts [51], see also d'Andrea [24]. Similar tools were also developed for Maple in Varga and Looye [53, 52].

The mechanism behind these manipulations is rather simple. We are going to clarify it by considering the addition of two LFR-objects. Let us consider two objects identical to that of Figure 2.4. They are denoted $\mathcal{F}_u(D',\Delta')$ and $\mathcal{F}_u(D'',\Delta'')$ where

$$D' = \begin{bmatrix} D'_{11} & D'_{12} \\ D'_{21} & D'_{22} \end{bmatrix} ; D'' = \begin{bmatrix} D''_{11} & D''_{12} \\ D''_{21} & D''_{22} \end{bmatrix}$$

The transfers between "u" and "y" are respectively

$$\mathcal{F}_u(D',\Delta') = D'_{21}\Delta'(I - D'_{11}\Delta')^{-1}D'_{12} + D'_{22}$$

and

$$\mathcal{F}_u(D'',\Delta'') = D''_{21}\Delta''(I - D''_{11}\Delta'')^{-1}D''_{12} + D''_{22}$$

Example 1: sum. The sum of $\mathcal{F}_u(D',\Delta')$ and of $\mathcal{F}_u(D'',\Delta'')$ can be written

$$\begin{bmatrix} D'_{21} & D''_{21} \end{bmatrix} \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix} \left(I - \begin{bmatrix} D'_{11} & 0 \\ 0 & D''_{11} \end{bmatrix} \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix} \right)^{-1} \begin{bmatrix} D'_{12} \\ D''_{12} \end{bmatrix} + D'_{22} + D''_{22}$$

so,

$$\begin{aligned} \mathcal{F}_u\left(D',\Delta'\right) + \mathcal{F}_u\left(D'',\Delta''\right) &= \mathcal{F}_u\left(\mathcal{S}\left(D',D''\right), \begin{bmatrix} \Delta' & \mathbf{0} \\ \mathbf{0} & \Delta'' \end{bmatrix}\right) \\ \text{with} \qquad \mathcal{S}\left(D',D''\right) &= \left[\begin{array}{cc|c} D'_{11} & \mathbf{0} & D'_{12} \\ \mathbf{0} & D''_{11} & D''_{12} \\ \hline D'_{21} & D''_{21} & D'_{22}+D''_{22} \end{array}\right] \end{aligned} \tag{2.23}$$

It remains to reorder the δ_i's.

Example 2: product. As above, the product of two LFRs can be written:

$$\begin{aligned} \mathcal{F}_u\left(D',\Delta'\right)\cdot\mathcal{F}_u\left(D'',\Delta''\right) &= \mathcal{F}_u\left(\mathcal{P}\left(D',D''\right),\begin{bmatrix}\Delta' & \mathbf{0}\\ \mathbf{0} & \Delta''\end{bmatrix}\right) \\ \text{with}\qquad \mathcal{P}\left(D',D''\right) &= \left[\begin{array}{cc|c} D'_{11} & D'_{12}D''_{21} & D'_{12}D''_{22}\\ \mathbf{0} & D''_{11} & D''_{12}\\ \hline D'_{21} & D'_{22}D''_{21} & D'_{22}D''_{22}\end{array}\right] \end{aligned} \tag{2.24}$$

It remains as above to reorder the δ_i's.

It is very easy to automate such manipulations. The set of required formulae for the other operations is given in §7.1. It remains to create the basic elements which will be treated as above.

Construction of elementary LFR-objects. Simple objects like I/s or uncertain parameters (for example $a = a_0 + a_1\delta$) are easily realized (it suffices to identify term by term their expression with the one of $\mathcal{F}_u(M,\Delta)$) :

$$\frac{I}{s} = \mathcal{F}_u(\begin{bmatrix}0 & I\\ I & 0\end{bmatrix}, \frac{I}{s})$$

and

$$a = a_0 + a_1\delta = \mathcal{F}_u(\begin{bmatrix}0 & 1\\ a_1 & a_0\end{bmatrix}, \delta)$$

Object-oriented realization. When the above elementary objects are available, it is possible to build any realization provided that it involves matrix operations compatible with LFTs (addition, multiplication, concatenation, inversion, transposition, conjugation, real and imaginary parts, see Appendix §7.1, page 147).

Examples 2.11 and 2.12 illustrate the object-oriented realization. Briefly, the function `lfrs` generates the above elementary objects, after that, the object-oriented realization consists of using standard operations as for building more complex objects. All operation directly applied to LFR-objects can be found in the `@lfr` sub-directory of the toolbox installation directory.

The function `sym2lfr` performs automatically the object oriented realization from a symbolic expression. However, if the considered symbolic expression comes from Maple, it can be very badly conditioned, for example:

$$M = 10^{-40} a\, b\, (2\times 10^{+10} + 10^{+10}\, a + 4\times 10^{+10}\, c\, b)^2 (1 + 6\times 10^{+10}\, c)^2$$

The function `sym2lfr` from version 1.3 treats this problem in an efficient way by normalizing recursively the leading coefficient of each expression between parenthesis before applying the object-oriented realization.

Software relative to this section

Illustrated functions:

- `lfrs` generates elementary 1×1 LFR-objects, see Examples 2.11 and 2.12.
- `sym2lfr` transforms a symbolic expression into an LFR-object, see Example 2.13.
- `distlfr` computes a lower bound of the distance between two LFR-objects (for an upper bound, see `udistlfr`).

These examples show the complementarity of the object-oriented realization and the direct realization from a symbolic object.

Example 2.11 First, let us introduce the function `lfrs`. This function is very convenient for defining elementary 1×1 LFR-objects. It admits three kinds of input arguments (without parenthesis and comas).

- The first group of arguments is a set of strings corresponding to the names of the 1×1 elementary LFR-objects to be created
- Then, an optional string `'real'` (default) or `'complex'` indicates the nature of the elementary objects to be created.
- The last two (three) arguments are optional vectors of *numerical* values giving the minimum, maximum (and nominal) values of real parameters (by default the nominal values are in the middle of intervals). Note that the square brackets (*e.g.*, `[2]`) must be used even for a single value.

```
>> lfrs Int
>> lfrs a b 'real' [1 1] [3 3]
>> lfrs c [1] [3] [2.5]
>> lfrs d 'complex' [0] [2]
>> size(a*b*c*d)

LFR-object with 1 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal   Bounds
```

```
a     1x1   LTI      r          s        [1,3]
b     1x1   LTI      r          s        [1,3]
c     1x1   LTI      r          s        [1,3], nominal=2.5
d     1x1   LTI      c          s        |0 - d| < 2
```

Let us considered the following system

$$y(s) = (\frac{\delta_1^2}{s^2} + \frac{\delta_1\delta_3}{s} + \delta_1^2\delta_3^2)u(s) \tag{2.25}$$

The object-oriented realization technique consists of realizing separately $1/s$, δ_1 and δ_2 and then to combine these realizations in order to build the realization of a more complex system:.

```
>> lfrs Int d1 d3
>> sys1 = d1^2*Int^2 + d1*d3*Int + d1^2*d3^2;
>> size(sys1)

LFR-object with 1 output(s), 1 input(s) and 3 state(s).
Uncertainty blocks (globally (7 x 7)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    4x4   LTI      r           s          [-1,1]
 d3    3x3   LTI      r           s          [-1,1]
```

We shall compare this result to similar results obtained in different ways in Examples 2.12 and 2.13.

■■■■■■

Example 2.12 We solve the same problem as above but using a different form of the transfer between u and y. We shall use:

$$\frac{\delta_1^2}{s^2} + \frac{\delta_1\delta_3}{s} + \delta_1^2\delta_3^2 = \delta_1 \begin{bmatrix} \frac{1}{s} & \delta_1\delta_3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_1\frac{1}{s} \\ \delta_3 \end{bmatrix} \tag{2.26}$$

So we shall have

```
>> lfrs Int d1 d3
>> sys2 = d1*[Int d1*d3]*[1 1;0 1]*[d1*Int ; d3];
```

We can check the size

```
>> size(sys2)
LFR-object with 1 output(s), 1 input(s) and 2 state(s).
Uncertainty blocks (globally (5 x 5)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    3x3   LTI       r           s          [-1,1]
 d3    2x2   LTI       r           s          [-1,1]
```

The Δ matrix is now 5×5 instead of 8×8. Using the function `distlfr` we can check that we have computed equivalent objects:

```
>> distlfr(sys1,sys2)

ans = 0
```

As a conclusion, considering the size of the Δ-block, the object-oriented LFR construction might be more or less efficient. Indeed, the size of this block is clearly equal to the number of times the parameters (uncertain parameters and $1/s$) appear in the symbolic form that is considered (compare Equations (2.25) and (2.26)).

■■■■■■

Example 2.13 The construction proposed in Example 2.12 is treated now using the symbolic approach. The function that is illustrated is `sym2lfr`. This function transforms a symbolic expression into an LFR-object.

```
>> syms Int d1 d3
>> syss = d1*[Int d1*d3]*[1 1;0 1]*[d1*Int ; d3];
>> sys3 = sym2lfr(syss);
```

Note that $1/s$ must be denoted using the symbol `Int`. It remains to check that we have modelled the same object as above:

```
>> distlfr(sys2,sys3)

ans = 0
```

On account of the fact that Maple (behind the Symbolic Toolbox) handles the operations needed for computing the symbolic expression `syss`, we have no more control on the number of times parameters are really repeated. But, using some Maple functions for factorizing we can get an automatic reduction (see § 3.4 page 80) which is useful for very large systems. The symbolic approach will be even more useful, for example, when the original transfer is nonlinear, and when we look for a symbolic expression of the continuum of linearized models (to be transformed into a single LFR-object). In that case differentiation needed for linearizing can be performed by symbolic computation (provided that the equilibrium surface is available in closed form).

2.6 Discussion on minimality and commutativity

We have just established some basic notions in order to be able to introduce the difficult problem of minimality.

At first, let us note that it is essential to obtain models of reasonable size to be able to use them, for example to run a μ-analysis. To calculate the upper bound of μ by resolving LMI systems it is necessary to avoid having parameters repeated too many times (let us say more than 25 times). Experience shows that it is "very easy" to exceed this limit, see for example Varga and Looye [52]. This reference shows that careless modeling of a single coefficient of the state matrix of a system may lead to a Δ block of the respectable size 293×293 (one parameter is repeated 136 times!) and that the size of this block can be divided by ten, without approximation. Without trying to minimize in an optimal way the size of Δ it is absolutely necessary to avoid this kind of situations.

To define the minimality, it is necessary at first to define a notion of *equivalence of models*. Indeed, having this notion, a representation is said to be minimal if there is no equivalent representation with a Δ of "smaller size". In view of the discussion about μ-analysis, one could consider that the relevant definition of "the size of Δ is the dimension of its biggest block". A definition more usually accepted is the total dimension of the matrix Δ (square matrix). It is this measure that we shall adopt.

Strong input / output equivalence is the notion of equivalence of interest (see Doyle *et al* [28] for a discussion about the other types of equivalences). It is defined in the following way.

Definition of the equivalence of two LFR-objects. $\mathcal{F}_u(M'(s),\Delta')$ and $\mathcal{F}_u(M''(s),\Delta'')$ are said to be equivalent if and only if

$$\forall \delta_i,\ \mathcal{F}_u(M'(s),\Delta') = \mathcal{F}_u(M''(s),\Delta'')$$

In fact, as $1/s$ and the parameters δ_i play a role almost identical[2], we shall use the following definition: $\mathcal{F}_u(M'(s),\Delta')$ and $\mathcal{F}_u(M''(s),\Delta'')$ are equivalent if and only

$$\forall \delta_i, \forall s, \mathcal{F}_u(\overline{M}', \mathrm{diag}\{I/s, \Delta'\}) = \mathcal{F}_u(\overline{M}'', \mathrm{diag}\{I/s, \Delta''\})$$

where the matrices $\overline{M}'$ and $\overline{M}''$ are now as in Figure 2.2, that is

$$M(s) = \begin{bmatrix} M_{11}(s) & M_{12}(s) \\ M_{21}(s) & M_{22}(s) \end{bmatrix} \text{ (see (2.12))} \rightarrow \overline{M} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix}$$

[2]With the difference that $1/s$ is not bounded and takes values in $\mathbb{C}$ This difference should be taken into account in the section relative to the approximation of LFR-objects.

These remarks justify the use of simplified notations.

Simplified notations. We shall use the first version of the definition by omitting dependence with regard to s, that is:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \text{ for } \left[\begin{array}{cc|c} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ \hline C_2 & D_{21} & D_{22} \end{array}\right]$$

The Δ block (representing at the same time dynamics and uncertainties, for example $\delta_1 = 1/s$) will have a block-diagonal structure

$$\Delta = \text{diag}\{I_{n_1}\delta_1, \ldots, I_{n_q}\delta_q\}$$

The size of Δ is identical to that of the matrix M_{11}. The matrix M_{11} being $(n \times n)$ it follows:

$$\sum_{i=1,\ldots,q} n_i = n$$

$\mathcal{F}_u(M', \Delta')$ and $\mathcal{F}_u(M'', \Delta'')$ are equivalent if and only if

$$\forall \delta_i, \mathcal{F}_u(M', \Delta') = \mathcal{F}_u(M'', \Delta'')$$

Definition of minimality. An LFT representation is said to be *minimal* if there is no equivalent representation (strong inputs / outputs equivalence) having a block Δ of lower dimension (value of n as above).

There are naturally no uniqueness of minimal representations. In particular we might have $\sum_{i=1,\ldots,q} n_i = n$ for n minimal but with more or less variable values of the n_i's.

Similarity and equivalence. All the references of literature which refer to the notion of equivalence, use, in fact, for calculation another much stricter notion. It is a kind of equivalence defined with a particular class of transformations of the form:

$$T = \text{diag}\{T_1, \ldots, T_q\} \text{ where } T_i \in \mathbf{R}^{n_i \times n_i} \text{ and } \det(T_i) \neq 0 \tag{2.27}$$

where T commutes with Δ, so,

$$T\Delta = \Delta T \text{ and } \Delta T^{-1} = T^{-1}\Delta$$

Applying this transformation to $\mathcal{F}_u(M, \Delta)$ (that will be denoted $\mathcal{F}_u(\mathcal{T}(M), \Delta)$) leads to

$$\mathcal{T}(M) = \begin{bmatrix} T^{-1}M_{11}T & T^{-1}M_{12} \\ M_{21}T & M_{22} \end{bmatrix} \tag{2.28}$$

Lemma 2.6.1 $\mathcal{F}_u(M,\Delta) = \mathcal{F}_u(\mathcal{T}(M),\Delta)$.

Proof.

$$\mathcal{F}_u(\mathcal{T}(M),\Delta) = M_{21}T\Delta(I - T^{-1}M_{11}T\Delta)^{-1}T^{-1}M_{12} + M_{22}$$

Using the commutativity of T and Δ

$$\mathcal{F}_u(\mathcal{T}(M),\Delta) = M_{21}\Delta T(I - T^{-1}M_{11}T\Delta)^{-1}T^{-1}M_{12} + M_{22}$$

moving T and T^{-1} into the parenthesis,

$$\mathcal{F}_u(\mathcal{T}(M),\Delta) = M_{21}\Delta(I - TT^{-1}M_{11}T\Delta T^{-1})^{-1}M_{12} + M_{22}$$

But the commutativity of T and Δ implying the commutativity of T^{-1} and Δ, we have

$$\mathcal{F}_u(\mathcal{T}(M),\Delta) = M_{21}\Delta(I - TT^{-1}M_{11}TT^{-1}\Delta)^{-1}T^{-1}M_{12} + M_{22} = \mathcal{F}_u(M,\Delta)$$

■

Definition of similarity. Considering the above result, $\mathcal{F}_u(M',\Delta')$ and $\mathcal{F}_u(M'',\Delta'')$ will be said to be similar if and only if there is a non-singular matrix T which commutes with Δ, s.t. $\mathcal{T}(M') = M''$.

Use of similarity "to minimize" order. Similarity is used as in the case of standard linear systems, it serves for putting in evidence non controllability or non observability (see §4.2 for a more complete treatment of this approach) to exhibit terms which can be eliminate without modifying the input / output transfer. Let us take an example.

$$\begin{bmatrix} \delta_2\delta_1 \\ \delta_1 \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right], \begin{bmatrix} \delta_1 & 0 & 0 \\ 0 & \delta_1 & 0 \\ 0 & 0 & \delta_2 \end{bmatrix} \right) \tag{2.29}$$

Using

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

we obtain the following similar form

$$\begin{bmatrix} \delta_2\delta_1 \\ \delta_1 \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{array}\right], \begin{bmatrix} \delta_1 & 0 & 0 \\ 0 & \delta_1 & 0 \\ 0 & 0 & \delta_2 \end{bmatrix} \right) \tag{2.30}$$

of which the second line corresponds to a non controllability. As for standard systems it is possible to eliminate the second row and the second column. Equivalent LFR-object:

$$\begin{bmatrix} \delta_2\delta_1 \\ \delta_1 \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{cc|c} 0 & 0 & 1 \\ 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 1 & 0 & 0 \end{array}\right], \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_2 \end{bmatrix}\right) \tag{2.31}$$

This form is now equivalent and of lower order.

Important remark. In the demonstration of Lemma 2.6.1, we do not use the fact that the δ_i's commute. So, the considered class of similarity is valid for the equivalence in the case where the δ_i's do not commute. It is the main weakness of all the reducing techniques as similarity excludes the equivalence of two LFR-objects as soon as parameters are in wrong order. For example $\delta_1\delta_2$ although equivalent is not similar to $\delta_2\delta_1$. To conclude this remark, all techniques based on the use of the similarity which claim to find minimal form, obtain really the minimality only in the case where the δ_i's do not commute. Let us return to the previous example and permute δ_1 and δ_2. A simple calculation shows that

$$\begin{bmatrix} \delta_1\delta_2 \\ \delta_1 \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right], \begin{bmatrix} \delta_1 & 0 & 0 \\ 0 & \delta_1 & 0 \\ 0 & 0 & \delta_2 \end{bmatrix}\right) \tag{2.32}$$

It can be shown that there exists no solution with T invertible s.t. T allows us to pass from (2.29) to (2.32): (2.29) and (2.32) are equivalent but not similar. Both LFR-object of equations (2.31) and (2.32), one of order 2, the other one of the order 3, *are both "minimal"*, but in the sense of literature!

Definition of the relative minimality. Considering the above discussion, from now on we shall distinguish between the absolute minimality first defined and the notion of "minimality" based on the similarity transformations. One will say that a LFT representation satisfies the *relative minimality* if there is no similarity transformation like in (2.27) such that some lines or columns can be eliminated without modifying the input / output transfer function.

In fact, it is not really necessary to consider difficult LFT theory to explain the difference between the minimality and the relative-minimality. In case of non commutativity, we have

$$\begin{bmatrix} \delta_2\delta_1 \\ \delta_1 \end{bmatrix} = \begin{bmatrix} \delta_2 \\ 1 \end{bmatrix}\delta_1 \tag{2.33}$$

and

$$\begin{bmatrix} \delta_1\delta_2 \\ \delta_1 \end{bmatrix} = \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_1 \end{bmatrix}\begin{bmatrix} \delta_2 \\ 1 \end{bmatrix} \tag{2.34}$$

Realization using the object-oriented approach (section 2.5) clearly leads to LFR-objects **of order equal to the number of times the parameters appear in the symbolic representation**. For simple transfers, it is easy to see if (absolute) minimality is reached (*i.e.* one can not write again an the symbolic expression with fewer parameters). The transfer of Equation (2.33) is of order 2 because each δ_1 and δ_2 appears only once. The transfer of Equation (2.34) is of order 3 because δ_1 appears twice and δ_2 once. It is also clear that without commutativity, it is not possible to reduce further the transfer of Equation (2.34).

More generally, non commutativity can lead to extremely different relative minimal orders for the same transfer. For example the following two transfers are equivalent, but their relative minimal orders[3] are respectively 9 and 3

$$\begin{bmatrix} \delta_1\delta_2\delta_3 \\ \delta_2\delta_3\delta_1 \\ \delta_3\delta_1\delta_2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \delta_1\delta_2\delta_3$$

Note that if

$$\begin{bmatrix} \delta_1\delta_2\delta_3 \\ \delta_1\delta_2\delta_3 \\ \delta_1\delta_2\delta_3 \end{bmatrix}$$

is "object-oriented built", respecting the order of δ_i's as above, we also reach a 9th order, but the algorithms of relative minimization (see chapter 4 for examples) would lead in that case to the minimal form of order 3.

[3] Instead of symbolic objects, imagine the realizations obtained using the object-oriented technique (§2.5) applied according to the symbolic notation (no commutations of δ_i's).

Software relative to this section

Illustrated functions:

- `lfrs` generates real scalar 1×1 LFR-objects,
- `minlfr` computes a "relative minimal" LFR-object.

Example 2.14 This example uses the n-D minimization approach (coded in function `minlfr`) that will be presented later in § 4.2 page 99. It suffices to have in mind that `minlfr` performs one of the most sophisticated LFR order reduction for a given realization. Let us consider the example used in the previous discussion. Let

$$S_1 = \begin{bmatrix} \delta_1\delta_2\delta_3 \\ \delta_2\delta_3\delta_1 \\ \delta_3\delta_1\delta_2 \end{bmatrix} \text{ and } S_2 = \begin{bmatrix} \delta_1\delta_2\delta_3 \\ \delta_1\delta_2\delta_3 \\ \delta_1\delta_2\delta_3 \end{bmatrix}$$

The realizations of S_1 and S_2 are computed

```
>> lfrs d1 d2 d3
>> S1 = [d1*d2*d3;d2*d3*d1;d3*d1*d2];
>> S2 = [d1*d2*d3;d1*d2*d3;d1*d2*d3];
```

Now, `minlfr` is used for reducing the size of the Δ matrix:

```
>> S1min = minlfr(S1);
>> S2min = minlfr(S2);
```

Resulting in

```
>> size(S1min)

LFR-object with 3 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (9 x 9)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds

 d1     3x3    LTI        r            s          [-1,1]
 d2     3x3    LTI        r            s          [-1,1]
```

```
 d3    3x3   LTI       r           s           [-1,1]

>> size(S2min)

LFR-object with 3 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (3 x 3)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    1x1   LTI       r           s           [-1,1]
 d2    1x1   LTI       r           s           [-1,1]
 d3    1x1   LTI       r           s           [-1,1]

>> distlfr(S1min,S2min)

ans = 3.3307e-16
```

As stated in this section, order reduction techniques are more or less efficient depending on the ordering of symbols appearing in the realized symbolic transfer. Here, the same LFR-object corresponds to a Δ-block of order 9 (`S1min`) or of order 3 (`S2min`), which is minimum in the sense of 4.2 in both cases.

2.7 General principles for parameter dependent system modelling

The discussion proposed all along this chapter allows us to propose a skeleton for modelling in LFT form. We propose to proceed in three main steps.

First step: Realization. Before using the algebraic reduction techniques it is necessary to take advantage of the commutativity of the δ_i's (during the realization phase). There are two main approaches for that, Belcastro's one and the tree decomposition.

Second step: Algebraic reduction. Two main approaches: d'Andrea's one and the one of Beck *et al*, see chapter 4. These two techniques are equivalent[4]. Both lead to the relative-minimal order.

Third step: Approximation. See Section 4.4.

[4]Only in theory, because numerically there are important differences.

EMPTY PAGE

Chapter 3

Realization of parameter dependent systems

We have already presented the object-oriented realization technique (§2.5). The other techniques are:

- The technique of Morton [46]. This technique is very restrictive from the point of view of the parameter dependency, but it is the first one which was published. Some extensions are rather natural (see for example [20]), but we shall not develop extensions preferring the more general approaches listed below.
- The technique proposed by Varga and Looye [52] who use Horner factorization.
- The tree decomposition. As for the previous approach, this very natural technique consists of making sum / product decompositions so as to reduce the number of times the parameters occur. It was introduced in Barmish *et al* [3] in another context. Cockburn and Moron [21, 22, 23] adapted this technique to the LFT-realization. It is the technique that is naturally used (manually) when one tries to apply efficiently the "object-oriented" approach.
- The technique of Belcastro *et al* [12, 14, 15, 16, 17] (see in particular [12, 13] which contain all the demonstrations). This approach is purely matrix-based. The sizes of all blocks is chosen *a priori*. This choice leads to a polynomial expansion with regard to parameter powers. By comparison of this expansion to the symbolic object to be realized the equations to be solved can be derived. If this set of equations has no solution, blocks sizes are augmented so that a solution can be found.

- The graph manipulation approach. It is clear that symbolic forms can be represented as graphs. Minimizing an LFT form is therefore equivalent to simplifying a graph. See Font [30] for a general graph approach to LFT modelling. See also Döll [26] for a Simulink-based approach.

Before considering realization techniques, we propose two sections treating generalities (might be considered as appendices). At first, we show how a rational dependency can be treated as a polynomial dependency, which justifies the hypothesis that all considered models will have a symbolic **polynomial** form. Second, we discuss the fact that the symbolic state-space form is more natural than the symbolic transfer matrix form, which justifies the hypothesis that all models considered for realization will be in symbolic state-space (polynomial) form.

3.1 Left and right factorizations

This section consider considers the transformations

$$\begin{bmatrix} N \\ D \end{bmatrix}(\Delta) \rightarrow N(\Delta)D(\Delta)^{-1}$$

and

$$\begin{bmatrix} N' & D' \end{bmatrix}(\Delta) \rightarrow D'(\Delta)^{-1}N'(\Delta)$$

The results of Lemmas 3.1.1 and 3.1.2 given below are often used with D, D', N and N' in polynomial form (see §3.5 (tree decomposition)). However the results of these lemmas are more general and can be used in any case provided that D or D' can be inverted (regardless of polynomial or rational considerations). The main advantage of using these results is that the size of Δ is not modified while using a more naive transformation, *e.g.*

$$\begin{bmatrix} N \\ D \end{bmatrix} \rightarrow \left(\begin{bmatrix} I & 0 \end{bmatrix}\begin{bmatrix} N \\ D \end{bmatrix}\right)\left(\begin{bmatrix} 0 & I \end{bmatrix}\begin{bmatrix} N \\ D \end{bmatrix}\right)^{-1}$$

the size of Δ would be multiplied by two.

