LFT Solver 4.8

A MATLAB Toolbox for Parameter Identification of Nonlinear LFT models

User Manual

Alessandro Della Bona
Gianni Ferretti

Preface

This user manual refers to the LFT Identification Toolbox originally created for academic usage by Alessandro Della Bona, and developed over the year by himself and others, in particular Alessandro Ros. The version of the tool here considered is the **LFT Solver 4.8**. The Toolbox can be simply installed by adding the relevant directory to the MATLAB path.

The manual is built around a practical example of parameter identification of a simple non-linear model, which will be presented in the next sections. This will give the opportunity to present and explain all the functions of the toolbox relevant to the identification procedure.

Contents

Chapter 1

The LFT Approach to Parameter Identification of Nonlinear Systems

The problem of parameter identification formulated over Linear Fractional Transform (LFT) model structures has been a subject of active research for more than 10 years, see, e.g., [5, 2, 4]. In particular, the parameter estimation method proposed in [4] is here extended to account for nonlinear models.

Consider a nonlinear, time invariant, multi-input multi-output, continuous-time system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\delta}^o) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\delta}^o) \end{aligned} \tag{1.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^p$, are the state and noise-free output vectors, with $\mathbf{u} \in \mathbb{R}^m$ being the input vector, and $\boldsymbol{\delta}^o \in \mathbb{R}^q$ a vector of unknown parameters, and assume as the output observation equation

$$\check{\mathbf{y}}(t_k) = \mathbf{y}(t_k) + \boldsymbol{\varepsilon}(t_k) \tag{1.2}$$

where t_k, $k = 1, \ldots, N$ denotes the sampling instant, and $\varepsilon_i(t_k)$ is a discrete-time, zero-mean, white noise of variance σ_i^2.

The identification problem can be formulated as follows: given the sampled data $\{\mathbf{u}(t_k), \check{\mathbf{y}}(t_k)\}_{k=1}^N$, find the values of parameters $\tilde{\boldsymbol{\delta}}$ minimizing the cost function

$$J(\boldsymbol{\delta}) = \frac{1}{2N} \sum_{k=1}^{N} \mathbf{e}^T(t_k, \boldsymbol{\delta}) \mathbf{W} \mathbf{e}(t_k, \boldsymbol{\delta}) \tag{1.3}$$

where $\mathbf{e}(t_k, \boldsymbol{\delta}) = \check{\mathbf{y}}(t_k) - \hat{\mathbf{y}}(t_k, \boldsymbol{\delta})$ is the prediction error between the measured output $\check{\mathbf{y}}(t_k)$ and the output $\hat{\mathbf{y}}(t_k, \boldsymbol{\delta})$ predicted by model (1.1), using parameters $\boldsymbol{\delta}$ instead of the true parameters $\boldsymbol{\delta}^o$ and $\mathbf{W}$ is a weight matrix.

As it is well known, $\tilde{\boldsymbol{\delta}}$ is a *maximum-likelihood* estimate of the model parameters $\boldsymbol{\delta}$ for output-error plants [6], and can be obtained through well known iterative optimization procedures such as, for example, the Gauss-Newton algorithm:

$$\hat{\boldsymbol{\delta}}(\nu+1) = \hat{\boldsymbol{\delta}}(\nu) - \alpha(\nu)\hat{\mathbf{H}}^{-1}(\hat{\boldsymbol{\delta}}(\nu))\mathbf{g}(\hat{\boldsymbol{\delta}}(\nu)) \tag{1.4}$$

where ν is the iteration number, $\alpha(\nu)$ is the step size, $\mathbf{g}(\boldsymbol{\delta}) : \mathbb{R}^q \to \mathbb{R}^q$ and $\hat{\mathbf{H}}(\boldsymbol{\delta}) : \mathbb{R}^q \to \mathbb{R}^{q\times q}$ are the gradient vector and a positive semi-definite approximation of the Hessian of the cost function with respect to the unknown parameters, respectively:

$$\mathbf{g}(\boldsymbol{\delta}) = \frac{1}{N}\sum_{k=1}^{N}\mathbf{E}^T(t_k,\boldsymbol{\delta})\mathbf{W}\mathbf{e}(t_k,\boldsymbol{\delta}), \tag{1.5}$$

$$\hat{\mathbf{H}}(\boldsymbol{\delta}) = \frac{1}{N}\sum_{k=1}^{N}\mathbf{E}^T(t_k,\boldsymbol{\delta})\mathbf{W}\mathbf{E}(t_k,\boldsymbol{\delta}) \tag{1.6}$$

where $\mathbf{E}(t_k,\boldsymbol{\delta}) \in \mathbb{R}^{p\times q}$ is the Jacobian of $\mathbf{e}(t_k,\boldsymbol{\delta})$ and is given by:

$$\mathbf{E}(t_k,\boldsymbol{\delta}) = \begin{bmatrix} \frac{\partial \mathbf{e}(t_k,\boldsymbol{\delta})}{\partial \delta_1} & \dots & \frac{\partial \mathbf{e}(t_k,\boldsymbol{\delta})}{\partial \delta_q} \end{bmatrix} \tag{1.7}$$

In turn, rewriting model (1.1) in a Linear Fractional Transform (LFT) formulation allows for a direct computation by simulation of the gradient and approximated Hessian of the cost function [1]:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_1\mathbf{w}(t) + \mathbf{B}_2\boldsymbol{\zeta}(t) + \mathbf{B}_3\mathbf{u}(t) \tag{1.8}$$

$$\mathbf{z}(t) = \mathbf{C}_1\mathbf{x}(t) + \mathbf{D}_{11}\mathbf{w}(t) + \mathbf{D}_{12}\boldsymbol{\zeta}(t) + \mathbf{D}_{13}\mathbf{u}(t) \tag{1.9}$$

$$\boldsymbol{\omega}(t) = \mathbf{C}_2\mathbf{x}(t) + \mathbf{D}_{21}\mathbf{w}(t) + \mathbf{D}_{22}\boldsymbol{\zeta}(t) + \mathbf{D}_{23}\mathbf{u}(t) \tag{1.10}$$

$$\mathbf{y}(t) = \mathbf{C}_3\mathbf{x}(t) + \mathbf{D}_{31}\mathbf{w}(t) + \mathbf{D}_{32}\boldsymbol{\zeta}(t) + \mathbf{D}_{33}\mathbf{u}(t) \tag{1.11}$$

$$\mathbf{w}(t) = \boldsymbol{\Delta}\mathbf{z}(t) = \mathrm{diag}\{\delta_1^o\mathbf{I}_{r_1},\dots,\delta_q^o\mathbf{I}_{r_q}\}\mathbf{z}(t) \tag{1.12}$$

$$\boldsymbol{\zeta}(t) = \boldsymbol{\Theta}(\boldsymbol{\omega}(t)) \tag{1.13}$$

where $\mathbf{z} \in \mathbb{R}^{n_z}$, $\boldsymbol{\omega} \in \mathbb{R}^{n_\omega}$, $\mathbf{w} \in \mathbb{R}^{n_w}$, $\boldsymbol{\zeta} \in \mathbb{R}^{n_\zeta}$ are vectors of auxiliary variables, $\mathbf{A}$, $\mathbf{B}_i$, $\mathbf{C}_i$, $\mathbf{D}_{ij}$ are 16 known constant matrices, r_i are the sizes of the corresponding identity matrices $\mathbf{I}_{r_i}$ in the $\boldsymbol{\Delta}$ block and $\boldsymbol{\Theta}(\boldsymbol{\omega}) : \mathbb{R}^{n_\omega} \to \mathbb{R}^{n_\zeta}$ is a known nonlinear vector function.

