Nonlinear $H_\infty$ Control
Derivation and Implementation

Abstract
Mathematical and computational aspects of nonlinear $H_\infty$ control or, more precisely, of $L_2$ gain synthesis are considered. The output feedback problem is solved for smooth plants slightly more general than those found in literature so far. The main contribution, however, is that the solution to the resulting Hamilton-Jacobi-Isaacs inequalities is approximated by power series and implemented in Maple V. The controllers are obtained as C coded functions in a form suitable for simulations with SIMULINK; they could be used for implementation on DSPs such as dSPACE as well. Since the matrix inversion necessary for calculating the output injection gain usually cannot be performed symbolically, it is carried out on line in the C coded function. The Maple routines are tested on a benchmark example.
# Table of Contents

1 Introduction ................................................................. 3
  1.1 Potential of Nonlinear $H_{\infty}$ Control ......................... 4

2 Derivation of the Controller ............................................ 5
  2.1 Problem Formulation .................................................. 5
  2.2 State Feedback Controller ........................................... 6
  2.3 Output Feedback Controller ......................................... 8

3 Approximated Controller ................................................ 19
  3.1 State Feedback Controller ....................................... 20
  3.2 Output Feedback Controller ...................................... 23
  3.3 Alternative Output Feedback Controller ....................... 29

4 Listings for Maple V ...................................................... 36
  4.1 hinfSF: State Feedback ........................................... 36
  4.2 hinfOF\_x: Output Feedback ..................................... 38
  4.3 hinfOF\_p: Alternative Output Feedback .................. 43
  4.4 Auxiliary Subroutines ........................................... 47
  4.5 Generation of Code for SIMULINK ......................... 50

5 Example ................................................................. 58
  5.1 State Feedback Controller ................................... 58
  5.2 Output Feedback Controllers ................................. 62

Appendix: Derivatives ................................................... 66

References ............................................................... 68

Urs Christen was with the Department of Systems Science and Mathematics, Washington University in St. Louis, Missouri, USA, while this report was being completed.
1 Introduction

Shortly after the publication of the state-space solution to the linear $H_\infty$ control problem in 1989 [7], solutions to its nonlinear equivalent, the $L_2$ gain problem, began to appear in literature; for references see [2], [12], [13]. Those publications, however, mainly deal with the mathematical aspects of the $L_2$ gain problem. The question of how these methods may be applied to solve engineering problems remains unaddressed. In those references, the structure of the plant is, therefore, restricted to a minimal structure needed to show the relevant mathematical aspects of the solution. In the dissertation [2], the solution for input-affine plants can be found, and in [13], the solution to the more general plant

\begin{align*}
\dot{x} &= X(x, w, u) \\
z &= Z(x, u) \\
y &= Y(x, w)
\end{align*}

is studied.

In this report, we closely follow Isidori and Kang [13] but remove one structural assumption in order to have more freedom when considering the engineering aspects in the future. The plant considered is given by

\begin{align*}
\dot{x} &= X(x, w, u) \\
z &= Z(x, w, u) \\
y &= Y(x, w).
\end{align*}

The first equation in (1.0.1) describes the state $x$ of the plant subject to the exogenous input $w$, which consists of disturbance and reference signals, and to the control input $u$. The second equation defines the controlled variable $z$, which has to be kept small. The last expression is the measurement equation, which describes the output signal $y$ provided to the controller. As discussed in [1], it is not possible to extend the measurement equation to $y = Y(x, w, u)$ since the trick used in linear $H_\infty$ synthesis to deal with such systems cannot be transferred to nonlinear systems. This is not a big restriction, however, since the input of the plant always can be filtered with a sufficiently fast filter so that no feed-through occurs.

The aim of the control is to attenuate the influence of the exogenous input signal $w$ on the controlled variable $z$. The attenuation is defined in terms of the ratio of the $L_2$ norms of $w$ and $z$, the so-called $L_2$ gain. For a nonlinear system, the $L_2$ gain is smaller or equal to a constant $\gamma > 0$ if

$$
\frac{\|z\|_2}{\|w\|_2} = \frac{\sqrt{\int_0^\infty z(t)^T z(t) dt}}{\sqrt{\int_0^\infty w(t)^T w(t) dt}} \leq \gamma
$$

for all $w(\cdot)$ for which the $L_2$ norm exists.
1.1 Potential of Nonlinear $H_\infty$ Control

This report does not deal with the engineering aspects of nonlinear $H_\infty$ control. It rather provides the tools needed for future research in that direction. Nevertheless, some preliminary reflections about the potential of this method are in place.

Nonlinear $H_\infty$ control is an extension of the linear $H_\infty$ method, which has been used successfully since the end of the 1980’s. The nonlinear counterpart, as it is described here, just considers some higher orders in a series approximation of the plant and thus should be at least as good as the linear method applied to the first-order approximation of the plant. When a single linear, time-invariant controller is not sufficient to achieve the desired performance over the whole operating range, gain-scheduled controllers may be used. However, the “classical” gain scheduling is a very *ad hoc* procedure. The more systematic gain scheduling based on linear matrix inequalities [9], on the other hand, has the disadvantage that the parameters are assumed to have an infinitely high rate of variation. This assumption may lead to conservative control laws. Moreover, for the method described in [9], the parameters must enter the plant affinely. These two disadvantages could be overcome by nonlinear $H_\infty$ control.

As an alternative to the Taylor approximations, approximation methods which yield good results in a region rather than in a single point could be considered [3, 4]. However, these methods are — in their present form — computationally even more involved than those described in this report.

**Note**

Throughout this report, we tried to keep the contents on a self-explanatory level. For this reason, where it was found to be appropriate, we included many of the intermediate steps in the calculations. They appear in a smaller font.
2 Derivation of the Controller

2.1 Problem Formulation

We consider a system of the form

\[ \begin{align*}
\dot{x} &= X(x, w, u) \\
z &= Z(x, w, u) \\
y &= Y(x, w)
\end{align*} \tag{2.1.1} \]

with \( x \) defined in a neighborhood of the origin in \( \mathbb{R}^n \) and \( w \in \mathbb{R}^{m_1}, u \in \mathbb{R}^{m_2}, z \in \mathbb{R}^{p_1}, y \in \mathbb{R}^{p_2} \). It is assumed that \( X(x, w, u), Z(x, w, u), \) and \( Y(x, w) \) are smooth mappings of class \( C^k \), \( k \) being sufficiently large.

2.1.1 Assumptions

In addition to smoothness, the following is assumed:

A1: System (2.1.1) has an equilibrium point at \( (x, w, z) = (0, 0, 0) \):

\[ \begin{align*}
X(0, 0, 0) &= 0, \\
Z(0, 0, 0) &= 0, \\
Y(0, 0) &= 0.
\end{align*} \]

A2: The matrix

\[ D_{11} = \left. \frac{\partial}{\partial w} Z(x, w, u) \right|_{(0, 0, 0)} \]

satisfies

\[ \sigma(D_{11}) < \gamma \]

where \( \gamma \) is the bound on the \( L_2 \) gain of the closed-loop system.

A3: The matrix

\[ D_{12} = \left. \frac{\partial}{\partial u} Z(x, w, u) \right|_{(0, 0, 0)} \]

has full column rank \( m_2 \).

A4: The matrix

\[ D_{21} = \left. \frac{\partial}{\partial w} Y(x, w) \right|_{(0, 0)} \]

has full row rank \( p_2 \).

A5: For any bounded trajectory of system (2.1.1) with input \( w = 0 \) \( \forall t \)

\[ Z(x, 0, u) = 0 \quad \forall t \quad \Rightarrow \quad \lim_{t \to \infty} x(t) = 0. \]

Assumption A1, which says that the origin must be an equilibrium point of the system, is made for ease of derivation. If an equilibrium point exists, A1 can always be guaranteed by suitable variable transformations. A2 presupposes that the \( H_\infty \) problem is solved for the feed-through from \( w \) to \( z \) in the linear problem and ensures that the problem has a saddle point solution (cf. p. 7). Satisfying A2 is
never a problem since the control engineer designs the output \( z \) when defining the engineering problem.

Assumptions A3 and A4 ensure that the \( H_\infty \) problem is nonsingular and therefore the squares \( D_{12}^T D_{12} \) and \( D_{21}^T D_{21} \) are invertible in the Riccati equations for the linear problem. The last assumption, A5, says that the system with output \( z \), input \( u \), and \( w = 0 \) must not have a neutrally stable \cite[p. 388]{11} zero dynamics.

### 2.1.2 Goal

The controller has to achieve two goals: it must stabilize the closed-loop system and attenuate the influence of the exogenous input signal \( w \) on the controlled signal \( z \), i.e., it has to bound its \( L_2 \) gain by a given value \( \gamma \). The disturbance attenuation property can be characterized by dissipativity (see also \cite{13} and the references therein). A nonlinear system

\[
\dot{x} = f(x, w) \\
z = h(x, w)
\]

is said to be locally dissipative near \((x, w) = (0, 0)\) with respect to a given supply rate \( s(w, z) \) if there exists a smooth, nonnegative function \( V(x) \) vanishing at \( x = 0 \) such that

\[
V_x(x) f(x, w) + s(w, h(x, w)) \leq 0
\]

for all \((x, w)\) in a neighborhood of \((0, 0)\). If (2.1.2) is locally asymptotically stable and locally dissipative with respect to the supply rate

\[
s(w, z) = \gamma^2 \|w\|^2 - \|z\|^2,
\]

its output response for a sufficiently small input and for \( x(0) = 0 \) satisfies

\[
\int_0^t \left( \gamma^2 \|w(\tau)\|^2 - \|z(\tau)\|^2 \right) d\tau \geq V(x(t)) \geq 0
\]

for all \( t > 0 \). Hence, (2.1.2) has an \( L_2 \) gain less than or equal to \( \gamma \).

### 2.2 State Feedback Controller

In this section, a feedback law \( u = \alpha_2(x) \) is sought which renders the closed-loop system

\[
\dot{x} = X(x, w, \alpha_2(x)) \\
z = Z(x, w, \alpha_2(x))
\]

(locally) dissipative with respect to the supply rate \( s(w, z) = \gamma^2 \|w\|^2 - \|z\|^2 \). This problem can be described as a zero sum differential game with two players \cite{13}.

Define

\[
v = \begin{bmatrix} w \\ u \end{bmatrix},
\]

with which the system equations read

\[
\dot{x} = \tilde{X}(x, v) \\
z = \tilde{Z}(x, v).
\]

Then
\[ \gamma^2 \|w\|^2 = v^T \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} v, \]

and the Hamiltonian function for this differential game can be found to be

\[ H(x, \lambda, v) = \lambda^T \dot{X}(x, v) + \|\dot{Z}(x, v)\|^2 - v^T \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} v. \]

In a neighborhood of \((x, \lambda) = (0, 0)\), there exists a smooth function

\[ v_a(x, \lambda) = \begin{bmatrix} w_a(x, \lambda) \\ u_a(x, \lambda) \end{bmatrix} \]

satisfying

\[ \frac{\partial}{\partial v} H(x, \lambda, v) \bigg|_{v = v_a(x, \lambda)} = 0 \]

\[ v_a(0, 0) = 0. \]

The Hessian of \(H\) satisfies

\[ \frac{\partial^2}{\partial v^2} H(x, \lambda, v) \bigg|_{(x, \lambda, v) = (0, 0, 0)} = 2R = 2 \begin{bmatrix} D_{11}^T D_{11} - \gamma^2 I_{m_1} & D_{11}^T D_{12} \\ D_{12}^T D_{11} & D_{12}^T D_{12} \end{bmatrix}. \]

The matrix \(R\) may be factored as \(R = N^T M N\) with

\[ M = \begin{bmatrix} D_{11}^T D_{11} - \gamma^2 I_{m_1} - D_{11}^T D_{12} (D_{12}^T D_{12})^{-1} D_{12}^T D_{11} & 0 \\ 0 & D_{12}^T D_{12} \end{bmatrix} \]

\[ N = \begin{bmatrix} I_{m_1} \\ (D_{12}^T D_{12})^{-1} D_{12}^T D_{11} I_{m_2} \end{bmatrix} \]

where the upper diagonal element of \(M\) is negative definite due to Assumption A2 and the lower diagonal element is positive definite. Hence, the Hamiltonian function \(H\) has a saddle point in \((w, u)\) for each \((x, \lambda)\) in the neighborhood of \((x, \lambda, w, u) = (0, 0, 0, 0)\):

\[ H(x, \lambda, v) \bigg|_{v = \begin{bmatrix} w \\ u_a(x, \lambda) \end{bmatrix}} \leq H(x, \lambda, v) \bigg|_{v = \begin{bmatrix} w_a(x, \lambda) \\ u_a(x, \lambda) \end{bmatrix}} \leq H(x, \lambda, v) \bigg|_{v = \begin{bmatrix} w_a(x, \lambda) \\ u \end{bmatrix}}. \]

Define a smooth, nonnegative function \(V: \mathbb{R}^n \rightarrow \mathbb{R}\) in a neighborhood of \(x = 0\) such that \(V(0) = 0\) and the Hamilton-Jacobi-Isaacs inequality

\[ H(x, V_x^T(x), v_a(x, V_x^T(x))) \leq 0 \]

holds for each \(x\) in a neighborhood of zero. Then

\[ u = u_a(x, V_x^T(x)) \]
yields a closed-loop system satisfying
\[ V_x(x)X(x, w, u_*) + \| Z(x, w, u_*) \|^2 - \gamma^2 \| w \|^2 \leq 0 , \] (2.2.5)
i.e., a system which has the required dissipativity property in a neighborhood of \((x, w) = (0, 0)\).

Due to Assumption A5, the feedback law (2.2.4) locally asymptotically stabilizes the system if \( V(x) \) is positive definite. This can be seen by a Lyapunov type of argument. For \( w = 0 \), expression (2.2.5) reads
\[ V_x(x)X(x, 0, u_*) + \| Z(x, 0, u_*) \|^2 \leq 0 \]
where \( Z(x, 0, u_*) \) can only be zero for asymptotically stable trajectories of \( x \) (Assumption A5). Thus, \( V(x) \) being positive definite and
\[ \frac{d}{dt} V(x) = \frac{\partial}{\partial t} V(x) + V_x(x) \frac{\partial x}{\partial t} = V_x(x)X(x, 0, u_*) \]
being negative proves asymptotic stability of the closed-loop system.

### 2.3 Output Feedback Controller

If only the output \( y \) rather than the state \( x \) is available for feedback, the state has to be reconstructed by an observer. In this section, the conditions on such an observer are derived which must be met to ensure that the composition of the plant and the controller containing the observer satisfy the \( H_\infty \) norm.

#### 2.3.1 A Necessary Condition for Output Feedback

Define the Hamiltonian function \( K: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m_1 \times \mathbb{R}^p_2 \rightarrow \mathbb{R} \) as
\[ K(x, p, w, y) = p^T X(x, w, 0) - y^T Y(x, w) + Z^T(x, w, 0)Z(x, w, 0) - \gamma^2 w^T w . \]

Since
\[ \frac{\partial^2}{\partial w^2} K(x, p, w, y) \bigg|_{(x, p, w, y) = (0, 0, 0, 0)} = 2(D_{11}^T D_{11} - \gamma^2 I) < 0 \]
\[ = 2Z^T_w(x, w, 0)Z_w(x, w, 0) \bigg|_{(x, p, w, y) = (0, 0, 0, 0)} - 2\gamma^2 I \]
due to Assumption A2, there exists a smooth function \( \hat{w}(x, p, y) \) in a neighborhood of \((0, 0, 0)\) such that
\[ \frac{\partial}{\partial w} K(x, p, w, y) \bigg|_{w = \hat{w}(x, p, y)} = 0 \quad \hat{w}(0, 0, 0) = 0 . \] (2.3.1)

Hence, locally
\[ \hat{w}(x, p, y) = -(2(D_{11}^T D_{11} - \gamma^2 I))^{-1}(2D_{11}^T C_1 x + B_1^T p - D_{21}^T y) , \] (2.3.2)
where \( C_1 = Z_x(x, w, u) \big|_{(0, 0, 0)} \) and \( B_1 = X_w(x, w, u) \big|_{(0, 0, 0)} \).

First order terms of a Taylor series expansion for (2.3.1) – valid in a neighborhood of \((0, 0, 0, 0)\):
\[ \frac{\partial}{\partial w} K(x, p, w, y) = \sum_i i^T \left( \frac{\partial}{\partial w} K(x, p, w, y) \right) \bigg|_{(0, 0, 0, 0)} = 0 \quad i = x, p, w, y \]

Hence,
\[
\hat{w}^T(x, p, y) = \left\{ \begin{array}{l}
\frac{d}{dx} \frac{d}{dp} K(x) \\
\frac{d}{dp} \frac{d}{dp} K(x) \\
\end{array} \right|_{(0, 0, 0)} + p^T \left( \frac{d}{dp} \frac{d}{dp} K(x) \right)_{(0, 0, 0)} + \\
\gamma T \left( \frac{d}{dy} \frac{d}{dw} K(x) \right) \left( \frac{d^2}{dw} K(x, p, w, y) \right)^{-1} \right|_{(0, 0, 0)}
\]

Moreover,
\[
\frac{d^2}{dy^2} K(x, p, \hat{w}(x, p, y), y) \bigg|_{(x, p, y) = (0, 0, 0)} = -\frac{1}{2} D_{21} (D_{11} D_{11} - \gamma^2 I)^{-1} D_{21} > 0
\]

\[
= \frac{d^2}{dy^2} (-y^T (x, \hat{w}) + Z^T (x, \hat{w}, 0) \dot{Z}(x, \hat{w}, 0) - \gamma^2 \hat{w}^T \hat{w}) \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
= \frac{d}{dy} (-\dot{Y}(x, \hat{w}) - \gamma \frac{d}{dy} \dot{Y}(x, \hat{w}, 0) \frac{d}{dy} \hat{w} + \frac{d}{dy} (Z^T) \frac{d}{dy} \dot{Z}(x, \hat{w}, 0) \frac{d}{dy} \hat{w} - \gamma^2 \frac{d}{dy} \dot{w} \frac{d}{dy} \hat{w}) \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
\frac{d}{dy} \hat{w}(x, p, y) \bigg|_{(x, p, y) = (0, 0, 0)} = (2(D_{11} D_{11} - \gamma^2 I)^{-1} D_{21}
\]

\[
\frac{d}{dy} (-\dot{Y}(x, \hat{w}) - \gamma \frac{d}{dy} \dot{Y}(x, \hat{w}, 0) \frac{d}{dy} \hat{w}) \bigg|_{(x, p, y) = (0, 0, 0)} = -\frac{1}{2} D_{21} (D_{11} D_{11} - \gamma^2 I)^{-1} D_{21}
\]

\[
\frac{d}{dy} \left( \frac{d^2}{dy^2} Z(x, \hat{w}, 0) \frac{d^2}{dy^2} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = 2 \left( \frac{d}{dy} Z(x, \hat{w}, 0) \frac{d}{dy} \hat{w} \right)^T \frac{d}{dy} \frac{d}{dy} Z(x, \hat{w}, 0) \frac{d}{dy} \hat{w} \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
= \frac{1}{2} D_{21} (D_{11} D_{11} - \gamma^2 I)^{-1} D_{21} (D_{11} D_{11} - \gamma^2 I)^{-1} D_{21}
\]

\[
\frac{d}{dy} \left( \frac{d^2}{dy^2} (\hat{w}^T \hat{w}) \frac{d}{dy} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = \gamma \frac{d}{dy} \left( \frac{d}{dy} (\hat{w}^T \hat{w}) \frac{d}{dy} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
= -\gamma^2 \frac{d}{dy} \left( \frac{d}{dy} \hat{w} \right)^T \frac{d}{dy} \hat{w} \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
= \frac{1}{2} \gamma^2 D_{21} (D_{11} D_{11} - \gamma^2 I)^{-1} (D_{11} D_{11} - \gamma^2 I)^{-1} D_{21}
\]
\[
\frac{\partial}{\partial y} \left( -Y(x, \hat{w}) - y^T \frac{\partial}{\partial w} Y(x, \hat{w}) \frac{\partial}{\partial y} + \frac{\partial}{\partial z} \left( Z^T(x, \hat{w}, 0)Z(x, \hat{w}, 0) \right) \frac{\partial}{\partial y} Z(x, \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - \gamma^2 \frac{\partial}{\partial w} \hat{w}^T \hat{w} \frac{\partial}{\partial y} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = 0
\]

Thus, there exists a smooth function \( y_s(x, p) \) in a neighborhood of \((0, 0)\) such that

\[
\frac{\partial}{\partial y} K(x, p, \hat{w}(x, p, y), y) \bigg|_{y = y_s(x, p)} = 0 \quad y_s(0, 0) = 0. \tag{2.3.3}
\]

Locally

\[
y_s(x, p) = -(D_{21}(D_{11}^TD_{11} - \gamma^2I)^{-1}D_{21}^T) \times (2(C_2 - D_{21}(D_{11}^TD_{11} - \gamma^2I)^{-1}D_{11}^T C_1)x - D_{21}(D_{11}^TD_{11} - \gamma^2I)^{-1}B_1^Tp)
\]