Lemma 3.1.1 *Let $M(\Delta) = N(\Delta)D(\Delta)^{-1}$ where the entries of the matrices are polynomials in the δ_i's. If*

$$\begin{bmatrix} N \\ D \end{bmatrix}(\Delta) = \mathcal{F}_u\left(\left[\begin{array}{c|c} Q_{11} & Q_{12} \\ \hline \begin{matrix} Q_{21N} \\ Q_{21D} \end{matrix} & \begin{matrix} Q_{22N} \\ Q_{22D} \end{matrix} \end{array}\right], \Delta\right)$$

so,

$$M(\Delta) = \mathcal{F}_u\left(\left[\begin{array}{c|c} P_{11} & P_{12} \\ \hline P_{21} & P_{22} \end{array}\right], \Delta\right)$$

where

$$\begin{aligned} P_{11} &= Q_{11} - Q_{12}Q_{22D}^{-1}Q_{21D} \\ P_{12} &= Q_{12}Q_{22D}^{-1} \\ P_{21} &= Q_{21N} - Q_{22N}Q_{22D}^{-1}Q_{21D} \\ P_{22} &= Q_{22N}Q_{22D}^{-1} \end{aligned} \tag{3.1}$$

We shall only prove the dual result that is:

Lemma 3.1.2 *Let $M(\Delta) = D'(\Delta)^{-1}N'(\Delta)$ where the coefficients of matrices N' and D' have polynomial dependency in the δ_i's. If*

$$\begin{bmatrix} N' & D' \end{bmatrix}(\Delta) = \mathcal{F}_u\left(\left[\begin{array}{c|cc} Q_{11} & Q_{12N} & Q_{12D} \\ \hline Q_{21} & Q_{22N} & Q_{22D} \end{array}\right], \Delta\right)$$

so

$$M(\Delta) = \mathcal{F}_u\left(\left[\begin{array}{c|c} P_{11} & P_{12} \\ \hline P_{21} & P_{22} \end{array}\right], \Delta\right)$$

where

$$\begin{array}{l} P_{11} = Q_{11} - Q_{12D}Q_{22D}^{-1}Q_{21} \\ P_{12} = Q_{12N} - Q_{12D}Q_{22D}^{-1}Q_{22N} \\ P_{21} = Q_{22D}^{-1}Q_{21} \\ P_{22} = Q_{22D}^{-1}Q_{22N} \end{array} \tag{3.2}$$

Proof. This result is demonstrated in Belcastro [12], page 206. Note that the Δ-block of $D'(\Delta)^{-1}N'(\Delta)$ is the same as the one of $[N' \;\; D'](\Delta)$. Our justification reduces to checking the following identity:

$$D'(\Delta)^{-1}N'(\Delta) = -\left[\begin{array}{cc} 0 & I \end{array}\right]\left[\begin{array}{cc} N'(\Delta) & D'(\Delta) \\ I & 0 \end{array}\right]^{-1}\left[\begin{array}{c} 0 \\ I \end{array}\right]$$

This formula is sufficient for computing $D'(\Delta)^{-1}N'(\Delta)$ without increase of the size of Δ. The derivation of the P'_{ij}'s as given above is straightforward by using the formulas of the appendix (see page 148): (7.4) for horizontal concatenation of $[N'(\Delta) \;\; D'(\Delta)]$ and $[0 \;\; I]$, (7.7) for inversion and so on. ■

Comment 3.1.3 *Using the same ideas as in the above proof we have:*

$$D_1^{-1}(\Delta)N(\Delta)D_2^{-1}(\Delta) = -\left[\begin{array}{cc} 0 & I \end{array}\right]\left[\begin{array}{cc} D_1(\Delta) & N(\Delta) \\ 0 & D_2(\Delta) \end{array}\right]^{-1}\left[\begin{array}{c} 0 \\ I \end{array}\right]$$

This formula will be useful in §3.5 for replacing rational by polynomial dependency in order to use the tree decomposition.

Software relative to this section

Illustrated functions:

- `rf2lfr` computes ND^{-1} (see (3.1)) from a vertical arrangement of $N(\Delta)$ and $D(\Delta)$.
- `lf2lfr` computes $D'^{-1}N'$ (see 3.2) from an horizontal arrangement of $N'(\Delta)$ and $D'(\Delta)$.

Example 3.1 This example illustrates the use of the function `rf2lfr`. This function performs the transformation described in Equation (3.1). First a realization of a given matrix

$$\texttt{NDv} = \begin{bmatrix} N(\Delta) \\ D(\Delta) \end{bmatrix}$$

is performed, note that Δ might contain dynamics.

```
lfrs Int x y z
NDv = [1+x*Int x*y+z;0 x^2*Int^2;1+Int 0;0 2+x*y*Int];
```

Then we apply the function that computes the corresponding system transfer matrix (`sys` $=N(\Delta)D(\Delta)^{-1}$).

```
sys = rf2lfr(NDv);
```

In order to check the results, we generate this object in an alternative way

```
N = [eye(2,2) zeros(2,2)]*NDv;
D = [zeros(2,2) eye(2,2)]*NDv;
sys2 = N*D^(-1);
```

The comparison leads to

```
>>      size(sys)

LFR-object with 2 output(s), 2 input(s) and 5 state(s).
Uncertainty blocks (globally (8 x 8)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
```

```
 x     5x5   LTI       r           s           [-1,1]
 y     2x2   LTI       r           s           [-1,1]
 z     1x1   LTI       r           s           [-1,1]

>>      size(sys2)

LFR-object with 2 output(s), 2 input(s) and 10 state(s).
Uncertainty blocks (globally (16 x 16)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 x     10x10 LTI       r           s           [-1,1]
 y     4x4   LTI       r           s           [-1,1]
 z     2x2   LTI       r           s           [-1,1]

>>    distlfr(sys,sys2)

ans = 0
```

The problem of Equation (3.2) can be considered in a similar way using `lf2lfr` instead of `rf2lfr`.

3.2 Input/output and state-space realizations

It is advisable to wonder about the *a priori* form of models stemming from the physical equations. This interrogation is justified by the fact that certain authors speak about *direct* and *indirect* approaches. "Direct approach" means that the transfer $M(s,\Delta)$ is directly realized.

$$y = M(s,\Delta)u \tag{3.3}$$

assuming that it is in symbolic form. The "indirect approach" consists of realizing

$$S(\Delta) = \begin{bmatrix} A(\Delta) & B(\Delta) \\ C(\Delta) & D(\Delta) \end{bmatrix} \tag{3.4}$$

corresponding to

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A(\Delta) & B(\Delta) \\ C(\Delta) & D(\Delta) \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

Let us take for example the missile model considered further on. Without entering into details, the physical equations are

$$\begin{aligned} \dot{x}_1 &= a_{11}(\delta_1,\delta_2)x_1 + a_{12}(\delta_1,\delta_2)x_2 + b_1(\delta_1,\delta_2)u \\ \dot{x}_2 &= a_{21}(\delta_1,\delta_2)x_1 + b_2(\delta_2)u \\ y &= c_1(\delta_1,\delta_2)x_1 + d(\delta_2)u \end{aligned}$$

Where a_{11}, a_{12}, b_1, b_2, c_1 and d are polynomial[1]. The elimination of states is rather simple: After introducing the integration symbol $1/s$, the second equation is substituted into the first one and the resulting symbolic form of x_1 is substituted into the third one. We obtain

$$M(s,\Delta) = c_1(\delta_1,\delta_2)\frac{b_1(\delta_1,\delta_2) + \frac{1}{s}a_{12}(\delta_1,\delta_2)b_2(\delta_2)}{1 - \frac{1}{s}(a_{11}(\delta_1,\delta_2) + a_{12}(\delta_1,\delta_2)a_{21}(\delta_1,\delta_2))} + d(\delta_2)$$

Using the indirect approach we would have to find a realization of

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ y \end{bmatrix} = \begin{bmatrix} a_{11}(\delta_1,\delta_2) & a_{12}(\delta_1,\delta_2) & b_1(\delta_1,\delta_2) \\ a_{21}(\delta_1,\delta_2) & 0 & b_2(\delta_2) \\ c_1(\delta_1,\delta_2) & 0 & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \end{bmatrix}$$

$$S(\Delta) = \begin{bmatrix} a_{11}(\delta_1,\delta_2) & a_{12}(\delta_1,\delta_2) & b_1(\delta_1,\delta_2) \\ a_{21}(\delta_1,\delta_2) & 0 & b_2(\delta_2) \\ c_1(\delta_1,\delta_2) & 0 & d(\delta_2) \end{bmatrix}$$

The following lemma shows that state-space and input/output models are equivalent as it suffices to permute some sub-matrices to pass from one form to the other one.

[1]The parameters δ_1 and δ_2 stand for the variations of the Mach number and of the angle of incidence. The states are the variation of the angle of incidence and the pitch rate.

Lemma 3.2.1 *If an input/output LFR-realization is given as in (2.3):*

$$y = \mathcal{F}_u\left(\left[\begin{array}{cc|c} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ \hline C_2 & D_{21} & D_{22} \end{array}\right], \left[\begin{array}{cc} \frac{I}{s} & 0 \\ 0 & \Delta \end{array}\right]\right) u \tag{3.5}$$

the equivalent state-space LFR-realization is given by

$$\left[\begin{array}{c} \dot{x} \\ y \end{array}\right] = \mathcal{F}_u\left(\left[\begin{array}{c|cc} D_{11} & C_1 & D_{12} \\ \hline B_1 & A & B_2 \\ D_{21} & C_2 & D_{22} \end{array}\right], \Delta\right) \left[\begin{array}{c} x \\ u \end{array}\right] \tag{3.6}$$

Proof. Let us denote $v = \Delta z$ and $x = \dot{x}/s$ then (3.5) and (3.6) respectively mean

$$\left[\begin{array}{c} \dot{x} \\ z \\ y \end{array}\right] = \left[\begin{array}{cc|c} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ \hline C_2 & D_{21} & D_{22} \end{array}\right] \left[\begin{array}{c} x \\ v \\ u \end{array}\right] \text{ and } \left[\begin{array}{c} z \\ \dot{x} \\ y \end{array}\right] = \left[\begin{array}{c|cc} D_{11} & C_1 & D_{12} \\ \hline B_1 & A & B_2 \\ D_{21} & C_2 & D_{22} \end{array}\right] \left[\begin{array}{c} v \\ x \\ u \end{array}\right]$$

which justifies the equivalence stated in the lemma. ■

Software relative to this section

Illustrated functions:

- `abcd2lfr` converts a system matrix LFR to an input/output LFR-object, see Examples 3.2 and 3.3.
- `bnds2lfr` converts a matrix with uncertainty bounds to an LFR-object, see Example 3.3.

Example 3.2 This example illustrates the use of the function `abcd2lfr` that performs the transformation from Equation (3.6) to Equation (3.5). In other words, this function permits us to compute $M(s,\Delta)$ (see (3.3)) as an LFR-object when we have at our disposal the LFR form of $S(\Delta)$ (see (3.4)).

```
lfrs x y z
A = [1+x x*y+z;0 x^2]; B = [1;y]; C=[y+z x*y]; D=1+y^2;
S = [A B;C D];
sys = abcd2lfr(S,2);
```

In order to check this result, `sys` is computed in an alternative way:

```
I2 = eye(2,2);
sys2 = C*feedback(I2*Int,A,1)*B + D;
```

the following comparison shows that we have generated equivalent objects.

```
>> distlfr(sys,sys2)

ans = 0
```

Note that this example might be misleading because we do not take advantage of the fact that we have a single object $S(\Delta)$ instead of four objects $A(\Delta)$, $B(\Delta)$, $C(\Delta)$ and $D(\Delta)$. Indeed, more simplifications can be expected using the single object because in this case factorization might involve more than one of the sub-matrices $A(\Delta)$, $B(\Delta)$, $C(\Delta)$ and $D(\Delta)$. This topic is the main purpose of the next sections of this chapter.

Example 3.3 This example illustrates the use of the function `bnds2lfr` combined with `abcd2lfr`. It is assumed that bounds relative to the elements of the matrices `A`, `B`, `C`, `D` of a quadruple (*A*,*B*,*C*,*D*) are known. The considered problem consists of building first an LFR-object for these matrices. Then by concatenation, the LFR form of the matrix `S = [A B;C D]` is found. In the last step, the LFR form of `S` is transformed to an input/output LFR-object using `abcd2lfr`.

```
>> minA = [-2 -2 -4;0 -5 -5;0 0 -6];
>> maxA = [ 0 -2 -2;0 -3 -5;0 0 -6];
>> minB = [0.5;0.5;0.5];
>> maxB = [1.5;1.5;1.5];
>> C=[1 1 1];
>> D=1;
```

The matrices `minA`, `maxA`, `minB` and `maxB` explicit the possible variations of the entries of the matrices `A` and `B`: `A` has 3 uncertain parameters that are `A(1,1)` (varying between `-2` and `0`), `A(1,3)` (varying between `-4` and `-2`), and `A(2,2)` (varying between `-5` and `-3`). The entries of `B` vary between `0.5` and `1.5`. The matrices `C` and `D` are fixed. The LFR-objects corresponding to `A` and `B` are built as follows:

```
>> lfrA = bnds2lfr('A_',minA,maxA);
>> lfrB = bnds2lfr('B_',minB,maxB);
```

The third input argument means that the 3 uncertain parameters of `A` are ordered (column-wise) from 1 to 3 (= `lastrank`) and that the uncertain parameters of `B` are ordered from 4 (= `lastrank + 1`). The corresponding input / output LFR-object is computed as follows:

```
>> sys = abcd2lfr([lfrA lfrB;C D],3);
>> size(sys)

LFR-object with 1 output(s), 1 input(s) and 3 state(s).
Uncertainty blocks (globally (6 x 6)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 A_1_1 1x1   LTI       r           s          [-2,0]
 A_1_3 1x1   LTI       r           s          [-4,-2]
 A_2_2 1x1   LTI       r           s          [-5,-3]
 B_1_1 1x1   LTI       r           s          [0.5,1.5]
 B_2_1 1x1   LTI       r           s          [0.5,1.5]
 B_3_1 1x1   LTI       r           s          [0.5,1.5]
```

The function `ordelta` can be used for modifying the default ordering of uncertain parameters.

3.3 Morton's method

Let us assume that matrix $S(\Delta)$ of Equation (3.4) admits a development of the form

$$S(\Delta) = \begin{bmatrix} A(\Delta) & B(\Delta) \\ C(\Delta) & D(\Delta) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \delta_1 \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} + \ldots + \delta_q \begin{bmatrix} A_q & B_q \\ C_q & D_q \end{bmatrix} \tag{3.7}$$

To reduce the realization order, every term ($i \geq 1$) is decomposed using the SVD (Singular Value Decomposition) that is

$$\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} = \begin{bmatrix} U_{i11} & U_{i12} \\ U_{i21} & U_{i22} \end{bmatrix} \begin{bmatrix} S_i & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{i11} & V_{i12} \\ V_{i21} & V_{i22} \end{bmatrix}$$

(Only the non-negligible singular values are retained in S_i) leading to a decomposition

$$\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} = \begin{bmatrix} U_{i1} \\ U_{i2} \end{bmatrix} \begin{bmatrix} V_{i1} & V_{i2} \end{bmatrix}$$

Therefore,

$$S(\Delta) = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \delta_1 \begin{bmatrix} U_{11} \\ U_{12} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \end{bmatrix} + \ldots + \delta_q \begin{bmatrix} U_{q1} \\ U_{q2} \end{bmatrix} \begin{bmatrix} V_{q1} & V_{q2} \end{bmatrix} \tag{3.8}$$

The final representation of $S(\Delta)$ can be written $S(\Delta) = \mathcal{F}_u(P,\Delta)$ where

$$P = \left[\begin{array}{cccc|c} A_0 & U_{11} & \ldots & U_{q1} & B_0 \\ V_{11} & 0 & \ldots & 0 & V_{12} \\ \vdots & & & & \vdots \\ V_{q1} & 0 & \ldots & 0 & V_{q2} \\ \hline C_0 & U_{12} & \ldots & U_{q2} & D_0 \end{array}\right]$$

and

$$\Delta = \mathrm{diag}\{I/s, \delta_1 I_{n_1}, \ldots, \delta_q I_{n_q}\}$$

Proof. To check this result, let us consider the case of two uncertain parameters (the generalization is straightforward). Writing $S(\Delta) = \mathcal{F}_u(P,\Delta)$ with P and Δ as above means that

$$\begin{bmatrix} \dot{x} \\ z_1 \\ z_2 \\ y \end{bmatrix} = \begin{bmatrix} A_0 & U_{11} & U_{21} & B_0 \\ V_{11} & 0 & 0 & V_{12} \\ V_{21} & 0 & 0 & V_{22} \\ C_0 & U_{12} & U_{q2} & D_0 \end{bmatrix} \begin{bmatrix} x \\ w_1 \\ w_2 \\ u \end{bmatrix} \text{ with } \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \delta_1 z_1 \\ \delta_2 z_2 \end{bmatrix}$$

but

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \delta_1 V_{11} & \delta_1 V_{12} \\ \delta_2 V_{21} & \delta_2 V_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

so

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} U_{11} & U_{21} \\ U_{12} & U_{22} \end{bmatrix} \begin{bmatrix} \delta_1 V_{11} & \delta_1 V_{12} \\ \delta_2 V_{21} & \delta_2 V_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

or

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \left(\begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \delta_1 \begin{bmatrix} U_{11} \\ U_{12} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \end{bmatrix} + \delta_2 \begin{bmatrix} U_{21} \\ U_{22} \end{bmatrix} \begin{bmatrix} V_{21} & V_{22} \end{bmatrix} \right) \begin{bmatrix} x \\ u \end{bmatrix}$$

which in the form of the development of the equation (3.8). ■

Possible Extensions. One can go farther by considering that the coefficients of the development (3.7) are not any more the δ_i's but rational functions of the δ_i's (see [20]). One operates as above, but with an intermediate step where the elements of the block Δ are rational functions of δ_i. It remains to use the Redheffer's star product (see [27], or page 28 of this report) to obtain the usual form with δ_i in the block Δ. The functions `depl2lfr` performs the standard Morton realization and `deps2lfr`, the extended one.

Software relative to this section

Illustrated functions:

- `gmorton` generalized Morton's realization.
- `abcd2lfr` converts a state-space representation in LFR form to an input/output LFR-object.

Example 3.4 This example illustrates the generation of an LFT-object given in the expended form of Equation (3.7).

$$S(\Delta) = \begin{bmatrix} A(\Delta) & B(\Delta) \\ C(\Delta) & D(\Delta) \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \delta_1 \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} + \delta_3 \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix}$$

It is assumed that this expression depends on δ_1 and δ_2 that will be denoted `d1` and `d2`. The coefficients of the expansion are chosen at random:

```
>> lfrs d1 d2
>> sys0 = rss(4,2,3);
>> sys1 = rss(4,2,3);
>> sys2 = rss(4,2,3);
>> sys = gmorton({sys0,sys1,sys2},[1 d1 d2]);
```

In order to check this result, an alternative construction based on the use of the function `abcd2lfr` (see Example 3.2 on page 78) is proposed.

```
>> abcd = [sys0.a sys0.b;sys0.c sys0.d] + ...
>> d1 * [sys1.a sys1.b;sys1.c sys1.d] + ...
>> d2 * [sys2.a sys2.b;sys2.c sys2.d];
>> newsys = abcd2lfr(abcd,4);
```

Which results in

```
>> distlfr(sys,newsys)

ans = 3.8722e-14
```

The function `gmorton` is said to generalize Morton's realization technique because this function accepts any rational combinations of parameters as its second argument, in other words, the coefficients of the expansion of Equation (3.7) can be any rational expression. For example

$$S(\Delta) = (1+\delta_2^2)\begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \delta_1\delta_2\begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} + 1/(1+\delta_2)\begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix}$$

```
lfrs d1 d2
renewsys = gmorton({sys0,sys1,sys2},[1+d1^2 d1*d2 1/(1+d2)]);
```

3.4 Realization using Horner factorization

It is the simplest realization method. To illustrate it, let us consider an example.

$$f(\delta_1,\delta_2,\delta_3) = 2\delta_1^3\delta_2^2\delta_3 + 3\delta_1^2\delta_2^3 + 4\delta_1\delta_3 + 5$$

where $f(\delta_1,\delta_2,\delta_3)$ is some coefficient of $S(\Delta)$ (see (3.4)) or of $M(\Delta)$ (see (3.3)). The "step by step" construction of the corresponding LFT model (without factorization) leads to an LFR-object of order 13. Horner factorization concerns single variable polynomials, its objective consists of avoiding calculation of all the powers of the δ_i's. Therefore, it allows us to reduce the number of times the parameter appear and by the way to reduce the size of LFT representations. Assume that the parameter considered for factorizing is δ_1. So,

$$f(\delta_1,\delta_2,\delta_3) = \delta_1(\delta_1(2\delta_1\delta_2^2\delta_3 + 3\delta_2^3) + 4\delta_3) + 5$$

The corresponding "step by step" realization would lead to an LFR-object of order 10 instead of 13. One can again apply the same factorization with regard to δ_2:

$$f(\delta_1,\delta_2,\delta_3) = \delta_1(\delta_1(\delta_2^2(3\delta_2 + 2\delta_1\delta_3)) + 4\delta_3) + 5$$

of which order is now 8. Note that the minimization algorithms (relative minimality) of Chapter 4 applied to the initial form of f and to the above form lead respectively to orders equal to 9 and 7.

This technique can be very simply implemented because there is a Maple function which calculates Horner factorizations (function `convert` with option `horner`). The matrix extension is not very difficult. See [52] for a spectacular order reduction (from 293 to 59).

Software relative to this section

Illustrated functions:

- `sym2lfr` transforms a symbolic expression to an LFR-object.
- use of Maple *via* the Symbolic Toolbox.

Example 3.5 Let us consider the expression of Examples 2.11 and 2.13 on page 52.

$$y(s) = (\frac{\delta_1^2}{s^2} + \frac{\delta_1\delta_3}{s} + \delta_1^2\delta_3^2)u(s) \tag{3.9}$$

As in Example 2.13

```
>> syms d1 d2 d3 Int
>> sys_sym1 = d1^2*Int^2 + d1*d3*Int + d1^2*d3^2;
>> sys_lfr1 = sym2lfr(sys_sym1);
```

lead to an 11th order LFR-object. It is not useful as in the non-symbolic approach to consider factorized expressions in which the number of times parameters appear is reduced because Maple reorganizes internally polynomial expansions. However, some Maple functions permits the user to have some control on the way polynomials are expanded. For example we can use:

```
>> sys_sym2 = maple('convert',sys_sym,'horner',d1);
>> sys_lfr2 = sym2lfr(sys_sym2);
```

Resulting in

```
>> size(sys_lfr1)

LFR-object with 1 output(s), 1 input(s) and 3 state(s).
Uncertainty blocks (globally (8 x 8)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds

 d1     5x5    LTI        r            s          [-1,1]
 d3     3x3    LTI        r            s          [-1,1]
```

```
>> size(sys_lfr2)

LFR-object with 1 output(s), 1 input(s) and 3 state(s).
Uncertainty blocks (globally (5 x 5)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    2x2   LTI       r           s          [-1,1]
 d3    3x3   LTI       r           s          [-1,1]
```

The reduction of the number of times `d1` is repeated comes from Horner factorization w.r.t. `d1` in the used symbolic expression that is

```
sys_sym2 = (d3*Int+(Int^2+d3^2)*d1)*d1
```

It is possible to reduce further the order by applying additional Horner factorizations with respect to other parameters. (Note that to get Maple help messages *via* the Symbolic Toolbox use `mhelp` instead of `help`, for example `mhelp convert`.)

3.5 The structured tree decomposition

The structured tree decomposition is a generalization of the idea consisting of factorizing parameters so that they appear a minimum times as possible before proceeding to the realization.

In this section, it will be assumed that a model as in Equation (3.4) (page 71) is available in **polynomial form**. See Comment 3.1.3 (page 68) and Comment 3.5.2 for a transformation between rational and polynomial forms.

Two types of transformations are considered: **factorization** and **sum decomposition** (in fact two kinds of sum decompositions will be defined). The "tree decomposition" naming comes from the fact that factorizations and decompositions yield sub-matrices of $S(\Delta)$, in turn, factorizations and decompositions are applied to these sub-matrices and so on, until these transformations cannot be applied any more.

Factorization. Let us consider an example.

$$S(\Delta) = \begin{bmatrix} 4\delta_1^2\delta_3 & 3\delta_1 & 0 \\ \delta_3\delta_5 & 5\delta_2^2\delta_4 & \delta_2\delta_4^2 \end{bmatrix} \tag{3.10}$$

which is of order equal to 12. One can make two factorizations which decrease order (each one reduces it of one unity):

$$S(\Delta) = \begin{bmatrix} \delta_1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 5\delta_2^2\delta_4 & \delta_2\delta_4^2 \end{bmatrix} \begin{bmatrix} \delta_3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that there is usually non uniqueness and that *a priori* it is not known which factorization is the best one.

The direct sum decomposition. Decompositions, as a matter of fact, have as objective to make factorizations possible. The most evident decomposition will consist of separating a matrix in two parts in which appear two *complementary* subsets of parameters ("direct sum"). For example:

$$\begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 5\delta_2^2\delta_4 & \delta_2\delta_4^2 \end{bmatrix} = \begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5\delta_2^2\delta_4 & \delta_2\delta_4^2 \end{bmatrix}$$

which, as expected allows us to factorize:

$$\begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 5\delta_2^2\delta_4 & \delta_2\delta_4^2 \end{bmatrix} = \begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & \delta_2\delta_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5\delta_2 & \delta_4 \end{bmatrix}$$

Note that combining decomposition and factorization we have:

$$S(\Delta) = \begin{bmatrix} \delta_1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 4\delta_1 & 3 & 0 \\ \delta_5 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & \delta_2\delta_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5\delta_2 & \delta_4 \end{bmatrix} \right) \begin{bmatrix} \delta_3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The order has been reduced from 12 to 8.

Alternative sum decomposition. As a matter of fact, direct decomposition is not always possible. But it is always possible[2] either to operate a factorization, or to decompose into two parts in the following way:

1 select one (or some) parameter(s).

2 isolate on one side of the decomposition the entries that contain the (or at least one of the) chosen parameter(s).

3 the other term of the sum decomposition is independent of the selected parameter(s).

For example

$$S(\Delta) = \begin{bmatrix} \delta_1\delta_2 + \delta_3 \\ \delta_1\delta_2\delta_3 \end{bmatrix}$$

δ_3 is selected:

$$S(\Delta) = \begin{bmatrix} \delta_1\delta_2 \\ 0 \end{bmatrix} + \begin{bmatrix} \delta_3 \\ \delta_1\delta_2\delta_3 \end{bmatrix} \left(= \begin{bmatrix} \delta_1\delta_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ \delta_1\delta_2 \end{bmatrix} \delta_3 \right)$$

otherwise if we select δ_1 and δ_2

$$S(\Delta) = \begin{bmatrix} \delta_3 \\ 0 \end{bmatrix} + \begin{bmatrix} \delta_1\delta_2 \\ \delta_1\delta_2\delta_3 \end{bmatrix} \left(= \begin{bmatrix} \delta_3 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ \delta_3 \end{bmatrix} \delta_1\delta_2 \right)$$

In both cases, one can factorize the selected parameter(s). In the first case, the factorization of δ_3 leads to a 5th order. In the second, the factorization of δ_1 and δ_2 leads to a 4th order.