The model in the LFT formulation, although being formally equivalent to the original one, is now clearly divided in 3 parts (Fig. 1.1):

1. a *linear* part: equations (1.8 – 1.11);
2. a *nonlinear* part: equation (1.13), defined by the vector function $\boldsymbol{\Theta}(\boldsymbol{\omega}(t))$;
3. an *uncertain* part: equation (1.12), where vector $\mathbf{z}(t)$ multiplies matrix $\boldsymbol{\Delta} = \mathrm{diag}\{\delta_1^o\mathbf{I}_{r_1},\dots,\delta_q^o\mathbf{I}_{r_q}\}$, collecting the unknown parameters.

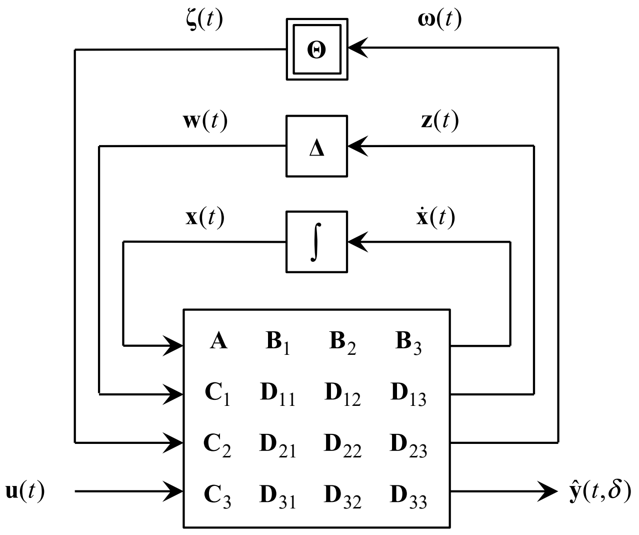
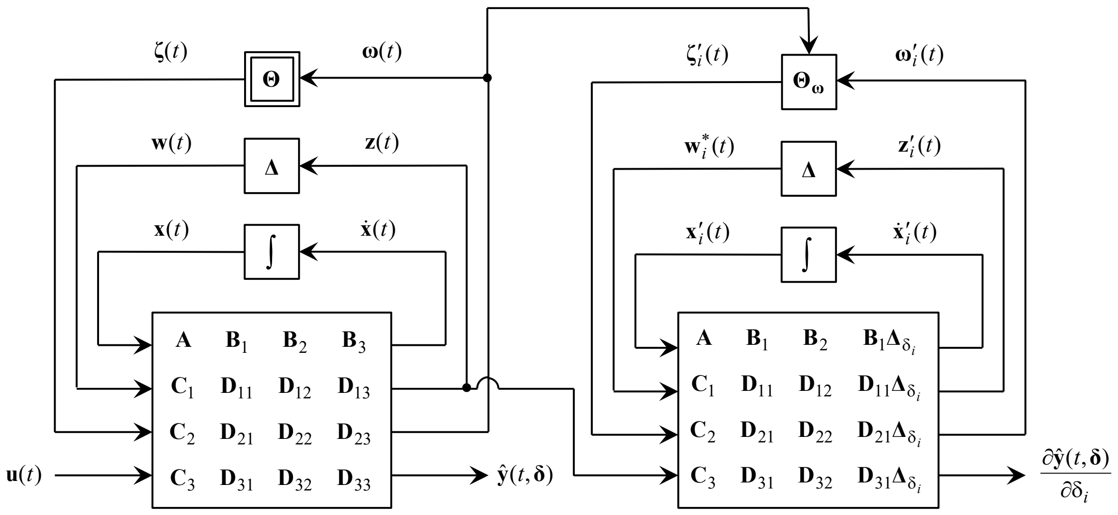


Figure 1.1: LFT formulation



$\zeta(t)$
$\omega(t)$
Θ
$w(t)$
$z(t)$
Δ
$x(t)$
$\dot{x}(t)$
$\int$
A B_1 B_2 B_3
C_1 D_{11} D_{12} D_{13}
C_2 D_{21} D_{22} D_{23}
C_3 D_{31} D_{32} D_{33}
$u(t)$
$\hat{y}(t,\delta)$
$\zeta_i'(t)$
$\omega_i'(t)$
Θ_ω
$w_i^*(t)$
$z_i'(t)$
Δ
$x_i'(t)$
$\dot{x}_i'(t)$
$\int$
A B_1 B_2 $B_1\Delta_{\delta_i}$
C_1 D_{11} D_{12} $D_{11}\Delta_{\delta_i}$
C_2 D_{21} D_{22} $D_{21}\Delta_{\delta_i}$
C_3 D_{31} D_{32} $D_{31}\Delta_{\delta_i}$
$\frac{\partial \hat{y}(t,\delta)}{\partial \delta_i}$

Figure 1.2: Simulation scheme for the computation of the sensitivity functions

The computation of the predicted output $\hat{\mathbf{y}}(t_k,\boldsymbol{\delta})$ (first stage of the scheme in Fig. 1.2) can be dealt with by rewriting model (1.8–1.13) as follows:

$$\begin{aligned}
\mathbf{M}\dot{\tilde{\mathbf{x}}}(t) &= \tilde{\mathbf{f}}(\tilde{\mathbf{x}}(t),\mathbf{u}(t,\boldsymbol{\tau}),\boldsymbol{\delta}) && (1.14)\\
\mathbf{y}(t) &= \tilde{\mathbf{g}}(\tilde{\mathbf{x}}(t),\mathbf{u}(t,\boldsymbol{\tau}),\boldsymbol{\delta}) && (1.15)
\end{aligned}$$

where $\tilde{\mathbf{x}}(t)=\left[\begin{array}{ccc}\mathbf{x}(t)^T & \mathbf{z}(t)^T & \boldsymbol{\omega}(t)^T\end{array}\right]^T$ and

$$\begin{aligned}
\mathbf{M} &= \left[\begin{array}{ccc}\mathbf{I}_n & \mathbf{0}_{n\times n_z} & \mathbf{0}_{n\times n_\omega}\\ \mathbf{0}_{n_z\times n} & \mathbf{0}_{n_z\times n_z} & \mathbf{0}_{n_z\times n_\omega}\\ \mathbf{0}_{n_\omega\times n} & \mathbf{0}_{n_\omega\times n_z} & \mathbf{0}_{n_\omega\times n_\omega}\end{array}\right] && (1.16)\\
\tilde{\mathbf{f}}(\tilde{\mathbf{x}},\mathbf{u},\boldsymbol{\delta}) &= \left[\begin{array}{c}\mathbf{A}\mathbf{x}+\mathbf{B}_1\boldsymbol{\Delta}\mathbf{z}+\mathbf{B}_2\boldsymbol{\Theta}(\boldsymbol{\omega})+\mathbf{B}_3\mathbf{u}\\ \mathbf{C}_1\mathbf{x}+(\mathbf{D}_{11}\boldsymbol{\Delta}-\mathbf{I}_{n_z})\,\mathbf{z}+\mathbf{D}_{12}\boldsymbol{\Theta}(\boldsymbol{\omega})+\mathbf{D}_{13}\mathbf{u}\\ \mathbf{C}_2\mathbf{x}+\mathbf{D}_{21}\boldsymbol{\Delta}\mathbf{z}+\mathbf{D}_{22}\boldsymbol{\Theta}(\boldsymbol{\omega})-\boldsymbol{\omega}+\mathbf{D}_{23}\mathbf{u}\end{array}\right] && (1.17)\\
\tilde{\mathbf{g}}(\tilde{\mathbf{x}},\mathbf{u},\boldsymbol{\delta}) &= \mathbf{C}_3\mathbf{x}+\mathbf{D}_{31}\boldsymbol{\Delta}\mathbf{z}+\mathbf{D}_{32}\boldsymbol{\Theta}(\boldsymbol{\omega})+\mathbf{D}_{33}\mathbf{u} && (1.18)
\end{aligned}$$

thus by sampling the output of a dynamic system defined by the algebraic transformation output (1.15) and by an index-1, semi-explicit DAE system defined by eq. (1.14), fed by the sampled input $\mathbf{u}(t_k)$.