Again, derived from the first-order terms of a series expansion:

\[
\frac{\partial}{\partial dy} K(x, p, \hat{w}(x, p, y), y) \bigg|_{(x, p, y) = (0, 0, 0)} = 0, \quad i = x, p
\]

\[
\frac{\partial}{\partial dp} \left( p^T X(x, \hat{w}, 0) - y^T Y(x, \hat{w}) + Z^T(x, \hat{w}, 0)Z(x, \hat{w}, 0) - \gamma^2 \hat{w}^T \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = 0 + \left( \frac{\partial}{\partial d} X(x, \hat{w}, 0) \frac{\partial}{\partial d} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
\frac{\partial}{\partial dy} \hat{w}(x, p, y) \bigg|_{(x, p, y) = (0, 0, 0)} = (2(D_{11}^TD_{11} - \gamma^2I)^{-1}D_{21}^T)
\]

\[
\frac{\partial}{\partial dp} \left( p^T \frac{\partial}{\partial w} X(x, \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = B_1(2(D_{11}^TD_{11} - \gamma^2I)^{-1}D_{21}^T)
\]

\[
\frac{\partial}{\partial dx} \left( -Y(x, \hat{w}) - y^T \frac{\partial}{\partial w} Y(x, \hat{w}) \frac{\partial}{\partial x} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = \left( \frac{\partial}{\partial x} Y(x, \hat{w}) + \frac{\partial}{\partial w} Y(x, \hat{w}) \frac{\partial}{\partial x} \right) \bigg|_{(x, p, y) = (0, 0, 0)}
\]

\[
\frac{\partial}{\partial dx} \hat{w}(x, p, y) \bigg|_{(x, p, y) = (0, 0, 0)} = -(D_{11}^TD_{11} - \gamma^2I)^{-1}D_{11}^TC_1
\]

\[
\frac{\partial}{\partial dp} \left( -Y(x, \hat{w}) - y^T \frac{\partial}{\partial w} Y(x, \hat{w}) \frac{\partial}{\partial p} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = -C_1^T + C_1^T D_{11}^TD_{11} - \gamma^2I)^{-1}D_{21}^T
\]

\[
\frac{\partial}{\partial dp} \left( -Y(x, \hat{w}) - y^T \frac{\partial}{\partial w} Y(x, \hat{w}) \frac{\partial}{\partial p} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = \left( \frac{\partial}{\partial y} Y(x, \hat{w}) \frac{\partial}{\partial p} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)}
\]
\[
\frac{\partial}{\partial p} \mathbf{u}(x, p, y) \bigg|_{(x, p, y) = (0, 0, 0)} = -(2(D_1^T D_{11} - \gamma^2 I))^{-1} B_1^T \\
\frac{\partial}{\partial p} \left( -Y^T(x, \hat{w}) - Y^T(x, \hat{w}) \frac{\partial}{\partial y} Y(x, \hat{w}) \frac{\partial}{\partial y} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = B_1(2(D_1^T D_{11} - \gamma^2 I))^{-1} D_{21}^T
\]

\[
\frac{\partial}{\partial p} \left( Z^T(x, \hat{w}, 0) Z(x, \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = 0
\]

\[
\frac{\partial}{\partial p} \left( 2Z^T(x, \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} \right) \bigg|_{(x, p, y) = (0, 0, 0)} = 0
\]

\[
\frac{\partial}{\partial p} \left( \frac{\partial}{\partial x} \left( \frac{\partial}{\partial y} \hat{w} \right) \right) \bigg|_{(x, p, y) = (0, 0, 0)} = \gamma^2 C_1^T D_{11} (D_1^T D_{11} - \gamma^2 I)^{-1} (D_1^T D_{11} - \gamma^2 I)^{-1} D_{21}^T
\]

\[
\frac{\partial}{\partial p} \left( \frac{\partial}{\partial w} \left( \frac{\partial}{\partial y} \hat{w} \right) \right) \bigg|_{(x, p, y) = (0, 0, 0)} = \frac{1}{2} \gamma B_1 (D_1^T D_{11} - \gamma^2 I)^{-1} (D_1^T D_{11} - \gamma^2 I)^{-1} D_{21}^T
\]
$$
\frac{\partial}{\partial p} y(x, p, \hat{w}(x, p, y), y) \bigg|_{(x, p, y) = (0, 0, 0)} = 
= B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T + B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T +
- \frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{11}^T D_{11} (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T +\frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} (D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T
= B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T + B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T +
- \frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{11}^T D_{11} (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T +\frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} (D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T
= \frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T + \frac{1}{2} B_1 (2D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T - \frac{1}{2} B_1 (D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T
= \frac{1}{2} B_1 (D_{11}^T D_{11} - \gamma I)^{-1} D_{21}^T
$$

Define

$$w_{**}(x, p) = \hat{w}(x, p, y_*(x, p)).$$

By construction, for all \((x, p, w, y)\) in a neighborhood of \((0, 0, 0, 0)\),

$$K(x, p, w, y) \leq K(x, p, \hat{w}(x, p, y), y)$$

$$K(x, p, \hat{w}(x, p, y), y) \geq K(x, p, w_{**}(x, p), y_*(x, p)).$$

(2.3.5)

**Theorem 2.1:** Necessary condition for disturbance attenuation by output feedback

Consider the plant (2.1.1) for which Assumption A4 holds. Suppose the feedback law

$$\dot{\xi} = \eta(\xi, y) \quad u = \theta(\xi)$$

solves the problem of local disturbance attenuation. Let \(U(x, \xi)\) be a positive definite smooth function satisfying

$$U_\xi(x, \xi) U_\xi(x, \xi) + \left[ \begin{array}{c} X(x, w, \theta(\xi)) \\ \eta(\xi, Y(x, w)) \end{array} \right] + \|Z(x, w, \theta(\xi, y))\|_2 - \gamma^2 \|w\|_2 \leq 0$$

(2.3.6)

for all \((x, \xi, w)\) in a neighborhood of \((0, 0, 0, 0)\). Then the positive definite function \(W(x) = U(x, 0)\) satisfies

$$K(x, W_x^T(x), w_{**}(x, W_x^T(x)), y_*(x, W_x^T(x))) \leq 0$$

(2.3.7)

for each \(x\) in a neighborhood of zero.

**Proof:**

Setting \(\xi = 0\) in (2.3.6) yields

$$W_x(x)X(x, w, 0) + U_\xi(x, 0)\eta(0, Y(x, w)) + \|Z(x, w, 0)\|^2 - \gamma^2 \|w\|^2 \leq 0.$$
Since the function $\eta(0, y)$ vanishes at $y = 0$, a vector $P(x, y)$ of smooth functions exists such that

$$U_\xi(x, 0)\eta(0, y) = -PT(x, y)y$$

and therefore

$$W_x(x)X(x, w, 0) - PT(x, Y(x, w))Y(x, w) + \|Z(x, w, 0)\|^2 - \gamma^2\|w\|^2 \leq 0.$$ 

With $w = \hat{w}(x, W^T_x(x), y)$, this may be written as

$$K(x, W^T_x(x), \hat{w}(x, W^T_x(x), y), P(x, Y(x, \hat{w}(x, W^T_x(x), y)))) \leq 0.$$ 

Let $\tilde{y}(x)$ denote the solution of

$$\tilde{y}(x) = P(x, Y(x, \hat{w}(x, W^T_x(x), \tilde{y}(x))))$$

and set $y = \tilde{y}(x)$ in the previous inequality:

$$K(x, W^T_x(x), \hat{w}(x, W^T_x(x), \tilde{y}(x)), \tilde{y}(x)) \leq 0.$$ 

Expression (2.3.5) then shows that

$$K(x, W^T_x(x), \hat{w}(x, W^T_x(x), y_*(x, W^T_x(x))), y_*(x, W^T_x(x)))$$

$$= K(x, W^T_x(x), w_*(x, W^T_x(x)), y_*(x, W^T_x(x))) \leq 0.$$ 

2.3.2 The Output Injection Gain

**Theorem 2.2: Output feedback controller**

Consider the plant (2.1.1) and suppose that

i) Assumptions A1 to A5 hold,

ii) inequality (2.2.3) has a smooth solution $V(x) > 0 \ \forall x \neq 0, V(0) = 0$,

iii) the inequality

$$K(x, W^T_x(x), w_*(x, W^T_x(x)), y_*(x, W^T_x(x))) - H(x, V^T_x(x), v_*(x, V_x(x))) < 0$$

(2.3.8)

has a smooth solution $W(x) > 0 \ \forall x \neq 0, W(0) = 0$,

iv) $W_x(x) - V_x(x) > 0 \ \forall x \neq 0,$

v) the Hessian matrix of

$$K(x, W^T_x(x), w_*(x, W^T_x(x)), y_*(x, W^T_x(x))) - H(x, V^T_x(x), v_*(x, V_x(x)))$$

is nonsingular at $x = 0$, and the equation

$$(W(x) - V(x))G(x) = y^T(x, W^T_x(x))$$

(2.3.9)

has a smooth solution $G(x)$.

Then, the problem of local $L_2$ gain synthesis is solved by the output feedback

$$\dot{\xi} = X(\xi, w_*(\xi, V^T_\xi(\xi)), u_*(\xi, V^T_\xi(\xi))) + G(\xi)(y - Y(\xi, w_*(\xi, V^T_\xi(\xi))))$$

$$u = u_*(\xi, V^T_\xi(\xi)).$$
Proof:
Set \( Q(x) = W(x) - V(x) \) and define

\[
S(x, w) = Q_x(x)(X(x, w, 0) - G(x)Y(x, w)) + H(x, V^T_x(x), \begin{bmatrix} w \\ 0 \end{bmatrix}) - H(x, V^T_x(x), v_a(x, V^T_x(x))).
\]

\( S(x, w) \) satisfies

\[
S(x, w) = W_x(x)X(x, w, 0) - y^T_x(x, W^T_x(x))Y(x, w) - V_x(x)X(x, w, 0) +
\]

\[
+ H(x, V^T_x(x), \begin{bmatrix} w \\ 0 \end{bmatrix}) - H(x, V^T_x(x), v_a(x, V_x(x)))
\]

\[
= W_x(x)X(x, w, 0) - y^T_x(x, W^T_x(x))Y(x, w) + \|Z(x, w, 0)\|^2 - \gamma^2\|w\|^2 +
\]

\[
- H(x, V^T_x(x), v_a(x, V_x(x)))
\]

\[
= K(x, W^T_x(x), w, y_a(x, W^T_x(x))) - H(x, V^T_x(x), v_a(x, V_x(x)))
\]

\[
\leq K(x, W^T_x(x), w_a(x, W^T_x(x)), y_a(x, W^T_x(x))) - H(x, V^T_x(x), v_a(x, V_x(x)))
\]

\[
= x^T M(x) x
\]

where \( M(x) \) is a matrix of smooth functions which is negative definite at \( x = 0 \) (due to Assumptions iii and v of Theorem 2.2).

Define \( x^e = \begin{bmatrix} x \\ \xi \end{bmatrix} \) and

\[
U(x^e) = Q(x - \xi) + V(x).
\]

With this function it will be proven that the closed-loop system

\[
\begin{align*}
\dot{x}^e &= X^e(x^e, w) = \begin{bmatrix} X(x, w, u_a(\xi, V^T_\xi(\xi))) \\
X(\xi, w_a(\xi, V^T_\xi(\xi)), u_a(\xi, V^T_\xi(\xi))) + G(\xi)(Y(x, w) - Y(\xi, w_a(\xi, V^T_\xi(\xi)))) \end{bmatrix} \\
z &= Z^e(x^e, w) = Z(x, w, u_a(\xi, V^T_\xi(\xi)))
\end{align*}
\]

has the following two properties:

a) It satisfies the dissipativity condition

\[
U_{x^e}(x^e)X^e(x^e, w) + \|Z^e(x^e, w)\|^2 - \gamma^2\|w\|^2 < 0.
\]

b) It has a locally asymptotically stable equilibrium at \( x^e = 0 \).
For a, consider

\[ U_x(c, x^e)X^e(c, w) + \|Z^e(c, w)\|^2 - \gamma^2\|w\|^2 \]

\[ = Q_x(x - \xi) \left[ X(x, w, u_x(\xi, V_{\xi}^T(\xi))) - X(\xi, w_x(\xi, V_{\xi}^T(\xi)), u_x(\xi, V_{\xi}^T(\xi))) \right. \]

\[ - G(\xi)(Y(x, w) - Y(\xi, w_x(\xi, V_{\xi}^T(\xi)))) + V_x(x, w, u_x(\xi, V_{\xi}^T(\xi))) \]

\[ + \|Z(x, w, u_x(\xi, V_{\xi}^T(\xi)))\|^2 - \gamma^2\|w\|^2 \]

\[ = Q_x(x - \xi) \left[ X(x, w, u_x(\xi, V_{\xi}^T(\xi))) - X(\xi, w_x(\xi, V_{\xi}^T(\xi)), u_x(\xi, V_{\xi}^T(\xi))) \right. \]

\[ - G(\xi)(Y(x, w) - Y(\xi, w_x(\xi, V_{\xi}^T(\xi)))) + H(x, V_x(x), \left[ \begin{array}{c} w \\ u_x(\xi, V_{\xi}^T(\xi)) \end{array} \right] ) \]

\[ \leq Q_x(x - \xi) \left[ X(x, w, u_x(\xi, V_{\xi}^T(\xi))) - X(\xi, w_x(\xi, V_{\xi}^T(\xi)), u_x(\xi, V_{\xi}^T(\xi))) \right. \]

\[ - G(\xi)(Y(x, w) - Y(\xi, w_x(\xi, V_{\xi}^T(\xi)))) + H(x, V_x(x), \left[ \begin{array}{c} w \\ u_x(\xi, V_{\xi}^T(\xi)) \end{array} \right] ) \]

\[ \hspace{1cm} (2.3.10) \]

\[ - H(x, V_x^T(x), v_x(\xi, V_x(x))) \]

In a neighborhood of \((x, \xi) = (0, 0)\), a function \(\tilde{w}(x, \xi)\) exists such that

\[ L_w(x, \xi, w)|_{w = \tilde{w}(x, \xi)} = 0 \quad \tilde{w}(0, 0) = 0 \]

where \(L(x, \xi, w)\) is the right-hand side of inequality (2.3.10). Then

\[ L(x, \xi, w) \leq L(x, \xi, \tilde{w}(x, \xi)) \]

for all \((x, \xi, w)\) in a neighborhood of \((0, 0, 0)\), since the Hessian of \(L\) is negative definite:

\[ L_{ww}(x, \xi, w) = 2(D_{11}^T D_{11} - \gamma^2 I) < 0 \]

\[ = \frac{\partial^2}{\partial w^2}(Q_x(x - \xi)(X(x, w, u_x) - G(x)Y(x, w)) + \left. V_x(x, w, u_x) + (Z^T Z - \gamma^2 w^T w) \right|_{(x, \xi, w) = (0, 0, 0)} \]

\[ = \frac{\partial^2}{\partial w^2}(W_x(0), 0)(X(x, w, u_x) - G(x)Y(x, w)) + \left. V_x(0)(x, w, u_x) \right|_{(x, \xi, w) = (0, 0, 0)} + 2(D_{11}^T D_{11} - \gamma^2 I) \]

due to \(V_x(0)\) and \(W_x(0)\) being zero and due to Assumption A2. Thus, in this neighborhood,

\[ U_x(x^e)X^e(x^e, w) + \|Z^e(x^e, w)\|^2 - \gamma^2\|w\|^2 \leq L(x, \xi, \tilde{w}(x, \xi)), \]

and the function \(\tilde{w}(x, \xi)\) locally is given by

\[ \tilde{w}(x, \xi) = -\left( \frac{\partial^2}{\partial w^2}L(x, \xi, w) \right)^{-1}\left( x^T \frac{\partial}{\partial w^2}L(x, \xi, w) \right)_{(0, 0, 0)} + \]

\[ + x^T \left( \frac{\partial}{\partial \xi \partial w}L(x, \xi, w) \right)_{(0, 0, 0)} \]

\[ = -\frac{1}{2}(D_{11}^T D_{11} - \gamma^2 I)^{-1}((B_1^T - D_{11}^T G^T(0))Q_{xx}(0)(x - \xi) + \]

\[ + (B_1^T V_{xx}(0) + 2D_{11}^T C_1)x - 2D_{11}^T D_{12}F_2^T \xi) \]
Then, (2.3.11) holds since $\lambda_1, \lambda_2 > 0$.

In order to prove this, set $\xi = x - \xi$ and define

$$H(e, \xi) = L(x, \xi, \hat{w}(x, \xi)) \bigg|_{x = e + \xi}.$$ 

Then, (2.3.11) holds since $H(0, \xi) = 0$ and $H(e, \xi) \big|_{e = 0} = 0$.

$$H(0, \xi) = L(x, \xi, \hat{w}(x, \xi)) \big|_{x = \xi} = Q_{\xi}(0)[X(x, \xi, \hat{w}, u_\ast) - X(\xi, \hat{w}, u_\ast) - G(\xi)(Y(x, \hat{w}) - Y(\xi, \hat{w}))]_{x = \xi} + \left( H(x, V^T_\xi(x), [\xi]) - H(x, V^T_\xi(x), v_\ast(x, V_\xi(x))) \right)_{x = \xi} = 0$$

since $Q_{\xi}(0) = 0$ and

$$\hat{w}(\xi, \xi) = -\frac{1}{2}(D_{11}^T D_{11} - \gamma^2 I)^{-1}(B_{12}^T V_{xx}(0) + 2 D_{11}^T C_1)_{x = \xi}$$

$$= w_\ast(\xi, V^T_\xi(\xi))$$

in a neighborhood of $x = \xi = 0$.

$$H_e(e, \xi) \big|_{e = 0} = L_s(x, \xi, \hat{w}(x, \xi)) \big|_{x = \xi} = \left[ X(x, \xi, \hat{w}, u_\ast) - X(\xi, \hat{w}, u_\ast) + G(\xi)(Y(x, \hat{w}) - Y(\xi, \hat{w})) \right] Q_{xx}(x - \xi) + \left[ Q_{\lambda}(x - \xi)[X_s(x, \xi, \hat{w}, u_\ast) + G(\xi)Y_s(x, \hat{w})] + H_s(x, V^T_\xi(x), [\xi]) - H_s(x, V^T_\xi(x), v_\ast(x, V_\xi(x))) \right]_{x = \xi} = \left( H_u(x, V^T_\xi, [\xi]) u_\ast(x, V^T_\xi(x)) \right)_{x = \xi} = 0$$

since $H$ is minimized w.r.t. $u$ at $u_\ast$.

The matrix $R(x, \xi)$ consists of smooth functions and satisfies

$$R(0, 0) = M(0).$$
Thus

\[ R(0, 0) = \frac{1}{2} H_{ee}(e, \tilde{\xi}) \bigg|_{(0,0)} \]

\[ = \frac{1}{2} L_{xx}(x, \tilde{\xi}, \tilde{w}(x, \tilde{\xi})) \bigg|_{(0,0)} \]

\[ = \frac{1}{2} \left( (X_{x}(x, \tilde{w}, 0) - G(\tilde{\xi})Y_{x}(x, \tilde{w}))^{T} Q_{xx}(0) + Q_{xx}(0) [X_{x}(x, \tilde{w}, 0) - G(\tilde{\xi})Y_{x}(x, \tilde{w})] + \right. \]

\[ + H_{xx}(x, V_{x}^{T}(x)) \left[ \begin{array}{c} \tilde{w} \\ \end{array} \right] - H_{xx}(x, V_{x}^{T}, v_{x}) \bigg|_{(0,0)} \]

\[ = \frac{1}{2} (A + B_{1} \tilde{w}_{x} - G(0)(C_{2} + D_{21} \tilde{w}_{x}))^{T} Q_{xx}(0) + \frac{1}{2} Q_{xx}(0) (A + B_{1} \tilde{w}_{x} - G(0)(C_{2} + D_{21} \tilde{w}_{x})) + \]

\[ + \frac{1}{2} (A + B_{1} \tilde{w}_{x})^{T} V_{x}^{T}(x) + \frac{1}{2} V_{x}^{T}(x) (A + B_{1} \tilde{w}_{x}) + (C_{1} + D_{11} \tilde{w}_{x})^{T} (C_{1} + D_{11} \tilde{w}_{x}) - \gamma^{2} \tilde{w}_{x}^{T} \tilde{w}_{x} + \]

\[ - \frac{1}{2} H_{xx}(x, V_{x}^{T}, v_{x}) \bigg|_{(0,0)} \]

With \( G(0) \) written as (see remark on p. 18 below)

\[ G(0) = Q_{xx}^{-1}(0) y_{x}(0, 0) \]

\[ = -2 Q_{xx}^{-1}(0) \left( C_{2} - D_{21} R_{1}^{T} \left( D_{11}^{T} C_{1} + \frac{1}{2} B_{1}^{T} W_{x}(0) \right) \right)^{T} \left( D_{11}^{T} R_{1}^{-1} D_{21}^{T} \right)^{-1}, \]

we get

\[ \tilde{w}_{x}(0, 0) = -\frac{1}{2} R_{1}^{-1} \left( (B_{1}^{T} - D_{21}^{T} G(0)) Q_{xx}(0) + B_{1}^{T} V_{x}^{T}(0) + 2 D_{21}^{T} C_{1} \right) \]

\[ = -R_{1}^{-1} \left( \frac{1}{2} B_{1}^{T} Q_{xx}(0) + D_{21}^{T} (D_{21}^{T} R_{1}^{-1} D_{21}^{T})^{-1} \left( C_{2} - D_{21} R_{1}^{T} \left( D_{11}^{T} C_{1} + \frac{1}{2} B_{1}^{T} W_{x}(0) \right) \right) + \frac{1}{2} B_{1}^{T} V_{x}^{T}(0) + D_{11}^{T} C_{1} \right) \]

\[ = -R_{1}^{-1} \left( D_{11}^{T} C_{1} + \frac{1}{2} B_{1}^{T} W_{x}(0) + D_{21}^{T} (D_{21}^{T} R_{1}^{-1} D_{21}^{T})^{-1} \left( C_{2} - D_{21} R_{1}^{T} \left( D_{11}^{T} C_{1} + \frac{1}{2} B_{1}^{T} W_{x}(0) \right) \right) \right) \]

\[ = w_{xx}(0, 0). \]

Thus

\[ R(0, 0) = \]

\[ = \frac{1}{2} (A + B_{1} w_{xx})^{T} W_{x}(0) + \frac{1}{2} W_{x}(0) (A + B_{1} w_{xx}) + (C_{1} + D_{11} w_{xx})^{T} (C_{1} + D_{11} w_{xx}) - \gamma^{2} w_{xx}^{T} w_{xx} + \]

\[ - \frac{1}{2} (C_{2} + D_{21} w_{xx})^{T} y_{x}(0, 0) - \frac{1}{2} y_{x}(0, 0) (C_{2} + D_{21} w_{xx}) - \frac{1}{2} H_{xx}(x, V_{x}^{T}, v_{x}) \bigg|_{(0,0)} \]

\[ = M(0) \]

Hence, \( R(x, \tilde{\xi}) \) is negative definite in the neighborhood of \((0, 0)\), and property a follows.

In order to prove asymptotic stability (property b), set \( w = 0 \). Then we conclude stability from property a which yields
\[
\frac{d}{dt}U(x^e(t)) \leq -\|Z(x^e, 0)^2 \leq 0.
\]

Asymptotic stability can be shown by considering a trajectory \(x^e(t)\) which yields
\[
Z(x^e(t), 0) = Z(x(t), 0, u_*(\xi, V^T_\xi(\xi))) = 0 \text{ for all } t \geq 0. \tag{2.3.12}
\]

From Assumption A5, it follows that
\[
\lim_{t \to \infty} x(t) = 0
\]
and from A3 that the function \(u = u(x)\) which yields
\[
Z(x, 0, u(x)) = 0 \text{ and } u(0) = 0
\]
is unique. Hence, (2.3.12) implies that
\[
\lim_{t \to \infty} u_*(\xi(t), V^T_\xi(\xi(t))) = 0.
\]

Now, it is sufficient to show that
\[
\dot{x} = X(x, w_*(x, V^T_\chi(x)), 0) - G(x)Y(x, w_*(x, V^T_\chi(x))) \tag{2.3.13}
\]
is asymptotically stable. For this purpose, set \(w = w_*(x, V^T_\chi(x))\) in the definition of \(S(x, w)\). Then
\[
0 > S(x, w_*(x, V^T_\chi(x))) \geq Q_\chi(x)(X(x, w_*(x, V^T_\chi(x)), 0) - G(x)Y(x, w_*(x, V^T_\chi(x))))
\]
shows that \(Q(x)\) is a Lyapunov function for (2.3.13).