Combination of factorizations and sum decompositions. There is no general method. It is clear that final result depends on the order in which one operates these transformations. Furthermore, transformations are not unique (see for example the sum decomposition of the last example). In the paper that introduced this technique ([23]) there is a simple natural (trivial) recommendation relative to factorization: try all the possibilities and retain the best one (there is not naturally any guarantee that such a choice is *globally* the best one).

Comment 3.5.1 *The main limitation of the tree decomposition is the following: If the considered symbolic expression contains complex factorizations, e.g.,*

$$M = (1 + a^2 + ba)^5 - (1 + a + b)^3$$

[2]Except when all sub-matrices entries are monomials depending on a single parameter, that is at the end of algorithm.

it is likely that the direct realization (function `sym2lfr`*) which doesn't remove factorizations will lead to much better results than the tree decomposition (function* `symtreed`*) which expands symbolic expressions. For the above example, the direct realization leads to order 26, and the tree decomposition, to order 39 (the expanded form of M that is treated by the tree decomposition is of order 156).*

Comment 3.5.2 *Transformation of rational objects to polynomial ones. For applying the tree decomposition to a rational symbolic object, first, find the polynomial symbolic objects* $N(\Delta), D_1(\Delta), D_2(\Delta)$ *satisfying* $S(\Delta) = D_1^{-1}(\Delta)N(\Delta)D_2^{-1}(\Delta)$ *and then apply the tree decomposition to*

$$S'(\Delta) = \begin{bmatrix} D_1(\Delta) & N(\Delta) \\ 0 & D_2(\Delta) \end{bmatrix}$$

Having $S'(\Delta)$ *as an LFR-object, it remains to apply the transformation of Comment 3.1.3 (page 68) to obtain the LFR-form of the original symbolic object. The advantage of using this formula is that the size of* Δ *is not the sum of the sizes of the* Δ*'s of* $D_1(\Delta), N(\Delta)$ *and* $D_2(\Delta)$ *but reduces to the size of the* Δ *of* $S'(\Delta)$*.*

Software relative to this section

Illustrated function:

- `symtreed` structured tree decomposition from a symbolic expression.

Example 3.6 Let us consider the symbolic expression of Equation (3.10) for illustrating realization by the tree decomposition..

```
>> syms d1 d2 d3 d4 d5
>> sys_sym = [4*d1^2*d3 3*d1 0 ; d3*d5 5*d2^2*d4 d2*d4^2];
>> sys1 = symtreed(sys_sym);
```

The order `sys1` is equal to 9. If in the last command, `symtreed` is replaced by `sym2lfr` for a brute force realization:

```
>> sys2 = sym2lfr(sys_sym);
```

The order is equal to 12.

In order to introduce Chapter 4, let us apply an "after realization" order reduction technique:

```
>> sys3 = minlfr(sys1);
>> sys4 = minlfr(sys2);
```

The order is 8 in the first case, 10 in the second case. This example shows that if realization is not optimized, after realization it is too late, some factorization are no longer feasible (commutativity problem).

3.6 The matrix method

Belcastro's ([12]) matrix method is by far the most technical. A risk of conservatism lies in the fact that the matrix "A" of certain sub-systems is supposed to be nilpotent. So, it is difficult to estimate with certainty the sub-optimality of this technique. See [15] for more details than in this section.

Principle of the method. The principle is rather simple but the implementation is dreadful. Assume that we have at hand a symbolic expression of $S(\Delta)$ (see (3.4)) of which coefficients are multivariate polynomials (the variables being the δ_i's).

The expression of P in Equation (3.11)

$$S(\Delta) = \mathcal{F}_u(P,\Delta) \tag{3.11}$$

is partitioned accordingly to the structure of Δ:

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{c|cc} P_{11} & P_{12} & P_{13} \\ \hline P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{array}\right], \Delta \right) \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.12}$$

The expression of $\mathcal{F}_u(P,\Delta)$ is expended so as to write it under the form of a multivariate polynomial in the δ_i's. Note that in deriving this expansion, the term $(I - P_{11}\Delta)^{-1}$ is troublesome. It is then necessary to assume that P_{11} is nilpotent of order r *i.e.*

$$(I - P_{11}\Delta)^{-1} = I + (P_{11}\Delta) + (P_{11}\Delta)^2 + \ldots + (P_{11}\Delta)^r$$

with

$$(P_{11}\Delta)^{r+1} = 0$$

so that $\mathcal{F}_u(P,\Delta)$ can be written in closed form without matrix inversion, so,

$$\mathcal{F}_u(P,\Delta) = \begin{bmatrix} P_{22} & P_{23} \\ P_{32} & P_{33} \end{bmatrix} + \begin{bmatrix} P_{21} \\ P_{31} \end{bmatrix} \Delta(I + (P_{11}\Delta) + \ldots + (P_{11}\Delta)^r) \begin{bmatrix} P_{12} & P_{13} \end{bmatrix}$$

The value of r is a function of the sum the partial degrees of δ_i in the expression of $S(\Delta)$.

Identifying term by term the two polynomial expressions of $S(\Delta)$ and of $\mathcal{F}_u(P,\Delta)$, a certain number of equations is deduced. It remains to solve these equations for finding the sub-matrices of P_{ij}. The references [12, 15] show in details how these equations can be handled. We shall just sketch the resolution procedure.

Resolution procedure. The resolution of obtained equations is done following three main steps:

- The first one consists of taking $\Delta = 0$, from which P_{22}, P_{23}, P_{32} and P_{33} can be deduced.
- The second step, for every δ_i, comes down to 1-D realization sub-problems subject to an additional nilpotency condition (of the matrix "A", it is a specific realization problem treated in Chen [19]). Having achieved all the sub-problems of 1-D realization, P_{21}, P_{31}, P_{12}, P_{13} and the diagonal blocks of P_{11} are known.
- Third step consists of calculating the non diagonal blocks of P_{11}. This computation can be organized sequentially so that only linear problems are to be dealt with: the sub-matrices that are computed at a given step are plugged into the equations considered at the next steps.

Comment 3.6.1 *We do not propose a* MATLAB *function because we believe that this technique does not work systematically for non-academic systems. The main drawback lies in the third step: when some linear sub-problems have no solution, Belcastro states that it is always possible to increase the number of degrees of freedom (i.e. increase the size of uncertainty blocks initially chosen) so that a solution exists. Unfortunately, for the existence of a solution, some rank conditions are required (see Lemma 3.2 in [13]). As some of the involved sub-matrices are nilpotent and appear as powers in the linear equations to be solved, this nilpotency property induces difficulties for satisfying the rank conditions in a systematic way.*

3.7 Comment on normalization

The normalization of parameters must be made after the realization. For example consider two parameters δ_1 and δ_2. Let us assume that δ_1 and δ_2 have been replaced by normalized values:

$$\begin{aligned}\delta_1 &\rightarrow a_1 + b_1\delta_1\\ \delta_2 &\rightarrow a_2 + b_2\delta_2\end{aligned}$$

the simple product $\delta_1\delta_2$ becomes

$$\delta_1\delta_2 \rightarrow (a_1 + b_1\delta_1)(a_2 + b_2\delta_2) = a_1a_2 + a_1b_2\delta_2 + b_1\delta_1a_2 + b_1\delta_1b_2\delta_2$$

The resulting order is 4 instead of 2. After the product is performed, realization techniques cannot return to the initial factorization.

EMPTY PAGE

Chapter 4

Order reduction and approximation after realization

It is assumed that a realization is available (see 3). Three techniques for reducing the order of the available realization are briefly presented:

- The 1-D approach. It consists of considering that δ_i plays the role of $1/s$. Then, for each of them, a standard reduction technique is applied (balanced realization for example). This approach was introduced in [36].
- The approach based on the n-D (Kalman like) decomposition (see d'Andrea, Beck *et al* [25, 6, 9]). Earlier introduction to n-D decomposition can be found in Bose [18]. This approach generalizes the 1-D reduction based on controllable and observable sub-spaces.
- The generalized Gramian approach. This approach is evoked in numerous papers, for example [5, 8, 10, 11, 54]. D'Andrea and Beck [7] showed its equivalence with the one based on the n-D (Kalman like) decomposition. However the Gramian based technique can also be used for LFT *approximation*. We shall briefly evoke some approximation improvements reviewed in Hiret *et al* [34, 33]. Note that the approximation techniques presented here are not valid in the continuous time case because $1/s$ is not bounded, therefore approximation error cannot be evaluated. In that case it is necessary to use an IQC (Integral Quadratic Constraints) approach, see Andersson *et al* [1, 2].
- There are alternative ways for approximating LFR-objects (for example using engineering knowledge, using order reduction with more or less high tolerance parameter an so on). Approximation in that way is often very efficient, however, it is necessary to be able to evaluate the approximation errors. In §4.4 is proposed an original technique that

permits us to compute tight upper bounds of approximation errors (independently of the used approximation technique). This is also a modelling tool, because, knowing approximations of errors bounds, approximation errors can be converted to additional artificial uncertain parameters (of low order).

- This chapter ends with a section that is not related to order reduction as it concerns uncertain parameter normalization. However, as already explained, parameter normalization must not be performed before realization, that is why we put here this section (that should be considered as an appendix).

4.1 Order reduction: The 1-D approach

It consists of considering alternately each of the δ_i: It is assumed that there is only one uncertainty block, say $\delta_i I_{n_i}$. The inputs/outputs of the other blocks are considered as additional natural inputs/outputs of the system (like u and y). So, a linear system where δ_i may play the role of $1/s$ is obtained. Order reduction can then be performed by a classic 1-D technique.

Let us consider a simple illustrative example with two uncertain parameters δ_1 and δ_2. We have

$$\begin{bmatrix} \dot{x} \\ z_1 \\ z_2 \\ y \end{bmatrix} = \begin{bmatrix} A & B_{1a} & B_{1b} & B_2 \\ C_{1a} & D_{11aa} & D_{11ab} & D_{21a} \\ C_{1b} & D_{11ba} & D_{11bb} & D_{21b} \\ C_2 & D_{21a} & D_{21b} & D_{22} \end{bmatrix} \begin{bmatrix} x \\ w_1 \\ w_2 \\ u \end{bmatrix}$$

For the first 1-D reduction, the natural quadruple is considered:

$$\begin{array}{rcl} sx & = & Ax + \begin{bmatrix} B_{1a} & B_{1b} & B_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ u \end{bmatrix} \\ \begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix} & = & \begin{bmatrix} C_{1a} \\ C_{1b} \\ C_2 \end{bmatrix} x + \begin{bmatrix} D_{11aa} & D_{11ab} & D_{21a} \\ D_{11ba} & D_{11bb} & D_{21b} \\ D_{21a} & D_{21b} & D_{22} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ u \end{bmatrix} \end{array}$$

Having eliminated non controllable and non observable states (1-D reduction) certain matrices are modified. For simplicity, the corresponding notations are not modified. As a second step, permutation between x and w_1 and between $\dot{x}$ and z_1 are done. Then the following quadruple is obtained:

$$\begin{array}{rcl} (1/\delta_1) z_1 & = & D_{11aa} w_1 + \begin{bmatrix} C_{1a} & D_{11ab} & D_{21a} \end{bmatrix} \begin{bmatrix} x \\ w_2 \\ u \end{bmatrix} \\ \begin{bmatrix} \dot{x} \\ z_2 \\ y \end{bmatrix} & = & \begin{bmatrix} B_{1a} \\ D_{11ba} \\ D_{21a} \end{bmatrix} w_1 + \begin{bmatrix} A & B_{1b} & B_2 \\ C_{1b} & D_{11bb} & D_{21b} \\ C_2 & D_{21b} & D_{22} \end{bmatrix} \begin{bmatrix} x \\ z_2 \\ u \end{bmatrix} \end{array}$$

One can so operate the same reduction as during the first step ($x = (1/s)\dot{x}$ is replaced formally by $w_1 = \delta_1 z_1$). The notions of controllability and observability does not make sense any more, but formally, the reduction is possible. It remains to proceed in a similar way with regard to z_2 and w_2 to end the procedure of reduction.

Remark 1. It is sometimes necessary to repeat 1-D reductions a number of times larger than the number of blocks, for example successively with regard to $1/s$, δ_1, δ_2, δ_1, ... The example below illustrates this remark.

Example. Two parameters δ_1 and δ_2 are considered. This example illustrates the fact that 1-D reduction, respectively with respect to δ_1 and δ_2, is not sufficient because for reaching the minimal representation, it is needed to add a reduction step relative to δ_1. Let us consider an uncertain gain matrix

$$S(\Delta) = \begin{bmatrix} \delta_1\delta_2 \\ \delta_1\delta_2 \end{bmatrix} \tag{4.1}$$

$S(\Delta)$ is realized step by step without using one of the realization techniques of Chapter 3 because this technique would find the intermediate following minimal form:

$$S(\Delta) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \delta_1\delta_2$$

We are going to find this minimal form by the 1-D technique. First, LFT representations of δ_1 and δ_2 are created, then products (without changing order) and concatenation are performed. The result is:

$$M_{11} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} ; M_{12} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$M_{21} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} ; M_{22} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The first step is an attempt to reducing the following quadruple

$$\left[\begin{array}{cc|ccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

which is controllable and observable, so there *no possible reduction*. It can be explained simply: one can factorize δ_1 to the right of (4.1) without commuting it with δ_2. The second step consists of reducing the quadruple obtained after *permutation of blocks* relative to δ_1 and δ_2. So,

$$\left[\begin{array}{cc|ccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

which is not controllable (now δ_2 can be factorized on the right hand side in (4.1)). After elimination of a non controllable state, we obtain the following quadruple:

$$\left[\begin{array}{c|ccc} 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Third step: let us go back to the reduction relative to δ_1, a new block permutation of δ_1 and δ_2 is applied. The new quadruple is:

$$\left[\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right]$$

which is uncontrollable. In other words, now it is possible to factorize to the right δ_1 in (4.1). Eliminating one of the uncontrollable states, the equivalent quadruple is obtained:

$$\left[\begin{array}{c|cc} 0 & 1 & 0 \\ \hline 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{array}\right]$$

Therefore, we have found the minimal realization that is:

$$M_{11} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} ; M_{12} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} ; M_{21} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} ; M_{22} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Remark 2. The 1-D method is not limited to finding factorizations forgotten during the realization as could let believe the above example. In particular, it does not miss some factorizations which are not "seen" using realization techniques. For example, considering

$$S(\Delta) = \begin{bmatrix} \delta_3\delta_1 + \delta_3\delta_2 + \delta_4\delta_1 + \delta_4\delta_2 \\ \delta_1 + \delta_2 \end{bmatrix} \tag{4.2}$$

the current realization techniques are not sufficiently sophisticated to "see" that

$$S(\Delta) = \begin{bmatrix} \delta_3 + \delta_4 \\ 1 \end{bmatrix} (\delta_1 + \delta_2)$$

but the 1-D technique leads (in this case) to the minimal realization.

Software relative to this section

Illustrated functions:

- `minlfr1` 1-D order reduction.
  - Example 4.1 illustrates the fact that the order 1-D reductions are performed is important.
  - Example 4.2 illustrates the fact that reduction techniques can detect factorizations that are not found at the realization step.

Example 4.1 Let us illustrate the remark involving Equation (4.1). First a realization is computed.

```
>> lfrs d1 d2
>> S = [d1*d2;d1*d2];
```

The function for 1-D reduction is `minlfr1`. If we try to reduce the order of this LFR-object only by considering the first parameter `d1` (this parameter cannot be factorized on the right because commutativity of `d1` and `d2` is ignored) there is no reduction

```
>> Smin = minlfr1(S,[],1);
```

resulting in

```
>> size(Smin)

LFR-object with 2 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    2x2   LTI       r           s          [-1,1]
 d2    2x2   LTI       r           s          [-1,1]
```

In order to avoid this problem, in `minlfr1`, by default 1-D reduction is applied more than one time for each parameter until there is no more reduction as shown now:

```
>> Smin2 = minlfr1(S);
>> size(Smin2)

LFR-object with 2 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (2 x 2)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds

 d1     1x1    LTI        r            s          [-1,1]
 d2     1x1    LTI        r            s          [-1,1]
```

as expected both factorizations are “found”.

■■■■■■

Example 4.2 Let us illustrate the remark involving Equation (4.2). First a realization is computed.

```
lfrs d1 d2 d3 d4
S = [d3*d1+d3*d2+d4*d1+d4*d2;d1+d2];
```

Then, `minlfr1` is used for reducing the size of the Δ matrix:

```
Smin = minlfr1(S);
```

resulting in

```
>> size(S)

LFR-object with 2 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (10 x 10)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds

 d1     3x3    LTI        r            s          [-1,1]
 d2     3x3    LTI        r            s          [-1,1]
 d3     2x2    LTI        r            s          [-1,1]
 d4     2x2    LTI        r            s          [-1,1]

>> size(Smin)
```

```
LFR-object with 2 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    1x1   LTI       r           s          [-1,1]
 d2    1x1   LTI       r           s          [-1,1]
 d3    1x1   LTI       r           s          [-1,1]
 d4    1x1   LTI       r           s          [-1,1]
```

As claimed in Remark 2 above, the factorization $\delta_3\delta_1+\delta_3\delta_2+\delta_4\delta_1+\delta_4\delta_2 = (\delta_3+\delta_4)(\delta_1+\delta_2)$ was "found" because the order dropped from 10 to 4.

4.2 Order reduction: The n-D approach

Instead of considering successively $1/s$ and then every δ_i as with the previous method, the n-D approach treats simultaneously all the parameters of the block Δ, so, this technique is less conservative. It is shown in [25] that it leads to the minimal order for a given realization (that is "relative-minimality").

As in the case of standard linear systems, it is necessary to apply two steps, the first one consisting of identifying the "controllable part", the second one, of identifying the "observable part". Minimal form corresponds to the subsystem that is both "controllable" and "observable".

We are going to describe (for explanations, see [25]) the "controllability part" of the algorithm proposed in [25]. For better clarity we consider a simple case: only the first three steps are described, the general formulation follows easily. Furthermore, to facilitate joint reading of this document and of [25], we adopt the notations of this reference.

Let us consider again the following example:

$$\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix} = \begin{bmatrix} A^{11} & A^{12} & B^1 \\ A^{21} & A^{22} & B^2 \\ C^1 & C^2 & D \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ u \end{bmatrix}$$

For controllability we shall just consider the matrix

$$\begin{bmatrix} A^{11} & A^{12} & B^1 \\ A^{21} & A^{22} & B^2 \end{bmatrix} \tag{4.3}$$

The algorithm consists of applying at each step a treatment concerning six matrices A^{11}, A^{12}, B^1, A^{21}, A^{22} and B^2 (or to the similar matrices appearing at the following steps). This treatment is clarified below.

A basic step of the algorithm. We present the first step in detail, the following ones are similar (adaptations to be made are evoked later).

- A transformation relative to the pair (A^{11}, B^1) is computed:

$$A^{11} \rightarrow \begin{bmatrix} S^{1,1} \\ S^{1,2} \end{bmatrix} A^{11} \begin{bmatrix} T^{1,1} & T^{1,2} \end{bmatrix} = \begin{bmatrix} * & * \\ 0 & A_1^{11} \end{bmatrix} ; B^1 \rightarrow \begin{bmatrix} S^{1,1} \\ S^{1,2} \end{bmatrix} B^1 = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

  Note that the above transformation is such that A_1^{11} is the uncontrollable part, the size of null blocks in B^1 is maximized (so as to exhibit the maximum number of uncontrollable states). Similar for all the other transformations considered during this algorithm. One indicates by "*" sub-matrices having no particular interest in describing the functioning of the algorithm.

- As above, a transformation relative to the pair (A^{22}, B^2) is computed:

$$A^{22} \to \begin{bmatrix} S^{2,1} \\ S^{2,2} \end{bmatrix} A^{22} \begin{bmatrix} T^{2,1} & T^{2,2} \end{bmatrix} = \begin{bmatrix} * & * \\ 0 & A_1^{22} \end{bmatrix} ; B^2 \to \begin{bmatrix} S^{2,1} \\ S^{2,2} \end{bmatrix} B^2 = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

- In fact, these transformations must be applied to the global matrix (4.3), which means:

$$A^{12} \to \begin{bmatrix} S^{1,1} \\ S^{1,2} \end{bmatrix} A^{12} \begin{bmatrix} T^{2,1} & T^{2,2} \end{bmatrix} = \begin{bmatrix} * & * \\ B_1^{12} & A_1^{12} \end{bmatrix}$$

and

$$A^{21} \to \begin{bmatrix} S^{2,1} \\ S^{2,2} \end{bmatrix} A^{21} \begin{bmatrix} T^{1,1} & T^{1,2} \end{bmatrix} = \begin{bmatrix} * & * \\ B_1^{21} & A_1^{21} \end{bmatrix}$$

Now again six new matrices similar to (A^{11}, A^{12}, B^1) and (A^{22}, A^{21}, B^2) are available (but of reduced size), to which one can undergo the same treatment. These matrices are denoted $(A_1^{11}, A_1^{12}, B_1^{12})$ and $(A_1^{22}, A_1^{21}, B_1^{21})$.

Three steps briefly presented. In order to clarify the presentation, three steps are presented.

- Initial matrices

$$\left[\begin{array}{ccc} A^{11} & A^{12} & B^1 \\ A^{21} & A^{22} & B^2 \end{array}\right]$$

- After applying the aforementioned transformations

$$\left[\begin{array}{ccccc} * & * & * & * & * \\ 0 & A_1^{11} & B_1^{12} & A_1^{12} & 0 \\ \hline * & * & * & * & * \\ B_1^{21} & A_1^{21} & 0 & A_1^{22} & 0 \end{array}\right]$$

- The same transformations are applied to $(A_1^{11}, A_1^{12}, B_1^{12})$ and to $(A_1^{22}, A_1^{21}, B_1^{21})$, so

$$\left[\begin{array}{ccc||ccc||c} * & * & * & * & * & * & * \\ \hline 0 & * & * & * & * & * & 0 \\ 0 & 0 & A_2^{11} & 0 & B_2^{12} & A_2^{12} & 0 \\ \hline\hline * & * & * & * & * & * & * \\ \hline * & * & * & 0 & * & * & 0 \\ 0 & B_2^{21} & A_2^{21} & 0 & 0 & A_2^{22} & 0 \end{array}\right] \tag{4.4}$$

The algorithm ends when all pairs similar to (A_2^{11}, B_2^{12}) and (A_2^{22}, B_2^{21}) become completely uncontrollable (*i.e.*, $B_2^{12} = 0$ and $B_2^{21} = 0$) or vanish. For the actual algorithm, see the code of the function `minlfr`. Three points are mentioned:

- The above three steps are generalized considering more than two uncertainty blocks. With notations as above, assuming that there are q uncertainty blocks and that we are at step k of the algorithm: the matrix "B" used for decomposition is given by $B_k^i = [B_k^{i1} \dots B_k^{iq}]$ (one of these sub-matrices is equal to zero).
- The matrices A_k^{ij} $(i \neq j)$ are automatically computed by applying the transformations found at each step.
- For identifying the final controllable subspace (and by the way the reduced from of the system), at each step (k) and sub-step $(i = 1, \dots, q)$, the upper part of the matrix S arising from controllability decomposition is plugged into some matrix that will define the global controllable subspace at the end of the algorithm.

For a justification, see Diandra and Khatri [25].

Controllability and observability. To complete the procedure, it is necessary to consider the duality "controllability / observability".

Advantages with regard to the 1-D approach. For numerous examples, the same results were obtained using the 1-D or n-D approaches. However very simple academic examples can be built in order to put in evidence the interest of the n-D approach. Let us consider the following object

$$S(\Delta) = \begin{bmatrix} \frac{1}{1+\delta_1+\delta_2} \\ \frac{1}{1+\delta_1+\delta_2} \end{bmatrix} \tag{4.5}$$

$S(\Delta)$ is realized using the object-oriented approach, the obtained realization is of order 4. It is clear that the order of the minimal form is 2 because

$$S(\Delta) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \frac{1}{1+\delta_1+\delta_2}$$

This minimal form is found using the n-D technique but not found using the 1-D approach. This fact can be simply explained. By considering a single variable at a time, it is impossible to find any factorizations. On the other hand by, dealing simultaneously with δ_1 and δ_2, it is clear that factorization becomes possible.

Software relative to this section

Illustrated functions:

- `minlfr` n-D order reduction.
- `minlfr1` 1-D order reduction.

See Example 2.14 on page 61 for an illustration of the effect of (bad) parameter ordering.

Example 4.3 This example aims at illustrating the fundamental difference between the 1-D and the n-D approaches. For that purpose, let us consider the expression of Equation (4.5) in which factorization cannot be "found" considering only one parameter at a time.

```
>> lfrs d1 d2
>> S = [1/(1+d1+d2);1/(1+d1+d2)];
```

The minimum forms after using the 1-D and n-D approaches are respectively denoted `Smin1` and `Smin`

```
>> Smin1 = minlfr1(S);
>> Smin  = minlfr(S);
```

For comparison of the results:

```
>> size(Smin1)

LFR-object with 2 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (4 x 4)):
 Name   Dims   Type    Real/Cplx    Full/Scal       Bounds

 d1     2x2    LTI        r            s           [-1,1]
 d2     2x2    LTI        r            s           [-1,1]

>> size(Smin)

LFR-object with 2 output(s), 1 input(s) and 0 state(s).
```

```
Uncertainty blocks (globally (2 x 2)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 d1    1x1   LTI       r           s          [-1,1]
 d2    1x1   LTI       r           s          [-1,1]
```

As expected `Smin` is minimum but `Smin1` has the same complexity as the original LFR-object.

4.3 Order reduction and approximation: The generalized Gramian approach

This method allows us to make exact reductions but also approximate reductions. Approximate reductions are validated by an upper bound of the approximation error. It is very natural to limit the variations of Δ to be able to define such a bound. It will be assumed that all parameters of Δ vary between -1 and 1. It excludes to treat $1/s$ like δ_i (except in the case of exact reduction). If the natural variations of δ_i are not normalized, it is enough to know that a simple transformation of the matrix M in $\mathcal{F}_u(M,\Delta)$ allows us to satisfy this condition (see §2.3.1).

First, generalized Gramian are defined. Let us consider a realization

$$\mathcal{F}_u(M,\Delta) \text{ where } M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

The generalized Gramian are the solutions X and Y of the following LMI systems:

$$\begin{array}{c} X \geq 0 \\ M_{11}^T X M_{11} - X + M_{21}^T M_{21} \leq 0 \end{array} \tag{4.6}$$

and

$$\begin{array}{c} Y \geq 0 \\ M_{11} Y M_{11}^T - Y + M_{12} M_{12}^T \leq 0 \end{array} \tag{4.7}$$

where X and Y are matrices that commute with Δ (similar condition relative to T in (2.27)).