The numerical integration of the DAE system (1.14) can be dealt with in MATLAB through the `ode15s.m` function, which implements a variable order BDF method and allows to define separately the mass matrix $\mathbf{M}$ and the vector function $\tilde{\mathbf{f}}(\tilde{\mathbf{x}},\mathbf{u},\boldsymbol{\delta})$. Moreover, in order to improve reliability and efficiency, the Jacobian matrix $\partial\tilde{\mathbf{f}}/\partial\tilde{\mathbf{x}}$ should be analytically computed.

The sensitivity

$$\frac{\partial\mathbf{e}(t_k,\boldsymbol{\delta})}{\partial\delta_i}=-\frac{\partial\hat{\mathbf{y}}(t_k,\boldsymbol{\delta})}{\partial\delta_i}=-\mathbf{y}_i'(t_k) \qquad (1.19)$$

can be computed by sampling the output $\mathbf{y}_i'(t)$ of the following LFT system (second stage of the scheme in Fig. 1.2):

$$\begin{aligned}
\dot{\mathbf{x}}_i'(t) &= \mathbf{A}\mathbf{x}_i'(t)+\mathbf{B}_1\mathbf{w}_i^*(t)+\mathbf{B}_2\boldsymbol{\zeta}_i'(t)+\mathbf{B}_1\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.20)\\
\mathbf{z}_i'(t) &= \mathbf{C}_1\mathbf{x}_i'(t)+\mathbf{D}_{11}\mathbf{w}_i^*(t)+\mathbf{D}_{12}\boldsymbol{\zeta}_i'(t)+\mathbf{D}_{11}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.21)\\
\boldsymbol{\omega}_i'(t) &= \mathbf{C}_2\mathbf{x}_i'(t)+\mathbf{D}_{21}\mathbf{w}_i^*(t)+\mathbf{D}_{22}\boldsymbol{\zeta}_i'(t)+\mathbf{D}_{21}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.22)\\
\mathbf{y}_i'(t) &= \mathbf{C}_3\mathbf{x}_i'(t)+\mathbf{D}_{31}\mathbf{w}_i^*(t)+\mathbf{D}_{32}\boldsymbol{\zeta}_i'(t)+\mathbf{D}_{31}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.23)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\Delta}_{\delta_i} &= \frac{\partial\boldsymbol{\Delta}}{\partial\delta_i}=\mathrm{diag}\{\mathbf{0}_{r_1\times r_1},\ldots,\mathbf{I}_{r_i},\ldots,\mathbf{0}_{r_q\times r_q}\} && (1.24)\\
\mathbf{w}_i^*(t) &= \boldsymbol{\Delta}\mathbf{z}_i'(t) && (1.25)\\
\boldsymbol{\zeta}_i'(t) &= \left.\frac{\partial\boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial\boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)}\boldsymbol{\omega}_i'(t)=\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t))\boldsymbol{\omega}_i'(t) && (1.26)
\end{aligned}$$

Differentiating eqs. (1.8–1.13) with respect to parameter δ_i yields:

$$\begin{aligned}
\frac{\partial \dot{\mathbf{x}}(t)}{\partial \delta_i} &= \mathbf{A}\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{B}_1\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{B}_2\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} && (1.27)\\
\frac{\partial \mathbf{z}(t)}{\partial \delta_i} &= \mathbf{C}_1\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{11}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{12}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} && (1.28)\\
\frac{\partial \boldsymbol{\omega}(t)}{\partial \delta_i} &= \mathbf{C}_2\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{21}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{22}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} && (1.29)\\
\frac{\partial \hat{\mathbf{y}}(t)}{\partial \delta_i} &= \mathbf{C}_3\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{31}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{32}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} && (1.30)\\
\frac{\partial \mathbf{w}(t)}{\partial \delta_i} &= \frac{\partial \boldsymbol{\Delta}}{\partial \delta_i}\mathbf{z}(t) + \boldsymbol{\Delta}\frac{\partial \mathbf{z}(t)}{\partial \delta_i} && (1.31)\\
\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} &= \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \frac{\partial \boldsymbol{\omega}(t)}{\partial \delta_i} && (1.32)
\end{aligned}$$

which gives eqs. (1.20–1.26) by renaming the partial derivatives:

$$\mathbf{x}'_i = \frac{\partial \mathbf{x}}{\partial \delta_i}, \quad \mathbf{z}'_i = \frac{\partial \mathbf{z}}{\partial \delta_i}, \quad \boldsymbol{\omega}'_i = \frac{\partial \boldsymbol{\omega}}{\partial \delta_i}, \quad \mathbf{y}'_i = \frac{\partial \hat{\mathbf{y}}}{\partial \delta_i}, \quad \mathbf{w}'_i = \frac{\partial \mathbf{w}}{\partial \delta_i}, \quad \boldsymbol{\zeta}'_i = \frac{\partial \boldsymbol{\zeta}}{\partial \delta_i}$$

By substituting (1.25) and (1.26) in (1.20–1.23) and solving (1.21) and (1.22) with respect to $\mathbf{z}'_i(t)$ and $\boldsymbol{\omega}'_i(t)$ the following time-variant, linear system is obtained,

$$\begin{aligned}
\dot{\mathbf{x}}'_i(t) &= \tilde{\mathbf{A}}(\boldsymbol{\omega}(t))\mathbf{x}'_i(t) + \tilde{\mathbf{B}}(\boldsymbol{\omega}(t))\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.33)\\
\mathbf{y}'_i(t) &= \tilde{\mathbf{C}}(\boldsymbol{\omega}(t))\mathbf{x}'_i(t) + \tilde{\mathbf{D}}(\boldsymbol{\omega}(t))\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (1.34)
\end{aligned}$$

where

$$\begin{aligned}
\tilde{\mathbf{A}}(\boldsymbol{\omega}(t)) &= \mathbf{A} + \begin{bmatrix} \mathbf{B}_1\boldsymbol{\Delta} & \mathbf{B}_2\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} && (1.35)\\
\tilde{\mathbf{B}}(\boldsymbol{\omega}(t)) &= \mathbf{B}_1 + \begin{bmatrix} \mathbf{B}_1\boldsymbol{\Delta} & \mathbf{B}_2\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{D}_{11} \\ \mathbf{D}_{21} \end{bmatrix} && (1.36)\\
\tilde{\mathbf{C}}(\boldsymbol{\omega}(t)) &= \mathbf{C}_3 + \begin{bmatrix} \mathbf{D}_{31}\boldsymbol{\Delta} & \mathbf{D}_{32}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} && (1.37)\\
\tilde{\mathbf{D}}(\boldsymbol{\omega}(t)) &= \mathbf{D}_{31} + \begin{bmatrix} \mathbf{D}_{31}\boldsymbol{\Delta} & \mathbf{D}_{32}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{D}_{11} \\ \mathbf{D}_{21} \end{bmatrix} && (1.38)\\
\mathbf{F}(\boldsymbol{\omega}(t)) &= \begin{bmatrix} \mathbf{I}_{n_z} - \mathbf{D}_{11}\boldsymbol{\Delta} & -\mathbf{D}_{12}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \\ -\mathbf{D}_{21}\boldsymbol{\Delta} & \mathbf{I}_{n_\omega} - \mathbf{D}_{22}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix}^{-1} && (1.39)
\end{aligned}$$

It must be pointed out that, in the implementation, the Jacobian $\boldsymbol{\Theta}_\omega$ is simbolically computed directly from the definition of the function $\boldsymbol{\Theta}$.