**Remarks:**

As pointed out in [13], the observer gain \(G(x)\) which is implicitly defined in (2.3.9) can be computed by extracting \(x^T\) from \(W_\chi(x) - V_\chi(x)\) and from \(y^T_\psi(x, W^T_\chi(x))\):
\[
G(x) = L^{-1}(x)R(x) \tag{2.3.14}
\]
where \(R(x)\) and \(L(x)\) are defined by
\[
W_\chi(x) - V_\chi(x) = x^T L(x)
\]
\[
y^T_\psi(x, W^T_\chi(x)) = x^T R(x).
\]

Note that \(V(x)\) is required to be positive definite rather than only nonnegative definite. Positive definiteness is needed to prove stability, but it is not necessary for establishing the norm bound to hold if the system is known to be stable (see Section 2.1.2). This is well known from linear \(H_\infty\) synthesis where the solution to one of the Riccati equations may be zero if the plant is stable (e.g., [6], chapter 7).

The two signals \(w_*(x, W^T_\chi(x))\) and \(y_*(x, W^T_\chi(x))\) can be interpreted as follows. As seen in Section 2.3.1, \(w_*(x, W^T_\chi(x))\) maximizes the Hamiltonian \(K\) while \(y_*(x, W^T_\chi(x))\) minimizes it. The interpretation therefore is similar to that for the state-feedback case: \(w_*(x, W^T_\chi(x))\) is the worst-case exogenous input which tries to increase the norm of the controlled variable \(z\). The control signal this time is not \(u\) but \(y_*(x, W^T_\chi(x))\), which stabilizes the plant and reduces the influence of \(w\) on \(z\) by output injection.
3 Approximated Controller

The Hamilton-Jacobi-Isaacs inequalities cannot be solved explicitly, in general. However, following the work of Lukes [15], it is possible to compute an approximate solution based on series expansions which can be used for finding an approximate control law. Lukes has done this for optimal regulators, and several authors have since pointed out that the same idea may be used for the $H_\infty$ problem (e.g., [13], [14], [16]).

The approximation of the lowest order $^1$ corresponds to the linear problem. It yields an approximation of first order for the control law and of second order for the storage function $V(x)$.

Notation: The series expansions for $v_*(x)$ and $V(x)$ are denoted as follows:

$$v_*(x) = v_*^{(1)}(x) + v_*^{(2)}(x) + \ldots = \sum_{d=1}^{\infty} v_*^{(d)}(x)$$

$$V(x) = V^{(2)}(x) + V^{(3)}(x) + \ldots = \sum_{d=1}^{\infty} V^{(d+1)}(x).$$

As pointed out in [16], each term in these expansions is of the form

$$c x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n} \text{ with } i_1 + i_2 + \ldots + i_n = d.$$

Denote the vector consisting of all possible terms of order $d$ (without coefficients) as $x^{(d)}$, i.e., $x^{(1)}$ and $x^{(2)}$ with $n = 3$ are given by

$$x^{(1)} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$$

$$x^{(2)} = \begin{bmatrix} x_1^2 & x_1 x_2 & x_1 x_3 & x_2^2 & x_2 x_3 & x_3^2 \end{bmatrix}^T.$$

Now, the scalar homogeneous function $V^{(d)}(x)$ of order $d$ can be written as

$$V^{(d)}(x) = C_V x^{(d)}$$

where $C_V$ is a row vector of the same length as $x^{(d)}$ containing the coefficients.

For derivatives, $V_x^{(d)}(x)$ stands for

$$V_x^{(d)}(x) = \frac{\partial}{\partial x} (V^{(d)}(x))$$

and thus is of order $d-1$ in $x$.

Similar approximations are used for the functions involved in the observer design.

---

1. The term “order” in connection with approximation does not refer to the dimension of the state vector but rather to the order of the Taylor approximation of each function. Sometimes, we use the expression “approximation order” to clearly distinguish this notion from the other.


3.1 State Feedback Controller

The two basic equations (2.2.3) and (2.2.2), where for the former the equality rather than the inequality is considered, may be written as:

\[
V(x) \ddot{X}(x, v_*(x, V(x))) + \dot{Z}(x, v_*(x, V(x))) \ddot{Z}(x, v_*(x, V(x))) + 
- v_*(x, V(x)) \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} v_*(x, V(x)) = 0
\]  

(3.1.1)

\[
\lambda^T \frac{\partial}{\partial x} \ddot{X}(x, v) + \frac{\partial}{\partial x} (\ddot{Z}(x, v) \ddot{Z}(x, v)) - 2v^T \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} \lambda = V(x), v = v_*(x, V(x))
\]  

(3.1.2)

Step-by-step approximations of the solutions to these two equations are given in the following subsections.

3.1.1 Lowest Order Approximation: The Linear $H_\infty$ Problem

For the linearized system

\[
\dot{x} = Ax + Bv
\]

\[
z = C_1 x + D_1 v
\]

with

\[
A = \left. \frac{\partial}{\partial x} \ddot{X}(x, v) \right|_{(0,0)} \quad B = \left. \frac{\partial}{\partial v} \ddot{X}(x, v) \right|_{(0,0)}
\]

\[
C_1 = \left. \frac{\partial}{\partial x} \ddot{Z}(x, v) \right|_{(0,0)} \quad D_1 = \left. \frac{\partial}{\partial v} \ddot{Z}(x, v) \right|_{(0,0)}
\]

the solution to the $H_\infty$ problem is well known [7]. The storage function $V(x)$ is given by

\[
V^{(2)}(x) = x^T K x
\]

and the optimum $v_*$ by

\[
v_*^{(1)} = -Fx = -R^{-1}(B^T K + D_1^T C_1) x
\]  

(3.1.3)

where $K$ is the solution to (3.1.1):

\[
2x^T K (A - BF)x + x^T C_1^T C_1 x - 2x^T C_1^T D_1 Fx + x^T F^T D_1^T D_1 Fx - \gamma^2 x^T F^T \begin{bmatrix} I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} Fx = 0.
\]

Since this equality must hold for all $x$, it can be written as the Riccati equation

\[
(A - BR^{-1} D_1^T C_1) K + K (A - BR^{-1} D_1^T C_1) - KBR^{-1} B^T K + Q = 0
\]  

(3.1.4)

with

\[
R = D_1^T D_1, \quad \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad Q = C_1^T C_1 - C_1^T D_1^T R^{-1} D_1^T C_1.
\]
3.1.2 Higher Order Approximations

With the linearized plant given above, the plant dynamics (2.2.1) may be rewritten as

\[ \dot{x} = Ax + Bv + f(x, v) \]

with the supply rate \(-s(w, z)\)

\[
T_z - v^T \begin{bmatrix} \gamma^2 I_m & 0 \\ 0 & 0 \end{bmatrix} v = x^T C_1^T C_1 x + 2x^T C_1^T D_1 v + v^T D_1^T D_1 v + g_1(x, v) - v^T \begin{bmatrix} \gamma^2 I_m & 0 \\ 0 & 0 \end{bmatrix} v
\]

\[ = x^T C_1^T C_1 x + 2x^T C_1^T D_1 v + v^T R v + g_1(x, v). \]

With these definitions of \(f\) and \(g_1\), with the feedback law (3.1.3), and with

\[ v_*(x) = v_*(x, V_1^T(x)), \]

the equations (3.1.1) and (3.1.2) read:

\[
V_x(x)((A - BF)x + B(v_*(x) + Fx) + f(x, v_*(x))) + x^T C_1^T C_1 x + 2x^T C_1^T D_1 v_*(x) + v_*(x)R v_*(x) + g_1(x, v_*(x)) = 0
\]

\[
2x^T C_1^T D_1 + 2v_*(x)R + \frac{\partial}{\partial v} g_1(x, v) \bigg|_{v = v_*(x)} + V_x(x) \left( B + \frac{\partial}{\partial v} f(x, v) \right) \bigg|_{v = v_*(x)} = 0.
\]

Solving the first of these equations for \(V_x(x)(A - BF)x\) and the second for \(v_*\) yields:

\[
V_x(x)(A - BF)x = - V_x(x)(B(v_*(x) + Fx) + f(x, v_*(x)) + x^T C_1^T C_1 x - 2x^T C_1^T D_1 v_*(x) - v_*(x)R v_*(x) - g_1(x, v_*(x)) \tag{3.1.5}
\]

\[
v_*(x) = -\frac{1}{2} R^{-1} \left( 2D_1^T C_1 x + \left( \frac{\partial}{\partial v} g_1(x, v) \bigg|_{v = v_*(x)} \right)^T + \left( B + \frac{\partial}{\partial v} f(x, v) \bigg|_{v = v_*(x)} \right) V_x^T(x) \right) \tag{3.1.6}
\]

Equation (3.1.5) written as a power series of terms up to order \(m\) is given by:

\[
\left( \sum_{d=1}^{m-1} V_x^{(d+1)}(x) \right) (A - BF)x =
\]

\[
- \sum_{d=1}^{m-2} \sum_{j=1}^{m-d} V_x^{(d+1)}(x) [B(v_*(x) + Fx) + f(x, v_*(x))]^{(j)} - \sum_{d=1}^{m} g_1^{(d)}(x, v_*(x)) +
\]

\[
-x^T C_1^T C_1 x - 2x^T C_1^T D_1 \sum_{d=1}^{m-1} v_*(d)(x) - \sum_{d=1}^{m-1} \sum_{j=1}^{m-d} v_*(d)(x) R v_*(j)(x) \quad m = 3, 4, \ldots
\]

and its \(m\)th-order term, i.e., the difference between the approximations of order \(m\) and \(m - 1\), by:

\[
V_x^{(m)}(x)(A - BF)x =
\]

\[
- \sum_{d=1}^{m-2} V_x^{(d+1)}(x) [B(v_*(x) + Fx) + f(x, v_*(x))]^{(m-d)} - g_1^{(m)}(x, v_*(x)) +
\]

\[
-2x^T C_1^T D_1 v_*(m-1)(x) - \sum_{d=1}^{m-1} v_*(d)(x) R v_*(m-d)(x) \quad m = 3, 4, \ldots \tag{3.1.7}
\]
Equation (3.1.6) as a power series of terms up to order \( k \) is given by:

\[
\sum_{d=1}^{k} v_*^{(d)}(x) = \frac{1}{2} R^{-1} \left( 2D_1^T C_1 x + \sum_{d=2}^{k} \left( \frac{\partial}{\partial v} g_1(x, v) \right)^T \right)_{v=v_*(x)} + B^T \left( \sum_{d=1}^{k} (V_x^{(d+1)}(x))^T \right) + \\
+ \sum_{d=1}^{k-1} \sum_{d=1}^{k-d} \left( \frac{\partial}{\partial v} f(x, v) \right)^T \left( (V^{(j+1)}(x))^T \right)_{v=v_*(x)} k = 2, 3, \ldots
\]

and its \( k \)th-order term:

\[
v_*^{(k)}(x) = \frac{1}{2} R^{-1} \left( \left( \frac{\partial}{\partial v} g_1(x, v) \right)^T \right)_{v=v_*(x)} + B^T (V_x^{(k+1)}(x))^T + \\
+ \sum_{d=1}^{k-1} \left( \frac{\partial}{\partial v} f(x, v) \right)^T \left( (V_x^{(k-d+1)}(x))^T \right)_{v=v_*(x)} k = 2, 3, \ldots
\]

Note that

\[
f^{(j)}(x, \sum_{d=1}^{\infty} v_*^{(d)}(x)) = f^{(j)}(x, \sum_{d=1}^{j-1} v_*^{(d)}(x)) \quad (3.1.9)
\]

\[
g^{(m)}_1(x, \sum_{d=1}^{\infty} v_*^{(d)}(x)) = g^{(m)}_1(x, \sum_{d=1}^{m-2} v_*^{(d)}(x)) \quad (3.1.10)
\]

since \( f(x, v) \) and \( g_1(x, v) \) are power series starting with terms in \((x, v)\) of order two and three, respectively (for \( f, v_* \) must be of higher order or a multiplication of \( x \) and \( v_* \) must occur because the linear term with \( v_* \) is already considered in \( Bv_* \)). Hence, equation (3.1.7) is independent of \( v_*^{(m-1)}(x) \), its coefficient being

\[-(V_x^{(2)}(x))B + 2x^TC_1^TD_{1*} + 2v_*^{(1)}(x)R = -2x^T(KB + C_1^TD_{1*} - F^TR) \]

\[= -2x^T(KB + C_1^TD_{1*} - (KB + C_1^T D_{1*}R^{-1}R) = 0. \]

Thus

\[
V_x^{(m)}(x)(A - BF)x = \\
- \sum_{d=2}^{m-2} V_x^{(d+1)}(x)Bv_*^{(m-d)}(x) - \sum_{d=1}^{m-2} V_x^{(d+1)}(x)f^{(m-d)}(x, v_*(x)) + \\
- g^{(m)}_1(x, v_*(x)) - \sum_{d=2}^{m-2} v_*^{(d)}(x)Rv_*^{(m-d)}(x) m = 3, 4, \ldots
\]

where the convention

\[
\sum_{k}^{l} = 0 \quad \text{for } l < k
\]

is adopted.
The right-hand side of (3.1.11) depends on the first \((m-2)\) terms of \(v_\ast(x)\) and on the first \((m-1)\) terms of \(V(x)\), while the right-hand side of (3.1.8) is determined by the first \((k-1)\) terms of \(v_\ast(x)\) and the first \((k+1)\) terms of \(V(x)\). Therefore, \(V(x)\) and \(v_\ast(x)\) can be approximated by consecutively computing

\[
V(2), v_\ast(1), V(3), v_\ast(2), V(4), v_\ast(3), \ldots
\]

This computation does not have to be done in the way its feasibility is derived. Since Taylor approximations have to be computed in every step anyway, the easiest way to compute \(V\) is the following:

\[
V_x^{(m)}(x)(A - BF)x = [- V_x(x)B v_\ast(x) - V_\lambda(x)f(x, v_\ast(x)) - g_1(x, v_\ast(x)) - v_\ast^T(x)R v_\ast(x)]^{(m)}
\]

\[m = 3, 4, \ldots\] (3.1.12)

Similarly, for \(v_\ast\):

\[
v_\ast^{(k)}(x) = -\frac{1}{2}R^{-1}\left[\left(\frac{\partial}{\partial v}g_1(x, v)\right)^T\right]_{v = v_\ast(x)} + B^TV_x^T(x) + \left(\frac{\partial}{\partial v}f(x, v)\right)^T_{v = v_\ast(x)} V_x^T(x)^{(k)}
\]

\[k = 2, 3, \ldots\] (3.1.13)

For both of these equations, \(V\) and \(v_\ast\) on the right-hand sides stand for the sums over all terms already computed.

### 3.2 Output Feedback Controller

As a result of the computations of the state feedback controller, \(v_\ast(x)\) and \(V(x)\) are available, and \(H(x, V_x^T(x), v_\ast(x), V_\lambda(x)))\) can be computed which is a prerequisite for the solution of the second Hamilton-Jacobi-Isaacs inequality (2.3.8). In order to ensure that (2.3.8) is a strict inequality, we must modify it by an additional term \(\Phi\):

\[
K(x, W_x^T(x), w_\ast(x, W_x^T(x)), y_\ast(x, W_x^T(x))) - H(x, V_x^T(x), v_\ast(x, V_\lambda(x))) + \Phi(x) = 0 \quad (3.2.1)
\]

where

\[
\Phi(x) > 0 \quad \forall x \neq 0
\]

\[
\Phi(0) = 0.
\]

For reasons which will become clear in Section 3.3, \(\Phi\) is chosen to be

\[
\Phi(x) = \varphi W_x^T(x) W_x(x)
\]

where \(\varphi > 0\). Moreover, condition iv of Theorem 2.2 can be enforced to hold by choosing \(\varphi\) sufficiently large.

Solving the second HJI equation is similar to solving the first one, since equation (3.2.1) has the same structure as (3.1.1):

\[
W_x(x)X(x, w_\ast(x, W_x^T(x)), 0) - y_\ast^T(x, W_x^T(x)) Y(x, w_\ast(x, W_x^T(x))) +
+ Z^T(x, w_\ast(x, W_x^T(x)), 0) Z(x, w_\ast(x, W_x^T(x)), 0) - \gamma^2 w_\ast^T(x, W_x^T(x)) w_\ast(x, W_x^T(x)) +
- H(x, V_x^T(x), v_\ast(x, V_\lambda(x))) + \Phi(x) = 0
\]

(3.2.2)
The expression corresponding to (3.1.2) now consists of two equations, (2.3.1) and (2.3.3):

\[
\frac{\partial}{\partial w} K(x, W^T_x(x), W, y) \bigg|_{w = \hat{\nu}(x, W^T_x(x), y)} = 0 = \\
\left( W_x(x) \frac{\partial}{\partial w} X(x, w, 0) - y^T \frac{\partial}{\partial w} Y(x, w) + 2Z^T(x, w, 0) \frac{\partial}{\partial w} Z(x, w, 0) - 2\gamma^2 w^T \right) \bigg|_{w = \hat{\nu}(x, W^T_x(x), y)}
\]

and

\[
\frac{\partial}{\partial y} K(x, W^T_x(x), \hat{\nu}(x, W^T_x(x), y), y) \bigg|_{y = y_*(x, W^T_x(x))} = 0 = \\
\left( W_x(x) \frac{\partial}{\partial \hat{\nu}} X(x, \hat{\nu}, 0) \frac{\partial}{\partial \hat{\nu}} \hat{\nu} - Y^T(x, \hat{\nu}) \frac{\partial}{\partial \hat{\nu}} Y(x, \hat{\nu}) \frac{\partial}{\partial y} \hat{\nu} + \\
+ 2Z^T(x, \hat{\nu}, 0) \frac{\partial}{\partial \hat{\nu}} Z(x, \hat{\nu}, 0) \frac{\partial}{\partial y} \hat{\nu} - 2\gamma^2 \hat{\nu} \frac{\partial}{\partial y} \hat{\nu} \right) \bigg|_{y = y_*(x, W^T_x(x))}
\]

### 3.2.1 Lowest Order Approximation

Equation (3.2.2) leads to the Riccati equation for the second HJI equation if it is approximated with second order terms. With the ansatz \( W^{(2)}(x) = x^T P x \), \( p = W^{(2)}_x(x) = 2P x \), it reads

\[
2x^T P (Ax + B_1 w) - y^T (C_2 x + D_{21} w) + (C_1 x + D_{11} w)^T (C_1 x + D_{11} w) + \\
- \gamma^2 w^T w - x^T h x + 4x^T P \phi P x = 0
\]

where \( x^T h x = H^{(2)}(x, V^T_x(x), v_*(x, V_x(x))) \) and

\[
B_1 = \frac{\partial}{\partial w} X(x, w, u) \bigg|_{(0, 0, 0)} \\
D_{11} = \frac{\partial}{\partial w} Z(x, w, u) \bigg|_{(0, 0, 0)} \\
C_2 = \frac{\partial}{\partial x} Y(x, w) \bigg|_{(0, 0)} \\
D_{21} = \frac{\partial}{\partial w} Y(x, w) \bigg|_{(0, 0)}
\]

With the shorthand notation

\[
R_1 = D^T_{11} D_{11} - \gamma^2 I
\]

and

\[
y_*(1)(x, 2Px) = -2(D_{21} R_1^{-1} D^T_{21})^{-1} ((C_2 - D_{21} R_1^{-1} D^T_{11} C_1) x - D_{21} R_1^{-1} B^T_1 P x),
\]

from (2.3.4), equation (2.3.2):

\[
\hat{\nu}^{(1)}(x, 2Px, y) = -R_1^{-1} \left( D^T_{11} C_1 x + B^T_1 P x - \frac{1}{2} D^T_{21} y \right)
\]

reads
such that equation (3.2.5) may be written as

$$0 = PA_e + A_e^T P + PR_e P + Q_e$$  \hspace{1cm} (3.2.6)

where

$$A_e = A - B_1 R_1^{-1} (D_{11}^T C_1 + D_{21}^T (D_{21} R_1^{-1} D_{21}^T)^{-1} (C_2 - D_{21} R_1^{-1} D_{11}^T C_1))$$

$$R_e = B_1 (- R_1^{-1} + R_1^{-1} D_{21}^T (D_{21} R_1^{-1} D_{21}^T)^{-1} D_{21} R_1^{-1} ) B_1^T + 4 \phi I_n$$

$$Q_e = (C_2 - D_{21} R_1^{-1} D_{11}^T C_1)^T (D_{21} R_1^{-1} D_{21}^T)^{-1} (C_2 - D_{21} R_1^{-1} D_{11}^T C_1) +$$

$$- C_1^T D_{11} R_1^{-1} D_{11}^T C_1 + C_1^T C_1 - h.$$  

Equation (3.2.6) is not yet the Riccati equation we need to solve because the solution $P$ would stabilize

$$A_e + R_e P$$

which is an expression for state feedback rather than output injection. As in the derivation of [10, Chapter 5] for the linear case, (3.2.6) must be multiplied by $P^{-1}$ from both sides in order to yield the Riccati equation (in $P^{-1}$) which is relevant for the observer problem. The Riccati equation to be solved is thus

$$0 = A_e P^{-1} + P^{-1} A_e^T + P^{-1} Q_e P^{-1} + R_e.$$  \hspace{1cm} (3.2.7)