Exact reduction. The order of the LFT representations can be reduced of an amount equal to the difference between the size of the product XY its rank. To put in evidence the loss of rank of XY, this matrix is diagonalized:

$$XY = T\Sigma^2 T^{-1}$$

To obtain the system of reduced order, the similarity transformation T is applied to $\mathcal{F}_u(M,\Delta)$ (see Lemma 2.6.1).

$$\mathcal{F}_u(\mathcal{T}(M),\Delta) = \mathcal{F}_u\left(\begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ \tilde{M}_{21} & \tilde{M}_{22} \end{bmatrix},\Delta\right)$$

It suffices to eliminate

- the rows and columns of Δ,
- the rows and columns of $\tilde{M}_{11}$,
- the rows of $\tilde{M}_{12}$ and

- the columns of $\tilde{M}_{21}$.

which correspond to null eigenvalues in the matrix Σ^2.

Approximate reduction. It is the same procedure, but instead of considering the negligible values of Σ^2, those that are "small" are also taken into account, knowing that the truncation error (for normalized δ_i's) satisfies:

$$\|\mathcal{F}_u(M,\Delta) - \mathcal{F}_u(\hat{M},\hat{\Delta})\| \leq 2\sum_{i\in I}\sigma_i \tag{4.8}$$

here $\mathcal{F}_u(\hat{M},\hat{\Delta})$ is the LFT form after truncation, I indicates the indices of neglected eigenvalues and σ_i indicates the square roots of the value of XY (or of Σ_i^2). It remains to see how can be optimized the selection of X and Y satisfying (4.6) and (4.7) so as to make more favorable the truncation possibilities. See Hiret *et al* [34, 33] for a comparative analysis of the different techniques. To optimize truncation, it is necessary "to minimize the rank" of XY, for example by minimizing the trace of this product. We have the system coming from (4.6) and (4.7) (with a condition of structure on X and Y)

$$\begin{gathered} X \geq 0 \\ M_{11}^T X M_{11} - X + M_{21}^T M_{21} \leq 0 \\ Y \geq 0 \\ M_{11} Y M_{11}^T - Y + M_{12} M_{12}^T \leq 0 \\ \text{trace}(XY) \text{ minimum} \end{gathered}$$

The product XY is non-linear, so, the above system is not an LMI. One can however resolve iteratively:

- by fixing one of the matrices X or Y and by optimizing the other one.
- by linearizing: $X = X_0 + dX$ and $Y = Y_0 + dY$, so, $XY \approx X_0 dY + dX Y_0$. The LMIs in dX and dY are resolved, X and Y are updated, and so on.

These algorithms can be improved by considering X^{-1} and Y^-1 instead of X and Y.

4.4 Interval of variations of a Linear Fractional Representations

The problem of finding the minimum and maximum values of a (1×1) real LFR-object will be shown to be a μ-analysis problem (see Lemma 4.4.1 below). Before considering technical issues, let us discuss the interest of such a tool for LFR-object approximation.

The main issue of this theory is approximate modelling as illustrated in Examples 4.4 and 4.5.

4.4.1 Necessity of having a reliable distance

Computing a non conservative distance between LFR-objects. The distance between two objects can be defined by computing the maximum and minimum values of the difference between both considered objects. If the considered objects are not scalar, the distance can be defined as the maximum of the absolute values of the extremal values of all the (1×1) inputs / outputs transfers. For example if two objects $S_1(\Delta)$ and $S_2(\Delta')$ (same real uncertainties in Δ and Δ') have m inputs and p outputs:

$$d(S_1(\Delta), S_2(\Delta')) = \text{Max}\{|\ \text{Max and Min of } e_i^T (S_1(\Delta) - S_2(\Delta')) e_j|, i \in [1 \dots m], j \in [1 \dots p]\}$$

in which e_i and e_j are vectors of the canonical basis of $\mathbf{R}^m$ and $\mathbf{R}^p$.

The technical result that will be stated in Lemma 4.4.1 is based on the computation of a μ measure. More precisely, it is the upper bound of μ that will be used. It means that LMI systems must be solved. At first sight, the order of each entry of $S_1(\Delta) - S_2(\Delta')$ is equal to the sum of the sizes of Δ and Δ' that might correspond to very high dimensional LMI systems. But in fact, each entry is considered separately, moreover there is a difference that might cancel some terms, therefore, after use of the reduction techniques (without approximation) of §4.1 or 4.2 it is likely that the order of each considered entry of $S_1(\Delta) - S_2(\Delta')$ will be far lower than the sum of the sizes of Δ and Δ'. Keeping in mind this remark, the proposed distance is likely to be computed much faster than the Generalized Gramian one of §4.3.

Discussion on approximations. There are a lot of approximation techniques that are naturally used by designers trying to find a model of low order. For example:

- The most natural technique consists of identifying the uncertain parameters that can be neglected. For that, it is possible to use engineering knowledge or Monte-Carlo analysis[1].

[1] For each random trials of uncertain parameters a model is derived. Then model variations are analyzed versus parameter variations, *etc*.

The uncertain parameters with the weakest effect are fixed to their nominal or worst case values.

- In aeronautics, coefficients entering into systems definition are often given as tables of numerical values that must be interpolated. Some degrees of precision in the interpolation formulas can be considered. Simplified interpolation formulas are a good way to reduce LFR-object complexity.
- The iterative 1-D *order reductions* of §4.1 can be replaced by iterative 1-D *approximations*.

These more or less heuristic techniques present a major drawback that lies in the fact that there is no guarantee that approximations are valid.

The (usually non conservative) distance that is presented in this section will permit us to alleviate this problem as it will permit us to compare actual LFR-objects to possible approximations.

Modelling approximation error. Having a tool that computes the extremal values that takes a (1×1) real LFR-object is not only useful for computing the approximation error, it is even more useful for modelling.

For all approximations discussed above, it becomes possible to know exactly the bounds of the approximation errors. Therefore, new artificial uncertainties can be added to the models in order to capture these errors. For example let $S_1(\Delta)$ be an accurate model and $S_2(\Delta')$ be an approximation (same real uncertainties in Δ and Δ'). Instead of considering $S_2(\Delta')$, we suggest to consider

$$S_3(\mathrm{Diag}(\Delta', \varepsilon_{ij})) = S_2(\Delta') + \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \dots \\ \varepsilon_{21} & \ddots & \\ \vdots & & \end{bmatrix}$$

in which ε_{ij} denotes the approximation relative to the entry (i, j) of $S_1(\Delta)$. These new parameters present some interesting features

- they are likely to be less repeated than the natural uncertainties,
- variations bounds are precisely known,
- if some μ-based test applied to the resulting LFR-object reveals that the worst case has a non negligible component corresponding to some ε_{ij}, it might be concluded that the corresponding approximation is not valid.

A similar idea for approximate modelling is used in [43]. But in this reference, simplified interpolation formulas are restricted to linear expansions, and error bounds are computed by comparison of the exact and approximate models on a gridding.

4.4.2 Technical result

The main result of this section is stated in the following lemma.

Lemma 4.4.1 *Let us consider $M(\Delta)$ a (1×1) non-dynamic LFR-object with real (repeated) uncertainties. It is assumed that $M(\Delta)$ is well-posed (bounded) in the unit ball:*

$$M(\Delta) = \mathcal{F}_u(\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \Delta)$$

Let us define

$$S(\lambda) = M_{11} + M_{12}(\lambda - M_{22})^{-1} M_{21}$$

For λ varying from $-\infty$ to $+\infty$:

- *The minimum value of $M(\Delta)$ over the unit ball is the first value of λ at which $\mu(S(\lambda)) = 1$.*
- *The maximum value of $M(\Delta)$ over the unit ball is the last value of λ at which $\mu(S(\lambda)) = 1$.*

Proof. The LFT form of $M(\Delta) - \lambda I$ is:

$$M(\Delta) - \lambda I = \mathcal{F}_u(\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} - \lambda I \end{bmatrix}, \Delta)$$

From (2.22) (see, page 47), the non-singularity radius of $M(\Delta) - \lambda I$ is given by

$$\frac{1}{\mu(S(\lambda))}$$

Now, let us define as a and b the lower and upper bounds of variations of $M(\Delta)$ when Δ varies in the unit ball (these values are bounded because it is assumed that $M(\Delta)$ is well-posed in the unit ball):

$$a \leq M(\Delta) \leq b \quad \forall \Delta \text{ s.t. } \|\Delta\| \leq 1$$

So, clearly, the non-singularity radius of $M(\Delta) - \lambda I$ is larger than 1 if $\lambda < a$ or if $\lambda > b$, in other words:

$$\forall \lambda \notin [a \ \ b], \frac{1}{\mu(S(\lambda))} > 1$$

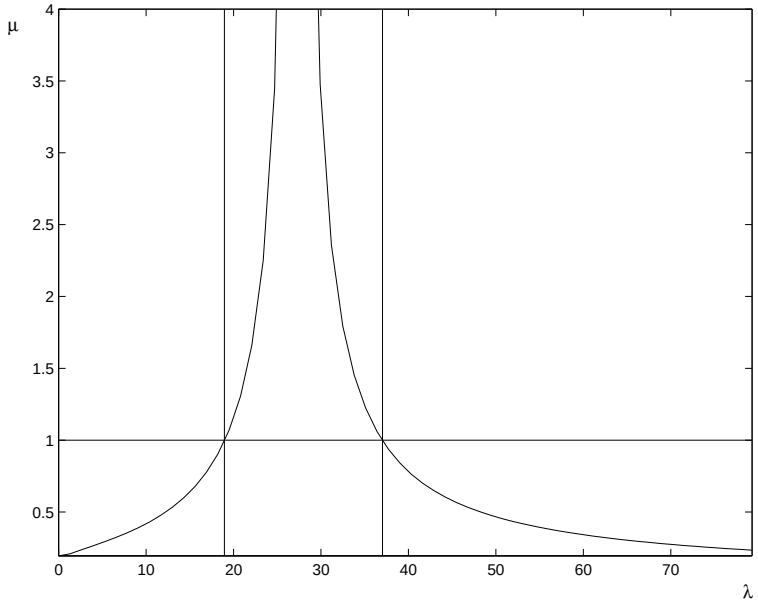

Figure 4.1: Example of an artificial μ analysis for finding minimum and maximum values of an LFR-object (here 19 and 37).

which proves the lemma. Note that the interval of variation of $M(\Delta) - \lambda I$ is not necessarily connected, in this case, $1/\mu(S(\lambda))$ becomes larger than one between a and b too, this is why we considered the "first" or "last" value of λ in the lemma. ■

Comment 4.4.2

- *The nominal value of $M(\Delta)$ that is $M(\Delta = 0) = M_{22}$ must not belong to the set of values that takes λ in the considered gridding. Otherwise, μ would be infinite at that point.*
- *It is clear that μ-curves must look like the one of Figure 4.1, i.e. lower than the unity for large values of λ and up to infinity when λ takes the nominal value of $M(\Delta)$.*
- *As μ cannot be computed exactly in most cases, it is an upper bound of μ that must be used. By analyzing Figure 4.1, it is clear that the conservatism of the upper bound will result in an interval of variations of $M(\Delta)$ larger than the actual one.*
- *As for all gridding technique there is a risk of missing a peak of the μ-curve. If such a problem is encountered, the computed interval of variations under-estimates the ac-*

tual one. Therefore, the frequency sweeping technique of [42] should be adapted to this problem.

Lower bounds. The upper bound computation can be validated by a lower bound of μ. Unfortunately, we consider here real μ-analysis, in this case lower bounds are not easy to compute. There is however a possibility that consists of adding a small imaginary component to the uncertain parameters. After this transformation, lower bound algorithms (those based on fixed point algorithms) are more likely to converge. It remains to use an iterative technique for getting rid of the imaginary components.

There is an alternative way for validating the μ upper-bound results (by comparison to a lower bound). Indeed, if we evaluate $M(\Delta)$ at several points in the parameter space preferably at the vertices of the parameter box, we obtain

- a lower bound of the maximum value of $M(\Delta)$ and
- an upper bound of the minimum value of $M(\Delta)$.

Extension to the non-real case. We shall not state the complex counterpart of Lemma 4.4.1 because we do not need this result here. However the result of this lemma can easily be generalized. If uncertainties are complex, more generally, if the considered LFR-object takes complex values, the analysis based on the limit of stability (§ 7.2, page 159) must be slightly adapted. Here instead of considering a real number λ as in Figure 7.1 we may consider a circle of radius λ, say

$$\lambda \, \frac{1-\delta^2+2j\delta}{1-\delta^2-2j\delta}$$

in which $\delta \in [-1\ 1]$ describes the unit circle. The parameter λ, radius of the circle, is varied from $+\infty$ towards zero in order to shrink the circle until the limit of stability is reached for the first time. The LFR-object describing the circle is of minimal order equal to 2 (with respect to δ). This object must be combined with $M(\Delta)$ for obtaining the system to be treated by an artificial μ-analysis (one μ-analysis for each value of λ). The first time λ is such that the maximum value of μ over all frequencies is equal to the unity, gives the radius of the smaller circle containing all the values of $M(\Delta)$ (for uncertain parameters in the unit ball). Here, μ analysis must be performed for all frequencies, but as we only need to compare the peak value of μ to one, there is an efficient algorithm in the LMI toolbox (`mustab` or `robuststab`) that avoids gridding with respect to the frequencies. Alternatively, a 2-D (gridding with respect to frequencies and λ) μ-analysis can be performed.

Software relative to this section

Illustrated function:

- `min_max`. Computes the minimum and maximum values (bounds) of a real valued 1×1 LFR-object.
- `reduclfr`. This function optimizes the tolerance argument used in reduction techniques (`minlfr1`, `minlfr`) for approximation with limited error.
- `udistlfr`. Computes the minimum and maximum values (bounds) of a real valued MIMO LFR-object.
- `bnds2lfr`. This function is complementary to `min_max` and `udistlfr` as it permits the designer to define systematically new uncertainties that take into account approximation errors.

Example 4.4 Let us consider an academic example in which simplifications can easily be guessed. Let $M_0 = (3\delta_1^5 + 0.0001\delta_1\delta_2\delta_3\delta_4)(1 - 0.0001\delta_4^4 + \delta_2^2\delta_3^2)$. It is natural to consider $M_1 = 3\delta_1^5(1 + \delta_2^2\delta_3^2)$ as an approximation of M_0 (all uncertainties are normalized).

The problem we shall treat consists of computing the approximation error bounds and the replacement of this approximation error by a new uncertain parameter.

```
lfrs d1 d2 d3 d4
M0 = (3*d1^5+.0001*d1*d2*d3*d4)*(1-.0001*d4^4+d2^2*d3^2);
M1 = (3*d1^5)*(1+d2^2*d3^2);
```

For computing the approximation error, we consider the difference

```
DeltM = M1 - M0;
DeltM = minlfr(DeltM);
```

Note that the size of `DeltM` was 26 before using `minlfr` and 16 after. For computing the error bounds:

```
[min_val,max_val,min_int,max_int] = min_max(DeltM);
```

The results are: `min_val = -9.2486e-04` and `max_val = 8.2905e-04` which means that `DeltM` varies in an interval included between these two values. It is possible to improve these bounds by switching to an LMI-based μ-upper bound (option `'accurate'` of `min_max`). The output arguments `min_int` and `max_int` give intervals respectively for `min_val` and `max_val`.

It remains to build M_3, an approximation of M_1. For that, we replace the neglected part of M_1 by an uncertainty (`err`).

```
lfrs err 'real' [-9.2486e-04] [8.2905e-04]
M3 = (3*d1^5)*(1+d2^2*d3^2)+err;
size(M3)

LFR-object with 1 output(s), 1 input(s) and 0 state(s).
Uncertainty blocks (globally (10 x 10)):
 Name   Dims   Type    Real/Cplx   Full/Scal      Bounds

 d1     5x5    LTI        r           s          [-1,1]
 d2     2x2    LTI        r           s          [-1,1]
 d3     2x2    LTI        r           s          [-1,1]
 err    1x1    LTI        r           s          [-0.00092486,0.00082905]
```

■■■■■■

Example 4.5 This example is similar to the previous one but considers a MIMO system. Approximation error modelling is performed in a more systematic way by using `bnds2lfr`. Let us consider a MIMO academic example.

$$S1 = \begin{bmatrix} 3+0.001*a^5-b*c & a^4*c \\ a*b*c^3+2 & a^2-0.001*b*c^3+1 \end{bmatrix}$$

Model:

```
>> lfrs a b c
>> S1 = [3+0.001*a^5-b*c a^4*c;a*b*c^3+2 a^2-0.001*b*c^3+1];
```

Approximation using `reduclfr`. This function chooses automatically the tolerance argument of order reduction algorithms (`minlfr` or `minlfr1`) by a dichotomy search so that the approximation error remains less than a given bound (here `0.01`, the option `'a'` means that we consider absolute rather than relative error).

```
>> S2 = reduclfr(S1,0.01,'a');
```

The function `reduclfr` considers a lower bound of the approximation error. The next function computes outer bounds of the intervals of variations:

```
>> [distu,dist2,mindiff,maxdiff] = udistlfr(S1,S2);
```

This function returns `mindiff` and `maxdiff` that are matrices having the same size as `S1`. These matrices give term by term

- a lower bound of the minimum value of the approximation error (`mindiff`)
- an upper bound of the maximum value of the approximation error (`maxdiff`)

It remains to replace the approximation errors by additional normalized real parameters

```
>> syserr = bnds2lfr('s_',mindiff,maxdiff);
>> S3 = S2 + syserr;
>> size(S3)

LFR-object with 2 output(s), 2 input(s) and 0 state(s).
Uncertainty blocks (globally (14 x 14)):
 Name   Dims   Type    Real/Cplx    Full/Scal      Bounds

 a      6x6    LTI        r            s          [-1,1]
 b      2x2    LTI        r            s          [-1,1]
 c      4x4    LTI        r            s          [-1,1]
 s_1_1  1x1    LTI        r            s          [-0.0012252,0.0012252]
 s_2_2  1x1    LTI        r            s          [-0.0012221,0.0012221]
```

Now `S3` has a Δ-block of size 14×14 to be compared to the original size of `S1` that is 23×23.

EMPTY PAGE

Chapter 5

Dynamic uncertainties modelling

5.1 Full complex blocks

Dynamic uncertainty modelling is encountered in several cases.

- The most natural case corresponds to *neglected dynamics modelling*. For example, for an aircraft, neglected dynamics might correspond to the structural modes. There are several ways to represent these uncertainties, for example direct additive like in Figure 5.1 or feedback multiplicative as in Figure 5.2.
- μ-analysis was introduced for analyzing stability robustness. Artificial dynamic blocks are often considered in order to extend the natural potentialities of μ-analysis to *performance analysis*.
- *Performance robustness analysis* (*e.g.* based on the Main Loop Theorem) is also based on the use of an artificial dynamical block.

The weighting matrix denoted $W(s)$ in Figures 5.1 and 5.2 comes from the fact that a bound on uncertainties $\Delta(j\omega)$ is more or less known under the following form

$$\forall \omega \in [0, \infty], \ \overline{\sigma}(\Delta(j\omega)) < W(j\omega)$$

This inequality can also be written

$$W(j\omega)\ \overline{\sigma}(\frac{\Delta(j\omega)}{W(j\omega)}) < W(j\omega)$$

So, the set of uncertainties over all frequencies can be represented by

$$W(j\omega)\Delta^C \text{ s.t. } \overline{\sigma}(\Delta^C) < 1$$

in which Δ^C replaces the ratio $\frac{\Delta(j\omega)}{W(j\omega)}$. It is in that way that are normalized dynamic uncertainties.

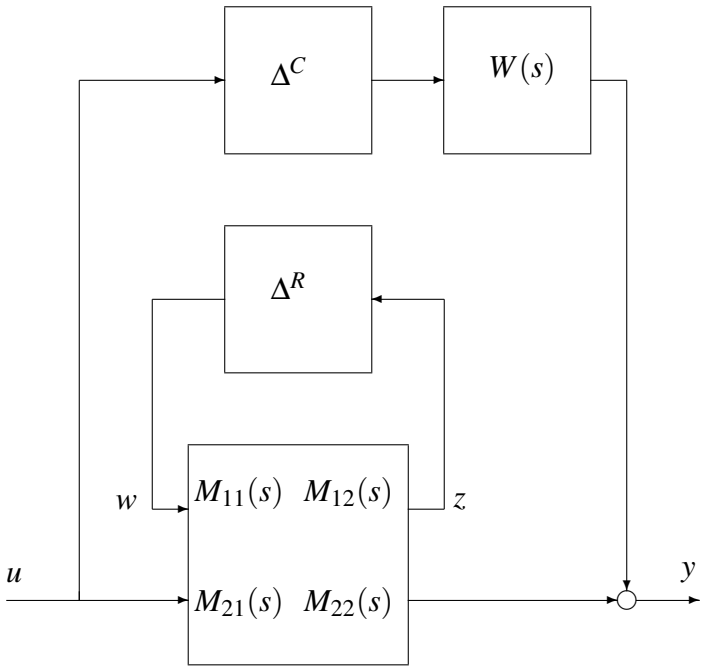

Figure 5.1: Direct additive neglected dynamics

The treatment of dynamic uncertainties is completely different from the one of real uncertain parameters for several reasons. As shown above, normalization is specific (done by means of weighting functions depending on the frequency). In addition, noting that dynamic uncertainties are usually well located in a bock diagram (not scattered in several equations like real parameter uncertainties), these uncertainties are not repeated, so, the problem of minimality is not relevant.

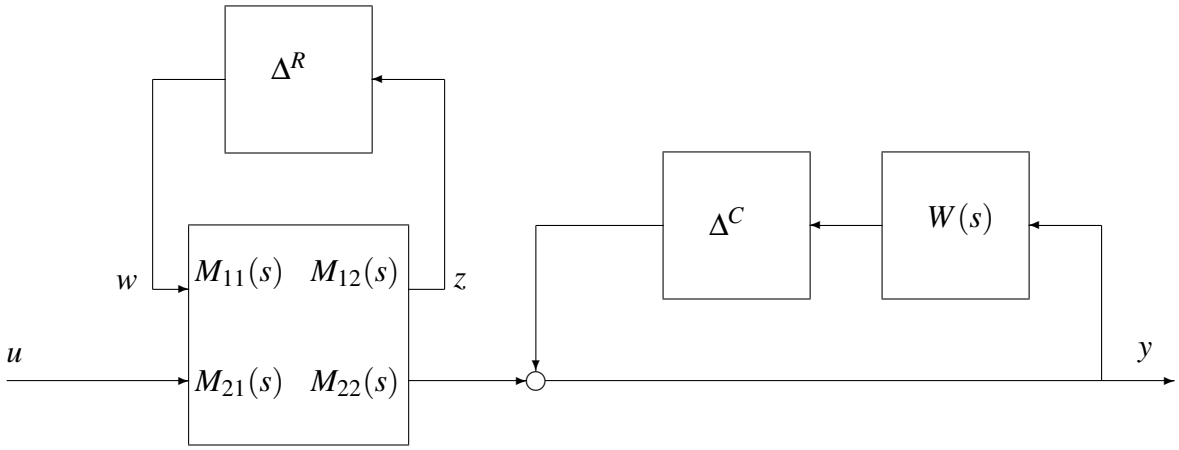

Figure 5.2: Feedback multiplicative neglected dynamics

Software relative to this section

Illustrated function:

- `lfr` for generation of full complex blocks. For use as in Figure 5.2 it suffices to invoke the function `feedback`. In the case of Figure 5.1, a simple sum of LFR-objects suffices.

Example 5.1 The function `lfr` offers an option for defining full blocks with frequency domain bounds. However, this feature is not already interfaced with MATLAB functions, therefore, it is suggested to build normalized full blocks and to add manually the weighting functions $W(s)$ as in Figures 5.1 and 5.2.

```
>> M = rlfr(5,2,3,4,4,4,'m');
>> W = ss(tf([30 100],[1 100])^2)*eye(2,2);
>> Delta = lfr('Delta','ltifc',[2,2]);
```

The transfer form u to y of Figure 5.2 is given by

```
>> sys = feedback(lfr(eye(2,2)),Delta*W,1)*M;

>> size(sys)

LFR-object with 2 output(s), 3 input(s) and 9 state(s).
Uncertainty blocks (globally (14 x 14)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 Delta 2x2   LTI       c           f          [-1,1]
 m1    4x4   LTI       r           s          [-1,1]
 m2    4x4   LTI       r           s          [-1,1]
 m3    4x4   LTI       r           s          [-1,1]
```

5.2 Complex scalar uncertainties

Scalar dynamic uncertainties are a special case of full complex blocks. Such uncertainties are often introduced for adaptation of μ-analysis to performance analysis. For example an ellipsoidal exclusion area in the Nichol chart can be defined as follows (Fielding [29]).

$$\{\frac{1+\beta(\delta_1+\delta_2)+\alpha\delta_1\delta_2}{1-\beta(\delta_1+\delta_2)+\alpha\delta_1\delta_2} \text{ s.t. } |\delta_1|<1 \text{ and } |\delta_2|<1\} \tag{5.1}$$

in which δ_1 and δ_2 are complex scalars. The transfer in this formula is to be considered in series with the open-loop system for which performance is analyzed (see [29] for details).

Building components that contain scalar uncertainties, is similar in both real and complex cases using the function `lfrs` with the option `'complex'`.

Software relative to this section

Illustrated functions:

- `lfrs` for generation of real or complex scalars 1×1 LFR-objects.

Example 5.2 This example shows how to generate the LFR-object of Equation (5.1) in which $\alpha = 1$ and $\beta = 2$. The uncertain parameters δ_1 and δ_2 are respectively denoted `d1` and `d2`. Two real parameters a and b are also considered. The LFR-object to be generated corresponds to:

$$\frac{a+1/s}{1+b/s}\ \frac{1+2(\delta_1+\delta_2)+\delta_1\delta_2}{1-2(\delta_1+\delta_2)+\delta_1\delta_2}$$

So we have:

```
>> lfrs  Int a b
>> lfrs  d1 d2 'complex'
>> sys = (1+2*(d1+d2)+d1*d2)/(1-2*(d1+d2)+d1*d2);
>> sys = (a+Int)/(1+b*Int)*sys;
>> size(sys)

LFR-object with 1 output(s), 1 input(s) and 2 state(s).
Uncertainty blocks (globally (10 x 10)):
 Name   Dims   Type    Real/Cplx    Full/Scal       Bounds

 a      1x1    LTI        r            s           [-1,1]
 b      1x1    LTI        r            s           [-1,1]
 d1     4x4    LTI        c            s           [-1,1]
 d2     4x4    LTI        c            s           [-1,1]
```

Chapter 6

Extensions to modelling of uncertain nonlinear systems

6.1 Introduction

The LFT approach can be used for modelling nonlinear systems at least in two ways.

- *The quasi-LPV (Linear Parameter Varying) approach*. It consists of modelling exactly the non-linear system (not a continuum of linearized models) by isolating in the Δ-block the nonlinear components of the system. There are usually many ways for designing such models. In [49] is proposed an interesting idea using the knowledge of the equilibrium surface (illustration considering a simplified nonlinear missile model), see also [50, 47, 44, 45].
- *Modelling a continuum of linearized models along the equilibrium surface*. Only this approach will be detailed here. Alternative techniques where it is not assumed that non-linear trajectories remain close to the equilibrium surface are proposed in literature, see for example [37, 38, 35].