Since the second stage requires the value of $\boldsymbol{\omega}(t)$ computed in the first stage, the two systems must be run in cascade, as in Fig. 1.2, that shows the complete procedures.

It is also important to underline the fact that the second stage must be executed as many times as the number of parameters in the $\boldsymbol{\delta}$ vector. This is a very critical part from the computational cost point of view, since the second stage is repeated many times during the whole estimation procedure.

A solution to increase the computational efficiency was implemented in the Toolbox. Given the linear ODE (1.33) it is possible to rewrite it as

$$\dot{\mathbf{x}}_i'(t) = \mathbf{\Gamma}_2(\omega) \begin{bmatrix} \mathbf{x}_i'(t) \\ \mathbf{\Delta}_{\delta_i}\mathbf{z}(t) \end{bmatrix} \tag{1.40}$$

with

$$\mathbf{\Gamma}_2(\boldsymbol{\omega}) = \left[\begin{bmatrix} \mathbf{A} & \mathbf{B}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1\mathbf{\Delta} & \mathbf{B}_2 \left.\frac{\partial \mathbf{\Theta}(\omega)}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \end{bmatrix} \mathbf{\Gamma}_1(\omega) \right] \tag{1.41}$$

$$\mathbf{\Gamma}_1(\boldsymbol{\omega}) = \begin{bmatrix} \mathbf{I}_{n_z} - \mathbf{D}_{11}\mathbf{\Delta} & -\mathbf{D}_{12} \left.\frac{\partial \mathbf{\Theta}(\omega)}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \\ -\mathbf{D}_{21}\mathbf{\Delta} & \mathbf{I}_{n_\omega} - \mathbf{D}_{22} \left.\frac{\partial \mathbf{\Theta}(\omega)}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{C}_1 & \mathbf{D}_{11} \\ \mathbf{C}_2 & \mathbf{D}_{21} \end{bmatrix} \tag{1.42}$$

Analyzing the use of $\mathbf{\Gamma}_1(\boldsymbol{\omega})$ and $\mathbf{\Gamma}_2(\boldsymbol{\omega})$, it can be noticed that their components can be evaluated and saved in a `CommonTerms` vector, during the first stage of the LFT parameter estimator at all the time instants of the simulation. These terms are then interpolated and evaluated in every time instant of the second stage in order to greatly reduce the overall computational cost of the parameter identification procedure.

It is also worth mentioning that, in order to deal with parameter estimation, it is essential to rewrite model (1.8−1.13) by introducing normalized unknown parameters $\bar{\delta}_i$, varying between ± 1 as parameter δ_i varies between a maximum $\delta_i^{\max}$ and a minimum $\delta_i^{\min}$ with

$$\delta_i = \frac{(\delta_i^{\max} + \delta_i^{\min})}{2} + \frac{\bar{\delta}_i(\delta_i^{\max} - \delta_i^{\min})}{2} \tag{1.43}$$

Deriving the LFT formulation could be non-trivial if carried out manually, to this aim, a symbolic computing approach, partially developed in [3] with reference to linear models, has been first fully implemented for application to the general non-linear case in [1]. Moreover, there is no unique solution.

1.1 Example

As an example consider this simple, purely academic, nonlinear model

$$\dot{x}_1 = 5x_1\delta_1 + 3\frac{x_2}{1+\delta_2} \tag{1.44}$$

$$\dot{x}_2 = u \tag{1.45}$$

$$y = x_1 + x_2 \tag{1.46}$$

where δ_1 and δ_2 are the uncertain parameters. The relevant LFT formulation can be derived according to the following steps:

1. Collect the uncertain parameters in the vector $\boldsymbol{\delta}$:

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_1 & \delta_2 \end{bmatrix}^T \tag{1.47}$$

2. Each parameter δ_i has to multiply another varying quantity, otherwise, equations are modified to get the result, in our example:

$$\frac{x_2}{1+\delta_2} \rightarrow \frac{x_2^2}{x_2 + x_2\delta_2} \tag{1.48}$$

3. Define:

$$w_j = \delta_i z_j \quad , \quad \begin{cases} i = 1, \dots, q, \\ j = r_{i-1}+1, \dots, r_{i-1}+r_i \end{cases} \tag{1.49}$$

   in our example:

$$\begin{aligned} w_1 &= \delta_1 z_1 & (1.50) \\ w_2 &= \delta_2 z_2 & (1.51) \end{aligned}$$

4. Compute

$$\mathbf{z} = \mathbf{C}_1\mathbf{x} + \mathbf{D}_{11}\mathbf{w} + \mathbf{D}_{12}\boldsymbol{\zeta} + \mathbf{D}_{13}\mathbf{u} \tag{1.52}$$

   keeping linear dependencies of $\mathbf{z}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:

$$\begin{aligned} z_1 &= x_1 & (1.53) \\ z_2 &= x_2 & (1.54) \end{aligned}$$

5. Compute

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}_1\mathbf{w} + \mathbf{B}_2\boldsymbol{\zeta} + \mathbf{B}_3\mathbf{u} & (1.55) \\ \mathbf{y} &= \mathbf{C}_3\mathbf{x} + \mathbf{D}_{31}\mathbf{w} + \mathbf{D}_{32}\boldsymbol{\zeta} + \mathbf{D}_{33}\mathbf{u} & (1.56) \end{aligned}$$

   keeping linear dependencies of $\dot{\mathbf{x}}$ and $\mathbf{y}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:

$$\begin{aligned} \dot{x}_1 &= 5w_1 + 3\zeta & (1.57) \\ \dot{x}_2 &= u & (1.58) \\ y &= x_1 + x_2 & (1.59) \end{aligned}$$

6. Define

$$\boldsymbol{\zeta} = \boldsymbol{\Theta}(\boldsymbol{\omega}) \tag{1.60}$$

   in our example

$$\zeta = \frac{\omega_1^2}{\omega_1 + \omega_2} \tag{1.61}$$

7. Compute
$$\boldsymbol{\omega} = \mathbf{C}_2\mathbf{x} + \mathbf{D}_{21}\mathbf{w} + \mathbf{D}_{22}\boldsymbol{\zeta} + \mathbf{D}_{23}\mathbf{u} \tag{1.62}$$
keeping linear dependencies of $\boldsymbol{\omega}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:
$$\omega_1 = x_2 \tag{1.63}$$
$$\omega_2 = w_2 \tag{1.64}$$