Based on equation (2.3.14), the observer gain matrix for the linearized problem is then given by

$$G = - \frac{1}{2} (P - K)^{-1} [2 (D_{21} R_1^{-1} D_{21}^T)^{-1} (C_2 - D_{21} R_1^{-1} D_{11}^T C_1 - D_{21} R_1^{-1} B_1^T P)]^T$$

$$= - (P - K)^{-1} (C_2 - D_{21} R_1^{-1} (D_{11}^T C_1 + B_1^T P)) (D_{21} R_1^{-1} D_{21}^T)^{-1}.$$

### 3.2.2 Higher Order Approximations

With the previous definitions of $f$ and $g_1$ and with

$$w_{**}(x) = w_{**}(x, W_x^T(x)),$$

the equations (3.2.2), (3.2.3), and (3.2.4) read as follows:

$$W_x(x) \left( Ax + B_1 w_{**}^T(x) - GC_2 x + B_1 (w_{**}(x) - w_{**}^T(x)) + GC_2 x + f(x, \begin{bmatrix} w_{**}^T(x) \\ 0 \end{bmatrix}) \right) +$$

$$- y_{**}^T(x, W_x^T(x)) Y(x, w_{**}(x)) +$$

$$+ x^T C_1^T C_1 x + 2 x^T C_1^T D_{11} w_{**}(x) + w_{**}^T(x) R_1 w_{**}(x) + g_1(x, \begin{bmatrix} w_{**}^T(x) \\ 0 \end{bmatrix}) +$$

$$- H(x, V_x^T(x), v_{**}(x, V_x(x))) + \Phi(x) = 0$$
\[2x^T C_1^T D_{11} + 2 \hat{\omega}^T (x, W^T(x), y)R_1 + \left. \frac{\partial}{\partial w} g_1(x, \left[ \begin{array}{c} \hat{\omega} \\ 0 \end{array} \right] ) \right|_{w = \hat{\omega}(x, W^T(x), y)} +
-\left. y^T \left( \frac{\partial}{\partial w} Y(x, w) \right) \right|_{w = \hat{\omega}(x, W^T(x), y)} +
+ \left. W_x(x) \left( B_1 + \frac{\partial}{\partial w} f(x, \left[ \begin{array}{c} \hat{\omega} \\ 0 \end{array} \right] ) \right) \right|_{w = \hat{\omega}(x, W^T(x), y)} = 0 \]

Solving the first of these equations for \( W_x(x) (Ax + B_1 w^{(1)}(x) - GC_2 x) \) and the second for \( \hat{\omega} \) yields:

\[ W_x(x) (Ax + B_1 w^{(1)}(x) - GC_2 x) =
- W_x(x) \left( B_1 (w^{(0)}(x) - w^{(1)}(x)) + GC_2 x + f(x, \left[ \begin{array}{c} w^{(0)}(x) \\ 0 \end{array} \right] ) \right) +
+ y^T (x, W^T(x)) Y(x, w^{(0)}(x)) +
+ \left. x^T C_1^T D_{11} w^{(0)}(x) + w^T_x(x) R_1 w^{(0)}(x) + g_1(x, \left[ \begin{array}{c} w^{(0)}(x) \\ 0 \end{array} \right] ) \right) +
+ H(x, V^T_x(x), v_x(x, V_x(x))) - \Phi(x) \]

\[ \hat{\omega}(x, W^T(x), y) = - \frac{1}{2} R^{-1} \left( 2D_{11}^T C_1 x + \left. \left( \frac{\partial}{\partial w} g_1(x, \left[ \begin{array}{c} \hat{\omega} \\ 0 \end{array} \right] ) \right) \right|_{w = \hat{\omega}(x, W^T(x), y)} \right)^T +
- \left. \left( \frac{\partial}{\partial w} Y(x, w) \right) \right|_{w = \hat{\omega}(x, W^T(x), y)} \right)^T y +
+ \left. \left( B_1 + \frac{\partial}{\partial w} f(x, \left[ \begin{array}{c} \hat{\omega} \\ 0 \end{array} \right] ) \right) \right|_{w = \hat{\omega}(x, W^T(x), y)} \right)^T W^T_x(x) \]

In order to solve (3.2.8) for \( y_2 \), define \( g_2 \), \( g_3 \), and \( g_4 \) by

\[ Y(x, \hat{\omega}) = C_2 x + \frac{1}{2} D_{21}^T C_1^T D_{11} y + g_2(x, y) \]

\[ \frac{\partial}{\partial \hat{\omega}} Y(x, \hat{\omega}) \frac{\partial}{\partial y} \hat{\omega} = \frac{1}{2} D_{21}^T C_1^T D_{11} + g_3(x, \hat{\omega}) \]

\[ 2\hat{\omega}^T R_1 \frac{\partial}{\partial y} \hat{\omega} = \frac{1}{2} y^T D_{21}^T C_1^T D_{11} + g_4(x, y) \]

such that
The $m$th-order term of (3.2.9) is given by:

$$
0 = \left( W_x(x) \frac{\partial}{\partial \hat{w}} X(x, \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - x^T C_2^T - \frac{1}{2} y^T D_{21} R_1^{-1} D_{21}^T - g_2^T(x, y) + \\
y^T \left( \frac{1}{2} D_{21} R_1^{-1} D_{21}^T + g_3(x, \hat{w}) \right) + \\
+ \left( 2x^T C_1^T D_{11} + \frac{\partial}{\partial \hat{w}} s_1(x, \begin{bmatrix} \hat{s} \\ 0 \end{bmatrix}) \right) \frac{\partial}{\partial y} \hat{w} + \frac{1}{2} y^T D_{21} R_1^{-1} D_{21}^T + g_4(x, y) \right) \bigg|_{y = y(x, W^T_x(x))}
$$

and hence

$$
y^*(x, W^T_x(x)) = 2(D_{21} R_1^{-1} D_{21}^T)^{-1} \left( \frac{\partial}{\partial y} \hat{w} \right)^T W_x(x) \frac{\partial}{\partial \hat{w}} X(x, \hat{w}, 0) + \\
+ 2x^T C_1^T D_{11} + \frac{\partial}{\partial \hat{w}} s_1(x, \begin{bmatrix} \hat{s} \\ 0 \end{bmatrix}) \right)^T + \\
- C_2 x - g_2(x, y) - g_3^T(x, \hat{w}) y + g_4^T(x, y) \bigg|_{y = y(x, W^T_x(x))}
$$

The $m$th-order term of (3.2.9) is given by:

$$
W^{(m)}_x(x)(Ax + B_1 w^{(1)}_*(x) - GC_2 x) = \\
- \sum_{d=1}^{m-2} W^{(d+1)}_x(x) \left( B_1 \left( w^{(1)}_*(x) - w^{(1)}_*(x) \right) + G C_2 x + f(x, \begin{bmatrix} w^{(1)}_*(x) \\ 0 \end{bmatrix}) \right)^{(m-d)} + \\
+ \sum_{d=1}^{m-1} y^T_x(x) W^T_x(x) Y^{(m-d)}(x, w^{(1)}_*(x)) + \\
- \left( 2x^T C_1^T D_{11} w^{(m-1)}_*(x) + \sum_{d=1}^{m-1} w^T_x(d)(x) R_1 w^{(m-d)}_*(x) + g^{(m)}(x, \begin{bmatrix} w^{(1)}_*(x) \\ 0 \end{bmatrix}) \right) + \\
+ [H(x, V^T_x(x), v_*(x, V^T_x(x))) - \Phi(x)]^{(m)} \quad m = 3, 4, \ldots
$$
The $k$th-order terms of (3.2.10) and (3.2.11) thus read:

\[
\hat{w}(k)(x, W^T(x), y) = \frac{1}{2} R_1^{-1} \left( \left( \frac{\partial}{\partial w} g_1(x, \left[ \begin{array}{c} w \\ \hat{w} \end{array} \right]_{w = \hat{w}(x, W^T(x), y)} \right)^T \right)^{(k)} + \\
- \left[ \left( \frac{\partial}{\partial w} y(x, w) \right)_{w = \hat{w}(x, W^T(x), y)} \right]^T(y)^{(k)} + B_1^T(W_x^{(k+1)}(x))^T + \\
+ \sum_{d=1}^{k-1} \left( \frac{\partial}{\partial w} f(x, \left[ \begin{array}{c} w \\ \hat{w} \end{array} \right]_{w = \hat{w}(x, W^T(x), y)} \right)^{(d)}(W_x^{(k-d+1)}(x))^T \right) \right)_{k = 2, 3, \ldots}
\]  

(3.2.13)

\[
y^{(k)}(x, W_x^T(x)) = 2(D_{21}R_1^{-1}D_{21}^T)^{-1} \left[ \left( \frac{\partial}{\partial y} \hat{w} \right)^T \left( W_x(x) \frac{\partial}{\partial w} X(x, \hat{w}, 0) + \\
+ 2x^T C_1^T D_{11} + \frac{\partial}{\partial w} g_1(x, \left[ \begin{array}{c} w \\ \hat{w} \end{array} \right]_{w = \hat{w}(x, W^T(x), y)} \right)^T + \\
- g_2(x, y) - g_3^T(x, \hat{w})y + g_4^T(x, y) \right|_{y = y_*(x, W^T(x))} \right)^{(k)} \right) \right)_{k = 2, 3, \ldots}
\]  

(3.2.14)

The signal $w^{(k)}_*(x) = \hat{w}(k)(x, W^T(x), y_*(x, W^T(x)))$ can then be calculated from $y^{(k)}$.

Due to (3.1.9) and (3.1.10), equation (3.2.12) is independent of $w^{(m-1)}_*(x)$, its coefficient being

\[
-W_x^{(2)}(x)B_1 + y^{(1)}_T(x)D_{21} - 2x^T C_1^T D_{11} - 2w^{(1)}_T(x)R_1 =
\]

\[
= -2x^T \left( PB_1 + (C_2 - D_{21}R_1^{-1}D_{21}^T C_1 - D_{21}R_1^{-1}B_1^TP)T(D_{21}R_1^{-1}D_{21}^T)^{-1}D_{21} + C_1^T D_{11} + \\
- [D_{11}^T C_1 + B_1^T P + D_{21}^T (D_{21}R_1^{-1}D_{21}^T)^{-1}(C_2 - D_{21}R_1^{-1}D_{11}^T C_1 - D_{21}R_1^{-1}B_1^TP)T R_{11}^{-1} R_1] \right)
\]

= 0

Hence, (3.2.12) may be rewritten as follows:

\[
W_x^{(m)}(x)(Ax + B_1w_*(x) - GC_2x) =
\]

\[
- \sum_{d=2}^{m-2} W_x^{(d+1)}(x)B_1 W_*(x) - \sum_{d=1}^{m-2} W_x^{(d+1)}(x) f^{(m-d)}(x, \left[ \begin{array}{c} w_*(x) \\ 0 \end{array} \right]) + \\
+ \sum_{d=1}^{m-1} y_*^T(x, W_x^T(x)) Y^{(m-d)}(x, w_*(x)) + \\
- \sum_{d=2}^{m-2} w_*^T(x, W_x^T(x)) R_1 w_*(x) - g_1^{(m)}(x, \left[ \begin{array}{c} w_*(x) \\ 0 \end{array} \right]) + \\
+ [H(x, V_x^T(x), v_*(x, V_x(x)) - \Phi(x)](m) \right)_{m = 3, 4, \ldots}
\]  

(3.2.15)

The right-hand side of equation (3.2.13) depends on $W(x)$ of terms up to order $k+1$ in $x$ and on $\hat{w}(x, W^T(x), y)$ of terms up to order $k-1$. The right-hand side of (3.2.14) depends on the same terms.
as those of $W(x)$ and on terms up to order $k-1$ of $y_x$. Thus, $W(x), y_x(x, W^T_x(x)), \text{ and } w_{zs}(x)$ can be approximated by consecutively computing

\[
W^{(2)}, \hat{w}^{(1)}, y_x^{(1)}, w_{zs}^{(1)}
\]

\[
W^{(3)}, \hat{w}^{(2)}, y_x^{(2)}, w_{zs}^{(2)}
\]

\[
W^{(4)}, \hat{w}^{(3)}, y_x^{(3)}, w_{zs}^{(3)}
\]

\[
\ldots
\]

Again, this computation does not have to be done in the way its feasibility is derived, but rather on the basis of the following:

\[
W^{(m)}(x)(Ax + B_1 w_{zs}^{(1)}(x) - GC_2 x) =
\]

\[
\left[- W_x(x)B_1 w_{zs}(x) - W_x(x)f(x, w_{zs}(x), 0) + y_x^T(x, W^T_x(x))Y(x, w_{zs}(x)) +
\right.
\]

\[
-w_1 \hat{w}_1(x)R_1 w_{zs}(x) - g_1(x, \left[\begin{array}{c}
w_{zs}(x) \\ 0 \end{array}\right]) + H(x, V^T_x(x), v_x(x, V_x(x))) - \Phi(x)\right]^{(m)}
\]

\[
m = 3, 4, \ldots
\]

\[
\hat{w}^{(k)}(x, W^T_x(x), y) = \frac{1}{2} R_1^{-1}\left(\frac{\partial}{\partial w} g_1(x, \left[\begin{array}{c}
w \\ 0 \end{array}\right]) - y^T \frac{\partial}{\partial w} Y(x, w) +
\right.
\]

\[
+ W_x(x) \frac{\partial}{\partial w} X(x, w, 0)\left[\begin{array}{c}
(k) \\ w = \hat{w}(x, W^T_x(x), y) \end{array}\right]
\]

\[
k = 2, 3, \ldots
\]

For both of these equations, $W(x), y_x(x, W^T_x(x))$ and $w_{zs}(x)$ on the right-hand sides are the sums over all terms already computed. Equation (3.2.14) has already a form suitable for computation.

### 3.3 Alternative Output Feedback Controller

Instead of directly solving the equations derived in Section 2.3, the variables are changed first so that the equations can be given in terms of the multiplicator $p$ rather than in terms of $x$. The advantage of this change of variables is that the correct Riccati equation will be obtained directly. (Probably, all the equations could be directly derived in this form.) Thus

\[
x = \overline{W^T_p(p)} \text{ and } W^T_x(x) = p.
\]

As before, the HJI equation (2.3.8) is modified by an additional term $\Phi$ in order to ensure that it is a strict inequality:

\[
K(\overline{W^T_p(p)}, p, w_{zs}(\overline{W^T_p(p)}, p), y_x(\overline{W^T_p(p)}, p)) +
\]

\[
-H(x, V^T_x(x), v_x(x, V_x(x)))\bigg|_{x = \overline{W^T_p(p)}} + \Phi(p) = 0
\]

(3.3.1)

where

\[
\Phi(p) > 0 \quad \forall p \neq 0
\]

\[
\Phi(0) = 0.
\]
In fact, \( \Phi \) may be chosen to be

\[
\Phi(p) = p^T \phi
\]

where \( \phi \in \mathbb{R}^{n \times n} \) is a positive definite matrix. This explains why \( \Phi \) had been chosen that way in the previous subsection. As before, condition iv of Theorem 2.2 can be enforced to hold by choosing \( \Phi \) sufficiently large.

Solving the second HJI equation is similar to solving the first one since equation (3.3.1) has the same structure as (3.1.1):

\[
px(x, w_\ast(\overline{W}_p^T(p), p), 0) - y^T(\overline{W}_p^T(p), p)Y(x, w_\ast(\overline{W}_p^T(p), p)) + \\
+ ZT(\overline{W}_p^T(p), w_\ast(\overline{W}_p^T(p), p), 0)Z(\overline{W}_p^T(p), w_\ast(\overline{W}_p^T(p), p), 0) + \\
- \gamma^2w_\ast(\overline{W}_p^T(p), p)w_\ast(\overline{W}_p^T(p), p) + \\
- H(x, V_x^T(x), v_\ast(x, V_x(x))) \bigg|_{x = \overline{W}_p^T(p)} + \Phi(p) = 0
\]

The expression corresponding to (3.1.2) now consists of the two equations (2.3.1) and (2.3.3):

\[
\frac{\partial}{\partial w} K(\overline{W}_p^T(p), p, w, y) \bigg|_{w = \hat{w}(\overline{W}_p^T(p), p, y)} = 0 = \\
\left( p\frac{\partial}{\partial w} X(\overline{W}_p^T(p), w, 0) - y^T \frac{\partial}{\partial w} Y(\overline{W}_p^T(p), w) + \\
+ 2ZT(\overline{W}_p^T(p), w, 0) \frac{\partial}{\partial w} Z(\overline{W}_p^T(p), w, 0) - 2\gamma T (\right)
\]

(3.3.2)

\[
\frac{\partial}{\partial y} K(\overline{W}_p^T(p), p, \hat{w}(\overline{W}_p^T(p), p, y), y) \bigg|_{y = y_\ast(\overline{W}_p^T(p), p)} = 0 = \\
\left( p\frac{\partial}{\partial w} X(\overline{W}_p^T(p), \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - Y(\overline{W}_p^T(p), \hat{w}) - y^T \frac{\partial}{\partial w} Y(\overline{W}_p^T(p), \hat{w}) \frac{\partial}{\partial y} \hat{w} + \\
+ 2ZT(\overline{W}_p^T(p), \hat{w}, 0) \frac{\partial}{\partial w} Z(\overline{W}_p^T(p), \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - 2\gamma T (\right)
\]

(3.3.3)

The output injection gain \( \overline{G}(p) \) for the dynamic controller

\[
p = \overline{W}_{pp}^{-1}(p)[X(x, w_\ast(x, V_x^T(x)), u_\ast(x, V_x^T(x))), \overline{G}(p)(y - Y(x, w_\ast(x, V_x^T(x))))] \bigg|_{x = \overline{W}_p^T(p)}
\]

\[
u = u_\ast(x, V_x^T(x)) \bigg|_{x = \overline{W}_p^T(p)}
\]

now can be computed as

\[
\overline{G}(p) = \overline{L}^{-1}(p)\overline{R}(p)
\]

with \( \overline{L} \) and \( \overline{R} \) defined by

\[
p^T V_x(x) \bigg|_{x = \overline{W}_p^T(p)} = p^T \overline{L}(p)
\]

\[
y^T(\overline{W}_p^T(p), p) = p^T \overline{R}(p)
\]
3.3.1 Lowest Order Approximation

Equation (3.3.2) leads to the Riccati equation for the second HJI equation if it is approximated with second order terms. With the ansatz \( \tilde{W}^{(2)}(p) = p^T P p \), \( x = \tilde{W}_p^{(2)}(p) = 2Pp \), it reads

\[
p^T (2A PP + B_1 w) - y^T (2C_2 PP + D_{21} w) + (2C_1 Pp + D_{11} w)^T (2C_1 Pp + D_{11} w) + \gamma^2 w^T w - 4p^T PhPp + p^T \phi p = 0
\]

where \( x^T h x = H^{(2)}(x, V_x(x), v_*(x, V_x(x))) \).

With

\[
y_*(1)(2Pp, p) = -(D_{21} R_{11}^{-1} D_{21}^T)^{-1}(4(C_2 - D_{21} R_{11}^{-1} D_{11}^T C_1) Pp - D_{21} R_{11}^{-1} B_1^T p),
\]

from (2.3.4), equation (2.3.2):

\[
\hat{w}^{(1)}(2Pp, p, y) = -R_{11}^{-1} \left( 2D_{11}^T C_1 Pp + \frac{1}{2} B_1^T p - \frac{1}{2} D_{21}^T y \right)
\]

reads

\[
w^{(1)}_*(2Pp, p) = -R_{11}^{-1} \left( 2D_{11}^T C_1 Pp + \frac{1}{2} B_1^T p + D_{21}^T (D_{21} R_{11}^{-1} D_{21}^T)^{-1} \times \right.
\]

\[
\left. \left( 2(C_2 - D_{21} R_{11}^{-1} D_{11}^T C_1) Pp - \frac{1}{2} D_{21} R_{11}^{-1} B_1^T p \right) \right)
\]

such that equation (3.3.5) may be written as

\[
0 = PA_e^T + A_e P + P R_e P + Q_e
\]

(3.3.6)

where

\[
A_e = A - B_1 R_{11}^{-1} (D_{11}^T C_1 + D_{21}^T (D_{21} R_{11}^{-1} D_{21}^T)^{-1} (C_2 - D_{21} R_{11}^{-1} D_{11}^T C_1))
\]

\[
R_e = 4 \left( (C_2 - D_{21} R_{11}^{-1} D_{11}^T C_1)^T (D_{21} R_{11}^{-1} D_{21}^T)^{-1} (C_2 - D_{21} R_{11}^{-1} D_{11}^T C_1) + \right.
\]

\[
- C_1^T D_{11} R_{11}^{-1} D_{11}^T C_1 + C_1^T C_1 - h \right]
\]

\[
Q_e = \frac{1}{4} B_1 ( - R_{11}^{-1} + R_{11}^{-1} D_{21}^T (D_{21} R_{11}^{-1} D_{21}^T)^{-1} D_{21} R_{11}^{-1}) B_1^T + \phi.
\]

The observer gain matrix for the linearized problem then is given by

\[
G = (I - 4PK)^{-1} (B_1 R_{11}^{-1} D_{21}^T + 4P (C_1^T D_{11} R_{11}^{-1} D_{21}^T C_2 - C_2) (D_{21} R_{11}^{-1} D_{21}^T)^{-1}.
\]

3.3.2 Higher Order Approximations

With the previous definition of \( g_1 \) and with

\[
\dot{x} = Ax + \tilde{f}(x, w, u)
\]

\[
w^{(1)}_*(p) = w^{(1)}_*(\tilde{W}_p^T(p), p),
\]

the equations (3.3.2), (3.3.3), and (3.3.4) read as follows:
In order to solve (3.3.7) for \( \hat{w} \), define

\[
p^T [A \bar{W}_p^T(p) - GC_2 \bar{W}_p^T(p) + GC_2 \bar{W}_p^T(p) + \tilde{f}(\bar{W}_p^T(p), w_{**}(p), 0)] + \\
- y^T((\bar{W}_p^T(p), p)Y(\bar{W}_p^T(p), w_{**}(p)) + \\
+ \bar{W}_p(p)C_1^T C_1 \bar{W}_p^T(p) + \bar{W}_p^T(p)C_1^T D_{11} w_{**}(p) + w_{**}^T(p)R_1 w_{**}(p) + \\
+ g_1(p, \begin{bmatrix} w_{**}(p) \\ 0 \end{bmatrix}) - H(x, V_x^T(x), v_s(x, V_x(x)))) \bigg|_{x = \bar{W}_p^T(p)} + \Phi(p) = 0
\]

Solving the first of these equations for \( p^T (A - GC_2) \bar{W}_p^T(p) \) and the second for \( \hat{w} \) yields

\[
2 \bar{W}_p(p)C_1^T D_{11} + 2 \hat{w}^T(\bar{W}_p^T(p), p, y)R_1 + \frac{\partial}{\partial w}g_1(\bar{W}_p^T(p), \begin{bmatrix} \hat{w} \\ 0 \end{bmatrix}) \bigg|_{w = \hat{w}(\bar{W}_p^T(p), p, y)} + \\
- y^T \left( \frac{\partial}{\partial w}Y(\bar{W}_p^T(p), w) \right) \bigg|_{w = \hat{w}(\bar{W}_p^T(p), p, y)} + \\
+ p^T \left( \frac{\partial}{\partial w} \tilde{f}(\bar{W}_p^T(p), w, 0) \right) \bigg|_{w = \hat{w}(\bar{W}_p^T(p), p, y)} = 0
\]

\[
\left( p^T \frac{\partial}{\partial w} X(\bar{W}_p^T(p), \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - Y^T(\bar{W}_p^T(p), \hat{w}) - y^T \frac{\partial}{\partial w} Y(\bar{W}_p^T(p), \hat{w}) \frac{\partial}{\partial y} \hat{w} + \\
+ (2 \bar{W}_p(p)C_1^T D_{11} + 2 \hat{w}^T R_1 + \frac{\partial}{\partial w} g_1(\bar{W}_p^T(p), \begin{bmatrix} \hat{w} \\ 0 \end{bmatrix}) \frac{\partial}{\partial y} \hat{w}) \right) \bigg|_{y = y_s(\bar{W}_p^T(p), p)} = 0.
\]