We shall only consider the problem of modelling the continuum of linearized models along the equilibrium surface. Usually there are two stages related to two levels of complexity.

- First complexity level: The system is considered around equilibrium points *but parameter dependency at equilibrium as treated in §6.3 is ignored*.
- Second complexity level: The implicit dependency corresponding to the equilibrium hyper surface equation is identified and "plugged" into the previous LFR (see §6.3).

Usually the second level of complexity results in an "explosion" of the size of the LFR. Briefly, the first kind of models can be used for feedback design. For analysis, it is better to use the second kind of models in order to reduce conservatism.

Figure 6.1 illustrates the adequacy between accuracy of models (or banks of models) and available analysis tools. For the most accurate model that is the original nonlinear model, analysis is often limited to simulation. For the less accurate model (bank of linearized models) there are numerous tools. But each analysis tools must be applied to each model of the bank (the number of models of the considered bank increases exponentially with the number of varying / uncertain parameters). With LFR models, the number of criteria available is not so rich. But for applicable criteria, one-shoot worst case analysis is possible (without risk of missing the worst case as in the linearized models bank case).

The LFRs are somewhere in between linear and nonlinear models (for example non-linear components are included in the continuum of linearized models, dependency on uncertain parameters is also nonlinear).

The main causes of possible misfit with nonlinear models are

- the speed of parameter variations is not taken into account (the existing alternative tools that take it into account are very conservative),
- trajectories are expected to remain close to the equilibrium surface.

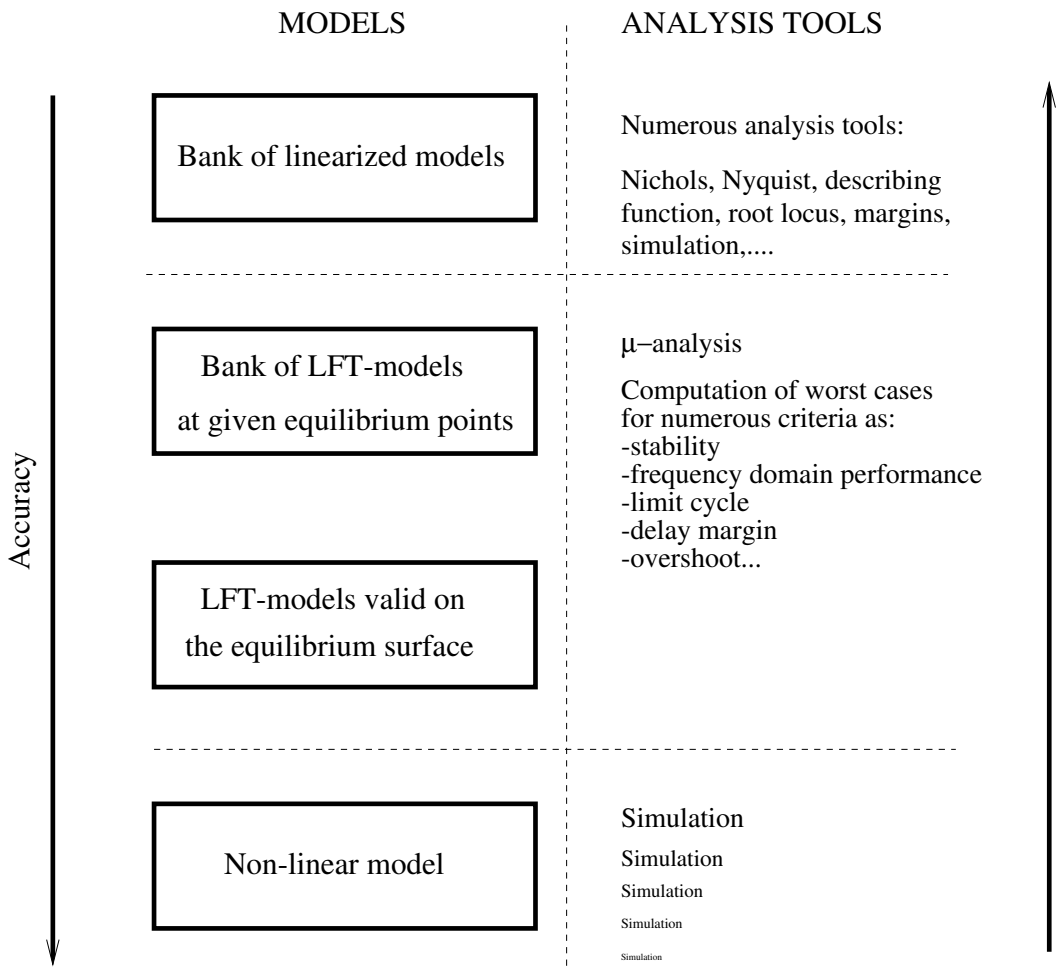

Figure 6.1: Models or banks of models and corresponding analysis tools

6.2 From a nonlinear model to a Linear Fractional Representation: Part 1

Assume that we have a non-linear model

$$\begin{aligned} \dot{x} &= f(x,u,p) \\ y &= g(x,u,p) \end{aligned} \tag{6.1}$$

in which $x \in \mathbf{R}^n, u \in \mathbf{R}^m, y \in \mathbf{R}^p$ are respectively the state, input and output vectors[1]. $p \in \mathbf{R}^q$ is the vector of varying parameters. The equilibrium hyper surface (depending on p) is defined by[2]

$$f(x_0,u_0,p) = 0 \tag{6.2}$$

There are n equations and $n+m+q$ unknowns. So, $m+q$ parameters including those in p, can be chosen (this new vector is denoted p'), the n remaining parameters (denoted ξ) are implicit functions of p'. The above equation will also be written

$$f(\xi,p') = 0 \tag{6.3}$$

In this section we shall ignore the fact that some parameters are implicit functions of others. This constraint will be considered in the next section.

6.2.1 Modelling ignoring parameter dependency at equilibrium

Let us compute the linearized model at a given point of the equilibrium surface. We shall have

$$\begin{aligned} \dot{x} &= A_{x_0,u_0,p}\, x + B_{x_0,u_0,p}\, u \\ y &= C_{x_0,u_0,p}\, x + D_{x_0,u_0,p}\, u \end{aligned} \tag{6.4}$$

in which x,u,y denote now the variations with respect to the equilibrium values (x_0,u_0,y_0). We have

$$A_{x_0,u_0,p} = \left.\frac{\partial f}{\partial x}\right|_{x_0,u_0,p} \;;\; B_{x_0,u_0,p} = \left.\frac{\partial f}{\partial u}\right|_{x_0,u_0,p} \tag{6.5}$$

$$C_{x_0,u_0,p} = \left.\frac{\partial g}{\partial x}\right|_{x_0,u_0,p} \;;\; D_{x_0,u_0,p} = \left.\frac{\partial g}{\partial u}\right|_{x_0,u_0,p} \tag{6.6}$$

[1] The notation p considered as the number of outputs is not used in this section, so, p indicates always the vector of parameters.

[2] Some integral states, for example coordinates of an aircraft, must be removed from (6.1) before computing equilibrium states.

The system of (6.4) will be written:

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A_{x_0,u_0,p} & B_{x_0,u_0,p} \\ C_{x_0,u_0,p} & D_{x_0,u_0,p} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{6.7}$$

In order to transform this equation to an LFR-object, in principle we have to transform (6.7) in the form of (2.3)-(2.4) (see Figure 2.2, page 19). We shall illustrate this task later considering a missile equation.

6.2.2 Differentiation of Linear Fractional Representations

The most natural way for deriving the matrices $A_{x_0,u_0,p}$, ... $D_{x_0,u_0,p}$ from (6.1) consists of computing a *symbolic form* of $f(x,u,p)$ and of $g(x,u,p)$ and then to use *symbolic computation* (function `sym/diff`) for the differentiation explicited in (6.5)-(6.6). See Example 6.1, page 130.

It is also possible to build $f(x,u,p)$ and of $g(x,u,p)$ as *LFR-objects* in which all the entries of the vectors x, u and p are considered as elementary LFR-objects. Then *LFR-object differentiation* (function `lfr/diff`, see §7.1.14, page 157) can be used as above. See Example 6.2, page 132.

The symbolic approach is more efficient because it is possible to use advanced realization tools (function `symtreed`) after differentiation. Generally, lower order are obtained in that way. The LFR approach is interesting if the Symbolic Toolbox is not available.

6.2.3 Missile model: Equations

This section introduces a simple nonlinear system that will permit us to illustrate the contents of Chapter 6 and of the corresponding software. The considered system is the missile model presented in [48].

Using standard notation (α angle of incidence, q pitch rate, Ma Mach number, δ_p tail plane deflection), the nonlinear longitudinal equation is

$$\begin{aligned} \dot{\alpha} &= K_1\, Ma\, C_z(\alpha, Ma, \delta_p)\cos(\alpha) + q \\ \dot{q} &= K_2\, Ma^2\, C_m(\alpha, Ma, \delta_p) \end{aligned} \tag{6.8}$$

in which

$$C_z(\alpha, Ma, \delta_p) = z_3\,\alpha^3 + z_2\,\alpha^2 + z_1\left(2 - \frac{1}{3}Ma\right)\alpha + z_0\,\delta_p \tag{6.9}$$

$$C_m(\alpha, Ma, \delta_p) = m_3\,\alpha^3 + m_2\,\alpha^2 + m_1\left(-7 + \frac{8}{3}Ma\right)\alpha + m_0\,\delta_p \tag{6.10}$$

The input is the signal δ_p. Numerical values are given in Table 6.1.

$Ma = \frac{V}{a}$	Mach number
$g = 32.2\, ft/s^2$	gravity
$\rho = 973.3\, lbs/ft^2$	static pressure at $20\,000\, ft$
$a = 1036.4\, ft/s$	sound velocity at $20\,000\, ft$
$S = 0.44\, ft^2$	reference surface
$m = 13.98\, slugs$	mass
$d = 0.75\, ft$	reference diameter
$I_y = 182.5\, slug \cdot ft^2$	inertia
$K_1 = 0.7\, \frac{\rho S}{ma}$	force coefficient
$K_2 = 0.7\, \frac{\rho S d}{I_y}$	torque coefficient
$K_3 = 0.7\, \frac{\pi}{180}\, \frac{\rho S}{mg}$	load factor coefficient
$z_3 = +19.347\, rad^{-3}$	C_z coefficient
$z_2 = -31.008\, rad^{-2}$	
$z_1 = -9.7174\, rad^{-1}$	
$z_0 = -1.9481\, rad^{-1}$	
$m_3 = +40.485\, rad^{-3}$	C_m coefficient
$m_2 = -64.166\, rad^{-2}$	
$m_1 = +2.9221\, rad^{-1}$	
$m_0 = -11.803\, rad^{-1}$	
$\zeta_a = 0.7$	actuator damping ratio
$\omega_a = 150\, rad/s$	actuator natural frequency

Table 6.1: Missile numerical data

The measure equation is (η load factor)

$$\eta = K_3\, Ma^2\, C_z(\alpha, Ma, \delta_p) \tag{6.11}$$

This system is in series with an actuator modelled as follows:

$$\delta_p = \frac{\omega_a^2}{s^2 + 2\,\zeta_a \omega_a\, s + \omega_a^2}\, \delta_c \tag{6.12}$$

The flight domain is defined by

$$\alpha = \alpha_0 + S_\alpha\, \delta\alpha \tag{6.13}$$

$$Ma = Ma_0 + S_{Ma}\, \delta Ma \tag{6.14}$$

with $\alpha_0 = 10^o = 0.1745\,rad$, $S_\alpha = 10^o = 0.1745\,rad$, $\delta\alpha \in [-1,\ 1]$ ($\delta\alpha$ is the normalized variation of α). Concerning the Mach number: $Ma_0 = 3$, $S_{Ma} = 1$, $\delta Ma \in [-1,\ 1]$. The flight domain is illustrated in Fig. 6.2.

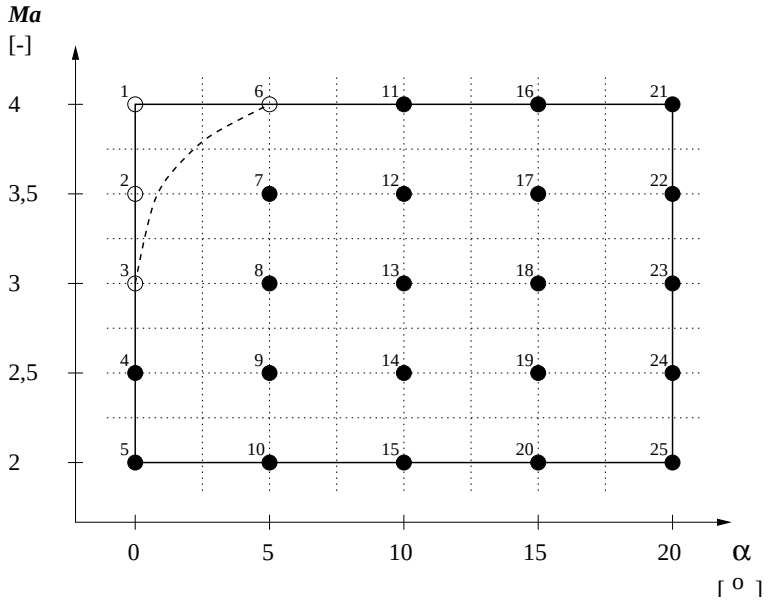

Figure 6.2: The missile flight domain

6.2.4 Missile model: Computation of the linearized models

Drawing a parallel with the generalities given in introduction of this section, the function $f(x,u,p)$ of Equation (6.1) corresponds to (6.8), the function $g(x,u,p)$ of Equation (6.1) corresponds to (6.11) and

$$x = \begin{bmatrix} \alpha \\ q \end{bmatrix} \ ;\ u = \delta_p \ ;\ p = Ma$$

So, the parameters to be considered at equilibrium are values of α, q, δ_p and Ma, satisfying "$f = 0$", that will be denoted α_0, q_0, δ_{p0} and Ma_0. The function $f(\alpha, q, \delta_p, Ma)$ consists of the substitution of Equations (6.9) and (6.10) into Equation (6.8).

Linearized models at equilibrium points. Let us compute the matrices of (6.5) and (6.6):

$$A_{\alpha_0,q_0,\delta_{p0},Ma_0} = \left.\frac{\partial f}{\partial x}\right|_{\alpha_0,q_0,\delta_{p0},Ma_0} \ ;\ B_{\alpha_0,q_0,\delta_{p0},Ma_0} = \left.\frac{\partial f}{\partial u}\right|_{\alpha_0,q_0,\delta_{p0},Ma_0}$$

$$C_{\alpha_0,q_0,\delta_{p0},Ma_0} = \left.\frac{\partial g}{\partial x}\right|_{\alpha_0,q_0,\delta_{p0},Ma_0} \; ; \; D_{\alpha_0,q_0,\delta_{p0},Ma_0} = \left.\frac{\partial g}{\partial u}\right|_{\alpha_0,q_0,\delta_{p0},Ma_0}$$

First, $cos(\alpha)$ is replaced by $1-\alpha^2/2$. After computation (we copy here the results obtained with the Symbolic Toolbox as detailed in Example 6.1), we obtain:

$$\begin{aligned} a_{11} = \; & K_1 M_{a_0}(3z_3\alpha_0^2 + 2z_2\alpha_0 + z_1(2-(1/3)M_{a_0}))(1-(1/2)\alpha_0^2) \\ & -K_1 M_{a_0}(z_3\alpha_0^3 + z_2\alpha_0^2 + z_1(2-(1/3)M_{a_0})\alpha_0 + z_0\delta_{p0})\alpha_0 \end{aligned}$$

$$A_{\alpha_0,q_0,\delta_{p0},Ma_0} = \begin{bmatrix} a_{11} & 1 \\ K_2 M_{a_0}^2(3m_3\alpha_0^2 + 2m_2\alpha_0 + m_1(-7+(8/3)M_{a_0})) & 0 \end{bmatrix} \tag{6.15}$$

$$B_{\alpha_0,q_0,\delta_{p0},Ma_0} = \begin{bmatrix} K_1 M_{a_0} z_0(1-(1/2)\alpha_0^2) \\ K_2 M_{a_0}^2 m_0 \end{bmatrix} \tag{6.16}$$

$$C_{\alpha_0,q_0,\delta_{p0},Ma_0} = \begin{bmatrix} K_3 M_{a_0}^2(3z_3\alpha_0^2 + 2z_2\alpha_0 + z_1(2-(1/3)M_{a_0})) & 0 \end{bmatrix} \tag{6.17}$$

$$D_{\alpha_0,q_0,\delta_{p0},Ma_0} = K_3 M_{a_0}^2 z_0 \tag{6.18}$$

Software relative to this section

Illustrated functions:

- `symtreed` for structured tree decomposition of a symbolic expression (Symbolic Toolbox required),
- `diff` for differentiating LFR-objects,
- `abcd2lfr` converts a system matrix LFR to an input/output LFR-object,

Brief presentation of the illustrative examples.

- Example 6.1 illustrates modelling of the *continuum* of linearized models of the missile ignoring parameter dependency on the equilibrium surface. The *symbolic* approach is used. Next step (considering the dependency on the equilibrium surface) in Example 6.3 (page 137).
- Example 6.2: Similar but the *LFR-object differentiation* is used instead symbolic differentiation. Next step in Example 6.4 (page 138).
- Example 6.3 (page 137) the dependencies on the equilibrium surface are handled by mean of symbolic computation (function `sym/eval`).
- Example 6.4 (page 138) the dependencies on the equilibrium surface are handled by mean of LFR-object manipulation (function `lfr/eval`).
- Example 6.5 (page 142) proposes an interpolation approach for identifying the dependencies of parameters on the equilibrium surface.
- Example 6.6 (page 143) proposes an interpolation approach for building the *continuum* of linearized models of the missile (the state-space matrices A, B, C, D that are interpolated come from a Simulink model of the missile).

Examples 6.1 to 6.4 need a common initialization of numerical data (given in Table 6.1). This initialization will not be repeated. These data can be loaded in Matlab workspace by invoking `missiledata`.

```
g = 32.2; rho = 973.3;  a = 1.0364e+03; s = 0.44; m = 13.98;
d = 0.75; Iy = 182.5;

K1 =     0.0207;  % K1 = 0.7*((rho*s)/(m*a));
K2 =     1.2320;  % K2 = 0.7*((rho*s*d)/Iy);
K3 =     0.0116;  % K3 = 0.7*(3.14/180)*((rho*s)/(m*g));

z3 =   19.3470;
z2 =  -31.0084;
z1 =   -9.7174;
z0 =   -1.9481;

m3 =   40.4847;
m2 =  -64.1657;
m1 =    2.9221;
m0 =  -11.8029;


% Actuator
kia = 0.7;
omegaa = 150;

% alpha in [Al_0-Al_S Al_0+Al_S]
Al_0 = 0.1745;
Al_S = 0.1745;

% mach in [Ma_0-Ma_S Ma_0+Ma_S]
Ma_0 = 3;
Ma_S = 1;
```

■■■■■■

Example 6.1 This example illustrates the symbolic approach to modelling in LFR form the *continuum* of linearized models of a nonlinear system. First, the symbolic expression of Equations (6.8) and (6.11) are computed (notation easily understandable from context).

```
>> missiledata
>> syms Al q Ma dp
```

```
>> Cz = z3*Al^3 + z2*Al^2 + z1*( 2 -(1/3)*Ma)*Al + z0*dp;
>> Cm = m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al + m0*dp;

>> A1 = q+K1*Ma*Cz*(1-Al^2/2);%+Al^4/24);
>> A2 = K2*Ma^2*Cm;
>> C1 = K3*Ma^2*Cz;
```

Now, the state-space matrices (see (6.5) and (6.6)) are computed by deriving the above expressions

```
>> fg = [A1;A2;C1];
>> ABCD = [diff(fg,'Al') diff(fg,'q') diff(fg,'dp')];
```

Then, the system matrix `[A B;C D]` is realized using the structured tree decomposition (`symtreed`), then, the input / output corresponding form is computed using `abcd2lfr`. The result is reduced using `minlfr`.

```
>> ABCD = symtreed(ABCD);
>> sys = abcd2lfr(ABCD,2);
>> sys = minlfr(sys,1000*eps);
```

The variations of `Al` and `Ma` must be normalized. respectively in `[Al_0-Al_S Al_0+AL_S]` and `[Ma_0-Ma_S Ma_0+Ma_S]`. Note also that the third parameter `dp` is entered with zero nominal value. It will be replaced later by a combination of `Ma` and `Al` (equilibrium constraint), so, its nominal value will be automatically computed from `Ma_0` and `Al_0`.

```
>> min = [Al_0-Al_S Ma_0-Ma_S];
>> max = [Al_0+Al_S Ma_0+Ma_S];
>> sys = normalizelfr(sys,{'Al','Ma'},min,max);
```

Finally, the actuator is added in series at the system input:

```
>> sys = sys*ss(tf([omegaa^2],[1 2*kia*omegaa omegaa^2]));
>> size(sys)

LFR-object with 1 output(s), 1 input(s) and 4 state(s).
Uncertainty blocks (globally (11 x 11)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds
```

```
Al    4x4   LTI       r          s          [-1,1]
Ma    6x6   LTI       r          s          [-1,1]
dp    1x1   LTI       r          s          [-1,1]
```

The resulting system has three uncertainty blocks of respective sizes 4 (α, angle of incidence), 6 (Ma, Mach number) and 1 (δp, tail plane deflection).

■■■■■■

Example 6.2 This example illustrates an alternative way to differentiate the functions f and g (see (6.1)) using a purely object oriented approach. The main function illustrated here is `diff`. The technique is very similar to the one illustrated in the previous example.

Here, the variation bounds of `Al` and `Ma` are specified when they are defined in the workspace:

```
>> missiledata
>> lfrs Al Ma [0.0 2.0] [0.349 4.0]
>> lfrs q dp

>> Cz = z3*Al^3 + z2*Al^2 + z1*( 2 -(1/3)*Ma)*Al + z0*dp;
>> Cm = m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al + m0*dp;

>> A1 = q+K1*Ma*Cz*(1-Al^2/2);%+Al^4/24);
>> A2 = K2*Ma^2*Cm;
>> C1 = K3*Ma^2*Cz;

>> fg = [A1;A2;C1];
>> fg = minlfr(fg,10000*eps);
```

Now, the state-space matrices (see (6.5) and (6.6)) are computed by deriving `fg` The system matrix `[A B;C D]` is realized in one step.

```
>> ABCD = [diff(fg,'Al') diff(fg,'q') diff(fg,'dp')];
```

Then, the input / output corresponding form is computed using `abcd2lfr`. The result is reduced using `minlfr`.

```
>> sys = abcd2lfr(ABCD,2);
>> sys = minlfr(sys,10000*eps);
```

The actuator is added in series at the system input:

```
>> sys_nn = sys*ss(tf([omegaa^2],[1 2*kia*omegaa omegaa^2]));
```

The parameter variations must be normalized, but as we have already defined variation bounds, it suffices to invoke the function `normalizelfr` without additional arguments:

```
>> sys = normalizelfr(sys_nn);
>> size(sys)

LFR-object with 1 output(s), 1 input(s) and 4 state(s).
Uncertainty blocks (globally (12 x 12)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 Al    4x4   LTI       r           s          [-1,1]
 Ma    7x7   LTI       r           s          [-1,1]
 dp    1x1   LTI       r           s          [-1,1]
```

The resulting system has three uncertainty blocks of respective sizes 4 (α, angle of incidence), 7 (Ma, Mach number) and 1 (δp, tail plane deflection). It can be checked using `distlfr` that this result is the same as the one of Example 6.1 (but with one block of order 7 instead of 6).

6.3 From a nonlinear model to a Linear Fractional Representation: Part 2

In Section 6.2 a preliminary LFR form of the *continuum* of linearized models of a nonlinear system was derived. In this section, it is shown that the dependency of the parameters at equilibrium can be viewed as an LFR-object that can be "plugged" into the *continuum* of linearized models. In that way, the resulting model will only depend on independent parameters.

6.3.1 Modelling considering parameter dependency at equilibrium

In Equations (6.4)-(6.7) the parameters x_0, u_0, p are not independent as they satisfy (6.2). In order to have a minimum set of independent parameters we have to choose $m+q$ (number of inputs plus number of uncertain / varying parameters) trim parameters within the entries of x_0, u_0 and p (all parameters in p must be selected). Let us denote p' the vector of the $m+q$ trim parameters. We shall denote ξ the remaining n parameters so that

$$\{\xi, p'\} = \{x_0, u_0, p\} \tag{6.19}$$

Now, in view of Equation (6.3), the entries of ξ are functions of p', so (6.7) has to be written:

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A_\xi & B_\xi \\ C_\xi & D_\xi \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{6.20}$$

A problem is often encountered when we try to write ξ as a function of p' because (6.2) is often an implicit relation. When this relation cannot be solved in an explicit way, we suggest to use interpolation (see §6.4). The next paragraph consider a large class of systems for which the relation is explicit.

6.3.2 Derivation of the equilibrium surface for a large class of systems

It is claimed in various references on qLPV modelling that for a large class of systems (including in particular aeronautical models) the state vector can be split into two parts x_1 and x_2 so that:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} k_1(x_1,p) \\ k_2(x_1,p) \end{bmatrix} + \begin{bmatrix} A_{11}(x_1,p) & A_{12}(x_1,p) \\ A_{21}(x_1,p) & A_{22}(x_1,p) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1(x_1,p) \\ B_2(x_1,p) \end{bmatrix} u \tag{6.21}$$

in which the vectors $x_1 \in \mathbf{R}^{n_1}$ can be measured and its length is equal to the length of the vector $u \in \mathbf{R}^m$.

$$n_1 = m \tag{6.22}$$

Equation (6.21) permits us to write the equilibrium surface in an explicit way. Indeed, we have

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} k_1(x_1,p) \\ k_2(x_1,p) \end{bmatrix} + \begin{bmatrix} A_{11}(x_1,p) & A_{12}(x_1,p) \\ A_{21}(x_1,p) & A_{22}(x_1,p) \end{bmatrix} \begin{bmatrix} x_1 \\ x_{20} \end{bmatrix} + \begin{bmatrix} B_1(x_1,p) \\ B_2(x_1,p) \end{bmatrix} u_0 \tag{6.23}$$

so, after some permutations

$$\begin{bmatrix} k_1(x_1,p) \\ k_2(x_1,p) \end{bmatrix} + \begin{bmatrix} A_{11}(x_1,p) \\ A_{21}(x_1,p) \end{bmatrix} x_1 = - \begin{bmatrix} A_{12}(x_1,p) & B_1(x_1,p) \\ A_{22}(x_1,p) & B_2(x_1,p) \end{bmatrix} \begin{bmatrix} x_{20} \\ u_0 \end{bmatrix}$$

Finally:

$$\begin{bmatrix} x_{20} \\ u_0 \end{bmatrix} = - \begin{bmatrix} A_{12}(x_1,p) & B_1(x_1,p) \\ A_{22}(x_1,p) & B_2(x_1,p) \end{bmatrix}^{-1} \begin{bmatrix} k_1(x_1,p) + A_{11}(x_1,p)x_1 \\ k_2(x_1,p) + A_{21}(x_1,p)x_1 \end{bmatrix} \tag{6.24}$$

From (6.22), the matrix inverted in the above equation is square ($(n_1+n_2)\times(n_2+m)$).