The final LFT formulation is thus given by:

$$\dot{\mathbf{x}} = \begin{bmatrix} 5w_1 + 3\zeta & u \end{bmatrix}^T \tag{1.65}$$
$$\mathbf{z} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \tag{1.66}$$
$$\boldsymbol{\omega} = \begin{bmatrix} x_2 & w_2 \end{bmatrix}^T \tag{1.67}$$
$$y = x_1 + x_2 \tag{1.68}$$
$$\mathbf{w} = \begin{bmatrix} \delta_1 z_1 & \delta_2 z_2 \end{bmatrix}^T \tag{1.69}$$
$$\zeta = \frac{\omega_1^2}{\omega_1 + \omega_2} \tag{1.70}$$

Chapter 2

User Manual

2.1 A nonlinear model

The nonlinear system chosen to help understanding the use of the toolbox is a simple mass, spring and damper system, depicted in Fig. 2.1 and described by the following model:

$$\dot{x}_1 = x_2 \tag{2.1}$$

$$\dot{x}_2 = -\frac{k_1}{m}x_1 - \frac{k_2}{m}x_1^3 - \frac{c}{m}x_2 + \frac{1}{m}u \tag{2.2}$$

$$y = x_1 \tag{2.3}$$

where x_1 is the position of the mass, x_2 is the velocity, the constants m, k_1, k_2 and c are respectively the mass, the linear stiffness coefficient, the nonlinear stiffness coefficient and the damping coefficient, the input u is an external force applied to the mass and the output y of the system is the position of the mass.

The mass and the linear stiffness coefficient are assumed as known and are equal to $m = 1$ Kg and $k_1 = 20$ N/m respectively, while the nonlinear stiffness and the damping coefficients have to be identified, thus $\delta_1 = c$, $\delta_2 = k_2$.

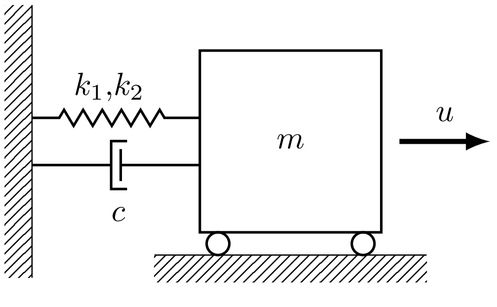


Figure 2.1: A nonlinear mass, spring, damper system

An equivalent LFT reformulation of the model is thus given by:

$$\begin{aligned}
\dot{\mathbf{x}} &= \begin{bmatrix} x_2 \\ -\frac{k_1}{m}x_1 - \frac{1}{m}w_1 - \frac{1}{m}w_2 + \frac{1}{m}u \end{bmatrix} && (2.4)\\
\mathbf{z} &= \begin{bmatrix} x_2 & \zeta \end{bmatrix}^T && (2.5)\\
\omega &= x_1 && (2.6)\\
y &= x_1 && (2.7)\\
\mathbf{w} &= \begin{bmatrix} \delta_1 z_1 & \delta_2 z_2 \end{bmatrix}^T && (2.8)\\
\zeta &= \omega^3 && (2.9)
\end{aligned}$$

and the only non-null matrices relevant to formulation (1.8–1.13) are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k_1}{m} & 0 \end{bmatrix} \quad , \quad \mathbf{B}_1 = \begin{bmatrix} 0 & 0 \\ -\frac{1}{m} & -\frac{1}{m} \end{bmatrix} \quad , \quad \mathbf{B}_3 = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \tag{2.10}$$

$$\mathbf{C}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad , \quad \mathbf{D_{12}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.11}$$

$$\mathbf{C_2} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.12}$$

$$\mathbf{C_3} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.13}$$

In order to perform parameter identification two scripts must be created:

- `spring_lft.m`: the function script, called in the main script, where the LFT form is defined.
- `main.m`: the main script where the data are loaded, the identification problem is defined and all the main functions are called.

2.2 spring_lft.m

As already mentioned, the `spring_lft.m` script is a function called in the main script where the LFT model is defined.

In this first line (Lis. 2.1) the function output `lftfun` is declared, which is the description of the LFT model. The input parameters are the known parameters of the model, m and k_1, and as many (2-dimensional) row vectors as the unknown parameters, containing the limits for the constrained search optimization. In this example we have two vectors `k2lim` and `clim`.

Listing 2.1: Declaration of the function

```
function [lftfun] = spring_lft(k1,m,k2lim,Clim)
```

Then, we have to specify the number of inputs $\mathbf{u}$, states $\mathbf{x}$, outputs $\mathbf{y}$ and omega variables $\boldsymbol{\omega}$ of the LFT model, these definitions will later be used to correctly create and size the linear time invariant matrices of the LFT model.

Listing 2.2: Definition of the number of inputs $\mathbf{u}$, states $\mathbf{x}$, outputs $\mathbf{y}$ and omega variables $\boldsymbol{\omega}$ of the LFT model

```
%definition of the number of inputs, states, outputs and
    omega of the LFT
u_count = 1;
x_count = 2;
y_count = 1;
om_count = 1;
```

2.2.1 DeltaSym and theta_sym

We will now define the matrix $\boldsymbol{\Delta}$, named `DeltaSym`, as shown in Lis. 2.3.

Listing 2.3: Definition of the matrix $\boldsymbol{\Delta}$

```
%definition of the matrix delta
DeltaSym = {
    'parName' 'indStartDiag' 'indStopDiag' 'LowerBound'
       'HigherBound' 'toIdentify', 'lb', 'ub';
    'c' 1 1 Clim(1) Clim(2) 1 -1 1;
    'k2' 2 2 k2lim(1) k2lim(2) 1 -1 1;
        };
delta_count = 0;
for i=2:size(DeltaSym,1)
    delta_count = delta_count + DeltaSym{i,3} - DeltaSym
       {i,2} + 1;
end
```

In this matrix the first row, not to be modified, simply acts as a guide for the construction of the matrix itself.

The first column contains the names of the unknown parameters, in our case `'c'` (second row) and `'k2'` (third row).

The second and third column are used to indicate the first and last position of the unknown parameter on the $\boldsymbol{\Delta}$ matrix's diagonal. In our case the unknown parameters appears only once in the $\boldsymbol{\Delta}$ matrix, respectively $c = \delta_1$ in position (1,1) and $k_2 = \delta_2$ in position (2,2).

In the fourth and fifth column the lower and upper bound of the unknown parameters are defined.

The sixth column can be filled with 1 or 0 if we, respectively, want or don't want to identify the parameter. In the latter case the parameter will be maintained fixed at its initial value.

The seventh and eighth column should always be respectively filled with -1 and 1, defining the lower and upper bound of the limits in normalized form.

At last, once completed the definition of the `DeltaSym` matrix, the number of elements in the diagonal of matrix $\boldsymbol{\Delta}$ is computed.

The next lines (Lis. 2.4) create the column vector $\boldsymbol{\zeta}$ named `theta_sym`, after having defined an array of symbolic $\boldsymbol{\omega}$.

Please note that in case of two or more $\boldsymbol{\omega}$ the initial `if`' condition can be skipped.

Finally, the number of components of the vector $\boldsymbol{\zeta}$ is determined.

Listing 2.4: Definition of the vector ζ

```
%definition of the matrix theta (= ZETA)
if om_count == 1
    om = sym(['om1_1']);
else
om = sym('om', om_count);
end

theta_sym = [
    om(1)*om(1)*om(1);
    ];
theta_count = size(theta_sym,1);
```

2.2.2 LTI matrices

Initially, all matrices of the LFT formulation are filled with zeros (Lis. 2.5), then the only non-null elements are defined (Lis. 2.6).