In order to solve (3.3.7) for \( y_s \), define \( g_2 \), \( g_3 \), and \( g_4 \) as

\[
Y(x, \hat{w}) = C_2 x + \frac{1}{2} D_{21} R_1^{-1} D_{21}^T y + g_2(x, y)
\]
\[
\frac{\partial}{\partial \hat{w}} Y(x, \hat{w}) \frac{\partial}{\partial y} \hat{w} = \frac{1}{2} D_{21} R_{1}^{-1} D_{21}^T + g_3(x, \hat{w})
\]

\[
2\hat{w}^T R_1 \frac{\partial}{\partial y} \hat{w} = \frac{1}{2} y^T D_{21} R_{1}^{-1} D_{21}^T + g_4(x, y)
\]
such that

\[
0 = \left( p^T \frac{\partial}{\partial \hat{w}} X(\hat{W}_p^T(p), \hat{w}, 0) \frac{\partial}{\partial y} \hat{w} - \hat{W}_p(p) C_1^T - \frac{1}{2} y^T D_{21} R_{1}^{-1} D_{21}^T - g_2(\hat{W}_p^T(p), y) \right) + \nonumber
\]

\[
- y^T \left( \frac{1}{2} D_{21} R_{1}^{-1} D_{21}^T + g_3(\hat{W}_p^T(p), \hat{w}) \right) + \nonumber
\]

\[
+ \left( 2 \hat{W}_p(p) C_1 D_{11} + \frac{\partial}{\partial \hat{w}} g_1(\hat{W}_p^T(p), \hat{w}) \right) \frac{\partial}{\partial y} \hat{w} + \nonumber
\]

\[
+ \frac{1}{2} y^T D_{21} R_{1}^{-1} D_{21}^T + g_4(\hat{W}_p^T(p), y) \right) \bigg|_{y = y_*(\hat{W}_p^T(p), p)}
\]

and hence

\[
y_*(\hat{W}_p^T(p), p) = 2(D_{21} R_{1}^{-1} D_{21}^T)^{-1} \left( \frac{\partial}{\partial \hat{w}} \hat{w} \right)^T \left[ p^T \frac{\partial}{\partial \hat{w}} X(\hat{W}_p^T(p), \hat{w}, 0) + \nonumber
\right.
\]

\[
+ 2 \hat{W}_p(p) C_1 D_{11} + \frac{\partial}{\partial \hat{w}} g_1(\hat{W}_p^T(p), \hat{w}) \right]^T - C_2 \hat{W}_p^T(p) + \nonumber
\]

\[
- g_2(\hat{W}_p^T(p), y) - g_3(\hat{W}_p^T(p), \hat{w}) y + g_4(\hat{W}_p^T(p), y) \bigg|_{y = y_*(\hat{W}_p^T(p), p)}
\]

The \( m \)-th order term of (3.3.8) is thus given by:

\[
(p^T A - p^T G C_2) \hat{W}_p^T(m)(p) =
\]

\[
- p^T \left( G C_2 \hat{W}_p^T(p) + \tilde{f}(\hat{W}_p^T(p), w_*(p), 0) \right)^{(m-1)} + \nonumber
\]

\[
+ \sum_{d=1}^{m-1} y_*(d)(\hat{W}_p^T(p), p) Y(m-d)(\hat{W}_p^T(p), w_*(p)) + \nonumber
\]

\[
- \sum_{d=1}^{m-1} \hat{W}_p(d)(p) C_1 C_1 \hat{W}_p^T(m-d)(p) - \sum_{d=1}^{m-1} 2 \hat{W}_p(d)(p) C_1 D_{11} w_*(m-d)(p) + \nonumber
\]

\[
(3.3.11)
\]

\[
- \sum_{d=1}^{m-1} w_*(d)(p) R_1 w_*(m-d)(p) - g_1(m)(\hat{W}_p^T(p), \left[ \begin{array}{c} w_*(p) \\ 0 \end{array} \right]) + \nonumber
\]

\[
+ \left( H(x, V_x^T(x), v_*(x), V_x(x)) \bigg|_{x = \hat{W}_p(p)} - \Phi(p) \right)^{(m)}
\]

\[
m = 3, 4, \ldots
\]
The \( k \)-th order terms of (3.3.9) and (3.3.10) read:

\[
\hat{w}^{(k)}(\overline{W}_p^T(p), p, y) = \frac{1}{2} R_1^{-1} \left( 2 D_{11}^T C_1 \overline{W}_p^{(k+1)}(p) + \\
+ \left( \frac{\partial}{\partial w} g_1(\overline{W}_p^T(p), [s]^T) \bigg|_{w = \hat{w}(\overline{W}_p^T(p), p, y)} \right)^T(k) + \\
- \left( \frac{\partial}{\partial w} Y(\overline{W}_p^T(p), w) \bigg|_{w = \hat{w}(\overline{W}_p^T(p), p, y)} \right)^T(k) y + \\
+ \left( \frac{\partial}{\partial w} \hat{f}(\overline{W}_p^T(p), w, 0) \bigg|_{w = \hat{w}(\overline{W}_p^T(p), p, y)} \right)^T(k-1) p \right)
\]

\( k = 2, 3, \ldots \)  

(3.3.12)

\[
y^{(k)}_s(\overline{W}_p^T(p), p) = 2(D_{21} R_1^{-1} D_{21}^T)^{-1} \left( \left( \frac{\partial}{\partial y} \hat{w} \right)^T \left[ p^T \frac{\partial}{\partial w} X(\overline{W}_p^T(p), \hat{w}, 0) \right] + \\
+ 2 \overline{W}_p(p) C_1^T D_{11} + \frac{\partial}{\partial w} g_1(\overline{W}_p^T(p), [s]^T) \right)^T - C_2 \overline{W}_p^T(p) - g_2(\overline{W}_p^T(p), y) + \\
- g_3^T(\overline{W}_p^T(p), \hat{w}) y + g_4^T(\overline{W}_p^T(p), y) \bigg|_{y = y_s(\overline{W}_p^T(p), p)} \right)^{(k)}
\]

\( k = 2, 3, \ldots \)  

(3.3.13)

The signal \( w^{(k)}_s(p) = \hat{w}^{(k)}(\overline{W}_p^T(p), p, y_s(\overline{W}_p^T(p), p)) \) can then be calculated.

Similar to (3.1.9), (3.1.10),

\[
\left( p^T \hat{f}(\sum_{d=1}^{\infty} \overline{W}_p^{T(d)}(p), \sum_{d=1}^{\infty} w^{(d)}_s(p), 0) \right)^{(m)} = p^T \hat{f}^{(m-1)}(\sum_{d=1}^{m-1} \overline{W}_p^{T(d)}(p), \sum_{d=1}^{m-1} w^{(d)}_s(p), 0),
\]

\[
g^{(m)}_1(\sum_{d=1}^{\infty} \overline{W}_p^{T(d)}(p), \sum_{d=1}^{\infty} w^{(d)}_s(p)) = g^{(m)}_1(\sum_{d=2}^{m-1} \overline{W}_p^{T(d)}(p), \sum_{d=1}^{m-2} w^{(d)}_s(p)),
\]

Hence, equation (3.3.11) is independent of \( w^{(m-1)}_s(p) \), its coefficient being

\[
-p^T B_1 + y^{(1)}_s(\overline{W}_p^T(p), p) D_{21} - 2 \overline{W}_p(p) C_1^T D_{11} - 2 w^{(1)}_s(p) R_1 =
\]

\[
= p^T \left(- B_1 - (4(C_2 - D_{21} R_1^{-1} D_{11}^T C_1) P - D_{21} R_1^{-1} B_1^T)(D_{21} R_1^{-1} D_{21}^T)^{-1} D_{21} + \\
- 4 P C_1^T D_{11} + 2 \left(2 D_{11} C_1 P + \frac{1}{2} B_1^T + D_{21}^T (D_{21} R_1^{-1} D_{21}^T)^{-1} \times \\
\left(2(C_2 - D_{21} R_1^{-1} D_{11}^T C_1) P - \frac{1}{2} D_{21} R_1^{-1} B_1^T \right) \right)^T R_1 \right)
\]

\[= 0\]
The right-hand side of equation (3.3.12) depends on $W(p)$ of terms up to order $k+1$ in $p$ and on $\dot{w}(\overline{W}_p^T(p), p, y)$ of terms up to order $k-1$. The right-hand side of (3.3.13) depends on the same terms of $\overline{W}(p)$ and on terms up to order $k-1$ of $y_*$. Thus, $\overline{W}(p)$, $y_*(\overline{W}_p^T(p), p)$, and $w_{**}(p)$ can be approximated by consecutively computing

\[
\overline{W}^{(2)}, \dot{w}^{(1)}, y_*^{(1)}, w_{**}^{(1)}
\]
\[
\overline{W}^{(3)}, \dot{w}^{(2)}, y_*^{(2)}, w_{**}^{(2)}
\]
\[
\overline{W}^{(4)}, \dot{w}^{(3)}, y_*^{(3)}, w_{**}^{(3)}
\]
...

As before, this computation does not have to be done in the way its feasibility is derived, but rather as follows:

\[
(p^TA - p^TGC_2)\overline{W}_p^{T(m)}(p) = \\
\left[- p^TGC_2 \overline{W}_p^T(p) - p^T\tilde{f}(\overline{W}_p^T(p), w_{**}(p), 0) + y_p^T(\overline{W}_p^T(p), p)Y(\overline{W}_p^T(p), w_{**}(p)) + \\
w_*^T(p)R_1 w_{**}(p) - \overline{W}_p(p)C_1^T C_1 \overline{W}_p^T(p) - g_1(\overline{W}_p^T(p), \left[ \begin{array}{c} w_{**}(p) \\ 0 \end{array} \right]) + \\
H(x, V_x^T(x), v_*(x, V_x(x))) \bigg|_{x = \overline{W}_p^T(p)} - \Phi(p) \bigg]^{(m)}
\]

\[
\dot{w}^{(k)}(x, W_x^T(x), y) = -\frac{1}{2}R_1^{-1}\left(2x^T C_1^T D_1 + \frac{\partial}{\partial w}(g_1(x, \left[ \begin{array}{c} y_* \end{array} \right])) - y_*^T \frac{\partial}{\partial w}Y(x, w) + \\
p^T \frac{\partial}{\partial w} \tilde{f}(x, w, 0) \right)^T\bigg|_{x = \overline{W}_p^T(p), w = \dot{w}(\overline{W}_p^T(p), p, y)}^{(k)} \quad k = 2, 3, \ldots
\]

For both of these equations, $\overline{W}(p)$, $y_*(\overline{W}_p^T(p), p)$, and $w_{**}(p)$ on the right-hand sides are the sums over all terms already computed. Equation (3.3.13) already has a form suitable for computation.
4 Listings for Maple V

This chapter contains the Maple V code which implements the algorithms developed above. The first three listings implement the state feedback and the two approaches for the output feedback controllers. They all invoke the subroutines contained in Section 4.4 to solve the algebraic Riccati equation and to improve the numerics by balancing the controller.

The last section lists the code for converting the controllers to C coded s-functions for use with SIMULINK 1.3 and 2.0 (mex-files). This allows for simulations in SIMULINK and for implementation on DSPs such as, e.g., the dSPACE systems by using MATLAB’s Real Time Workshop. Two versions of these conversions are available. The first one writes code for general state space systems which hence do not necessarily have to be controllers. The second one allows for coding output feedback controllers even if the inversion involved in the computation of the output injection gain (equation (2.3.14)) cannot be computed symbolically due to expressions which contain too many terms. The inversion is carried out on line by the code which is included in this last section as well.

4.1 hinfSF: State Feedback

The code is written for Maple V Release 2 (except for the state feedback controller, which is written for Release 4) and should also run with Release 3. The routines will be updated to Release 4 as soon as it becomes available at the Washington University’s Department of Systems Science and Mathematics. The routines which write the mex files currently depend on the UNIX operating system.

Why Maple? For conveniently dealing with nonlinear systems, a program for symbolic mathematical computations must be used, even if the goal is to solve the problem numerically. At ETH, there are basically two such programs available: Maple and Mathematica. We preferred Maple since it is somewhat integrated in MATLAB (Symbolic Math Toolbox). If it were completely integrated, MATLAB could be used for the numerical calculations while the commands of Maple would be used for manipulating the equations.

The listings contain a section providing help on their usage which eventually will be accessible by Maple’s help command (only in Release 4 versions). An application of the procedures can be found in Chapter 5.

```maple
hinfSF := proc()  
    local eps, nargout, gamma, apxord, var, G, x, w, u, v, xv, X, Z, n, m, m2, m1, i, j, pass, A, Ae, B, C1, D1dot, R, Q, F, res, ham, vstar, gl, V, grad_V, xk, index, grad_gl, jacob_f, RHS, k, eq_V, A_V, ord, K, Tzw, wstar, x_list, indets_eq_V, to_be_assigned, tmp;  
    description  
        'FUNCTION: hinfSF',  
        '  numerical solution to the nonlinear state-feedback H_inf problem',  
        ' ',  
        'CALLING SEQUENCE:',  
        ' hinfSF(G, var, apxord, gamma, nargout, eps)',  
        ' ',  
        'PARAMETERS:',  
        ' G       - generalized plant \([X(x,w,u)], [Z(x,w,u)]\) where',  
        ' x_dot = X(x,w,u)',  
        ' z = Z(x,w,u)',  
        ' X and Z must be of type vector',  
        ' var - names of the variables used in G: var := [[x], [w], [u]]',  
        ' where x, w, and u must be of type vector',  
        ' apxord - approximation order for the controller (default: 3)',  
        ' gamma - upper bound for the L2-gain (default: 1)',  
        ' nargout - number of output arguments (default: 1) -- see Synopsis',  
        ' eps - tolerance for zero (default: 1e-10)',
```
`SYNOPSIS',
  - Computes the solution to the nonlinear L2-gain problem (H_inf problem).
  - Computes the solution to the nonlinear L2-gain problem (H_inf problem).
  - Computes the solution to the nonlinear L2-gain problem (H_inf problem).

  \[ \|Z(x, w, u)\|_2 / \|w\|_2 < \gamma \]

  If nargout := 3, the output of hinfSF is a list containing [K, Tzw, wstar] where K is the controller, Tzw is the closed-loop dynamics, and wstar is the worst-case excitation of the system.

  If nargout := 1 (default), only K is returned.

  # not optimized for speed! does not use evalhf
  # Urs Christen, Jan 1997

if nargs < 6 then
  eps := 1.0e-10;
elif not(type(args[6], numeric)) then
  ERROR(`'eps' must be a numerical value.');
else
  eps := args[6];
fi;

if nargs < 5 then
  nargout := 1;
elif not(type(args[5], integer)) or args[5] > 3 or args[5] <= 0 then
  ERROR(`'nargout' must be a positive integer <= 3.');
else
  nargout := args[5];
fi;

if nargs < 4 then
  gamma := 1.0;
elif not(type(args[4], numeric)) or args[4] <= 0 then
  ERROR(`'gamma' must be a positive number.');
else
  gamma := args[4];
fi;

if nargs < 3 then
  apxord := 3;
elif not(type(args[3], integer)) or args[3] <= 0 then
  ERROR(`'apxord' must be a positive integer <= 3.');
else
  apxord := args[3];
fi;

if nargs < 2 then
  ERROR(`not enough arguments!');
elif not(nops(args[2]) = 3) then
  ERROR(`'var' must consist of the three parts \[x, w, u\]');
elif convert([seq(not(type(op(i, args[2])), vector), i = 1..3)], 'or') then
  ERROR(`'var' must consist of three vectors.');
else
  var := args[2];
  G := args[1];
fi;

x := op(1, var);
4.2 hinfOF_x: Output Feedback

hinfOF_x := proc() ...
    # description
    "FUNCTION: hinfOF_x;",
    "numerical solution to the nonlinear output-feedback H_inf problem;",
    "CALLING SEQUENCE:",
    "hinfOF_x(G, var, apxord, gamma, nargout, eps, psi, phi, ordflag, filename);",
    "PARAMETERS:",
    "G - generalized plant [[X(x,w,u)], [Z(x,w,u)], [Y(x,w)]] where',
    "x = X(x,w,u),'
    "z = Z(x,w,u),'
    "u = U(x,w,u)';
    "var - names of the variables used in G: var := [[x], [w], [u]]',
    "where x, w, and u must be of type vector';

    local Phi, eps, nargout, gamma, apxord, var, balflag, fileflag, filename, 
    ordflag, 
    extract, tmp, G, x, w, u, v, xv, X, Y, R, indets_eq_VW,
    to_be_assigned, 
    n, m1, m2, m, pl, p2, l, j, pass, A, Ae, B, Cl, Dl, Dl1, Dl2, R, Q, F, res, Psi, 
    ham, vstar, gl, V, grad_V, xkk, index, grad_gl, f, jacob_f, k, c_VW, 
    eq_VW, 
    ord, usstar, wstar, Xstar, Zstar, Hstar, phi, x_list, K, 
    B1, D1l, D2l, D2l1, D2l2, C2, eym1, R1, R1l, DRD, DRD1, hstar, P, 
    ystar, y, 
    w_hat, wstarstar_1, W, grad_W, jacob_X, wstarstar, grad_Y, jacob_Y, 
    WXg, 
    g2, jacob_w, jacob_w_hat, g3, g4, Ystar, L, dx, xi, Kxi, Tzw, threshold, 
    T, x, 
    ...
end:

4.2 Listings for Maple V384.2
# apxord - approximation order for the controller (default: 3),
# gamma - upper bound for the L2-gain (default: 1),
# nargout - number of output arguments (default: 1) -- see Synopsis',
# eps - tolerance for zero (default: 1e-10) -- not utilized yet!,
# psi - additional term for the first HJI to ensure its solution being',
# phi - as psi but for 2nd HJI (default: 1e-8),
# ordflag - if true (default), the controller has terms up to order apxord',
# balflag - if true, the linear approximation of the controller is balanced;,
# filename - if specified, a Simulink mex-file containing code for the',
#           controller is written with the name filename (must be a string);,
# in this case, nothing is returned by hinfOF_x',
# SYNOPSIS:,
# Computes the solution to the nonlinear L2-gain problem (H_inf problem),
# \[ \|Z(x, w, u)\|_2 / \|w\|_2 < \gamma \],
# - If nargout := 2, the output of hinfOF_x is a list containing',
#   \([K, \tau_{zw}]\) where K is the controller and \(\tau_{zw}\) is the closed-loop',
#   dynamics.',
# - If nargout := 1 (default), only K is returned.',
# - If a filename is specified, the inversion necessary for the computation',
#   of the output injection gain is not carried out symbolically. Instead, code',
#   is generated which will compute this inversion numerically on-line.',
# Urs Christen, May 1997

if nargs < 11 then
  fileflag := false;
elif not(type(args[11], string)) then
  ERROR(`'filename' must be a string.'
else
  fileflag := true;
  filename := args[11];
if nargs < 10 then
  balflag := false;
elif not(type(args[10], boolean)) then
  ERROR(`'balflag' must be a boolean.'
else
  balflag := args[10];
if nargs < 9 then
  ordflag := true;
elif not(type(args[9], boolean)) then
  ERROR(`'ordflag' must be a boolean.'
else
  ordflag := args[9];
fi;
if nargs < 8 then
  Phi := 1.0e-8;
elif not(type(args[8], scalar) or type(args[8], numeric)) then
  ERROR(`'Phi' must be a scalar or a numerical value.'
else
  Phi := args[8];
fi;
if nargs < 7 then
  Psi := 0.1;
elif not(type(args[7], scalar) or type(args[7], numeric)) then
  ERROR(`'Psi' must be a scalar or a numerical value.'
else
  Psi := args[7];
fi;
if nargs < 6 then
  eps := 1.0e-10;
elif not(type(args[6], numeric)) then
  ERROR(`'eps' must be a numerical value.'
else
  eps := args[6];
fi;
if nargs < 5 then
  nargout := 1;
elif not(type(args[5], integer)) or args[5] > 3 or args[5] <= 0 then
  ERROR(`'nargout' must be a positive integer <= 3.'
else
  nargout := args[5];
fi;
if nargs < 4 then
  gamma := 1.0;
elif not(type(args[4], numeric)) or args[4] <= 0 then
  ERROR(`'gamma' must be a positive number.'
else
  gamma := args[4];
fi;
if nargs < 3 then
  apxord := 3;
elif not(type(args[3], integer)) or args[3] <= 0 then
  ERROR(`'apxord' must be a positive integer.'
else
  apxord := args[3];
fi;
if nargs < 2 then
  ERROR(`not enough arguments!');
elif not(nops(args[2]) = 3) then
  ERROR(`'var' must consist of the three parts [x, w, u].
else if convert([seq not(type(op(i, args[2]), vector)), i = 1 .. 3], 'or') then
  ERROR(`'var' must consist of three vectors.');
elif not(nops(args[1]) = 3) then
  ERROR(`'G' must consist of [X(x,w,u), Z(x,w,u), [Y(x,w)].
else if convert([seq not(type(op(i, args[1]), vector)), i = 1 .. 3], 'or') then
  ERROR(`'G' must consist of three vectors.');
elif not(type(args[2], boolean)) then
  var := args[2];
G := args[1];
fi;
hinfOF_x