It is necessary to check the invertibility of this matrix for all feasible values of the vector (x_1,p). For that purpose, it suffices to build its LFR-form and to check that its well-posedness radius (§ 2.4.1, page 45) is larger than the unity, assuming that (x_1,p) has been normalized.

In view of Equation (6.19), we have

$$p' = \{x_1, p\} \quad \text{and} \quad \xi = \{x_{20}, u_0\} \tag{6.25}$$

6.3.3 Application to the missile model

The missile model is quite simple, therefore, deriving the equilibrium surface equation is almost obvious. However, in order to illustrate the above paragraph we shall write the missile nonlinear model ((6.8)-(6.9)-(6.10)) as in Equation (6.21).

$$x_1 = \alpha \ ; \ x_2 = q \ ; \ u = \delta p \ ; \ n_1 = m = 1$$

$$\begin{array}{rcl} \dot{x}_1 & = & \left(K_1 Ma(z_3 x_1^2 + z_2 x_1 + z_1(2 - \frac{1}{3}Ma))\cos x_1\right) x_1 \ + \ x_2 \ + \ K_1 Ma\, z_0 \cos x_1\, u \\ \dot{x}_2 & = & \left(K_2 Ma^2(m_3 x_1^2 + m_2 x_1 + m_1(-7 + \frac{8}{3})\right) x_1 \ + \ K_2 Ma^2 m_0\, u \end{array} \tag{6.26}$$

So, (6.24) can be written as follows

$$\begin{bmatrix} x_{20} \\ u_0 \end{bmatrix} = - \begin{bmatrix} 1 & K_1 Ma z_0 \cos x_1 \\ 0 & K_2 Ma^2 m_0 \end{bmatrix}^{-1} \begin{bmatrix} \left(K_1 Ma(z_3 x_1^2 + z_2 x_1 + z_1(2 - \frac{1}{3}Ma))\cos x_1\right) x_1 \\ \left(K_2 Ma^2(m_3 x_1^2 + m_2 x_1 + m_1(-7 + \frac{8}{3}Ma)\right) x_1 \end{bmatrix}$$

The above inversion is obvious, so

$$u_0 = - \frac{K_2 Ma^2(m_3 x_1^2 + m_2 x_1 + m_1(-7 + \frac{8}{3}Ma)x_1}{K_2 Ma^2 m_0}$$

or

$$u_0 = -\frac{(m_3 x_1^2 + m_2 x_1^2 + m_1(-7+\frac{8}{3}Ma)x_1}{m_0}$$

coming back to the original notations:

$$\delta_{p0} = -\frac{m_3\alpha^3 + m_2\alpha^2 + m_1(-7+\frac{8}{3}Ma)\alpha}{m_0} \tag{6.27}$$

Next step: the above form of δ_{p0} is sufficient to terminate the missile modelling initialized in Examples 6.1 or 6.4: it suffices to use `eval` to plug this equation into the state-space matrices.

Software relative to this section

Illustrated functions:

- `symtreed` for tree decomposition of a symbolic expression.
- `abcd2lfr` converts a system matrix in LFR form to an input/output LFR-object.
- `eval` for replacing some entries of the matrix Δ by LFR-objects (advanced "star product").

Example 6.3 See Example 6.1 for explanations relative to the first lines of MATLAB code.

```
>> missiledata

>> syms Al q Ma dp

>> Cz = z3*Al^3 + z2*Al^2 + z1*( 2 -(1/3)*Ma)*Al + z0*dp;
>> Cm = m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al + m0*dp;

>> A1 = q+K1*Ma*Cz*(1-Al^2/2);%+Al^4/24);
>> A2 = K2*Ma^2*Cm;
>> C1 = K3*Ma^2*Cz;

>> fg = [A1;A2;C1];
>> ABCD = [diff(fg,'Al') diff(fg,'q') diff(fg,'dp')];
```

In view of Equation (6.27), the equilibrium surface is given by

```
>> dp = -(m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al)/m0;
```

This form of `dq` is substituted into the state-space matrices by invoking the `sym/eval` function

```
>> ABCD = eval(ABCD);
```

These matrices do not depend any more on `dp`.

The system matrix `[A B;C D]` is realized using the structured tree decomposition (`symtreed`), and then, the input / output corresponding form is computed using `abcd2lfr`. The result is reduced using `minlfr`.

```
>> ABCD = symtreed(ABCD);
>> sys = abcd2lfr(ABCD,2);
>> sys = minlfr(sys,1000*eps);
```

The parameter variations must be normalized.

```
>> min = [Al_0-Al_S Ma_0-Ma_S];
>> max = [Al_0+Al_S Ma_0+Ma_S];
>> sys = normalizelfr(sys,{'Al','Ma'},min,max);
```

Finally, the actuator is added in series at the system input:

```
>> sys = sys*ss(tf([omegaa^2],[1 2*kia*omegaa omegaa^2]));
```

The resulting system has two uncertainty blocks of sizes 4 (α) and 6 (Mach number).

■■■■■■

Example 6.4 See Example 6.2 for obtaining the matrix `sys_nn`. The first step, in order to take equilibrium dependency, consists of writing `dp` as an LFR-object:

```
>> lfrs Al Ma [0.0 2.0] [0.349 4.0]
>> lfrs dp
>> dp = -(m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al)/m0;
```

It remains to replace `dp` in the Δ block of `ABCD` invoking `lfr/eval`

```
>> sys = eval(sys_nn);
```

Normalization:

```
>> sys = normalizelfr(sys);
```

We obtain the same result as in the previous example.

6.4 Techniques based on a gridding

The first step that must be considered for using the techniques presented in this section consists of computing a set of points (ξ, p') belonging to the equilibrium surface. Equation (6.2) can be written as in (6.3):

$$f(\xi, p') = 0$$

Let us consider a gridding of N values of p'. For each point of this gridding, compute the corresponding value of ξ (Simulink function `trim`). Therefore, we have a set of known vectors

$$\{\xi^{(1)}, \ldots, \xi^{(N)}, p'^{(1)} \ldots, p'^{(N)}\}$$

satisfying:

$$\begin{aligned} f(\xi^{(1)}, p'^{(1)}) &= 0 \\ &\vdots \\ f(\xi^{(N)}, p'^{(N)}) &= 0 \end{aligned}$$

6.4.1 Interpolation

The interpolation problem presented in this section can be applied to several practical problems, for example

- Finding the equilibrium surface equation (see Example 6.5).
- System matrices interpolation (see Example 6.6).
- Transformation of numerical data arrays (*e.g.*, aerodynamic coefficients) to polynomial or rational forms.

In order to make explicit the dependency of ξ on p', an interpolation formula must be chosen, for example:

$$\xi = \alpha_1 p'_1 + \alpha_2 p'^2_1 + \alpha_3 p'_2 p'_1 + \ldots$$

In which the α_i's are unknown vectors having the same size as ξ.

The problem of identifying the parameters of this interpolation formula can be considered by mean square minimization, that is finding the vectors α_1, α_2,... which are least squares solution to

$$\begin{aligned} \xi^{(1)} &= \alpha_1 p'^{(1)}_1 + \alpha_2 p'^{(1)2}_1 + \alpha_3 p'^{(1)}_2 p'^{(1)}_1 + \ldots \\ &\vdots \\ \xi^{(N)} &= \alpha_1 p'^{(N)}_1 + \alpha_2 p'^{(N)2}_1 + \alpha_3 p'^{(N)}_2 p'^{(N)}_1 + \ldots \end{aligned}$$

This equation can be written in matrix form:

$$\begin{bmatrix} \xi^{(1)} & \dots & \xi^{(N)} \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \dots \end{bmatrix} \begin{bmatrix} p'^{(1)}_1 & & p'^{(N)}_1 \\ p'^{(1)2}_1 & \cdots & p'^{(N)2}_1 \\ p'^{(1)}_2 p'^{(1)}_1 & & p'^{(N)}_2 p'^{(N)}_1 \\ \vdots & & \vdots \end{bmatrix}$$

The interpolation problem as above is treated in the MATLAB function `data2lfr`.

6.4.2 Elementary system modelling

From a data base of linearized models, the maximal and minimal values of each entry of the matrices A, B, C and D might be considered as the variation bounds of the uncertain parameters. This technique is evaluated in Bates *et al* [4]. The MATLAB function that computes that kind of LFR-objects is `bnds2lfr`.

The disadvantages of using such models is that worst case combinations of parameters cannot be identified, some conservatism is introduced and we have to rely on a gridding for computing the maximum and minimum values of the system matrix entries. Nevertheless, as a first analysis step, or if we just look for sufficient robustness condition (without worst case identification), the simplicity of this approach makes it very attractive.

Software relative to this section

Illustrated functions:

- `data2lfr` interpolation, the result is an LFR-object.
  - in Example 6.5 this function is used for the computation of an equilibrium surface by interpolation
  - in Example 6.6 this function is used for modelling the continuum of linearized systems by interpolation.
- `plotlfr` plots the values (entry per entry) of an LFR-object.

Examples 6.5 and 6.6 need a common initialization that consists of trimming and linearizing the missile model on a gridding of `Al` and `Ma`. The considered Simulink model of the missile (`missile.mdl`) is presented in Figure 6.3. The states of this Simulink diagram are respectively the 2 actuator states, α and q.

```
missiledata;
kk = 1;

for ii = 0:5;
for jj = 0:5;

    % Trim parameters
    Al = ii*(0.349/5); % from 0 to 0.349
    Ma = 2 + jj*(2/5); % from 2 to 4

    % Computation of equilibrium points
    X0 = [0;0;Al;0];
    [X,U,Y,dX] = trim('missile',X0,0,0,3);

    % Computation of linearized models
    [A,B,C,D] = linmod('missile',X,U);
    B = B/B(1,1);
    C = C*B(1,1);

    % Storage of values for state-space model interpolation
```

```
    abcd_data{kk} = [A B;C D];
    al_ma_data{kk} = [Al,Ma];

    % Storage of values for equilibrium surface interpolation
    q_dp_data{kk} = [X(4) U];  % values of q and dp

    kk = kk + 1;

end;
end;
```

(This script is given in the file `ex_6_5_loop.m`.)

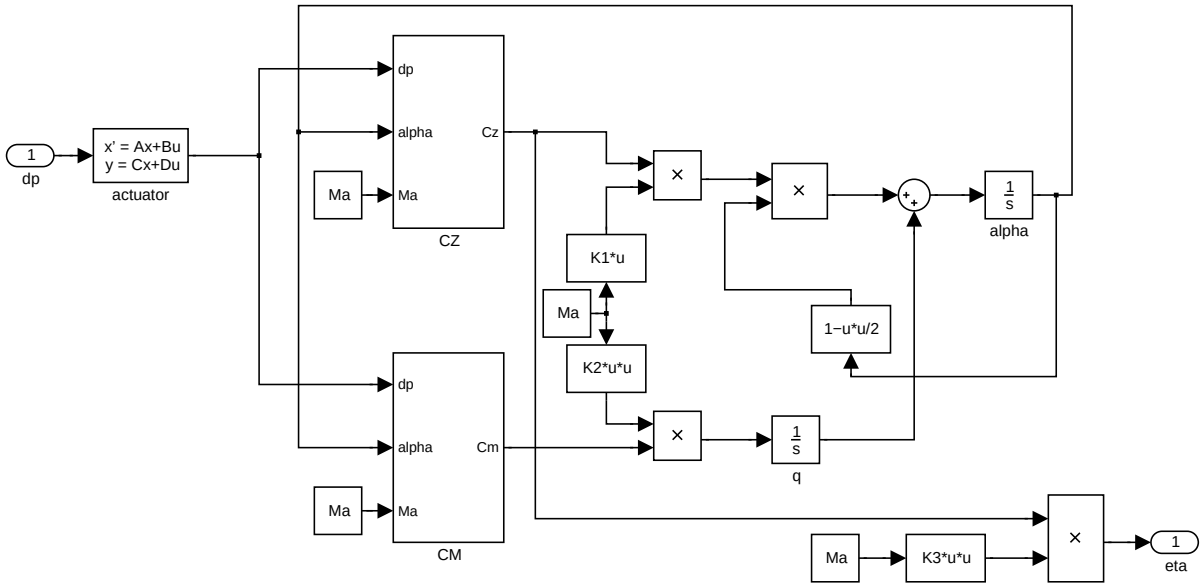

Figure 6.3: Simulink model of the missile

Example 6.5 *Equilibrium surface of the missile identified by interpolation:* The cell `al_ma_data` contains the values of α and of the Mach number defining the gridding. The cell `q_dp_data` contains the corresponding values of q and dp on the equilibrium surface.

The polynomial form of the equilibrium surface is chosen *a priori* as `lfrex = [1 Al Al*Ma Al^ 2]`. It means that we look for a_i's, b_i's such that q and dp are given by:

$$\begin{array}{rcl} q & = & a_0 + a_1\alpha + a_2\alpha Ma + a_3\alpha^2 \\ dp & = & b_0 + b_1\alpha + b_2\alpha Ma + b_3\alpha^2 \end{array} \tag{6.28}$$

```
>> lfrs Al Ma

>> lfrex = [1 Al Al*Ma Al^2];
>> ordlfr = {Al Ma};
>> q_dp = data2lfr(q_dp_data,al_ma_data,lfrex,ordlfr);

>> q  = q_dp(1);
>> dp = q_dp(2);
```

This result can be checked by comparison with the exact value of `dp` (denoted `dp2` below)

```
>> dp2 = -(m3*Al^3 + m2*Al^2 + m1*(-7 +(8/3)*Ma)*Al)/m0;
>> distlfr(dp,dp2)
```

There is a small discrepancy (distance about 0.0077), because `Al^3` is missing in the interpolation formula `lfrex` (in addition the linearization routine `linmod` introduces some approximations). It is also possible to compare the surfaces `dp` and `dp2`

```
>> figure
>> plotlfr(dp, {'Al',0,0.349,10},{'Ma',2,4,10});
>> hold on
>> plotlfr(dp2,{'Al',0,0.349,10},{'Ma',2,4,10});
```

■■■■■■

Example 6.6 *State-space matrices of the missile identified by interpolation:* The cell `al_ma_data` contains the values of α and of the Mach number defining the gridding. The cell `abcd_data` contains the corresponding state-space matrices.

Note that we have been obliged to normalize $B(1,1)$ in order to have continuity between the linearized models. In some cases, normalization is more complex.

The polynomial expansion (`lfrex`) below was chosen by "trial and errors" until the interpolation error became sufficiently small (for interpretation refer to (6.28)). From experience, it is not necessary to reduce so much interpolation error especially if the model is to be used for control design.

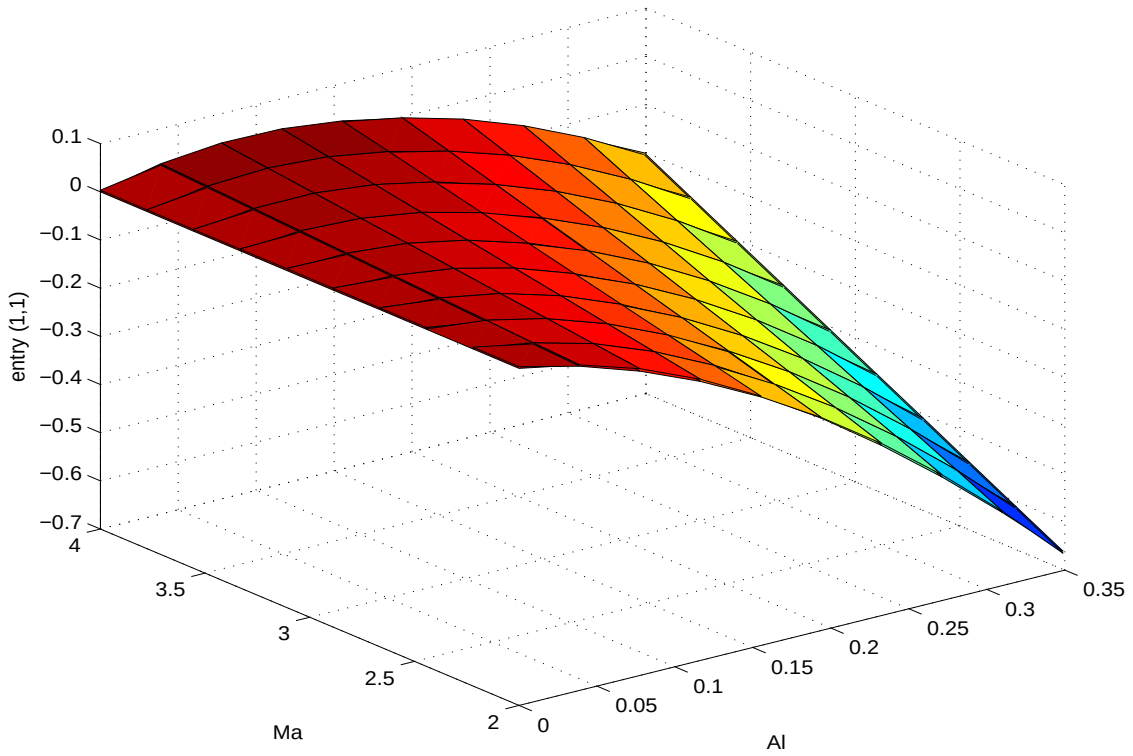

Figure 6.4: Exact and interpolated equilibrium surfaces cannot be distinguished.

```
>> lfrs Al Ma [0.0 2.0] [0.349 4.0]

>> lfrex = [1 Al Ma Al*Ma Al^2 Ma^2 Al^2*Ma Al*Ma^2 Al^3 Ma^3];
>> ordlfr = {Al Ma};
>> ABCD = data2lfr(abcd_data,al_ma_data,lfrex,ordlfr);
```

The result displayed to the screen is

```
Maximum absolute error = 1.296 at entry (4x3) of point 31
Maximum relative error = 7.8718 per cent at entry (4x3) of point 3
```

despite the appearances, the relative error is quite small, it seems to be large (7.5 %) because the coefficient (4,3) changes of sign, relative error is a bad measure around zero.

```
>> ABCD = minlfr(ABCD,0.0001);
```

The input/output form is computed as follows:

```
>> sys = abcd2lfr(ABCD,4);
>> sys = normalizelfr(sys);
>> size(sys)

LFR-object with 1 output(s), 1 input(s) and 4 state(s).
Uncertainty blocks (globally (11 x 11)):
 Name  Dims  Type   Real/Cplx   Full/Scal      Bounds

 Al    6x6   LTI       r           s          [-1,1]
 Ma    5x5   LTI       r           s          [-1,1]
```

It remains to analyse the distance from this system to the one of Example 6.4. The function `distlfr` gives a very pessimistic distance, however, the eigenvalues of both systems at random values of `Al` and `Ma` are almost equal.

EMPTY PAGE

Chapter 7

Appendix

7.1 Standard operation relative to LFTs

Let us consider $\mathcal{F}_u(M,\Delta'$, $\mathcal{F}_u(M',\Delta')$ and $\mathcal{F}_u(M'',\Delta'')$ where

$$M = \left[\begin{array}{cc} M_{11} & M_{12} \\ M_{21} & M_{22} \end{array}\right] ; M' = \left[\begin{array}{cc} M'_{11} & M'_{12} \\ M'_{21} & M'_{22} \end{array}\right] ; M'' = \left[\begin{array}{cc} M''_{11} & M''_{12} \\ M''_{21} & M''_{22} \end{array}\right]$$

So,

$$\begin{array}{l} \mathcal{F}_u(M,\Delta) = M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} + M_{22} \\ \mathcal{F}_u(M',\Delta') = M'_{21}\Delta'(I - M'_{11}\Delta')^{-1}M'_{12} + M'_{22} \\ \mathcal{F}_u(M'',\Delta'') = M''_{21}\Delta''(I - M''_{11}\Delta'')^{-1}M''_{12} + M''_{22} \end{array}$$

7.1.1 Transposition

Transposition holds for LFR-objects having only scalar repeated uncertainties (no full blocks so that $\Delta = \Delta^T$). In addition, if the uncertainties in the Δ-matrix are real, the following formula also holds for conjugate-transpose.

$$\mathcal{F}_u\left(M,\Delta\right)^T = \mathcal{F}_u\left(\left[\begin{array}{c|c} M_{11}^T & M_{21}^T \\ \hline M_{12}^T & M_{22}^T \end{array}\right], \Delta\right) \tag{7.1}$$

Proof.

$$\begin{array}{l} \mathcal{F}_u(M,\Delta)^T = M_{12}^T(I - \Delta^T M_{11}^T)^{-1}\Delta^T M_{21}^T + M_{22}^T \\ \mathcal{F}_u(M,\Delta)^T = M_{12}^T\Delta^T(I - M_{11}^T\Delta^T)^{-1}M_{21}^T + M_{22}^T \end{array}$$

$$\mathcal{F}_u(M,\Delta)^T = M_{12}^T\Delta(I-M_{11}^T\Delta)^{-1}M_{21}^T + M_{22}^T$$

Note that the restrictions listed above (no full blocks, no complex uncertainties in the conjugate-transpose case) come from the fact that Δ must be equal to Δ^T (or equal to Δ^* in the conjugate-transpose case). ■

7.1.2 Addition.

$$\mathcal{F}_u\left(M',\Delta'\right) + \mathcal{F}_u\left(M'',\Delta''\right) = \mathcal{F}_u\left(\left[\begin{array}{cc|c} M'_{11} & 0 & M'_{12} \\ 0 & M''_{11} & M''_{12} \\ \hline M'_{21} & M''_{21} & M'_{22}+M''_{22} \end{array}\right], \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix}\right) \tag{7.2}$$

Proof. This result can be justified considering the sum of $\mathcal{F}_u(M',\Delta')$ and $\mathcal{F}_u(M'',\Delta'')$. It can be written

$$\begin{bmatrix} M'_{21} & M''_{21} \end{bmatrix} \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix} \left(I - \begin{bmatrix} M'_{11} & 0 \\ 0 & M''_{11} \end{bmatrix} \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix}\right)^{-1} \begin{bmatrix} M'_{12} \\ M''_{12} \end{bmatrix} + M'_{22} + M''_{22}$$

which corresponds to (7.2). Usually the δ_i's are reordered so that they become contiguous (not scattered in two diagonal blocks). Reordering the uncertain parameters consists of permuting the columns and rows of

$$\begin{bmatrix} M'_{11} & 0 & M'_{12} \\ 0 & M''_{11} & M''_{12} \\ M'_{21} & M''_{21} & M'_{22}+M''_{22} \end{bmatrix} \text{ and } \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix}$$

in a similar way. ■

7.1.3 Multiplication.

$$\mathcal{F}_u\left(M',\Delta'\right)\mathcal{F}_u\left(M'',\Delta''\right) = \mathcal{F}_u\left(\left[\begin{array}{cc|c} M'_{11} & M'_{12}M''_{21} & M'_{12}M''_{22} \\ 0 & M''_{11} & M''_{12} \\ \hline M'_{21} & M'_{22}M''_{21} & M'_{22}M''_{22} \end{array}\right], \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix}\right) \tag{7.3}$$

It remains to reorder the δ_i's as above. Justification as above.

7.1.4 Concatenation.

$$\begin{bmatrix} \mathcal{F}_u\left(M',\Delta'\right) \\ \mathcal{F}_u\left(M'',\Delta''\right) \end{bmatrix} = \mathcal{F}_u\left(\left[\begin{array}{cc|c} M'_{11} & 0 & M'_{12} \\ 0 & M''_{11} & M''_{12} \\ \hline M'_{21} & 0 & M'_{22} \\ 0 & M''_{21} & M''_{22} \end{array}\right], \begin{bmatrix} \Delta' & 0 \\ 0 & \Delta'' \end{bmatrix}\right) \tag{7.4}$$

$$\left[\begin{array}{cc} \mathcal{F}_u(M',\Delta') & \mathcal{F}_u(M'',\Delta'') \end{array}\right] = \mathcal{F}_u\left(\left[\begin{array}{cc|cc} M'_{11} & 0 & M'_{12} & 0 \\ 0 & M''_{11} & 0 & M''_{12} \\ \hline M'_{21} & M''_{21} & M'_{22} & M''_{22} \end{array}\right], \left[\begin{array}{cc} \Delta' & 0 \\ 0 & \Delta'' \end{array}\right]\right) \tag{7.5}$$

In both cases it remains to reorder the δ_i's as above. Justification as above.

7.1.5 Juxtaposition.

This is the operation called "append" in toolboxes.

$$\left[\begin{array}{cc} \mathcal{F}_u(M',\Delta') & 0 \\ 0 & \mathcal{F}_u(M'',\Delta'') \end{array}\right] = \mathcal{F}_u\left(\left[\begin{array}{cc|cc} M'_{11} & 0 & M'_{12} & 0 \\ 0 & M''_{11} & 0 & M''_{12} \\ \hline M'_{21} & 0 & M'_{22} & 0 \\ 0 & M''_{21} & 0 & M''_{22} \end{array}\right], \left[\begin{array}{cc} \Delta' & 0 \\ 0 & \Delta'' \end{array}\right]\right) \tag{7.6}$$

In both cases it remains to reorder the δ_i's as above. Justification as above.

7.1.6 Inversion.

It is assumed that M_{22} is square invertible

$$\mathcal{F}_u(M,\Delta)^{-1} = \mathcal{F}_u\left(\left[\begin{array}{c|c} M_{11}-M_{12}M_{22}^{-1}M_{21} & M_{12}M_{22}^{-1} \\ \hline -M_{22}^{-1}M_{21} & M_{22}^{-1} \end{array}\right], \Delta\right) \tag{7.7}$$

Note that we have the same Δ-block for the considered LFR and its inverse.