Listing 2.5: Construction of the LTI matrices

```
%construction of the matrices of the LTI part
LTI = struct(...
    'A', zeros(x_count,x_count),...
    'B1', zeros(x_count,delta_count),...
    'B2', zeros(x_count,theta_count),...
    'B3', zeros(x_count,u_count),...
    'C1', zeros(delta_count,x_count),...
    'D11', zeros(delta_count,delta_count),...
    'D12', zeros(delta_count,theta_count),...
    'D13', zeros(delta_count,u_count),...
    'C2', zeros(om_count,x_count),...
    'D21', zeros(om_count,delta_count),...
    'D22', zeros(om_count,theta_count),...
    'D23', zeros(om_count,u_count),...
    'C3', zeros(y_count,x_count),...
    'D31', zeros(y_count,delta_count),...
    'D32', zeros(y_count,theta_count),...
    'D33', zeros(y_count,u_count));
```

Listing 2.6: Definition of the LTI matrices

```
%definition of the matrices
% dx1 = x2
LTI.A(1,2) = 1;
% dx2 = -k1/m*x1-1/m*w1-1/m*w2+1/m*u
LTI.A(2,1) = -k1/m;
LTI.B1(2,1) = -1/m;
LTI.B1(2,2) = -1/m;
LTI.B3(2,1) = 1/m;

% z1 = x2
LTI.D12(2,1) = 1;
% z2 = zeta
LTI.C1(1,2) = 1;

% om1 = x1
LTI.C2(1,1) = 1;
% LTI.C2(2,1) = 3;

% y1 = x1
LTI.C3(1,1) = 1;
```

We are finally able to construct the `lftfun` struct, which is the output of the function, by initially defining it and then calling the `lft_finalize` function.

Listing 2.7: Definition of the `lftfun` struct

```
%initial definition of the LFTfun
lftfun = struct(...
    'LTI', LTI,...
    'DeltaSym', {DeltaSym},...
    'DeltaVal', zeros(delta_count,delta_count)...
        );

% save additional data for reference purpose
lftfun.theta_sym = theta_sym;
lftfun.u_count = u_count;
lftfun.x_count = x_count;
lftfun.y_count = y_count;
lftfun.om_count = om_count;
lftfun.theta_count = theta_count;
lftfun.delta_count = delta_count;

%function to finalize the definition of LFTfun
lftfun = lft_finalize(lftfun);

end
```

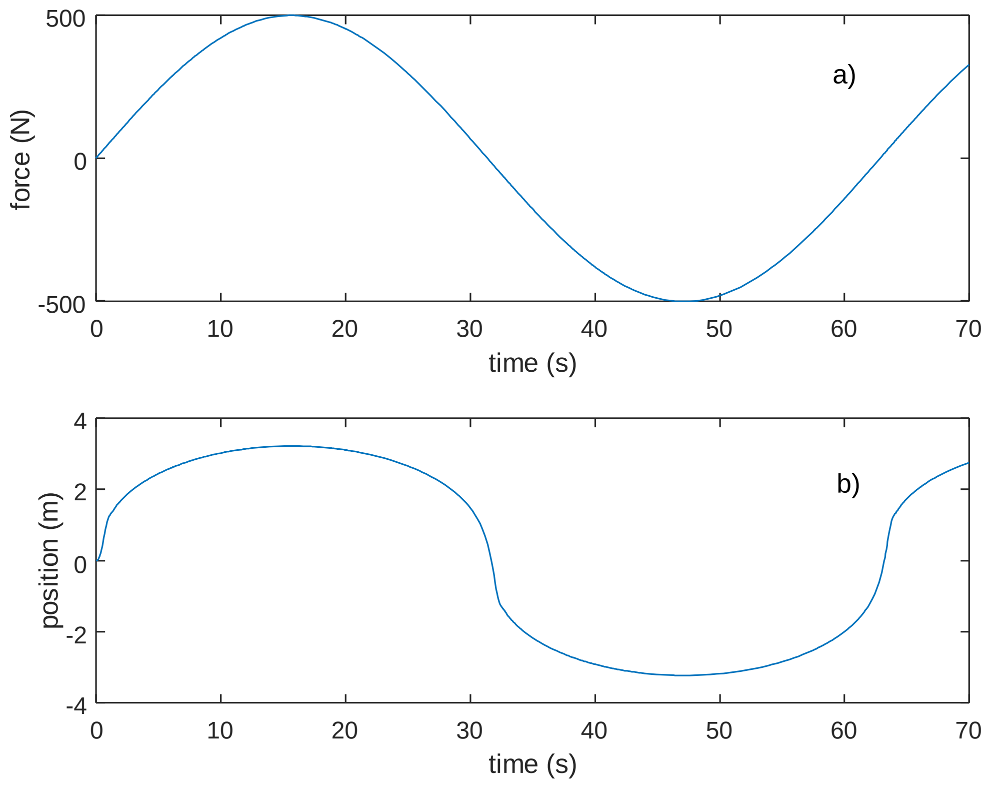


Figure 2.2: Input force a) and position b)

2.3 `main.m`

The `main.m` script is where all the data are loaded, where the solver options are defined and, finally, where all the main functions are called to tackle the identification process.

The data used for parameter identification will be collected by applying the input (Fig. 2.2 a))

$$u(t) = 500 \sin(0.1t) \tag{2.14}$$

and sampling the output of system (2.1–2.3) (Fig. 2.2 b)), assuming the following values for the parameters to be identified:

$$c = 8 \text{ Ns/m} \tag{2.15}$$
$$k_2 = 13 \text{ N/m}^3 \tag{2.16}$$

Of course, since this is an ideal case, an almost perfect convergence to the actual values of the parameters is expected.

The time vector and the input values are stored in the file `F.mat`, while the output values are stored in the file `lft_y.mat`. Note that the output values are always assumed to be evaluated at the time instants of the time vector in the input matrix.

As shown in Lis. 2.8, after a very generic clean up of the workspace, command window and of the possible figures open, the script begins by loading the input and output data of the system, that will be used for identification.

Listing 2.8: Loading of the input and output data

```
clc, clear all, close all

%import input (Force 'F') NB every input or state comes
   with its timevector
load F.mat;
t = F(:,1);
F = F(:,2);

%import the output (position 'x')
load('lft_y');
```

Then, before defining the `lftfun` struct, the known parameters of the model, in our case m and k_1, are defined, as well as the limits of the constrained search on the optimal values of the parameters (Lis. 2.9).

In this case, wide ranges have been defined, in order to show the capabilities of the tool. For the identification of multiple parameters of more complex systems, in order to speed up the estimation procedure, it is recommended to choose reasonable ranges in which the LFT identification Toolbox should find the optimal solution.

Listing 2.9: Definition of the `lftfun` struct

```
%definition of the constant and known parameters
k1=20; %[kN/m]
m=1; %[Kg]

%definition of the limits of the uncertain parameters
%in this case the values to be found is 'k2=13' and 'c
   =8'
k2lim = [1 200];
Clim = [1 200];

%load lft function and solver options
lftfun = spring_lft(k1,m,k2lim,Clim);
```

In the case of multiple outputs their influence on the cost function (1.3) can be weighted through the matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & W_p \end{bmatrix} \tag{2.17}$$

in turn, the weights W_i can be defined by defining the vector `lftfun.ISO.Nominal` as the vector

$$\boldsymbol{\alpha} = \begin{bmatrix}\alpha_1 & \alpha_2 & \dots & \alpha_p\end{bmatrix}^T \tag{2.18}$$

$$\alpha_i = \frac{1}{\sqrt{W_i}} \tag{2.19}$$

where `lftfun` is the structure containing the description of the LFT model.