\[ x := \text{op}(1, \text{var}); \]
\[ w := \text{op}(2, \text{var}); \]
\[ u := \text{op}(3, \text{var}); \]
\[ X := \text{op}(1, G); \]
\[ Z := \text{op}(2, G); \]
\[ Y := \text{op}(3, G); \]
\[ n := \text{vectdim}(x); \]
\[ m1 := \text{vectdim}(w); \]
\[ m2 := \text{vectdim}(u); \]
\[ m := m1 + m2; \]
\[ p1 := \text{vectdim}(Z); \]
\[ p2 := \text{vectdim}(Y); \]

```
if (not (type(Phi, scalar) and (Phi > 0))) then
    ERROR(`'Phi' must be a positive scalar number.');
fi;
if (not (type(Psi, scalar) and (Psi >= 0))) then
    ERROR(`'Psi' must be a non-negative scalar number.');
fi;
```

```
readlib(mtaylor);
```

```
# procedure for the extraction of a vector of indets from a vector of functions
extract := proc(f: vector, x: vector)
    local m, n, A, i, a, j, ord, k, b;
    m := vectdim(f);
    n := vectdim(x);
    A := matrix(m,n, 0.);
    for i from 1 to m do
        a := f[i];
        for j from 1 to n do
            a := collect(a, x[j]);
            ord := degree(a, x[j]);
            b := [seq(coeff(a, x[j]^k), k = 1..ord)];
            A[i,j] := sum('b[k]*x[j]^(k-1)', 'k' = 1..ord);
            # for Maple V.3; for V.4, "add" should be used
            a := simplify(a - A[i,j]*x[j]);
        od;
    od;
    RETURN(eval(A));
end;
```

```
v := vector([seq(w[i], i = 1..m1), seq(u[i], i = 1..m2)]);
xv := [seq(x[i], i = 1..n), seq(v[i], i = 1..m)];
x_list := convert(x, list);
tmp := seq(indets(subs([seq(xv[i]=rand()*1e-13, i=1..n+m)], X[j])), j=1..n);
tmp := (tmp, seq(indets(subs([seq(xv[i]=rand()*1e-13, i=1..n+m)], Z[j])), j=1..p1));
tmp := (tmp, seq(indets(subs([seq(xv[i]=rand()*1e-13, i=1..n+m)], Y[j])), j=1..p2));
if not indets(tmp) = {} then
    ERROR('Parameters must have numerical values:', indets(tmp));
fi;
```

```
lprint(`' Solving L2-gain problem');
```

```
lprint(`' ~~~~~~~~~~~~~~~~~~');
lprint(`' ');
print();
```

```
pass := true;
```

```
# controller
lprint(`' * first HJI *');
lprint(`' ');
print();
```
4  Listings for Maple V414.2

hinfOF_x

: Output Feedback

xkk := vector([seq([seq(x[i]*x[j], j = i..n), i = 1..n])];
# terms of x of order k+1 = 2
index := table([seq((x[i]) = i, i = 1..n)];
g1 := multiply(transpose(Z), Z) - multiply(transpose(x), transpose(C1), C1, x);
- 2*multiply(transpose(x), transpose(C1), Didot, v);
- multiply(transpose(v), transpose(Ddot), Ddot, v);
grad_g1 := grad(gl, v);
f := evalm(K - A &* x - B &* v);
jacob_f := transpose(jacobian(f, v));
for ord from 2 to apxord while pass do  # ord: order of vstar; ord+1: order of V
  lprint(` approximation order:`, ord);
  xkk := vector([seq([seq(xkk[i]*x[j], j = max(seq(index[op(k, indets(xkk[i])]), k = 1..nops(indets(xkk[i])))..n), i = 1..vectdim(xkk))]);
RHS := evalm(transpose(grad_V) &* (B &* vstar + f) + g1 + (*transpose(vstar), R, vstar));
RHS := eval(subs({seq(v[i] = vstar[i], i = 1..m)}, RHS));
RHS := mtaylor(RHS, x_list, ord+2) - mtaylor(RHS, x_list, ord+1);
c_VW := vector(vectdim(xkk));
eq_VW := simplify(evalm(*transpose(grad(multiply(transpose(c_VW), x)), (A - B &* F), x)) + RHS);
eq_VW := {coeffs(collect(eq_VW, x_list, distributed), x_list)};
  for i from 1 to nops(eq_VW) do
    if not type(eq_VW[i], constant) then
      tmp := {op(tmp), eq_VW[i]};
    fi;
  od;
eq_VW := tmp;
deq_VW := convert(c_VW, set) minus indets_eq_VW;
eq_VW := {coeffs(collect(eq_VW, x_list, distributed), x_list)};
for i from 1 to vectdim(c_VW) do
  if abs(c_VW[i]) < threshold then
    c_VW[i] := 0;
  fi;
od;
V := V + multiply(transpose(c_VW), xkk);
grad_V := grad(V, x);  # of order ord
RHS := evalm(-0.5*inverse(R) &* (grad_g1 + (transpose(B) + jacob_f) &* grad_V));
RHS := map(eval, subs({seq(v[i] = vstar[i], i = 1..m)}, eval(RHS)));
grad_g1 := grad(g1, w);
jacob_X := transpose(eval(subs({seq(u[i] = 0, i = 1..m2)}, jacobian(X, w))));
for ord from 2 to apxord while pass do
    # ord: order of wstarstar, ystar; ord+1: order of W
    lprint(`  approximation order:`, ord);
    xkk := vector([seq(seq(xkk[i]*x[j], j = max(seq(index[op(k, indets(xkk[i]))], k = 1..nops(indets(xkk[i]))))..n), i = 1..vectdim(xkk))]);
    wstarstar := map(eval, subs({seq(y[i] = ystar[i], i = 1..p2)}, evalm(w_hat)));
    RHS := evalm(transpose(grad_W) &* (B1 &* wstarstar + f) - transpose(ystar) &* Y + g1 + &* (transpose(wstarstar), R1, wstarstar) - Hstar + Phi*transpose(grad_W)*grad_W);
    RHS := eval(subs({seq(w[i] = wstarstar[i], i = 1..m1), seq(u[i] = 0, i = 1..m2)}, RHS));
    RHS := mtaylor(RHS, x_list, ord+2) - mtaylor(RHS, x_list, ord+1);
    c_VW := vector(vectdim(xkk));
    eq_VW := simplify(evalm(transpose(grad(multiply(transpose(c_VW), x)) &* ((A - G &* C2) &* x + B1 &* wstarstar_1)) + RHS);
    eq_VW := {coeffs(collect(eq_VW, x_list, distributed), x_list)};
    tmp := {};
    for i from 1 to nops(eq_VW) do
        if not type(eq_VW[i], constant) then
            tmp := {op(tmp), eq_VW[i]};
        fi;
    od;
    eq_VW := tmp;
    indets_eq_VW := indets(eq_VW);
to_be_assigned := convert(c_VW, set) minus indets_eq_VW;
    assign(fsolve(eq_VW, indets_eq_VW));
    assign({seq(to_be_assigned[i] = 0, i = 1 .. nops(to_be_assigned))});
    threshold := min(eps, 1000.*10^(-Digits)) * max(seq(abs(c_VW[i]), i=1..vectdim(c_VW)));
    for i from 1 to vectdim(c_VW) do
        if abs(c_VW[i]) < threshold then
            c_VW[i] := 0;
        fi;
    od;
    W := W + multiply(transpose(c_VW), xkk);
    grad_W := grad(W, x);  # of order ord
    grad_Y := grad(evalm(transpose(y) &* Y), w);
    WXg := evalm(jacob_X &* grad_W + grad_g1);
    RHS := eval(subs({seq(w[i] = w_hat[i], i = 1..m1), seq(u[i] = 0, i = 1..m2)}, RHS));
    w_hat := vector([seq(w_hat[i] = w_hat[i], i = 1..m1)], jacobian(RHS, x_list, ord+1));
    jacob_Y := eval(subs({seq(w[i] = w_hat[i], i = 1..m1)}, jacobian(Y, w)));
    g2 := eval(subs({seq(w[i] = w_hat[i], i = 1..m1)}, evalm(Y - C2 &* x - 0.5*DRD &* y)));
    jacob_w_hat := jacobian(w_hat, y);
    g3 := evalm((jacob_Y &* jacob_w_hat - 0.5*DRD));
    g4 := evalm(2*DRD1 &* (transpose(jacob_w_hat)) &* (WXg + 2 &* (D11t, C1, x)) - g2 - transpose(g3) &* y + g4));
RHS := map(eval, subs({seq(w[i] = w_hat[i], i = 1..m1), eval(RHS)});
RHS := map(eval, subs({seq(y[i] = ystar[i], i = 1..p2)}, eval(RHS)));
4.3 hinfOF_p: Alternative Output Feedback

hinfOF := proc()
local Phi, eps, nargout, gamma, apxord, var, balflag, fileflag, filename,
    ordflag, extract, tmp, G, x, w, u, v, xv, X, Z, Y, RHS, indets_eq_VW,
    to_be_assigned,
    n, m1, m2, m, pl, p2, i, j, pass, A, Ae, B, Cl, D, Ddot, R, Q, F, res, Psi,
    ham, ustar, gl, V, grad_V, xk, index, grad_gl, jacob_f, k, c_V, eq_VW,
    ord, ustar, wstar, Xstar, Zstar, Hstar, phi, x_list, K,
    B1, D1l, D2l, D1lt, D2lt, C2, eyem1, R1, Rl1, DRO, DROS, hstar, P,
    ystar, y, w_hat, wstarstar_1, W, grad_W, jacob_X, wstarstar, grad_yY, jacob_Y,
    pYpDCx, g2, jacob_w, jacob_w_hat, g3, g4, Ystar, L, Wpp, Wppi, dx, xi, Kxi, Tzw;

    if nargs < 11 then
        fileflag := false;
    elif not(type(args[11], string)) then
        ERROR(`'filename' must be a string.`);
    else
        fileflag := true;
        filename := args[11];
    fi;
    if nargout = 1 then
        RETURN(eval(K, 2));
    else
        RETURN(eval([K, Tzw], 3));
    fi;
end:

# `balflag - if true, the linear approximation of the controller is balanced;`, # the resulting coordinate transformation is applied to the', # controller (default: false);', # 'filename - If specified, a Simulink mex-file containing code for the', # controller is written with the name filename (must be a string);', # 'in this case, nothing is returned by hinfOF_p', # SYNOPSIS:', # Computes the solution to the nonlinear L2-gain problem (H_inf problem)', # ||Z(x, w, u)||2 / ||w||2 < gamma', # If nargout := 2, the output of hinfOF_p is a list containing', # [K, Tzw] where K is the controller and Tzw is the closed-loop', # dynamics.', # The controller computed by hinfOF_p has the Lagrangian p as', # its state and all the computations involving the 2nd HJI are', # in p (rather than in x as in hinfOF_x);', # If nargout := 1 (default), only K is returned.', # If a filename is specified, the inversion necessary for the', # computation of the output injection gain is not carried out symbolically. Instead, code', # is generated which will compute this inversion numerically on-line.', # Urs Christen, May 1997

if nargs < 11 then
    fileflag := false;
else if not(type(args[11], string)) then
    ERROR(`'filename' must be a string.`);
else
    fileflag := true;
    filename := args[11];
fi;
if nargs < 10 then
    balflag := false;
else
    balflag := true;
fi;

4.3 hinfOF_p: Alternative Output Feedback
error('balflag' must be a boolean.);
else
  balflag := args[10];
fi;
if nargs < 9 then
  ordflag := true;
else
  ordflag := args[9];
fi;
if nargs < 8 then
  Phi := 1.0e-8;
elif not(type(args[8], scalar) or type(args[8], numeric))
  then
  error('Phi' must be a scalar or a numerical value.);
else
  Phi := args[8];
fi;
if nargs < 7 then
  Psi := 1.0e-8;
elif not(type(args[7], scalar) or type(args[7], numeric))
  then
  error('Psi' must be a scalar or a numerical value.);
else
  Psi := args[7];
fi;
if nargs < 6 then
  eps := 1.0e-10;
elif not(type(args[6], numeric))
  then
  error('eps' must be a numerical value.);
else
  eps := args[6];
fi;
if nargs < 5 then
  nargout := 1;
else
  nargout := args[5];
fi;
if nargs < 4 then
  gamma := 1.0;
else
  gamma := args[4];
fi;
if nargs < 3 then
  apxord := 3;
else
  apxord := args[3];
fi;
if nargs < 2 then
  error('not enough arguments!');
elif nops(args[2]) = 3 then
  error('var' must consist of the three parts \(x, w, u\).');
elif convert([seq(not(type(op(i, args[2]), vector)), i = 1..3)], 'or')
  then
  error('var' must consist of three vectors.);
elif nops(args[1]) = 3 then
  \(G\) := args[2];
  \(f\) := args[1];
f := op(1, var);
w := op(2, var);
u := op(3, var);
X := op(1, G);
Z := op(2, G);
Y := op(3, G);

\(n :=\) vectdim(x);
ml := vectdim(w);
m2 := vectdim(u);
m := m1 + m2;
p1 := vectdim(Z);
p2 := vectdim(Y);

if (not(type(Psi, scalar) and (Psi >= 0)))
  then
  error('Psi' must be a non-negative scalar number.);
fi;
if (type(Phi, numeric))
  then
  Phi := evalm(Phi * transpose(x) &* x);
fi;
readlib(mtaylor);

# procedure for the extraction of a vector of indets from a vector of functions
extract := proc(f: vector, x: vector)
  local m, n, A, i, a, j, ord, k, b;
  m := vectdim(f);
  n := vectdim(x);
  A := matrix(m,n, 0.);
  for i from 1 to m do
    a := f[i];
    for j from 1 to n do
      a := collect(a, x[j]);
      ord := degree(a, x[j]);
      b := [seq(coeff(a, x[j]^k), k = 1..ord)];
      A[i,j] := sum('b[k]*x[j]^(k-1)', 'k' = 1..ord);
      a := simplify(a - A[i,j]*x[j]);
    od;
  od;
  RETURN(eval(A));
end;

v := vector([seq(w[i], i = 1..ml), seq(u[i], i = 1..m2)]);
xv := [seq(x[i], i = 1..l), seq(v[i], i = 1..m)];
x_list := convert([x, list]);
tmp := seq(indets(subs([seq(xv[i]=rand()*1e-13, i=1..n+m)], X[j])), j=1..n);
tmp := [tmp, seq(indets(subs([seq(xv[i]=rand()*1e-13, i=1..n+m)], Z[j])),
  j=1..p1)];
tmp := (tmp, seq(indets(subs({seq(xv[i]=rand()*1e-13, i=1..n+m)}, Y[j])), j=1..p2));
if not(indets(tmp) = {}) then
ERROR('Parameters must have numerical values:', indets(tmp));
fi;
lprint('  Solving L2-gain problem');
lprint('  =======================');
lprint('  ');
print();
pass := true;

# controller
lprint('  * first HJI * ');
lprint('  ');
print();
# first order approximation:
lprint('approximation order:', 1);
A := map(eval, subs({seq(xv[i] = 0., i = 1..n+m)}, jacobian(X, x)));
B := map(eval, subs({seq(xv[i] = 0., i = 1..n+m)}, jacobian(X, v)));
C1 := map(eval, subs({seq(xv[i] = 0., i = 1..n+m)}, jacobian(Z, x)));
D1dot := map(eval, subs({seq(xv[i] = 0., i = 1..n+m)}, jacobian(Z, v)));
R := evalm(transpose(D1dot)&*D1dot - diag(seq(gamma^2,i=1..m1), seq(0,i=1..m2)));
if max(op(singularvals(submatrix(D1dot, 1..p1, 1..m1)))) >= gamma then
lprint('    sigmamax(D11) < gamma:        FAIL');
pass := false;
else
lprint('    sigmamax(D11) < gamma:         ok');
fi;
if rank(R) < m then
lprint('    invertibility of R:           FAIL');
pass := false;
else
lprint('    invertibility of R:            ok');
fi;
print();
if pass then
Ae := evalm(A - &*(B, inverse(R), transpose(B), C1));
Q := evalm(Psi*&*() + transpose(C1) &* C1 - &*(transpose(C1), D1dot, inverse(R), transpose(D1dot), C1));
ham := stack(augment(Ae, -&*(B, inverse(R), transpose(B))), augment(-Q, -transpose(Ae)));
res := aresolve(ham, 'eigen2', 2, eps);
if not(res[2] and definite(res[1], 'positive_semidef')) then
lprint('    Riccati equation:             FAIL');
else
K := res[1];
F := evalm(inverse(R)*transpose(B)&*K + transpose(D1dot)&*C1));
vstar := vector([seq(vstar[i], i = 1..m1)]);
grad_V := grad(V, x);
fli;
fi;
fi;

RHS := evalm(transpose(grad_V) &* (B &* vstar + X) + grad_f &* (transpose(F, vstar)));
RHS := map(eval, subs({seq(v[i] = vstar[i], i = 1..m)}, eval(RHS)));
for i from 2 to apxord while pass do # ord: order of vstar; ord+1: order of V
lprint('  approximation order:', ord);
xkk := vector([seq(seq(x[i]*x[j], j = i..n), i = 1..n)]);
index := table([seq((x[i]) = i, i = 1..n)]);
g1 := multiply(transpose(Z), Z - multiply(transpose(C1), D1dot, v) - multiply(transpose(v), transpose(D1dot), D1dot, v));
for i from 2 to apxord while pass do # ord: order of V
lprint('  approximation order:', ord);
xkk := vector([seq(seq(xkk[i]*x[j], j = max(seq(index[op(k, indets(xkk[i])]), k = 1..nops(indets(xkk[i]))))..n), i = 1..vectdim(xkk))]);
RHS := evalm(transpose(grad_V) &* (B &* vstar + X) + grad_f &* (transpose(F, vstar)));
RHS := map(eval, subs({seq(v[i] = vstar[i], i = 1..m)}, eval(RHS)));
for i from 2 to apxord while pass do # ord: order of vstar; ord+1: order of V

Hstar := evalm(transpose(grad_V) &* Xstar + transpose(Zstar) &* Zstar - gamma^2 * transpose(wstar) &* wstar);

lprint(`  `);
lprint(`  * second HJI *`);
lprint(`  `);
# first order approximation:
# first order approximation order: 1;
print();
B1 := submatrix(B, 1..n, 1..m1);
D11 := submatrix(D1dot, 1..p1, 1..m1);
D21 := map(eval, subs({seq(xv[i] = 0., i = 1..n+m1)}, jacobian(Y, w)));
D21t := transpose(D21);
D11t := transpose(D11);
C2 := map(eval, subs({seq(xv[i] = 0., i = 1..n+m1)}, jacobian(Y, x)));
eym1 := diag(seq(1., i=1..m1));
R1 := evalm(D11t &* D11 - gamma^2 * eym1);
R1i := inverse(R1);
DRD := evalm(&*(D21, R1i, D21t));
DRDi := inverse(DRD);
hstar := evalm(0.5 * map(eval, subs({seq(x[i] = 0., i = 1..n)}, hessian(Hstar, x))));
phi := evalm(0.5 * map(eval, subs({seq(x[i] = 0., i = 1..n)}, hessian(Phi, x))));
Ae := evalm(A - &*(B1, R1i, (D11t &* C1 + &*(D21t, DRDi, (C2 - &*(D21, R1i, D11t, C1)))));
R := evalm(4. * (&*(transpose(C2 - &*(D21, R1i, D11t, C1)), DRDi) + &*(transpose(wstarstar), R1, wstarstar) + &*(transpose(x), transpose(C1), C1, x) - Hstar + Phi));
Q := evalm(0.25 * &*(B1, (-R1i + &*(R1i, D21t, DRDi, D21, R1i)), transpose(B1)) + phi);

# higher order approximations
# mtaylor: order must be desired order + 1
pkk := vector([seq(seq(p[i]*p[j], j = i..n), i = 1..n)]);
# terms of p of order k+1 = 2
index := table([seq(x[i] = 0., i = 1..n)]);
g1 := eval(subs({seq(u[i] = 0., i = 1..m2)}, jacobi(X, w)));
grad_gl := grad(gl, w);
jacob_X := transpose(eval(subs({seq(u[i] = 0., i = 1..m2)}, jacobi(X, w))));
for ord from 2 to apxord while pass do
  lprint(`  approximation order:`, ord);
  pkk := vector([seq(seq(pkk[i]*p[j], j = max(seq(index[op(k, indets(pkk[i])]), k = 1..nops(indets(pkk[i]))))..n), i = 1..vectdim(pkk))], k = 1..nops(indets(pkk[i])));
wstarstar := map(eval, subs({seq(y[i] = ystar[i], i = 1..p2), eval(w_hat)});
RHS := evalm(transpose(p) &* (hstarstar + &*(transpose(wstarstar), R1, wstarstar)) + &*(transpose(w_hat), transpose(C1), C1, x) - Hstar + Phi));
RHS := eval(subs({seq(x[i]=grad_W[i], i=1..n), seq(w[i]=wstarstar[i], i=1..m1), seq(u[i]=0, i = 1..m2)}, RHS));
RHS := mtaylor(RHS, p_list, ord+2) - mtaylor(RHS, p_list, ord+1);
c_VW := vector(mtaylor(pkk, p_list, ord+1));
eq_VW := simplify(evalm(&*(transpose(p), (A - G &* C2), grad(multiply(transpose(c_VW), pkk), p)) + RHS));
eq_VW := {coeffs(collect(eq_VW, p_list, distributed), p_list)};
tmp := [];
for i from 1 to nops(eq_VW) do
  if not type(eq_VW[i], constant) then
    tmp := {op(tmp), eq_VW[i]};
  fi;
  od;
  eq_VW := tmp;
  indets_eq_VW := indets(eq_VW);
to_be_assigned := convert(c_VW, set) minus indets_eq_VW;
assign(fsolve(eq_VW, indets_eq_VW));
assign({seq(to_be_assigned[i] = 0, i = 1 .. nops(to_be_assigned))});
W := W + multiply(transpose(c_VW), pkk);
grad_W := grad(W, p);  # of order ord