Proof. This is a form of the well known Matrix Inversion Lemma:

$$(A_{22}+A_{21}A_{11}^{-1}A_{12})^{-1} = A_{22}^{-1}+A_{22}^{-1}A_{21}(A_{11}-A_{12}A_{22}^{-1}A_{21})^{-1}A_{12}A_{22}^{-1}$$

in which we replace A_{11} by $I-M_{11}\Delta$, A_{12} by M_{12}, A_{21} by $-M_{21}\Delta$ and A_{22} by M_{22}:

$$\begin{aligned}(M_{22}+M_{21}\Delta(I-M_{11}\Delta)^{-1}M_{12})^{-1} = \\ M_{22}^{-1}-M_{22}^{-1}M_{21}\Delta(I-(M_{11}-M_{12}M_{22}^{-1}M_{21})\Delta)^{-1}M_{12}M_{22}^{-1}\end{aligned}$$

which is as stated in (7.7). ■

7.1.7 Feedback

The closed-loop of a system G with feedback K is $(I+GK)^{-1}G$. We shall not detail the realization of $(I+GK)^{-1}G$ as for the previous operations but give a formula involving already treated operations (addition, multiplication and concatenation of lfr-objects). The formula to be used must be such that K and G appear only once in order to reduce the complexity:

$$(I+GK)^{-1}G = \begin{bmatrix} -I & 0 \end{bmatrix} \left(I + \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} K & I \\ 0 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{7.8}$$

7.1.8 Kernel computation

Kernel or null space computation is a specific problem because a kernel is not a matrix but is a subspace. In the LFR framework, we shall compute an LFR-object that, for all fixed values of Δ, corresponds to a basis of the null space. The main problem is that a basis is defined up to the right multiplication by a nonsingular matrix (change of basis). In the LFT case, all possible basis have not the same well-posedness radius. For example the kernel of the realization of $[(1+a) \;\; 2]$ can be

$$\text{Ker} \begin{bmatrix} (1+a) & 2 \end{bmatrix} = \text{Im} \begin{bmatrix} \frac{1}{1+a} \\ \frac{-1}{2} \end{bmatrix} = \text{Im} \begin{bmatrix} 2 \\ -1-a \end{bmatrix}$$

A realization of the first form of the kernel is well-posed for $\|a\| < 1$, the second one is always well-posed. Optimizing the well-posedness radius is a difficult task but this problem must be considered in order to derive kernel representations having a too small well-posedness radius.

Algorithm 1. It is assumed that the $p \times m$ matrix M_{22} has maximal row rank (rank(M_{22}) = p).

- **Step 1.** Compute a $m \times (m-p)$ matrix Q' the columns of which span the kernel of M_{22}. We shall denote $Q = Q'^T$.
- **Step 2.** The kernel $X(\Delta)$ of $\mathcal{F}_u(M,\Delta)$ is given by

$$X(\Delta) = \begin{bmatrix} M(\Delta) \\ Q \end{bmatrix}^{-1} \begin{bmatrix} 0_{p\times(m-p)} \\ I_{m-p} \end{bmatrix} \tag{7.9}$$

Proof. Note that the formula of Equation (7.9) is written in such a way that $X(\Delta)$ has the same Δ matrix as $\mathcal{F}_u(M,\Delta) = \mathcal{F}_u(M,\Delta)$. In order to justify (7.9), it suffices to check that $\mathcal{F}_u(M,\Delta)X(\Delta) = 0$.

$$\mathcal{F}_u(M,\Delta)X(\Delta) = \mathcal{F}_u(M,\Delta) \begin{bmatrix} M(\Delta) \\ Q \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}$$

so,

$$\mathcal{F}_u(M,\Delta)X(\Delta) = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} 0 \\ I \end{bmatrix} = 0$$

which justifies the proposed algorithm. ■

Comment 7.1.1 *This choice of Q can be justified by considering the row spans of $\mathcal{F}_u(M,\Delta)$ and of Q. We selected Q as having a row span orthogonal to the one of $\mathcal{F}_u(M,\Delta)$ at $\Delta = 0$, because it is likely that Δ has to change quite a lot before both row spans have a non-zero intersection.*

Algorithm 2. It is assumed that the $p \times m$ matrix M_{22} has maximal row rank (rank(M_{22}) = p).

- **Step 1.** Compute the non-singularity radius of all the $p \times p$ minors of $\mathcal{F}_u(M,\Delta)$). Select the "best one". Let P_1 denote the $m \times p$ permutation matrix that moves the selected minor to the left, P_2 is a complementary $m \times (m-p)$ permutation matrix.
- **Step 2.** The kernel $X(\Delta)$ of $\mathcal{F}_u(M,\Delta)$ is given by

$$X(\Delta) = \begin{bmatrix} P_1 & P_2 \end{bmatrix} \begin{bmatrix} \mathcal{F}_u(M,\Delta)P_1 & \mathcal{F}_u(M,\Delta)P_2 \\ 0_{(m-p)\times p} & I_{m-p} \end{bmatrix}^{-1} \begin{bmatrix} 0_{p\times(m-p)} \\ I_{m-p} \end{bmatrix} \tag{7.10}$$

Proof. Note that Equation (7.10) is written in such a way that $X(\Delta)$ has the same matrix Δ as the original LFR-object $\mathcal{F}_u(M,\Delta)$. In order to justify (7.10), it suffices to check that $\mathcal{F}_u(M,\Delta)X(\Delta) = 0$.

$$\mathcal{F}_u(M,\Delta)X(\Delta) = \mathcal{F}_u(M,\Delta) \begin{bmatrix} P_1 & P_2 \end{bmatrix} \begin{bmatrix} \mathcal{F}_u(M,\Delta)P_1 & \mathcal{F}_u(M,\Delta)P_2 \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}$$

The above inverse can be written,

$$\begin{bmatrix} (\mathcal{F}_u(M,\Delta)P_1)^{-1} & -(\mathcal{F}_u(M,\Delta)P_1)^{-1}\mathcal{F}_u(M,\Delta)P_2 \\ 0 & I \end{bmatrix}$$

so,

$$\mathcal{F}_u(M,\Delta)X(\Delta) = -\mathcal{F}_u(M,\Delta)P_1(\mathcal{F}_u(M,\Delta)P_1)^{-1}\mathcal{F}_u(M,\Delta)P_2 + \mathcal{F}_u(M,\Delta)P_2 = 0$$

which justifies the algorithm. ■

Comment 7.1.2 *This procedure is again a technique for enlarging the domain in which the inversion appearing in (7.7) remains feasible. Here, μ-analysis is required (a single μ-test for each minor in order to compute its non-singularity radius), so, the computation is longer than using Algorithm 1. This idea can also be applied considering the minors of $\mathcal{F}_u(M,\Delta)$ for $\Delta = 0$ in order to avoid the use of μ-analysis.*

7.1.9 Real and imaginary parts

It is assumed that the entries of the matrix Δ are all real. Let us denote:

$$\begin{aligned} M_{11} &= M_{11}^R + jM_{11}^I \\ M_{12} &= M_{12}^R + jM_{12}^I \\ M_{21} &= M_{21}^R + jM_{21}^I \\ M_{22} &= M_{22}^R + jM_{22}^I \end{aligned}$$

The real part of $\mathcal{F}_u(M,\Delta)$ is given by:

$$\Re(\mathcal{F}_u(M,\Delta)) = \mathcal{F}_u\left(\left[\begin{array}{cc|c} M_{11}^R & -M_{11}^I & M_{12}^R \\ M_{11}^I & M_{11}^R & M_{12}^I \\ \hline M_{21}^R & -M_{21}^I & M_{22}^R \end{array}\right], \left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\right) \tag{7.11}$$

and the imaginary part of $\mathcal{F}_u(M,\Delta)$ is given by:

$$\Im(\mathcal{F}_u(M,\Delta)) = \left(\left[\begin{array}{cc|c} M_{11}^R & -M_{11}^I & M_{12}^R \\ M_{11}^I & M_{11}^R & M_{12}^I \\ \hline M_{21}^I & M_{21}^R & M_{22}^I \end{array}\right], \left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\right) \tag{7.12}$$

Proof. Only for the real part:

$$\mathcal{F}_u(M,\Delta) = M_{22}^R + jM_{22}^I + (M_{21}^R + jM_{21}^I)\Delta(I - (M_{11}^R + jM_{11}^I)\Delta)^{-1}(M_{12}^R + jM_{12}^I)$$

We shall denote

$$H^R + jH^I = (I - (M_{11}^R + jM_{11}^I)\Delta)^{-1}(M_{12}^R + jM_{12}^I) \tag{7.13}$$

so that

$$\Re(\mathcal{F}_u(M,\Delta)) = M_{22}^R + \left[\begin{array}{cc} M_{21}^R & -M_{21}^I \end{array}\right]\left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\left[\begin{array}{c} H^R \\ H^I \end{array}\right] \tag{7.14}$$

Multiplying (7.13) on the left by $(I - (M_{11}^R + jM_{11}^I)\Delta)$:

$$(I - (M_{11}^R + jM_{11}^I)\Delta)(H^R + jH^I) = M_{12}^R + jM_{12}^I$$

or, isolating imaginary and real parts

$$\left\{\begin{array}{rcl} (I - M_{11}^R\Delta)H^R + M_{11}^I\Delta H^I & = & M_{12}^R \\ (I - M_{11}^R\Delta)H^I - M_{11}^I\Delta H^R & = & M_{12}^I \end{array}\right.$$

which is equivalent to

$$\left(I - \left[\begin{array}{cc} M_{11}^R & -M_{11}^I \\ M_{11}^I & M_{11}^R \end{array}\right]\left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\right)\left[\begin{array}{c} H^R \\ H^I \end{array}\right] = \left[\begin{array}{c} M_{12}^R \\ M_{12}^I \end{array}\right]$$

H^R and H^I can be expressed from this equation, after substitution in (7.14) the following formula is obtained $\Re(\mathcal{F}_u(M,\Delta)) =$

$$M_{22}^R + \left[\begin{array}{cc} M_{21}^R & -M_{21}^I \end{array}\right]\left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\left(I - \left[\begin{array}{cc} M_{11}^R & -M_{11}^I \\ M_{11}^I & M_{11}^R \end{array}\right]\left[\begin{array}{cc} \Delta & 0 \\ 0 & \Delta \end{array}\right]\right)^{-1}\left[\begin{array}{c} M_{12}^R \\ M_{12}^I \end{array}\right]$$

which is as stated in (7.11). ■

Lemma 7.1.3 *The well-posedness radius of $\Re(M)$ and of $\Im(M)$ are equal to the well-posedness radius of M*

Proof. In view of the discussion of §2.4, it suffices to show that the matrices

$$I-(M_{11}^R+jM_{11}^I)\Delta \quad \text{and} \quad I-\begin{bmatrix} M_{11}^R & -M_{11}^I \\ M_{11}^I & M_{11}^R \end{bmatrix}\begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}$$

are rank deficient for the same values of Δ. This result is obvious from

$$(I-(M_{11}^R+jM_{11}^I)\Delta)(\xi^R+j\xi^I)=0 \Leftrightarrow \left(I-\begin{bmatrix} M_{11}^R & -M_{11}^I \\ M_{11}^I & M_{11}^R \end{bmatrix}\begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}\right)\begin{bmatrix} \xi^R \\ \xi^I \end{bmatrix}=0$$

■

7.1.10 Concatenation and conjugation

For control design we often need to manipulate the concatenation of a complex vector and of its conjugate:

$$[\mathcal{F}_u(M,\Delta) \quad \text{conj}(\mathcal{F}_u(M,\Delta))]$$

this object is not very interesting for computing feedback gains (with real coefficients) because it contains complex numbers in its realization. In fact, it can be shown that we can replace this object by the following one

$$[\Re(\mathcal{F}_u(M,\Delta)) \quad \Im(\mathcal{F}_u(M,\Delta))]$$

that has interesting properties (see (7.15))

- the size of the Δ matrix of this object is only twice the original size (from (7.11) and (7.12), at first sight, it seems that its size is four times the original one).
- the four matrices of its realization are real (if the original ones are real).

$$[\Re(\mathcal{F}_u(M,\Delta)) \quad \Im(\mathcal{F}_u(M,\Delta))]=$$

$$\mathcal{F}_u\left(\left[\begin{array}{cc|cc} M_{11}^R & -M_{11}^I & M_{12}^R & M_{12}^I \\ M_{11}^I & M_{11}^R & M_{12}^I & -M_{12}^R \\ \hline M_{21}^R & -M_{21}^I & M_{22}^R & M_{22}^I \end{array}\right], \begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}\right) \tag{7.15}$$

Proof. For simplifying notation, let us define

$$A = \begin{bmatrix} M_{11}^R & -M_{11}^I \\ M_{11}^I & M_{11}^R \end{bmatrix} ; B = \begin{bmatrix} M_{12}^R \\ M_{12}^I \end{bmatrix} ; \Delta = \mathrm{diag}\{\Delta, \Delta\}$$

$$C_1 = \begin{bmatrix} M_{21}^R & -M_{21}^I \end{bmatrix} ; C_2 = \begin{bmatrix} M_{21}^I & M_{21}^R \end{bmatrix} ; D_1 = M_{22}^R ; D_2 = M_{22}^I$$

and

$$Q = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}$$

>From (7.11) and (7.12), this notation leads to

$$\Re(\mathcal{F}_u(M,\Delta)) = \mathcal{F}_u\left(\begin{bmatrix} A & B \\ C_1 & D_1 \end{bmatrix}, \Delta\right)$$

$$\Im(\mathcal{F}_u(M,\Delta)) = \mathcal{F}_u\left(\begin{bmatrix} A & B \\ C_2 & D_2 \end{bmatrix}, \Delta\right)$$

It is clear that the matrix Q satisfies the following properties

$$Q^T Q = I \; ; \; C_1 Q = C_2 \; ; \; QA - AQ = 0 \; ; \; Q\Delta - \Delta Q = 0$$

we shall use the following system similarity transformation

$$\begin{bmatrix} I & -Q \\ 0 & I \end{bmatrix}$$

This system similarity transformation is compatible with the considered LFR-object because the Δ-matrix (here $\mathrm{diag}\{\Delta,\Delta\}$ is invariant under the following transformation

$$\begin{bmatrix} I & Q \\ 0 & I \end{bmatrix} \mathrm{diag}\{\Delta,\Delta\} \begin{bmatrix} I & -Q \\ 0 & I \end{bmatrix} = \mathrm{diag}\{\Delta,\Delta\}$$

therefore, this transformation can be applied to the realization of the concatenated object $[\Re(\mathcal{F}_u(M,\Delta)) \;\; \Im(\mathcal{F}_u(M,\Delta))]$.

$$[\Re(\mathcal{F}_u(M,\Delta)) \;\; \Im(\mathcal{F}_u(M,\Delta))] = \mathcal{F}_u\left(\left[\begin{array}{cc|cc} A & 0 & B & 0 \\ 0 & A & 0 & B \\ \hline C_1 & C_2 & D_1 & D_2 \end{array}\right], \begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}\right)$$

Applying the system similarity transformation, it is equal to

$$\mathcal{F}_u\left(\left[\begin{array}{c|c} \begin{bmatrix} I & Q \\ 0 & I \end{bmatrix}\begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix}\begin{bmatrix} I & -Q \\ 0 & I \end{bmatrix} & \begin{bmatrix} I & Q \\ 0 & I \end{bmatrix}\begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix} \\ \hline \begin{bmatrix} C_1 & C_2 \end{bmatrix}\begin{bmatrix} I & -Q \\ 0 & I \end{bmatrix} & \begin{bmatrix} D_1 & D_2 \end{bmatrix} \end{array}\right], \begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}\right)$$

that is

$$\mathcal{F}_u\left(\left[\begin{array}{cc|cc} A & 0 & B & QB \\ 0 & A & 0 & B \\ \hline C_1 & 0 & D_1 & D_2 \end{array}\right], \begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix}\right)$$

which reduces to (definition of $\mathcal{F}_u$)

$$\mathcal{F}_u\left(\left[\begin{array}{c|cc} A & B & QB \\ \hline C_1 & D_1 & D_2 \end{array}\right], \Delta\right)$$

which is as stated in (7.15) ■

Comment 7.1.4 *The object* $[\Re(\mathcal{F}_u(M,\Delta)) \;\; \Im(\mathcal{F}_u(M,\Delta))]$ *can be computed more simply using*

$$[\Re(\mathcal{F}_u(M,\Delta)) \;\; \Im(\mathcal{F}_u(M,\Delta))] = [\mathcal{F}_u(M,\Delta) \;\; conj(\mathcal{F}_u(M,\Delta))] \begin{bmatrix} 0.5 & -0.5j \\ 0.5 & 0.5j \end{bmatrix}$$

but this formula does not insure that all the entries of the realization matrices are real.

7.1.11 Closing partially the upper loop

In many cases, it is necessary to assign some δ_i's to numerical values. Let us assume that the δ_i's belong to two distinct subsets: Δ_1 corresponds to the evaluated δ_i's and Δ_2 is the complementary subset. The original LFR-object $M(\Delta)$ is

$$M(\Delta_1,\Delta_2) = \mathcal{F}_u\left(\left[\begin{array}{cc|c} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right], \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix}\right)$$

it is equivalent to::

$$M(\Delta_1,\Delta_2) = \mathcal{F}_u\left(\left[\begin{array}{c|c} \mathcal{F}_u(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},\Delta_1) & \mathcal{F}_u(\begin{bmatrix} A_{11} & B_1 \\ A_{21} & B_2 \end{bmatrix},\Delta_1) \\ \hline \mathcal{F}_u(\begin{bmatrix} A_{11} & A_{12} \\ C_1 & C_2 \end{bmatrix},\Delta_1) & \mathcal{F}_u(\begin{bmatrix} A_{11} & B_1 \\ C_1 & D \end{bmatrix},\Delta_1) \end{array}\right], \Delta_2\right) \tag{7.16}$$

in which $\mathcal{F}_u(\bullet,\Delta_1)$ can be evaluated.

Proof. For computing the new LFT (Δ_1 taking numerical values), consider the input/output version of the upper LFT:

$$\begin{array}{ll} y_1 = A_{11}u_1 + A_{12}u_2 + B_1 u & \text{with } u_1 = \Delta_1 y_1 \\ y_2 = A_{21}u_1 + A_{22}u_2 + B_2 u & \\ y = C_1 u_1 + C_2 u_2 + Du & \end{array}$$

So

$$u_1 = \Delta_1 (I - A_{11}\Delta_1)^{-1}(A_{12}u_2 + B_1 u)$$

and

$$y_2 = (A_{21}\Delta_1(I-A_{11}\Delta_1)^{-1}A_{12}+A_{22})u_2 + (A_{21}\Delta_1(I-A_{11}\Delta_1)^{-1}B_1+B_2)u$$
$$y = (C_1\Delta_1(I-A_{11}\Delta_1)^{-1}A_{12}+C_2)u_2 + (C_2\Delta_1(I-A_{11}\Delta_1)^{-1}B_1+D)u$$

We recognize here an upper LFT:

$$M(\Delta_1,\Delta_2) = \mathcal{F}_u\left(\left[\begin{array}{c|c} A_{21}\Delta_1(I-A_{11}\Delta_1)^{-1}A_{12}+A_{22} & A_{21}\Delta_1(I-A_{11}\Delta_1)^{-1}B_1+B_2 \\ \hline C_1\Delta_1(I-A_{11}\Delta_1)^{-1}A_{12}+C_2 & C_1\Delta_1(I-A_{11}\Delta_1)^{-1}B_1+D \end{array}\right], \Delta_2\right)$$

■

7.1.12 DC-gain computation

In dynamic LFR-objects, $1/s$ is considered as a parameter (s Laplace variable). So, for computing the DC gain, the parameter $1/s$ must be replaced by infinity (Δ_1 is infinity times identity in (7.16)). Numerically, replacing infinity by large values is not very efficient. The alternative approach proposed below is numerically stable:

$$\lim_{s\to 0} \mathcal{F}_u\left(\left[\begin{array}{cc|c} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right], \left[\begin{array}{cc} I/s & 0 \\ 0 & \Delta \end{array}\right]\right) = \mathcal{F}_u\left(\left[\begin{array}{cc|c} A_{11}+I & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right], \left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\right) \tag{7.17}$$

Proof.

$$\mathcal{F}_u\left(\left[\begin{array}{cc|c} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right], \left[\begin{array}{cc} I/s & 0 \\ 0 & \Delta \end{array}\right]\right) =$$

$$\left[\begin{array}{cc} C_1 & C_2 \end{array}\right]\left[\begin{array}{cc} I/s & 0 \\ 0 & \Delta \end{array}\right]\left(I-\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]\left[\begin{array}{cc} I/s & 0 \\ 0 & \Delta \end{array}\right]\right)^{-1}\left[\begin{array}{c} B_1 \\ B_2 \end{array}\right]+D =$$

$$\left[\begin{array}{cc} C_1 & C_2 \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\left(\left[\begin{array}{cc} Is & 0 \\ 0 & I \end{array}\right]-\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\right)^{-1}\left[\begin{array}{c} B_1 \\ B_2 \end{array}\right]+D =$$

for $s \to 0$

$$\left[\begin{array}{cc} C_1 & C_2 \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\left(\left[\begin{array}{cc} 0 & 0 \\ 0 & I \end{array}\right]-\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\right)^{-1}\left[\begin{array}{c} B_1 \\ B_2 \end{array}\right]+D =$$

$$\left[\begin{array}{cc} C_1 & C_2 \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\left(\left[\begin{array}{cc} I & 0 \\ 0 & I \end{array}\right]-\left[\begin{array}{cc} A_{11}-I & A_{12} \\ A_{21} & A_{22} \end{array}\right]\left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\right)^{-1}\left[\begin{array}{c} B_1 \\ B_2 \end{array}\right]+D =$$

$$\mathcal{F}_u\left(\left[\begin{array}{cc|c} A_{11}+I & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right], \left[\begin{array}{cc} I & 0 \\ 0 & \Delta \end{array}\right]\right) = \text{DC-gain}$$

■

7.1.13 Upper LFT computation without duplication of Δ

In some cases, Δ must be considered as an LFR-object. For example assume that some δ_i must be replaced by an expression involving other parameters (say, δ_1 replaced by $\delta_2+\delta_3$, see the function `pluglfr` of LFRT version 1.x or the function `eval` of version 2.x). The most straightforward technique for finding the resulting LFR-object consists of computing explicitly

$$M_{21}\Delta(I-M_{11}\Delta)^{-1}M_{12}+M_{22}$$

in which all the entries of Δ are considered as 1×1 LFR-objects (note that the sub-matrices M_{ij} might also be LFR-objects). Unfortunately, Δ is repeated twice in this formula, therefore, the complexity of the result would be artificially multiplied by two.

In order to avoid this problem, it is suggested to use the following identity:

$$M_{21}\Delta(I-M_{11}\Delta)^{-1}M_{12}+M_{22}=\left[\begin{array}{cc} M_{21} & 0 \end{array}\right]\left(I+\left[\begin{array}{cc} -\Delta & 0 \\ 0 & I \end{array}\right]\left[\begin{array}{cc} M_{11} & I \\ 0 & 0 \end{array}\right]\right)^{-1}\left[\begin{array}{c} 0 \\ M_{12} \end{array}\right]+M_{22} \tag{7.18}$$

in which Δ appears only once.

Note that it is also possible to use Lemma 3.1.1, page 67 for avoiding duplication od Δ.

7.1.14 Differentiation

Let us consider a non-dynamic LFR-object f

$$f=F_{21}\Delta(I-F_{11}\Delta)^{-1}F_{12}+F_{22}$$

for some matrices $(F_{11},F_{12},F_{21},F_{22})$. The matrix Δ is as follows

$$\Delta=\mathrm{Diag}\{\delta_1 I_{n_1},\delta_2 I_{n_2},\ldots\}$$

Lemma 7.1.5

$$\left.\frac{\partial f}{\partial \delta_i}\right|_{\Delta}=F_{21}(I-\Delta F_{11})^{-1}H_i(I-F_{11}\Delta)^{-1}F_{12}$$

where

$$H_i=Diag\{0_{n_1\times n_1},\ldots,0_{(n_i-1)\times(n_i-1)},I_{n_i},0_{(n_i+1)\times(n_i+1)},\ldots,\}$$

Proof. Let $d\Delta_i$ denote the variation of Δ induced by the variation of δ_i.

$$df = F_{21}d\Delta_i(I-F_{11}\Delta)^{-1}F_{12} + F_{21}\Delta(I-F_{11}\Delta)^{-1}F_{11}d\Delta_i(I-F_{11}\Delta)^{-1}F_{12}$$

after factorization

$$df = F_{21}(I+\Delta(I-F_{11}\Delta)^{-1}F_{11})d\Delta_i(I-F_{11}\Delta)^{-1}F_{12}$$

the first matrix Δ can be commuted as follows

$$df = F_{21}(I+(I-\Delta F_{11})^{-1}\Delta F_{11})d\Delta_i(I-F_{11}\Delta)^{-1}F_{12}$$

so,

$$df = F_{21}(I-\Delta F_{11})^{-1}d\Delta_i(I-F_{11}\Delta)^{-1}F_{12}$$

but $d\Delta$ can be written as $d\Delta_i = H_i d\delta_i$, finally,

$$\left.\frac{\partial f}{\partial \delta_i}\right|_{\Delta} = F_{21}(I-\Delta F_{11})^{-1}H_i(I-F_{11}\Delta)^{-1}F_{12}$$

■

This computation can be performed using the MATLAB function `lfr/diff` (see Example 6.2, page 132).

7.2 Alternative proof of Lemma 4.4.1

Lemma 4.4.1 *Let us consider $M(\Delta)$ a (1×1) non-dynamic LFR-object with real (repeated) uncertainties. It is assumed that $M(\Delta)$ is well-posed (bounded) in the unit ball:*

$$M(\Delta) = \mathcal{F}_u(\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \Delta)$$

Let us define

$$S(\lambda) = M_{11} + M_{12}(\lambda - M_{22})^{-1} M_{21}$$

For λ varying from $-\infty$ to $+\infty$:

- *The minimum value of $M(\Delta)$ over the unit ball is the first value of λ at which $\mu(S(\lambda)) = 1$.*
- *The maximum value of $M(\Delta)$ over the unit ball is the last value of λ at which $\mu(S(\lambda)) = 1$.*

Proof. This result can be understood very easily by considering the artificial closed-loop system of Figure 7.1. The stability of this system changes when $\lambda = M(\Delta)$ for some admissible value of Δ (*i.e.* some combination of the uncertainties in the unit ball).

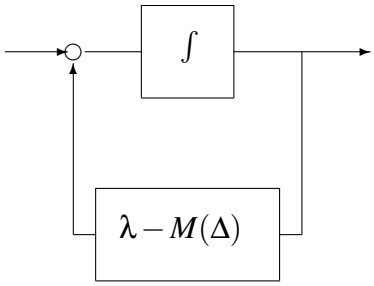

Figure 7.1: Artificial system for computing the min/max values of $M(\Delta)$

Discussion on stability. Let λ varies from $-\infty$ to $+\infty$. For large negative values of λ, there are no combinations of the parameters in the unit ball for which we can have $\lambda = M(\Delta)$ (boundedness hypothesis).

As λ increases, there exists a first value at which an admissible combination of parameters is such that $\lambda = M(\Delta)$ *i.e.* such that the system of Figure 7.1 is at the limit of stability. Therefore, we have justified that the minimum of $M(\Delta)$ is the first value of λ at which there exists a worst case (a combination of uncertain parameters) corresponding to the limit of stability of the system.

Similarly, the last value of λ for which there exists a worst case corresponding to the limit of stability is the maximum value of $M(\Delta)$ over the unit ball.