One possible choice is:

$$W_i = \frac{1}{(M_i - m_i)^2} \tag{2.20}$$

where M_i is the highest value of y_i and m_i is the lowest one, which can be easily implemented as:

Listing 2.10: Definition of the `lftfun.ISO.Nominal` vector

```
M = max(lft_y);
m = min(lft_y);
for i=1:length(M)
    alpha(i) = M(i) - m(i);
end
lftfun.ISO.Nominal = alpha;
```

The next piece of code defines the options for the LFT Solver (see Section 2.4), the (struct) input and the initial conditions of the model's state variables.

Listing 2.11: Definition of options for the LFT solver, the input and the initial conditions of the model's state variables

```
%load LFT solver options
LFTsolverOptions = lftSet('RelTol', 1e-4,...
    'AbsTol', 1e-8,...
    'SolutionInterpMethod', 'spline',...
    'SolutionTimeSpan', t,...
    'SensAlgorithm', 'ode15s',...
    'OversamplingMethod', 'spline' );

%define the input
Input = struct('Type', 'interpolated', 'Samples', F, '
   Time', t);

%define the initial conditions of the model's state
   variables
InitialConditions = struct('StateInitialConditions', [0;
    0]);
```

Similarly, the options for the LFT optimizer (see Section 2.5) and the initial guess of parameters in normalized form[1] are defined before running the estimation

[1]An initial guess equal to zero means that the initial guess of the parameter is equal to the mid-point in between the search limits.

through the optimizer.

The identified parameters (not normalized) are finally presented through the norm2abs function and the Hessian and its condition number are shown.

Listing 2.12: Definition of options of the LFT optimizer, the initial guess of parameters in normalized form and start of the identification

```
%load LFT estimator options
LFToptimOptions = lftOptSet('Display', 'iter',...
    'MaxIter', 120, ...
    'TolFun', 1e-8 ...
    );

% estimate
% initial guess of parameters in normalized form
lftfun.DeltaVal = diag([0 0]);

%estimation
[DELTA_opt,fval,J,grad,H,CN, history] = lftOptDelta(
   lftfun,Input,InitialConditions,LFTsolverOptions,lft_y
   ,LFToptimOptions);

norm2abs(lftfun, lftfun.DeltaVal, DELTA_opt);

H=H(:,:,end), CN=CN(end)
```

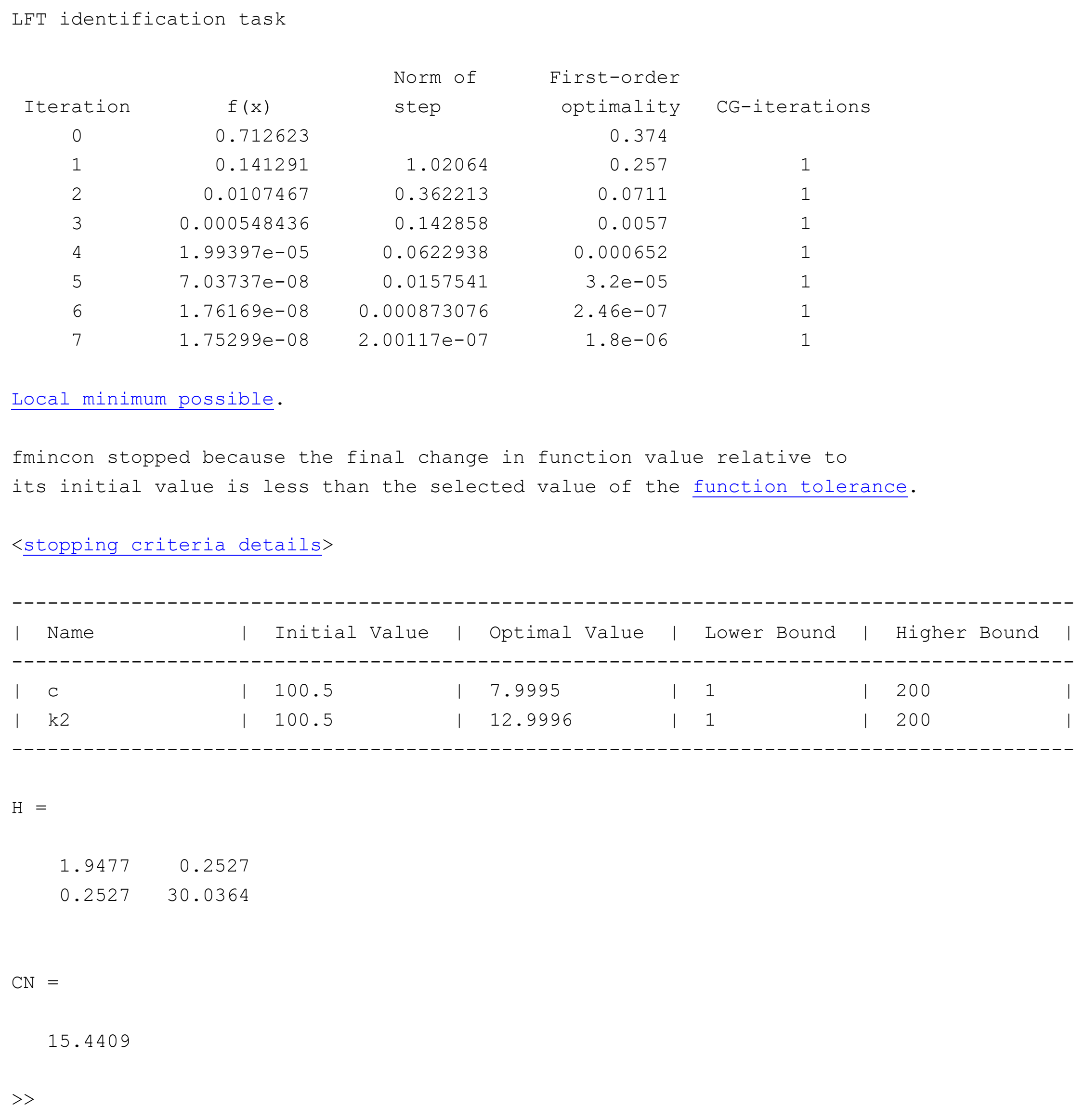
2.3.1 Results

The results obtained by running the main.m script are shown in the command window as in Fig. 2.3

In the first part of the command window informations on all iterations are shown: the first column reports the number of the iteration, the second the value of the cost function, the third the norm of each step, the fourth the first-order optimality measure and the last the number of conjugate gradient iterations taken.

At the end of the identification the fmincon function, that is the function used to minimize the cost function, will print the *stopping criteria* that stopped the identification. In our case the stopping criteria is the final change in function value relative to its initial value, which is less than the selected value of the function tolerance: 10^{-8}. It is also possible to click on <stopping criteria details> to have more details about the stopping criteria trigger.