Translation in terms of μ-analysis. In order to terminate the proof of the lemma, it suffices to check that the μ analysis problem that is stated in the lemma corresponds to the above discussion relative to limits of stability. For that purpose, the system of Figure 7.1 is written in the equivalent form of Figure 7.2 (we added λ to the direct transmission $-M_{22}$ of $-M(\Delta)$, pulled out the Δ matrix and pulled down the integrator. Therefore, we obtain a $(M-\Delta)$-form in which "M" (denoted $S(s,\lambda)$) is

$$S(s,\lambda) = \mathcal{F}_l\left(\begin{bmatrix} M_{11} & -M_{12} \\ M_{21} & \lambda - M_{22} \end{bmatrix},\ 1/s\right)$$

or

$$S(s,\lambda) = M_{11} - M_{12}(s-\lambda+M_{22})^{-1}M_{21}$$

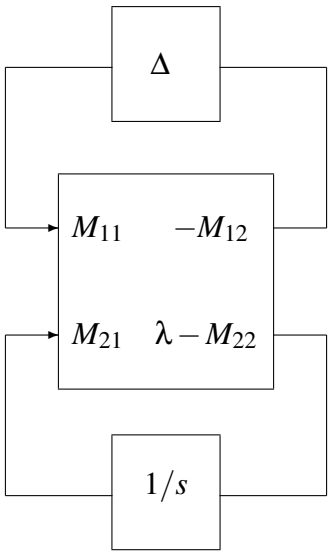

Figure 7.2: System equivalent to the one of Figure 7.1

Usually, stability analysis requires to compute μ over a gridding of frequencies. Here, we consider a (1×1) system with a single state. So, the limit of stability is necessarily reached when the single real system pole crosses the imaginary axis at the origin. Therefore, it suffices to compute μ at $\omega = 0$ that is, to compute

$$S(0,\lambda) = \mu(M_{11} + M_{12}(\lambda - M_{22})^{-1}M_{21})$$

as stated in the lemma. As uncertainties are assumed to be normalized, the limits of stability correspond to values of μ equal to the unity. ■

7.3 From version 1.x to version 2.0

Version 1.x and 2.0 are not compatible, so for using both versions simultaneously you need to open two instances of MATLAB with the `path` variables pointing separately to version 1 or 2.

There are two major improvements

- The variables are not ordered as in version 1 but are recognized by their names.
- It is no longer required to assign a non-zero nominal values before inverting a variable.

In addition to these major differences, many points have been improved for example:

- transformation from symbolic objects to LFR-objects,
- non-linear blocks are supported,
- parameter bounds of various kinds can be stored in the LFR-objects,
- compatibility with uncertain objects introduced in version 3 of the Robust Control Toolbox.

Note that all functions that have LFR-parameters as input arguments (*i.e.*, most functions) require a new syntax in version 2.0. Some functions have new names (see Table 7.1).

In version 1.x, the reserved name for $1/s$ (Laplace) was `lfrS`, in version 2.0 it is `Int` for $1/s$ and `Delay` for $1/z$. They are not in the workspace by default, they can be defined by invoking `lfrs Int` or `lfrs Delay`.

Version 2.0 focuses on modelling: tools for feedback design/analysis, for Simulink interfaces and possibly other purposes will be proposed as separate modules.

Version 1.x	Version 2.0
convert2lfr	lfr
difflfr	diff
depl2lfr deps2lfr	gmorton
flup	uplft
lfr2ss	lfr2rob
lfr2mua	lfr2mu lfr2mussv
lfr2lmip	lfr2mustab
lfr2lmim	lfr2mubnd lfr2mussv
normlfr	normalizelfr
ndv2lfr	rf2lfr
ndh2lfr	lf2lfr
pluglfr	eval
ss2lfr	lfr

Table 7.1: Names of functions from version 1.x to version 2.0

7.4 List of MATLAB-functions

The third column of the tables below gives a reference to a script illustrating the use of the corresponding function. For most functions, illustrative examples are also available by typing `help <name of the function>` (then it suffices to "copy and paste" the command lines displayed to the screen.

Functions in `@lfr`
The functions of the directory '`@lfr`' are `append`, `blkdiag`, `ctranspose`, `dcgain`, `diff`, `display`, `eig`, `eval`, `feedback`, `get`, `horzcat`, `inv`, `isempty`, `minus`, `mpower`, `mrdivide`, `mtimes`, `plus`, `set`, `size`, `subsasgn`, `subsref`, `transp`, `uminus`, `vertcat`. These functions permit the user to define and manipulate LFR-objects like matrices.

Elementary LFR-object realization		
`lfr`	Core of the Toolbox for LFR-object generation	Ex.2.1, p.22
`lfrdata`	Data recovery in an LFR-object	Ex.2.1, p.22
`lfrs`	Real or complex 1-by-1 LFR-objects	Ex.2.11, p.52
`rlfr`	Generates random LFR-objects	Ex.2.4, p.31
`bnds2lfr`	From min/max bounds to LFR-objects	Ex.3.3, p.74
`gmorton`	Generalized Morton's technique	Ex.3.4, p.78

LFR-object realization from symbolic objects		
`sym2lfr`	Symbolic expressions converted to LFRs	Ex.2.13, p.54
`symtreed`	Elementary structured tree decomposition	Ex.6.3, p.137

Miscellaneous conversions		
`lfr`	SS, PCK, USS, UMAT-objects $\rightarrow$ LFR-objects	
`lfr2rob`	LFR-objects $\rightarrow$ USS, UMAT-objects	
`abcd2lfr`	From A, B, C, D to input/output LFR	Ex.3.2, p.73
`lfr2abcd`	Converse	`help lfr2abcd`
`lf2lfr`	$[N\ D] \rightarrow D^{-1}N$	Ex.3.1, p.69
`rf2lfr`	$[N;D] \rightarrow ND^{-1}$	Ex.3.1, p.69

Operations on the Δ-block		
`normalizelfr`	Normalizes real uncertainty variations	Ex.4.4, p.111
`unnormalize`	Converse	Ex.2.9, p.42
`actualval`	Actual parmeter values from normalized values	Ex.2.9, p.42
`uplft`	Closes (partially) the Δ-loop	Ex.2.8, p.41
`eval`	Elaborated "star product" and much more	Ex.6.4, p.138
`starplfr`	Basic "star product"	`help starplfr`

LFR order reduction after realization		
`minlfr1`	Sequential 1-D LFR order reduction	Ex.4.1, p.96
`minlfr`	n-D Kalman like decomposition	Ex.4.3, p.102

LFR approximation		
`reduclfr`	Handling of the tolerance argument of `minlfr`	Ex.4.5, p.112
`distlfr`	Distance between two LFRs (lower bound)	Ex.2.12, p.53
`udistlfr`	Upper bound of the distance	`help udistlfr`

μ-analysis related functions		
`lfr2mubnd`	Generates the input arguments of `mubnd`	`help lfr2mubnd`
`lfr2mustab`	Generates the input arguments of `mustab`	`help lfr2mustab`
`lfr2mussv`	Generates the input arguments of `mussv`	`help lfr2mussv`
`lfr2mu`	Generates the input arguments of `mu`	`help lfr2mu`
`min_max`	Min/max values of real 1-by-1 LFR-object	Ex.4.4, p.111
`udistlfr`	Upper bound of the distance	`help udistlfr`
`ns_rad`	Non-singularity radius	Ex.2.10, p.48
`wp_rad`	Well-posedness radius	`help wp_rad`

Miscellaneous tools		
`diff`	Overloaded function for differentiation	Ex.6.2, p.132
`feedback`	Overloaded function for feedback	Ex.3.2, p.73
`distlfr`	Lower bound of the distance	Ex.2.12, p.53
`size`	Overloaded function for displaying size	Ex.2.1, p.22
`data2lfr`	Interpolation of matrices on an LFR basis	Ex.6.5, p.142
`plotlfr`	Plots entries of an LFR-object	Ex.6.5, p.142

7.5 Description of uncertainty blocks

This section explains the meaning of the entries of the structured array that is denoted `blk` in this manual. For a given LFR-object, say `sys`, `blk` can be obtained by typing `sys.blk`.

The structured array `sys.blk` contains

- a cell `sys.blk.names` which contains the names of the uncertain parameters.
- a matrix `sys.blk.desc` which contains information on uncertainty bloks (size, nature, bounds).

7.5.1 Names of variables

There are three reserved variable names

- `ConstBlock`: In principle the user doesn't need to manipulate this variable. This variable might appear in `sys.blk.names` as `'1'`. This variable is present in `blk` when a non-feasible inversion has been performed. When an LFR-object is normalized or evaluated this block should disappear automatically.
- `Int`: This variable stand for "integrator" it is represented in `sys.blk.names` for dynamic LFR-objects as `'1/s'`.
- `Delay`: This variable is represented in `sys.blk.names` for discrete time dynamic LFR-objects as `'1/z'`. `Int` and `Delay` cannot be combined in an LFR-object.

Example

```
   lfrs a b Int
   sys = (b+Int)/a;
   sys.blk.names

ans =

    '1'    '1/s'    'a'    'b'
```

Note that '1' ("constant block") appears in `sys.blk.names` because there is a division by the parameter `a` with default nominal value equal to zero. Similar example in discrete time:

```
   lfrs a b Delay
   sys = (b+Delay)/a;
   sys.blk.names

ans =

    '1'    '1/z'    'a'    'b'
```

7.5.2 Block description

Rows 1 to 6. Each column of `sys.blk.desc` describes the block structure relative to the corresponding variable in `sys.blk.names`.

m	block row dimension
n	block column dimension
$1/0$	1 for real, 0 for complex
$1/0$	1 for scalar repeated, 0 for full
$1/0$	1 for linear, 0 for nonlinear
$1/0$	1 for time-invariant, 0 for time-varying

Rows 7 and 8 can be used to identify the kind of bounds considered (min/max, disc, sector or frequency domain bound): In the frequency domain case, the argument defining the bound is an

	min/max	disc	sector	frequency	1/s,1/z,1
7th row	1	1	2	size(pck-matrix,1)	0
8th row	2	1	1	size(pck-matrix,2)	0

Table 7.2:

LTI system in pck format (MATLAB, not ported to SCILAB) therefore the sizes are both larger or equal to 2.

Other rows

- **min/max case**:

Row	description
9	minimum value of variation range (set to `-1` by `normalizelfr`)
10	maximum value of variation range (set to `+1` by `normalizelfr`)
11	nominal value (set to `0` by `normalizelfr`)
12	`normalizelfr` $\hookrightarrow$ the above minimum value is saved here
13	`normalizelfr` $\hookrightarrow$ the above maximum value is saved here
14	`normalizelfr` $\hookrightarrow$ the above nominal value is saved here

- **disc case**:

Row	description
9	real part of disc center (set to `0` by `normalizelfr`)
10	imaginary part of disc center (set to `0` by `normalizelfr`)
11	radius of disc (set to `1` by `normalizelfr`)
12	`normalizelfr` $\hookrightarrow$ the above real part is saved here
13	`normalizelfr` $\hookrightarrow$ the above imaginary part is saved here
14	`normalizelfr` $\hookrightarrow$ the above radius is saved here

- **frequency domain bound case**: the argument `BOUND` defining the bounds is something having the form

```
BOUND = [
A B 0 ;
C D 0 ;
0 0 -Inf ]
```

  the rest of the column after row number 8 is `BOUND(:)` (*i.e.*, all columns of `BOUND` appended in a long column). It is possible to recover the dynamic weighting function by using `reshape`, for example `reshape(sys.blk.desc(9:end,3),3,3)`.

- **sector bounded case**: Rows 9 and 10 give the minimum and maximum slopes defining the sector.

Example

```
lfrs a real [1] [4] [3.3]
lfrs b complex [-1+i] [2.2]
c = lfr('c','ltifc',[2 3],ltisys(-1,2,3,4));
```

```
   sys = [[a;b] c];
   sys.blk.desc

ans =

     1     1     2      <- rows
     1     1     3      <- columns
     1     0     0      <- real(1)/complex(0)
     1     1     0      <- scalar(1)/full(0)
     1     1     1      <- linear
     1     1     1      <- time invariant
     1     1     3      <- bounds type
     2     1     3          .
     1    -1    -1      <- bounds (and nominal values in scalar case)
     4     1     1          .
   3.3   2.2     0          .
     0     0     1          .
     0     0     0          .
     0     0     0          .
     0     0     1          .
     0     0     0          .
     0     0  -Inf          .
```

Remark. The objects compared by the function `distlfr` must have similar block descriptions for a common parameter name. In the min/max and disc cases, the values after the 10th row are ignored by this function.

7.6 Installation of the toolbox

Read first the license agreement in the file `LICENSE` in the toolbox root directory. Installation instructions are detailed in the `INSTALL` or `README` files.

System requirements MATLAB:

- MATLAB 5, 6 and the Control Toolbox (the μ-Analysis and Control Toolbox and the Robust Control Toolbox are also required by a few functions)
- MATLAB 7 and the Control Toolbox (the Robust Control Toolbox version 3 is also required by a few functions)
- It is also highly recommended to have the Symbolic Toolbox.

System requirements SCILAB: Version 3.1 or higher.

7.7 SCILAB specificities

Despite a very different approach to object definition in MATLAB and SCILAB, from the user point of view, object-oriented modelling is very similar in both versions.

However the objectives are not similar in both cases.

- In the MATLAB case we offer several features that are not available in the Robust Control Toolbox version 3. For example, using our toolbox, it is possible to treat and transform symbolic objects to LFR-objects, to divide by objects with zero-nominal value, intergration to Simulink is proposed using an independent module, and so on. In addition, it is possible to convert uncertain objetcs from one toolbox to the other one.
- In the SCILAB case, no symbolic objects are available (see below), our contribution consists of a port to SCILAB of the newly defined MATLAB objects like `ureal`, `ucomplex`, `umat`, `uss`. These objects are defined as special cases of our LFR-object. Tools for using efficiently LFR-objects are still missing but we hope that our contribution will help to port to SCILAB all these missing tools.

Symbolic objects. As symbolic objects are not available in SCILAB we propose to use first MAPLE, then, export a symbolic object in a file using the MAPLE script `maple2scilab.mpl`[1]. Then, the functions `symtreed` and `sym2lfr` can be invoked but the input argument is the name of the file where a MAPLE symbolic object was stored.

Functions not available in the SCILAB version.

- Functions related to μ-analysis: `lfr2mubnd`, `lfr2mustab`, `lfr2mussv`, `lfr2mu`, `min_max`, `udistlfr`, `ns_rad`, `wp_rad`.
- Irrelevant functions: `lfr2rob`, `minlfr1` (`minlfr` is always more efficient).
- Functions not ported on account of a SCILAB bug (feature ?) that prevents overloading: `size` (renamed `sizelfr`), `eval` (renamed `evallfr`), `diff` (renamed `difflfr`), `isempty`.

Illustrative examples that cannot be run. Ex. 2.10, page 48 (well-posedness, uses μ-analysis), Ex. 2.13, page 54 (realization from a symbolic object, requires a first step using MAPLE, see `hlp sym2lfr`), Ex. 3.5, page 81 (Horner factorization), Ex. 3.6, page 86 (structured tree decomposition, requires a first step using MAPLE, see `hlp symtreed`), Ex. 4.1, page 96 and

[1] It is in the subdirectory named `maple` of the SCILAB installation directory. Type SCI in a SCILAB command window to retrieve the path to the installation directoty.

Ex. 4.2, page 97 (use of `minlfr1`), Ex. 4.4, page 111 and Ex. 4.5, page 112 (approximate modelling, uses μ-analysis), Ex. 6.1, page 130 and Ex. 6.4, page 137 (requires a first step using MAPLE, treated in the last section of the HTML tuturial shipped with the toolbox).

Syntax adaptation. Cells are not easy to use in SCILAB, so, they are replaced by lists or arrays of strings (*e.g.*, input arguments of the functions `uplft`, `lfr`, `normalizelfr`). The functions `size` is replaced by `sizelfr`.

Functions specific to the SCILAB version. MATLAB function describing systems that have been ported to SCILAB: `ss`, `ureal`, `ucomplex`.

EMPTY PAGE

Bibliography

[1] L. Andersson and C. Beck.
Model comparison and simplification.
Proc. of 35th IEEE Conference on Decision and Control, Kobe, Japan, pages 3958–3963, December 1996.

[2] L. Andersson, A. Rantzer, and C. Beck.
Model comparison and simplification.
International Journal of Robust and Nonlinear Control, 9(4):157–181, March 1999.

[3] B.R. Barmish, J. Ackermann, and H.Z. Hu.
The tree structured decomposition.
In Proc. Conference on Information Sciences and Systems, Baltimore, MD, 1989.

[4] D.G. Bates, R. Kureemun, M.J. Hayes, and I. Postlethwaite.
Clearance of the HIRMplus RIDE flight control law: A μ-analysis approach.
Technical Report TP-119-11, Group for Aeronautical Research and technology in EURope GARTEUR-FM(AG11), 2000.

[5] C. Beck.
Minimality for uncertain systems and IQCs.
In *Proc. of 33th IEEE Conference on Decision and Control, Lake Buena Vista, Florida*, pages 1233–1238, December 1994.

[6] C. Beck and R. D'Andrea.
Minimality, controllability and observability for uncertain systems.
In *Proc. of the American Control Conference, Philadelphia, Pennsylvania*, pages 3130–3135, June 1997.

[7] C. Beck and R. D'Andrea.
Computational study and comparison of LFT reducibility methods.
In *Proc. of the American Control Conference, Philadelphia, Pennsylvania*, pages 1013–1017, June 1998.

[8] C. Beck, R. D'Andrea, F. Paganini, W.M. Lu, and J.C. Doyle.
A state-space theory of uncertain systems.

In *Proc. of the 13th IFAC Triennial World Congress, San Francisco, California*, pages 291–296. Pergamon, July 1996.

[9] C. Beck and J. Doyle.
A necessary and sufficient minimality condition for uncertain systems.
IEEE Transactions on Automatic Control, AC-44:1802–1813, october 1999.

[10] C. Beck and J.C. Doyle.
Mixed μ upper bound computation.
In *Proc. 31st IEEE Conference on Decision and Control*, pages 3187–3192, Tucson, Arizona, USA, December 1992.

[11] C. Beck and J.C. Doyle.
Reducing uncertain systems and behaviors.
In *Proc. of 35th IEEE Conference on Decision and Control, Kobe, Japan*, pages 712–714, December 1996.

[12] C.M. Belcastro.
Uncertainty modeling od real parameter variations for robust control applications.
In *PhD Thesis, University of Drexel, US*, pages 1–304, December 1994.

[13] C.M. Belcastro.
On the numerical formulation of parametric linear fractional transformation (LFT) uncertainty models for multivariate matrix polynomial problems.
In *Nasa / TM-1998-206939*, pages 1–35, November 1998.

[14] C.M. Belcastro.
Parametric uncertainty modeling: An overview.
In *Proc. of the American Control Conference, Philadelphia, Pennsylvania*, pages 992–996, June 1998.

[15] C.M. Belcastro and B.C. Chang.
LFT formulation for multivariate polynomial problems.
In *Proc. of the American Control Conference, Philadelphia, Pennsylvania*, pages 1002–1007, June 1998.

[16] C.M. Belcastro, B.C. Chang, and R. Fischl.
A matrix approach to low-order uncertainty modeling of real parameters.
In *Proc. of the 13th IFAC Triennial World Congress, San Francisco, California*, pages 297–302. Pergamon, July 1996.

[17] C.M. Belcastro, K.B. Lim, and E.A. Morelli.
Computer-aided uncertainty modeling of nonlinear parameter-dependent systems, part ii: F-16 example.
In *Proc. of the IEEE International Symposium on Computed Aided Control System Design, Hawai'i, USA*, pages 16–23, August 1999.

[18] N.K. Bose.
Applied Multidimensional Systems Theory.
Electrical/Computer Science and Engineering Series. Van Nostrand Reinhold Company, New-York Cincinnati Toronto London Melbourne, 1982.

[19] Chi-Tsong Chen.
Linear system theory and design.
Holt, Rinehart and Winston, New York, USA, 1984.

[20] Y. Cheng and B. De Moor.
A multidimensional realization algorithm for parametric uncertainty modeling and multi-parameter margin problems.
Int. J. Control, 60(5):789–807, 1994.

[21] J.C. Cockburn.
Linear fractional representation of systems with rational uncertainty.
In *Proc. of the American Control Conference, Philadelphia, Pennsylvania*, pages 1008–1012, June 1998.

[22] J.C. Cockburn and B.G. Morton.
On linear fractional representations of systems with parametric uncertainty.
In *Proc. of the 13th IFAC Triennial World Congress, San Francisco, California*, pages 315–320. Pergamon, July 1996.

[23] J.C. Cockburn and B.G. Morton.
Linear fractional representations of uncertain systems.
Automatica, 33(7):1263–1271, 1997.

[24] R. D'Andrea.
Software for modeling, analysis, and control design for multidimensional systems.
In *Proc. of the IEEE International Symposium on Computed Aided Control System Design, Hawai'i, USA*, pages 24–27, August 1999.

[25] R. D'Andrea and S. Khatri.
Kalman decomposition of linear fractional transformation representations and minimality.
In *Proc. of the American Control Conference, Albulquerque, New Mexico*, pages 3557–3561, June 1997.

[26] C. Döll.
La robustesse de lois de commande pour des structures flexibles en aéronautique et espace.
Thèse présentée à l'Ecole Nationale Supérieure de l'Aéronautique et de l'Espace (SUPAERO), Toulouse, France, 2001.

[27] J.C. Doyle, A. Packard, and K. Zhou.
Review of LFTs LMIs and μ.

In *Proc. of 30th IEEE Conference on Decision and Control, Brighton, U.K.*, pages 1227–1232, December 1991.

[28] J.C. Doyle, F. Paganini, R. D'Andrea, and S. Khatri.
Approximate behaviors.
In *Proc. of 35th IEEE Conference on Decision and Control, Kobe, Japan*, pages 688–693, December 1996.

[29] C. Fielding.
Application of μ-analysis to flight control systems.
Minutes of the First Meeting of GARTEUR FM(AG11), Appendix D.1(FM(AG11)/SR-1), May 1999.

[30] S. Font.
Méthodologie pour prendre en compte la robustesse des système asservis: Optimisation H_∞ et approche symbolique de la forme standard.
Thèse présentée à l'Université de Paris-Sud, Orsay, France, 1995.

[31] S. Hecker and A. Varga.
Generalized LFT-based representation of parametric uncertain models.
European Journal of Control, 10(4):326–337, 2004.

[32] S. Hecker, A. Varga, and J.F. Magni.
Enhanced LFR-Toolbox for Matlab.
Aerospace Science and Technology, 9:173–180, January 2005.

[33] A. Hiret.
Pilotage robuste d'un missile sur un large domaine de vol. Synthèse et analyse dans le cadre H_∞ et LPV.
Thèse présentée à l'Université de Paris-Sud, Orsay, France., 5879, Octobre 1999.

[34] A. Hiret, C. Valentin-Charbonnel, G. Duc, and J.P. Bonnet.
Different multidimensional reduction algorithms for the LFT model of a missile.
In Proc. CESA'98 Conference, Nabeul-Hammamet, Tunisia, 1:1016–1020, April 1998.

[35] I. Kaminer, A. Pascoal, P. Khargonekar, and E. Coleman.
A velocity algorithm for the implementation of gain-scheduled controllers.
Automatica, 31(8):1185–1191, 1995.

[36] P. Lambrechts, J. Terlouw, S. Bennani, and M. Steinbuch.
Parametric uncertainty modeling using LFTs.
In Proc. American Control Conference, San Francisco, CA, pages 267–272, 1993.

[37] D.J. Leith and W.E. Leithead.
Gain-scheduled and nonlinera systems: dynami analysis by velocity-based realisation families.

Int. J. Control, 70:289–317, 1998.

[38] D.J. Leith and W.E. Leithead.
Input-output linearizayion by velocity-based gain-scheduling.
Int. J. Control, 72:229–246, 1999.

[39] J.F. Magni.
Presentation of the linear fractionnal representation toolbox (LFRT).
In Proc. IEEE CACSD, Glasgow, September 2002.

[40] J.F. Magni.
Computation of the kernel of linear fractional representations.
In Proc. of the IEEE Conference on Control Applications, Istanbul, Turkey, June 2003.

[41] JF Magni.
Extension of the Linear Fractional Representation Toolbox (LFRT).
In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design, Taipei, Taiwa*, pages 261–266, September 2004.

[42] J.F. Magni, J.M. Biannic, and C. Döll.
An algorithm for computing an upper bound of the peak value of μ.
In Proc. IFAC Symposium on System Structure and Control, Prague, Czech Republic, August 2001.

[43] T. Mannchen, Y. Klett, C. Petermann, B. Weinert, and T Zöbelein.
Flight control law clearance of the HIRMplus fighter aircraft model using μ-analysis.
Technical Report TP-119-12, Group for Aeronautical Research and technology in EURope GARTEUR-FM(AG11), 2001.

[44] A. Marcos.
A linear parameter varying model of the Boeing 747-100/200, longitudinal motion.
Master's thesis, University of Minnesota, Minneapolis, USA, January 2001.

[45] A. Marcos and G.J. Balas.
Development of linear-parameter-varying models for aircraft.
Journal of Guidance Control, and Dynamics, 27(2):218–228, March-April 2004.

[46] B. Morton.
New applications of mu to real-parameter variation problems.
In *Proc. of 24th IEEE Conference on Decision and Control, Fort Lauderdale, Florida*, pages 233–238, December 1985.

[47] G. Papageorgiou.
Robust control system design: $\mathcal{H}_\infty$ loop shaping and aerospace applications.
PhD thesis, Darwin College, Cambrige, UK, July 1998.

[48] R.T. Reichert.

Robust auropilot design using μ-analysis.
In Proc. American Control Conference, pages 2368–2373, May 1990.

[49] W.J. Rugh and J.S. Shamma.
Research on gain scheduling.
Automatica, 36(10):1401–1425, 2000.

[50] J.S. Shamma and J. Cloutier.
Gain-scheduling missile autopilot design using linear parameter varying transformations.
Journal of Guidance, Control, and Dynamics, 16(2):256–261, 1993.

[51] J.C. Terlouw and P.F. Lambrechts.
A MATLAB Toolbox for Parametric Uncertainty Modelling.
Technical Report CR-93455-L, National Aerospace Laboratory, NLR, Amsterdam, 1993.

[52] A. Varga and G. Looye.
Symbolic and numerical software tools for LFT-based low order uncertainty modeling.
In *Proc. of the IEEE International Symposium on Computed Aided Control System Design, Kohala Coast, Hawai'i, USA*, pages 176–181, August 1999.

[53] A. Varga, G. Looye, D. Moormann, and Grübel G.
Automated generation of LFT-based parametric uncertainty descriptions from generic aircraft models.
Mathematical Modelling of Dynamical Systems, 4:249–274, 1998.

[54] W. Wang, J. Doyle, C. Beck, and K. Glover.
Model reduction of LFT systems.
In *Proc. of 30th IEEE Conference on Decision and Control, Brighton, England*, pages 1233–1238, December 1991.

Index

T

U

V

W

EMPTY PAGE