Then, the norm2abs function prints the optimal values found by the LFT Identification Toolbox for the unknown parameters, along with their name, the initial guess values and the search limits. In the example, the values found by the LFT



```
LFT identification task

                                Norm of      First-order
 Iteration         f(x)          step          optimality   CG-iterations
     0           0.712623                         0.374
     1           0.141291         1.02064         0.257              1
     2          0.0107467        0.362213        0.0711              1
     3        0.000548436        0.142858        0.0057              1
     4        1.99397e-05       0.0622938      0.000652              1
     5        7.03737e-08       0.0157541       3.2e-05              1
     6        1.76169e-08     0.000873076      2.46e-07              1
     7        1.75299e-08     2.00117e-07       1.8e-06              1

Local minimum possible.

fmincon stopped because the final change in function value relative to
its initial value is less than the selected value of the function tolerance.

<stopping criteria details>

-------------------------------------------------------------------------------------------
|  Name               |  Initial Value  |  Optimal Value  |  Lower Bound  |  Higher Bound  |
-------------------------------------------------------------------------------------------
|  c                  |  100.5          |  7.9995         |  1            |  200           |
|  k2                 |  100.5          |  12.9996        |  1            |  200           |
-------------------------------------------------------------------------------------------

H =

    1.9477    0.2527
    0.2527   30.0364


CN =

   15.4409

>>
```

Figure 2.3: Results of identification

Identification Toolbox, after 7 iterations, are:

$$c = 7.9995 \text{ Ns/m} \quad (2.21)$$

$$k_2 = 12.9996 \text{ N/m}^3 \quad (2.22)$$

which correspond respectively to a 0.0062% and 0.0031% error, with respect to the real values of the parameters.

2.4 LFT Solver

The LFT solver function, which is not directly used in the main script, is the most important part of the LFT Toolbox since it is the heart of the optimization function.

The LFT solver function, as its name suggests, solves the nonlinear, time invariant LFT model given the initial state, the input and the options. The solver function is called as follows:

Listing 2.13: Call of the LFT solver

```
[output,internalSolution,CommonTerms] =
    lftSolver(lftfun,Input,InitialConditions,
       lftSolverOptions);
```

The outputs of the function are:

- `output`: a matrix of $1+p$ columns, where p is the number of system outputs. The first column is the time vector t, while the other p columns are the output components of the system evaluated at time instants in vector t.
- `internalSolution`: a matrix of $1+m+n+n_z+n_\omega+p$ columns, where m, n, n_z and n_ω are respectively the number of inputs, states, $\mathbf{z}$ variables and $\boldsymbol{\omega}$ variables. The first column is the vector of the time instants chosen by the solver and in the other columns we find, in order, the inputs, the state variables, the $\mathbf{z}$ variables, the $\boldsymbol{\omega}$ variables and the outputs, evaluated at the time instants of the first column.
- `CommonTerms`: a struct containing the matrices of the LFT model both fixed, relative to the linear part of the system and time-variable, relative to the non-linear part of the system.

The inputs are the following:

- `lftfun`: struct containing the LFT model (in our example defined by the function `spring_lft.m`).
- `Input`: struct composed by:
  - `Type`: defines the type of the input: `discontinuous`, `interpolated` or `continuous`.

  - `Samples`: defines the values, for each time instants, of the inputs.
  - `Time`: defines the time vector of the inputs.
- `InitialConditions`: contains the vector of the initial conditions of the state variables of the LFT system.
- `lftSolverOptions`: struct containing the solver options composed by:
  - `RelTol`: relative tolerance on the solution (default 10^{-3}).
  - `AbsTol`: absolute tolerance on the solution (default 10^{-6}).
  - `MaxOrder`: maximum order of accuracy for `ode15s` solver (default 5).
  - `BDF`: variable setting the use of BDF method (default `off`).
  - `InitialStep`: initial stepsize (default 10^{-3}).
  - `MaxStep`: maximum stepsize (default not defined).
  - `NumberOfSteps`: maximum number of steps (default not defined).
  - `ShowIntTime`: boolean variable setting the printing in the command window of the time instant of each step (default `false`).
  - `SolutionInterpMethod`: type of interpolation that should be executed on the input values (default `linear`).
  - `OversamplingMethod`: type of interpolation that should be executed on the output values (default `linear`).
  - `SolutionTimeSpan`: time vector on which the output should be evaluated.
  - `Sensitivity`: For `Sensitivity = 0` the solver simply simulates the LFT system. For `Sensitivity = 'pX'`, with `X` equal to the number of the unknown parameter, it is possible to execute the sensitivity of the outuput with respect to the chosen unknown parameter (default `0`).
  - `SensAlgorithm`: algorithm to be used to compute the sensitivity (default `ode15s`).
  - `CommonTerms`: vector of common terms of the first stage (solver in simulation configuration) that can be passed to the sensitivity solver in order to be almost 100 times more rapid than the first stage.

2.5 LFT Optimizer

As for the `LFTsolverOptions` in Sec. 2.4 we are going to explain all the components of this struct:

- `TolFun`: lower bound on the change in the value of the objective function during a step (default 10^{-3}).

- `MaxStep`: maximum amplitude of the step in the optimization procedure (default not defined).
- `NumberOfSteps`: maximum number of steps of the optimization procedure (default 2).
- `PolynomialDegree`: degree of the interpolating polynomial in case of continuous type input (default empty).
- `RelTolX`: relative tolerance (default 10^{-3}).
- `MinNormGrad`: minimum value of the norm of the gradient (default 10^{-3}).
- `EpsilonLambda`: maximum value of tolerance for the zero (default 10^{-6}).
- `MaxIter`: maximum number of Newton iterations for the calculation of the step (default 20).
- `Display`: flag for printing in the command window the Newton iterations for the search of the optimal step (default `off`).
- `StartOptimSample`: first time instant, as position in the time vector, from which the algorithm should begin the optimization process (default 1).

To summarize, the function requires the following inputs:

- `lftfun`: LFT model of the system.
- `Input`: input variables struct.
- `InitialConditions`: initial conditions struct.
- `LFTsolverOptions`: solver options struct.
- `lft_y`: outputs matrix (in our example a vector).
- `LFToptimOptions`: optimizer options struct.

The output of the function are:

- `DELTA_opt`: row vector containing the identified parameter in normalized form.
- `fval`: final cost function's value.
- `J`: column vector containing the values of the cost function at each step.
- `grad`: matrix containing the value of the gradients at each step.
- `H`: struct containing the Hessian matrices at each step.
- `CN`: vector containing the condition number at every step.
- `history`: matrix containing the values of the unknown parameters at each step.

Bibliography

[1] A. D. Bona, G. Ferretti, E. Ficara, and F. Malpei. LFT modelling and identification of anaerobic digestion. *Control Engineering Practice*, 36:1 – 11, 2015. [cited on pages 6 and 10]

[2] F. Demourant and G. Ferreres. Closed loop identification of a LFT model. *Journal Européen des Systémes Automatisés*, 36(3):449–464, 2002. [cited on page 5]

[3] F. Donida, C. Romani, F. Casella, and M. Lovera. Integrated modelling and parameter estimation: an LFT-Modelica approach. In *2009 IEEE Conference on Decision and Control*, pages 8357–8362, Dec 2009. [cited on page 10]

[4] K. Hsu, T. Vincent, G. Wolodkin, S. Rangan, and K. Poolla. An LFT approach to parameter estimation. *Automatica*, 44(12):3087–3092, 2008. [cited on page 5]

[5] L. H. Lee and K. Poolla. Identification of Linear Parameter-Varying Systems Using Nonlinear Programming. *Journal of Dynamic Systems, Measurement, and Control*, 121:71–78, Mar. 1999. [cited on page 5]

[6] L. Ljung. *System Identification: Theory for the User.* Prentice Hall, Upper Saddle River, NJ, USA, 1999. [cited on page 6]