## Further code
# higher order approximations
# mtaylor: order must be desired order + 1
4.4 Auxiliary Subroutines

4.4.1 aresolve: Solution to the Algebraic Riccati Equation

aresolve := proc()  
  local eps, nargout, method, ham, n, m, k, EVham, i, XX, X, info;  
  
ystar := vector([seq(ystar[i] + mtaylor(RHS[i], p_list, ord+1)  
                 - mtaylor(RHS[i], p_list, ord), i = 1..p2]));  
  print();  
od;  
RHS := 'RHS'; eq_VW := 'eq_VW'; tmp := 'tmp'; indets_eq_VW := 'indets_eq_VW';  
pkk := 'pkk';  
if pass then  
  Ystar := map(eval, subs([seq(w[i] = wstar[i], i = 1..m1)], eval(Y)));  
  # calculation of observer gain  
  # order is restricted to apxord  
  lprint(`  * calculation of output injection gain *`);  
  print();  
  grad_V_p := map(eval, subs([seq(x[i] = grad_W[i], i = 1..n)],  
                           eval(grad_V)));  
  tmp := evalm(p - grad_V_p);  
  tmp := vector([seq(mtaylor(tmp[i], p_list, apxord+1), i = 1..n)]);  
  L := transpose(extract(tmp, p));  
  R := transpose(extract(ystar, p));  
  hess_W := hessian(W, p);  
  ustar_p := map(eval, subs([seq(x[i] = grad_W[i], i = 1..n)],  
                           eval(ustar)));  
  ustar_p := vector([seq(mtaylor(ustar_p[i], p_list, apxord+1), i = 1..m2)]);  
  if balflag then  
    A := evalm(&*(inverse(P), (A-B&*F-G&*C2), P));  
    B := evalm(0.5*inverse(P)&*G);  
    C := evalm(-2.*F&*P);  
    T := sysbal(A, B, C);  
    p_ := evalm(T &* p);  
    ustar_p := map(eval, subs([seq(p[i] = p_[i], i = 1..n)],  
                           eval(ustar_p)));  
    fi;  
  if fileflag then  
    G := linsolve(L, R);  
    # controller  
    K := [[evalm(inverse(hess_W) &* (Xstar + G &* (y - Ystar))), ustar_p],  
          [p, y]];  
    if ordflag then  
      K := [[map(eval, subs([seq(x[i] = grad_W[i], i = 1..n)],  
                             eval(K[1][1]))), K[1][2]], K[2]];  
      if balflag then  
        K := [[vector([seq(mtaylor(K[1][1][i], p_list, apxord+1), i = 1..n)]),  
               vector([seq(mtaylor(K[1][2][i], p_list, apxord+1), i = 1..m2)]), K[2]];  
        fi;  
      else  
        K := [[evalm(inverse(T) &* K[1][1]), K[1][2]], K[2]];  
        fi;  
      if nargout = 2 then  
        Kzw := [[map(eval, subs([seq(y[i] = Y[i], i = 1..p2)],  
                                eval(K[1][1]))),  
                  map(eval, subs([seq(y[i] = Y[i], i = 1..p2)],  
                                  eval(K[1][2])))]];  
        if balflag then  
          Tzw := [[vector([seq(eval(subs([seq(u[i] = Kzw[2][i], i = 1..m2),  
                                         X[j])], j = 1..n), seq(Kzw[1][j], j = 1..n)])),  
                      map(eval, subs([seq(u[i] = Kzw[2][i], i = 1..m2)],  
                                      eval(Z)))]];  
          fi;  
        else  
          K := [];  
          Tzw := [];  
          fi;  
        if nargout = 1 then  
          RETURN(eval(K, 2));  
        else  
          RETURN(eval([K, Tzw], 3));  
        fi;  
      fi;  
    fi;  
  fi;  
end:
# Algebraic Riccati Equation -- numerical solution
# [X, info] := aresolve(ham, method, nargout, eps)
# finds the stabilizing solution (A+R*X stable) of the
# Riccati equation
# A'*X + X*A + X*R*X + Q = 0.
# input:  ham = [A R; -Q -A']: Hamiltonian
# method: `eigen` or `eigen2` -- method used to compute the basis
#         of the stable subspace of ham
# nargout: number of output arguments (default: 1)
# output: X:       stabilizing solution to the Riccati equation
#         info:    flag: true if valid solution, false otherwise (due to
#                        jw-axis eigenvalues of ham or an unequal number of
#                        positive and negative eigenvalues).
# not optimized for speed! does not use evalhf
#
if nargs < 4 then
eps := 1.0e-10;
elif not(type(args[4], numeric)) then
ERROR(`'eps' must be a numerical value.`);
else
eps := args[4];
fi;
if nargs < 3 then
nargout := 1;
elif not(type(args[3], integer)) or args[3] <= 0 then
ERROR(`'nargout' must be a positive integer <= 2.`);
else
nargout := args[3];
fi;
if nargs < 2 then
method := `eigen`;
elif not(args[2] = `eigen` or args[2] = `eigen2`) then
ERROR(`'method' must be `eigen` or `eigen2`.
else
method := args[2];
fi;
if nargs < 1 then
ERROR(`no arguments!`);
elif not(type(args[1], 'matrix'(numeric, square))) then
ERROR(`'ham' must be a square matrix of numerical values.`);
else
ham := scalarmul(args[1], 1.0);
fi;
f;

n := rowdim(ham)/2;
if method = `eigen` then # using eigenvects
EVham := sort([eigenvects(ham)], (x,y) -> evalb(Re(op(1,x)) < Re(op(1,y))));
m := 0; k := 0;
while (m < n) do
  k := k + 1;
  m := m + op(2,op(k, EVham));
  od;
else
  EVham := sort([evalf(eigenvals(ham, 'vect'))], (x,y) -> evalb(Re(x) < Re(y)));
  m := 1; k := 0;
  while (m < 2*n) do
    if Re(EVham[m]) < -eps then
      k := k + 1;
      if type(EVham[m], realcons) then
        XX[k] := evalm([subvector(EVham, 1..2*n, m)]);
      else
        XX[k] := evalm([subvector(EVham, 1..2*n, m)] + I*subvector(EVham, 1..2*n, m+1));
        XX[k] := evalm([subvector(EVham, 1..2*n, m)] - I*subvector(EVham, 1..2*n, m+1));
      fi;
    fi;
    m := m + 1;
  od;
fi;

if k = n then
  # the first n eigenvalues are in the left half plane
  XX := transpose(matrix([seq(op(op(i, EVham)), i=1..k)]));
  X := linsolve(transpose(submatrix(XX, 1..n, 1..n)),
                transpose(submatrix(XX, n+1..2*n, 1..n)));
  X := map(Re, evalm(0.5*X + 0.5*transpose(X)));
  info := true;
else
  X := `X`;
  info := false;
fi;
evaluate(X, info := false);
else
  info := false;
fi;

if nargout = 1 then
RETURN(eval(X));
else
RETURN([X, info], 2);
fi;
evaluate;

'help/text/aresolve' := TEXT(
  'FUNCTION: aresolve: numerical solution to the algebraic Riccati equation',
  'CALLING SEQUENCE:',
  'aresolve(ham, method, nargout, eps)',
  'PARAMETERS:',
  'ham' = Hamiltonian [A R; Q -A'] for the Riccati equation',
  'method' = 'eigen' or 'eigen2' -- method used to compute the basis',
  'nargout' = number of output arguments (default: 1)',
  'method' = 'eigen' or 'eigen2' -- method used to compute the basis
)
4.4.2 lyapsolve: Solution to Lyapunov Equation

lyapsolve := proc()
local A, Q, n, EWEV_A, EW_A, EV_A, EV_Ai, i, j, X;

# Lyapunov Equation -- numerical solution
# X := lyapsolve(A, Q)
# finds the solution X of the Lyapunov equation
# A*X + X*A' + Q = 0
# if A has eigenvalues smaller than zero.
# not optimized for speed! does not use evalhf
# Urs Christen, Feb. 1997
if nargs < 2 then
ERROR('not enough arguments!');
elif not(type(args[1], 'matrix'(numeric, square))) then
ERROR('A must be a square matrix of numerical values.');
elif not(type(args[2], 'matrix'(numeric))) then
ERROR('Q must be a square matrix of numerical values.');
elif not(rowdim(args[2]) = rowdim(args[1])) then
ERROR('dimensions not compatible.');
else
A := scalarmul(args[1], 1.0);
Q := scalarmul(args[2], 1.0);
fi;

n := rowdim(A);
EWEV_A := [eigenvects(A)];
EW_A := vector([seq(seq(op(1, op(i, EWEV_A)), j=1..op(2, op(i,EWEV_A))), i=1..nops(EWEV_A))]);
EV_A := transpose(matrix([seq(op(op(3,op(i,EWEV_A))), i=1..nops(EWEV_A))]));
EV_Ai := inverse(EV_A);
X := evalm(&*(EV_Ai, Q, transpose(EV_Ai)));
X := matrix([seq([seq(X[i,j]/(EW_A[i]+EW_A[j]), j=1..n)], i=1..n)]);
X := evalm(&*(-EV_A, X, transpose(EV_A)));
X := map(Re, evalm(0.5*X + 0.5*transpose(X)));
RETURN(eval(X));
end:

4.4.3 sysbal: Balancing a Linear System

sysbal := proc()
local A, B, C, Wc, Wo, sig, T, U;

# Transformation matrix for balancing a linear system
# T := sysbal(A, B, C)
# computes the balancing transformation matrix for the system
# (A, B, C). The balanced system is given by (T\A*T, T\B, C*T).
# The system must be minimal and asymptotically stable.
# Urs Christen, Feb. 1997
if nargs < 3 then
ERROR('not enough arguments!');
elif not(type(args[1], 'matrix'(numeric, square))) then
ERROR('A must be a square matrix of numerical values.');
elif not(type(args[2], 'matrix'(numeric))) then
ERROR('B must be a matrix of numerical values.');
elif not(type(args[3], 'matrix'(numeric))) then
ERROR('C must be a matrix of numerical values.');
elif not(rowdim(args[1]) = rowdim(args[2]) and coldim(args[1]) = coldim(args[3])) then
ERROR('dimensions not compatible.');
else
A := scalarmul(args[1], 1.0);
B := scalarmul(args[2], 1.0);
C := scalarmul(args[3], 1.0);
fi;

Wc := lyapsolve(A, evalm(B &* transpose(B)));
Wo := lyapsolve(transpose(A), evalm(transpose(C) &* C));
sig := diag(op(convert(map(sqrt, evalf(Svd(Wc, U, 'left'))), list)));
T := evalm(U &* sig);
Wo := evalm(*transpose(T), Wo, T));
sig := diag(op(convert(map((x) -> x^(-0.25), evalf(Svd(Wo, U, 'left'))), list)));
T := evalm(*transpose(T), U, sig));
RETURN(eval(T));
end:
4.5 Generation of Code for SIMULINK

There are two different subroutines for the generation of SIMULINK mex files in C code. The first one, `wrtMEXfile`, can be used for any smooth, dynamical system. The second one, `wrtMEXctrl`, generates code for a controller for which the observer gain matrix $G(x)$ must be computed on line. It contains code for solving a system of linear equations (Gauß algorithm, code based on [8]). Both of these subroutines make use of the mex-file template `sfuntmpl.c` shipped with SIMULINK 1.3.

4.5.1 `wrtMEXfile`

```maple
wrtMEXfile := proc()
local var, G, X, Y, dx, y, xx, u, uu, n, m, p, i, line, replaceset,
     tmplname, mexfile, name, Xo, Yo, localsX, localsY, tmp;
# writing a mex-file for use in SIMULINK
#
# wrtMEXfile(G, var, system_name)
# writes a mex-file in C for the simulation of the
dynamical system given by
#       x_dot = X(x,u)
#       y     = Y(x,u).
#
# input:  G := [[X(x,u)], [Y(x,u)]]: dynamical system
#         var := [[x], [u]]:  variables in G
#         system_name: name of the system and file to be produced
# (without extension)
#
# Urs Christen, August 1996

if nargs < 3 then
    ERROR(`not enough arguments!`);
elif not(type(args[3], string)) then
    ERROR(`'system name' must be a string.`);
elif nops(args[2]) = 2 then
    ERROR(`'var' must consist of the two parts [x, u]`);
elif convert([seq(not(type(op(i, args[2])), vector)), i = 1..2], `or`) then
    ERROR(`'var' must consist of two vectors.`);
elif nops(args[1]) = 2 then
    ERROR(`'G' must consist of [[X(x,u)], [Y(x,u)]]`);
elif convert([seq(not(type(op(i, args[1])), vector)), i = 1..2], `or`) then
    ERROR(`'G' must consist of two vectors.`);
else
    var := args[2];
    G := args[1];
fi;

name := args[3];
xx := op(i, var);
uu := op(i, var);
X := op(i, G);
Y := op(i, G);
n := vectdim(xx);
m := vectdim(uu);
p := vectdim(Y);
```

4 Listings for Maple V514.5  Generation of Code for SIMULINK

```maple
line := readline(tmplname);
writeln(line);
od;
writeln(`#define S_FUNCTION_NAME `.name);
writeln(mexfile);
for i from 15 to 31 do
  line := readline(tmplname);
  writeln(line);
od;
writeln(`#include <math.h>`);
for i from 32 to 36 do
  line := readline(tmplname);
  od;
for i from 37 to 57 do
  line := readline(tmplname);
  writeln(line);
od;
for i from 58 to 61 do
  line := readline(tmplname):
  od;
for i from 62 to 74 do
  line := readline(tmplname):
  writeln(line):
od;
if n > 0 then
  writeln(`    int i;`);
  writeln(`    `);
  writeln(`    for(i = 0; i < `.n.`; i++)`);
  writeln(`        x0[i] = 0.;`);
fi;
for i from 75 to 77 do
  line := readline(tmplname);
  od;
for i from 78 to 88 do
  line := readline(tmplname);
  writeln(line);
od;
if n > 0 then
  writeln(`    # output function`);
  for i from 1 to nops(localsY) do
    tmp := op(i, localsY);
    writeln(`    static double `.tmp.`;`);
    writeln(`    static double s1, s2, s3, s4, s5, s6, s7, s8, s9, s10;`);
    writeln(mexfile);
    od;
  fi;
  for i from 79 to 91 do
    line := readline(tmplname);
    od;
for i from 92 to 118 do
  line := readline(tmplname);
  writeln(line);
done;
# derivative's function
if n > 0 then
  for i from 1 to nops(localsX) do
    tmp := op(i, localsX);
    writeln(`    static double `.tmp.`;`);
    writeln(`    static double s1, s2, s3, s4, s5, s6, s7, s8, s9, s10;`);
    writeln(mexfile);
    close();
    C(Xo, filename = mexfile);
    appendto(mexfile);
  fi;
for i from 119 to 122 do
  line := readline(tmplname);
  od;
end:

`help/text/wrtMEXfile` := TEXT(
  `FUNCTION: wrtMEXfile: produces a mex-file for use in SIMULINK`,
  `CALLING SEQUENCE:`,
  `wrtMEXfile(G, var, system_name)`,
  `PARAMETERS:`,
  `G       - dynamical system \
           \[X(x,u), Y(x,u)\] where`,
  `\quad x_dot = X(x,u)`,
  `\quad y = Y(x,u)`,
  `\quad X and Y must be of type vector`,
  `\quad var - names of the variables used in G; var := [x], [u]`,
  `\quad system_name - name of the system and file to be produced`,
  `\quad (without extension)`,
  `SYNOPSIS:`,
  `- writes a mex-file for use in SIMULINK`,
  `- For static systems, define X and x as vector([]),`,
  `- Path not handled yet; no initial conditions`);
4.5.2 \texttt{wrtMEXctrl}

\texttt{wrtMEXctrl := proc()}
local GG, RR, R, LL, L, X, dx, y, yy, xx, uu, n, m, p, i, j, line, replaceset, replaceset2, tmplname, invfile, mexfile, name, Xo, Yo, Ro, Lo, localsX, localsL, localsR, localsXRLW, localsY, tmp, AWBTflag, np, G, u, x, Wppflag, localsWpp, hess_W, hess_Wi, Wpp, Wppi, Wppo;

\# Urs Christen, Feb 1997

if nargs < 10 then
  Wppflag := false;
else
  Wppflag := true;
  hess_W := args[10];
  hess_Wi := args[11];
fi;
if nargs < 9 then
  AWBTflag := false;
else
  AWBTflag := args[9];
fi;
name := args[8];
GG := args[7];
yy := args[6]; \# inputs to controller, later u
xx := args[5]; \# outputs of controller, later y
uu := args[4];
L := args[3];
R := args[2];
X := args[1];
n := vectdim(xx);
m := vectdim(uu);
p := vectdim(yy);

readlib(C);
readlib(optimiz.e); # for releases R2 and R3 only
readlib(write);
interface(screenwidth=200);

replaceset := seq(yy[i] = u[i-1], i = 1..p), seq(xx[i] = x[i-1], i = 1..n);
replaceset2 := seq(seq(GG[i, j] = G[i-1, j-1], j = 1..p), i = 1..n);
if Wppflag then
  replaceset2 := `union`(replaceset2, seq(seq(hess_Wi[i, j] = Wppi[i-1, j-1], j = 1..n), i = 1..n));
fi;
X := map(eval, subs(replaceset, evalm(X)));
RR := map(eval, subs(replaceset, evalm(R)));
LL := map(eval, subs(replaceset, evalm(LL)));
Xo := map(eval, subs(replaceset, evalm(Xo)));

X := array(0..n-1);
for i from 1 to n do
  dx[i-1] := X[i];
od;

Xo := [optimize(dx)];
localsX := (op(2, `optimize/makeproc` (Xo)));
L := array(0..n-1, 0..n-1);
for i from 1 to n do
  for j from 1 to n do
    L[i-1, j-1] := LL[i, j];
  od;
od;

Yo := [optimize(y)];
localsY := (op(2, `optimize/makeproc` (Yo)));

tmplname := `sfuntmpl.c`;
n := vectdim(yy);
for i from 1 to n do
  y[i-1] := uu[i];
od;
localsWpp := {};

if Wppflag then
  hess_W := map(eval, subs(replaceset, evalm(hess_W)));
  Wpp := array(0..n-1, 0..n-1);
  for i from 1 to n do
    for j from 1 to n do
      Wpp[i-1, j-1] := hess_W[i, j];
    od;
  od;
  Wppi := [optimize(Wpp)];
  localsWpp := (op(2, `optimize/makeproc` (Wppi)));
else
  localsWpp := {};
fi;

localsXRLW := `union` (localsX, localsR, localsL, localsWpp);
y := array(0..m-1);
for i from 1 to m do
  y[i-1] := uu[i];
od;
localsY := (op(2, `optimize/makeproc` (Yo)));

tmplname := `sfuntmpl.c`;
invfile := `linsolv.c`;
traperror(fremove(mexfile));

open(mexfile);
line := readline(tmplname);
write1n(line);
write1n(’ * ’);
write1n(’ C-code for an s-function for use in Simulink’);
write1n(’ * ’);
/* generated by wrtMEXctrl in Maple */
close();
system(`date >>`.mexfile);
appendto(mexfile);
writeln(` * generated by wrtMEXctrl by U. Christen, Feb 1997 */
`);
for i from 1 to 10 do
  line := readline(invfile);
  writeln(line);
od;
writeln(`#define dim_n `.n.` /* Dimension n */`);
line := readline(invfile);
while line <> 0 do
  writeln(line);
  line := readline(invfile);
od;
for i from 2 to 6 do
  line := readline(tmplname);
for i from 7 to 13 do
  line := readline(tmplname);
writeln(`#define S_FUNCTION_NAME `.name);
writeln();
writeln(`#include <math.h>`);
for i from 15 to 31 do
  line := readline(tmplname);
for i from 32 to 36 do
  line := readline(tmplname);
for i from 37 to 57 do
  line := readline(tmplname);
for i from 58 to 61 do
  line := readline(tmplname);
for i from 62 to 74 do
  line := readline(tmplname):
if n > 0 then
  writeln(` int i;`);
  writeln(` `);
  writeln(` for(i = 0; i < `.n.`; i++)`);
  writeln(` x0[i][0] = 0.;`);
fi;
for i from 75 to 77 do
  line := readline(tmplname);
for i from 78 to 88 do
  line := readline(tmplname);
writeln(line);
# output function
for i from 1 to nops(localsY) do
  tmp := op(i, localsY);
  writeln(` static double `.tmp.`;`);
for i from 89 to 91 do
  line := readline(tmplname);
for i from 92 to 118 do
  line := readline(tmplname);
writeln(line);
# derivative's function
writeln(`    static int i, j, err, signdet, perm['`.n.`'];`);
writeln(`    static double sol['`.n.`'];`);
writeln(`    static double G['`.n.`']['`.p.`'];`);
writeln(`    static double R['`.p.`']['`.n.`'];`);
writeln(`    static double L['`.n.`']['`.n.`'];`);
if Wppflag then
  writeln(`    static double Wpp['`.n.`']['`.n.`'];` /* Hessian of W(p) */
  writeln(`    static double Wppi['`.n.`']['`.n.`'];` /* Inverse of Wpp */
  writeln(`    static double eye['`.n.`'];` /* Identity for inversion */
fi;
for i from 1 to nops(localsXRLW) do
  tmp := op(i, localsXRLW);
  writeln(`    static double `.tmp.`;`);
writeln(`    static double s1, s2, s3, s4, s5, s6, s7, s8, s9, s10;` /* often used */`);
close();
C(Yo, filename = mexfile);
C(Ro, filename = mexfile);
C(Lo, filename = mexfile);
if Wppflag then
  C(Wppo, filename = mexfile);
fi;
4.5.3 linsolve.c

/*  ***************************************************************  */
/*                             MODUL GAUSS                          */
/*  ***************************************************************  */

/* based on:  
G. Engeln-Muellges, F. Reutter: Formelsammlung zur numerischen  
Mathematik mit C-Programmen. Bibliographisches Institut, 
Mannheim, 1987, pp.314 - 322. */

#include <stdio.h>
#define dim_n 7  /* Dimension n */
#define MACH_EPS 2.220446049250313e-016 /* Maschinengenauigkeit */
/* IBM-AT: = 2 hoch -52 */
/* MACH_EPS ist die kleinste positive auf der Maschine darstellbare  */
/* Zahl x, die der Bedingung genuegt: 1.0 + x > 1.0 */
/* */
/* Definition von Funktionsmakros: */
#define max(X, Y) { (X) > (Y) ? (X) : (Y) }  /* Maximum von X,Y */

int main()  
{
    int i, n;
    double matrix[dim_n][dim_n],
            lumat[dim_n][dim_n],
            perm[dim_n],
            b[dim_n],
            x[dim_n],
            Wpp[dim_n][dim_n],
            Wppi[dim_n][dim_n],
    double det;

    det = 1.0;
    for (i = 1; i < dim_n; i++)
        det = det * Wppi[i][i];

    for (i = 0; i < dim_n; i++)
        x[i] = Wppi[i][i] / det;

    return 0;
}
/* Steuerparameter: */
/* --------------------- */
/* cas int cas; */
/* Aufrufart von gaudec: */
/* - 0 Bestimmung der Zerlegungsmatrix und Berechnung */
/* der Lösung des Gleichungssystems. */
/* - 1 Nur Berechnung der Zerlegungsmatrix lumat. */
/* - 2 Nur Lösung des Gleichungssystems; zuvor muss je- */
/* doch die Zerlegungsmatrix bestimmt sein. Diese */
/* Aufrufart wird verwendet, falls bei gleicher */
/* Matrix lediglich die rechte Seite des Systems vari- */
/* iert, z. B. zur Berechnung der Inversen. */
/* Eingabeparameter: */
/* --------------------- */
/* n int n: (n > 1) */
/* Dimension von matrix und lumat, */
/* Anzahl der Komponenten des b-Vektors, des Lö- */
/* sungsvektors x, des Permutationsvektors perm. */
/* matrix double matrix[n][n]; */
/* Matrix des Gleichungssystems. Diese wird als Vektor */
/* von Zeigern uebergeben. */
/* lumat double lumat[n][n]; */
/* (bei cas = 2) */
/* LU-Dekompositionsmaatrix, die die Zerlegung von */
/* matrix in eine untere und obere Dreiecksmatrix ent- */
/* haelt. */
/* perm int perm[n]; */
/* (bei cas = 2) */
/* Permutationsvektor, der die Zeilenvertauschungen */
/* von lumat enthält. */
/* b double b[n]; */
/* (bei cas = 0, 2) */
/* Rechte Seite des Gleichungssystems. */
/* signdet int *signdet; */
/* (bei cas = 2) */
/* Vorzeichen der Determinante von matrix; die De- */
/* terminante kann durch das Produkt der Diagonal- */
/* elemente mal signdet bestimmt werden. */
/* Ausgabeparameter: */
/* --------------------- */
/* lumat double lumat[n][n]; */
/* (bei cas = 0, 1) */
/* LU-Dekompositionsmaatrix, die die Zerlegung von */
/* matrix in eine untere und obere Dreiecksmatrix ent- */
/* haelt. */
/* perm int perm[n]; */
/* (bei cas = 0, 1) */
/* Permutationsvektor, der die Zeilenvertauschungen */
/* von lumat enthält. */
/* x double x[n]; */
/* (bei cas = 0, 2) */
/* Lösungsvector des Systems. */
/* signdet int *signdet; */
/* (bei cas = 0, 1) */
/* Vorzeichen der Determinante von matrix; die De- */
/* terminante kann durch das Produkt der Diagonal- */
/* elemente mal signdet bestimmt werden. */
/* Rückgabewert: */
/* --------------------- */
/* = 0 alles ok */
/* = 1 n < 2 gewählt oder unsinnige Eingabeparameter */

int gaudec(int n, double matrix[dim_n][dim_n], double lumat[dim_n][dim_n],
            int perm[dim_n], int *signdet) {
    /* gaudec: Bestimmt die LU-Zerlegung einer n x n Matrix */
    /* und berechnet die Lösung des Gleichungssystems. */
    if (n < 2) return(1);
    switch (cas) {
        case 0: if (res == 0) return(gausol(n, lumat, perm, b, x));
            else return(res);
        case 1: return(gausol(n, lumat, perm, signdet));
        case 2: return(gausol(n, lumat, perm, b, x));
    }
    return(5); /* Falsche Aufrufart */
}

4 Listings for Maple V554.5 Generation of Code for SIMULINK
/* Ausgabeparameter: */
/* --------------- */
/* lumat double lumat[n][n]; */
/* LU-Dekompositionsmatrix, die die Zeerlegung von */
/* matrix in eine untere und obere Dreiecksmatrix ent- */
/* haelt. */
/* perm int perm[n]; */
/* Permutationsvektor, der die Zeilenvertauschungen */
/* von lumat enthaelt. */
/* signdet int *signdet; */
/* Vorzeichen der Determinante von matrix; die De- */
/* terminante kann durch das Produkt der Diagonal- */
/* elemente mal signdet bestimmt werden. */
/* Rueckgabewert: */
/* = 0 alles ok */
/* = 1 n < 2 gewahlt oder unzulaessige Eingabeparameter */
/* = 2 zu wenig Speicherplatz */
/* = 3 Matrix ist singulaer */
/* = 4 Matrix rechnerisch singulaer */
/* Benutzte Funktionen: */
/* Aus der C Bibliothek: fabs() */
/* Benutzte Konstanten: NULL, MACH_EPS */
/* Makros: max */
/*====================================================================*/

int j0;
register k, j, m;
double piv, temp, d[dim_n], zmax;
double fabs();

if ( &n == NULL || n < 2 ) return(1); /* Unzulaessige Parameter */
    /* d = Skalierungsvector */
    /* fuer Pivotsuche */
if ( lumat != matrix ) /* Falls lumat u. matrix ver- */
    /* schieden gewaehlt sind, */
    for (j = 0; j < n; j++) /* kopiere matrix auf lumat. */
        lumat[k][j] = matrix[k][j];

for (k = 0; k < n; k++) { /* Initialisiere perm */
    perm[k] = k;
    for (zmax = 0.0, j = 0; j < n; j++)
        zmax = max(zmax, fabs(lumat[k][j]));
    if ( zmax == 0.0 ) return(3); /* matrix singulaer */
    d[k] = zmax;
}

*signdet = 1; /* Vorzeichen der Determinante */

for (k = 0; k < n-1; k++) { /* Schleife ueber alle Zeilen */
    piv = fabs(lumat[k][k]) / d[k];
    j0 = k;
    for (j = k+1; j < n; j++) { /* Suche aktuelles Pivotelem. */
        temp = fabs(lumat[j][k]) / d[j];
        if ( piv < temp ) {
            piv = temp; /* Merke Pivotelement u. */
            j0 = j; /* dessen Index */
        }
    }
    if ( piv < MACH_EPS ) { /* Wenn piv zu klein, so ist */
        *signdet = 0; /* matrix nahezu singulaer */
        return(4);
    }
    if ( j0 != k ) {
        *signdet = - *signdet; /* Vorzeichen Determinante */
        m = perm[j0]; /* Tausche Eintraege im Pivot- */
        perm[j0] = perm[k]; /* vektor */
        perm[k] = m;
        temp = d[j0]; /* Tausche Eintraege im */
        d[j0] = d[k]; /* Skalierungsvector */
        d[k] = temp;

        for (j = 0; j < n; j++) /* Tausche j0-te und k-te Zeile */
            temp = lumat[j][j0];
        lumat[j][j0] = lumat[k][j];
        lumat[k][j] = temp;
    }

for (j = k+1; j < n; j++) /* Gauss Eliminationsschritt */
    if ( lumat[j][k] == 0.0 ) {
        lumat[j][k] /= lumat[k][k];
        for (temp = lumat[j][k], m = k+1; m < n; m++)
            lumat[j][m] -= temp * lumat[k][m];
    }
}

/* end k */

if ( fabs(lumat[n-1][n-1]) < MACH_EPS ) {
    *signdet = 0;
    return(4);
}
return(0);

int gausol(int n, double lumat[dim_n][dim_n], int perm[dim_n],
        double b[dim_n], double x[dim_n])
    /*====================================================================*/
    /* gaussol bestimmt die Losung x des linearen Gleichungssystems */
    /* lumat * x = b mit der n x n Koeffizientenmatrix lumat, wobei */
/* lumat in zerlegter Form (LU-Dekomposition) vorliegt, wie          */
/* sie von gaudec als Ausgabe geliefert wird.                       */
/*--------------------------------------------------------------------*/
/* Eingabeparameter:                                                 */
/* ==============================================================*/
/* n int n; ( n > 1 )                                              */
/* Dimension von lumat,                                            */
/* Anzahl der Komponenten des b-Vektors, des Las-                 */
/* sungvektors x, des Permutationsvektors perm.                   */
/* lumat double lumat[n][n];                                       */
/* LU-Dekompositionsvektoren, wie sie von gaudec                   */
/* geliefert wird.                                                */
/* perm int perm[n];                                                 */
/* Permutationsvektor, der die Zeilenvertauschungen                */
/* von lumat enthält.                                              */
/* b double b[n];                                                   */
/* Rechte Seite des Gleichungssystems.                            */
/* Ausgabeparameter:                                                */
/* ==============================================================*/
/* x double x[n];                                                   */
/* Lösungsvektor des Systems                                       */
/* Rueckgabewert:                                                   */
/* ==============================================================*/
/* = 0 alles ok                                                     */
/* = 1 n < 2 gewählt oder unzulaessige Eingabeparameter             */
/*--------------------------------------------------------------------*/

/* Benutzte Konstanten: NULL                                    */
/*--------------------------------------------------------------------*/

{  register j, k;
    double sum;
    if ( &n == NULL || n < 2 ) return(1); /* Unzulässige Parameter */
    if ( b == NULL || perm == NULL ) return(1);
    x[0] = b[perm[0]]; /* Vorwärtselimination */
    for (k = 1; k < n; k++)
        for (x[k] = b[perm[k]], j=0; j < k; j++)
            x[k] -= lumat[k][j] * x[j];
    x[n-1] /= lumat[n-1][n-1]; /* Rückwärtselimination */
    for (k = n-2; k >= 0; k--)
        for (sum = 0.0, j = k+1; j < n; j++)
            sum += lumat[k][j] * x[j];
        x[k] = (x[k] - sum) / lumat[k][k];
    return(0);  
}
/*------------------------- ENDE GAUSS -------------------------------*/
/* ****************************************************************** */
5 Example

5.1 State Feedback Controller

The only $L_2$ gain problem with a numerical solution we have found in literature is used as an example to test the program. It is the Nonlinear Benchmark Problem proposed by Bupp et al. [5] for which in [16] an $H_{\infty}$ state-feedback controller is reported. A cart of mass $M$, constrained to move along a horizontal line and fixed to a wall by a spring of stiffness $k$, is stabilized by a rotational proof mass actuator (Fig. 5.1.1). The proof mass has mass $m$ and moment of inertia $I$ about its center of mass which is located a distance $e$ from the axis of rotation. Our control signal is the torque $N$, and $F$ is an external disturbance force acting on the cart.

![Figure 5.1.1: Translational oscillator with rotational actuator.](image)

After normalization, the equations of motion are given by

\[
\begin{align*}
\ddot{\xi} + \xi &= \varepsilon(\dot{\Theta}^2 \sin \Theta - \dot{\Theta} \cos \Theta) + w \\
\dot{\Theta} &= -\varepsilon \ddot{\xi} \cos \Theta + u
\end{align*}
\]

(5.1.1)

[5], where $\xi$ is the nondimensional translational position, $w$ and $u$ denote the scaled disturbance and torque, respectively, and $\varepsilon$ describes the coupling between the translational and the rotational motions:

\[
\varepsilon = \frac{me}{\sqrt{(I + me^2)(M + m)}}.
\]

Equation (5.1.1) written in the form required for the Maple program of the previous section reads

\[
\begin{bmatrix}
\dot{\xi} \\
\dot{\xi} \\
\dot{\theta} \\
\dot{\theta}
\end{bmatrix} =
\begin{bmatrix}
\dot{\xi} \\
\dot{\xi} \\
\dot{\Theta} \\
\dot{\Theta}
\end{bmatrix} =
\begin{bmatrix}
\dddot{\xi} + \xi \\
-\dddot{\xi} + \varepsilon \dot{\Theta}^2 \sin \Theta - \dot{\Theta} \varepsilon \cos \Theta + w \\
1 - \varepsilon^2 \cos^2 \Theta \\
\Theta \\
\varepsilon \cos \Theta(\dddot{\xi} - \dot{\Theta}^2 \sin \Theta) + u - w \varepsilon \cos \Theta \\
1 - \varepsilon^2 \cos^2 \Theta
\end{bmatrix}.
\]

(5.1.2)
As in [16], the following performance criterion is considered:

\[
\begin{bmatrix}
\sqrt{0.1}\xi \\
\sqrt{0.1}\xi \\
\sqrt{0.1}\Theta \\
u
\end{bmatrix}
\]  

(5.1.3)

With the parameter \(\epsilon = 0.5\) and the bound \(\gamma = 3\) for the \(L_2\) gain, the Maple commands for calculating approximations of the optimal \(u\) and \(w\) read as follows:

```maple
> restart;
with(linalg):
read `aresolve`:
read `hinfSF`;

> epsilon := 1/2:
x := vector(4):
X := vector([x[2],
(-x[1]+epsilon*x[4]^2*sin(x[3]) - (epsilon*cos(x[3])*u) + w) /
(1-epsilon^2*cos(x[3])^2),
x[4],
(epsilon*cos(x[3])*(x[1]-epsilon*x[4]^2*sin(x[3])) + u -
(epsilon*cos(x[3])*w)) / (1-epsilon^2*cos(x[3])^2))]);
Z := vector([seq(sqrt(0.1)*x[i], i = 1..4), u]);
G := [X, Z];
var := [x, vector([w]), vector([u])];

> printlevel:=1;
Gamma := 3;
apxord := 5;
result := hinfSF(G, var, apxord, Gamma,3);
```

As mentioned in [16], \(V^{(i)}\) is zero for uneven \(i\) whence \(u^{(i)}\) and \(w^{(i)}\) are zero for even \(i\). The terms of order 1 and 3 for \(u_s\) and \(w_s\) are exactly the same as those reported in [16] (top: \(u_s\), bottom: \(w_s\)):
The same is true for the simulations with the controller (carried out in MATLAB/SIMULINK) if terms up to third order are used (Fig. 5.1.2 as compared to Figures 2 to 4 in [16]). With the linear approximation $u_{s}^{(1)}$, the trajectory starting at $[5 \ 5 \ -2 \ 2]^T$ tends to a limit cycle while with $u_{s}^{(1)} + u_{s}^{(3)}$, it tends to the origin. With $u_{s}^{(1)} + u_{s}^{(3)} + u_{s}^{(5)}$, however, the trajectory is unstable. This is not surprising since the approximation of trigonometric functions by power series is not very good for large arguments (Fig. 5.1.3).

**Figure 5.1.2: Simulations with initial condition $[5 \ 5 \ -2 \ 2]^T$.**

**Figure 5.1.3: sin(x) and its approximation $x - \frac{1}{6} x^3 + \frac{1}{120} x^5$.**
As Figure 5.1.4 shows, for considerably smaller initial conditions \((1.5 1.5 -0.6 0.6)^T\), a simulation with \(u^{(1)} + u^{(3)} + u^{(5)}\) is stable. Again, the control signals including these fifth-order terms have the largest magnitude. The trajectories do not tend to the origin faster than with \(u^{(1)} + u^{(3)}\) alone.

No general conclusion should be drawn from this example since it is not clear at all why it should be reasonable to penalize all four state variables in the same way, as is done here. If \(Z(x, w, u)\) in expression (5.1.3) had been chosen differently, more reasonable control signals could result. In fact, it must be expected that the plant has to be augmented with dynamic weighting functions as in the linear case [6]. The investigation of such weightings for \(L_2\) gain synthesis is the topic of on-going research.
5.2 Output Feedback Controllers

The example for the output feedback case is constructed out of the previous example by feeding back only the two positions $\xi$ and $\Theta$ ($x_1$ and $x_3$). In order to satisfy all the assumptions, additional exogenous inputs must be introduced. The equations defining the plant now read:

\[
\begin{bmatrix}
\dot{\xi} \\
\dot{\xi} \\
\dot{\Theta} \\
\dot{\Theta}
\end{bmatrix} = \begin{bmatrix}
-\frac{\xi + \varepsilon \Theta^2 \sin \Theta - u \varepsilon \cos \Theta + w_1}{1 - \varepsilon^2 \cos^2 \Theta} & \xi \\
\Theta & 1 - \frac{\varepsilon \cos \Theta (\xi - \varepsilon \Theta^2 \sin \Theta) + u - w_1 \varepsilon \cos \Theta}{1 - \varepsilon^2 \cos^2 \Theta} & \xi \\
\end{bmatrix} + \alpha w_2 \]

\[
z = \begin{bmatrix} \sqrt{0.1} x_1 \\ \sqrt{0.1} x_2 \\ \sqrt{0.1} x_3 \\ \sqrt{0.1} x_4 \\ u \end{bmatrix} \]

\[
y = \begin{bmatrix} x_1 + \alpha w_2 \\ x_3 + \alpha w_3 \end{bmatrix}
\]

where $\alpha$ is a small number here chosen to be $\alpha = 0.0001$.

For the computation of the controllers, the additional term $\Phi(x) = \varphi W_x(x) W_x^T(x)$ of the second HJI equation introduced on page 23 is of crucial importance. With $\varphi = 10^{-8}$, some of the entries in the solution $P^{-1}$ of the Riccati equation (3.2.7) have an order of magnitude of $10^{-7}$. Hence, the second-order approximation of $W(x)$ has coefficients as large as $10^7$, while those of higher orders are even larger. The coefficients of the factors $L(x)$ and $R(x)$ of the output injection matrix $G(x)$ (equation (2.3.14)) are of the same magnitude, which obviously is not reasonable for any simulation. Only by increasing $\varphi$ considerably to $\varphi = 10^{-2}$, coefficients of an acceptable magnitude are obtained which allow for the simulations shown below.

The term $\Theta(x)$ corresponds to an additional exogenous input affecting all of the state variables. The larger $\varphi$ is chosen, the more uncertain the input of the plant relative to its measured output becomes. Hence, the reconstruction of the state must depend on the output injection rather than on the propagation of the input. The dynamics of the observer will be faster.
The following Maple commands are used to compute the controller of third approximation order:

```maple
# TORA Example
restart;
alias(hinfOF=hinfOF_x);
with(linalg):
read `aresolve`:
read `hinfOF`;
read `wrtMEXctrl`:
read `wrtMEXfile`:
read `lyapsolve`:
read `sysbal`:
Digits := 28;
epsilon := 1/2:
x := vector([x1, x2, x3, x4]):
w := vector(3):
X := vector([x[2],
            (-x[1]+epsilon*x[4]^2*sin(x[3]) - (epsilon*cos(x[3])*u) + w[1]) /
            (1-epsilon^2*cos(x[3])^2) + 0.0001*w[2],
            x[4],
            (epsilon*cos(x[3])*(x[1]-epsilon*x[4]^2*sin(x[3])) + u -
            (epsilon*cos(x[3])*w[1])) / (1-epsilon^2*cos(x[3])^2))):
Y := vector([x[1]+0.0001*w[2], x[3]+0.0001*w[3]]):
Z := vector([seq(sqrt(0.1)*x[i], i = 1..4), u]):
G := [X, Z, Y]:
var := [x, w, vector([u])]:
Gamma := 3;
approxord := 3;
psi := 0.;
phi := 1e-2;
orderOnly := true;
balance := false:
Name:=`TORAContr`.approxord.`xu`;
if true then
    K := hinfOF(G, var, approxord, Gamma,2, 1e-10, psi, phi, orderOnly, balance, Name):
else
    K := hinfOF(G, var, approxord, Gamma,2, 1e-10, psi, phi, orderOnly, balance):
    wrtMEXfile(K[1][1], K[1][2], Name);
fi;
cmd:=`mv `.Name.`.c ../TORA/`.Name.`.c`;
system(cmd);
```

In the third line of the listing, an alias is defined for hinfOF_x. This alias allows to conveniently switch to hinfOF_p by just replacing the x by a p in that line. The name for the file to which the controller is written is constructed to include the approximation order (for this third order example, TORAConstr3xu results). The last two lines move the controller file to the directory TORA, where the SIMULINK environment for the simulation is stored. These two lines only work within the UNIX operating system.
As before, controllers of first, third, and fifth approximation order are computed and compared in simulations. The initial conditions are set to $0.4 \cdot [1.5 \ 1.5 \ -0.6 \ 0.6]^T$ (Fig. 5.2.1).

The controllers for these simulations are computed with the first approach (approximation in $x$). For the higher approximation orders, the state space descriptions of the controllers can no longer be computed since the inversion for the computation of the output injection gain cannot be computed symbolically. Hence, the inversions are computed on-line.

Figure 5.2.1: Simulations with initial condition $0.4 \cdot [1.5 \ 1.5 \ -0.6 \ 0.6]^T$.
Controller approximated in $x$. 
Simulations with controllers based on the alternative approach (approximation in $p$) are shown in Figure 5.2.2.

**Figure 5.2.2: Simulations with initial condition $0.4\cdot[1.5\ 1.5\ -0.6\ 0.6]^T$. Controller approximated in $p$.**
Appendix: Derivatives

Derivative of a scalar $f(x)$ with respect to the vector $x \in \mathbb{R}^n$:

$$\frac{\partial}{\partial x} f(x) = \left[ \frac{\partial}{\partial x_1} f(x) \quad \frac{\partial}{\partial x_2} f(x) \quad \cdots \quad \frac{\partial}{\partial x_n} f(x) \right]$$

$\frac{\partial}{\partial x} f(x)$ may also be written as $f_x(x)$.

Derivative of a vector $F(x) \in \mathbb{R}^m$ with respect to the vector $x \in \mathbb{R}^n$:

$$\frac{\partial}{\partial x} F(x) = \left[ \frac{\partial}{\partial x_1} F_1(x) \quad \frac{\partial}{\partial x_2} F_1(x) \quad \cdots \quad \frac{\partial}{\partial x_n} F_1(x) \\
\frac{\partial}{\partial x_1} F_2(x) \quad \frac{\partial}{\partial x_2} F_2(x) \quad \cdots \quad \frac{\partial}{\partial x_n} F_2(x) \\
\vdots \quad \vdots \quad \ddots \quad \vdots \\
\frac{\partial}{\partial x_1} F_m(x) \quad \frac{\partial}{\partial x_2} F_m(x) \quad \cdots \quad \frac{\partial}{\partial x_n} F_m(x) \right] \in \mathbb{R}^{m \times n}$$

Hessian: second derivative of $f(x)$ with respect to $x$:

$$\frac{\partial^2}{\partial x^2} f(x) = \left[ \frac{\partial^2}{\partial x_1^2} f(x) \quad \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_2} f(x) \quad \cdots \quad \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_n} f(x) \\
\frac{\partial}{\partial x_2} \frac{\partial}{\partial x_1} f(x) \quad \frac{\partial^2}{\partial x_2^2} f(x) \quad \cdots \quad \frac{\partial}{\partial x_2} \frac{\partial}{\partial x_n} f(x) \\
\vdots \quad \vdots \quad \ddots \quad \vdots \\
\frac{\partial}{\partial x_n} \frac{\partial}{\partial x_1} f(x) \quad \frac{\partial}{\partial x_n} \frac{\partial}{\partial x_2} f(x) \quad \cdots \quad \frac{\partial^2}{\partial x_n^2} f(x) \right]$$

Second derivative of a scalar $g(x, y)$ with respect to the vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$:

$$\frac{\partial}{\partial y} \frac{\partial}{\partial x} g(x, y) = \left[ \frac{\partial}{\partial y_1} \frac{\partial}{\partial x_1} g(x, y) \quad \frac{\partial}{\partial y_1} \frac{\partial}{\partial x_2} g(x, y) \quad \cdots \quad \frac{\partial}{\partial y_1} \frac{\partial}{\partial x_n} g(x, y) \\
\frac{\partial}{\partial y_2} \frac{\partial}{\partial x_1} g(x, y) \quad \frac{\partial}{\partial y_2} \frac{\partial}{\partial x_2} g(x, y) \quad \cdots \quad \frac{\partial}{\partial y_2} \frac{\partial}{\partial x_n} g(x, y) \\
\vdots \quad \vdots \quad \ddots \quad \vdots \\
\frac{\partial}{\partial y_m} \frac{\partial}{\partial x_1} g(x, y) \quad \frac{\partial}{\partial y_m} \frac{\partial}{\partial x_2} g(x, y) \quad \cdots \quad \frac{\partial}{\partial y_m} \frac{\partial}{\partial x_n} g(x, y) \right]$$

$$= \left( \frac{\partial}{\partial y} \frac{\partial}{\partial x} g(x, y) \right)^T$$

$$= \left( \frac{\partial}{\partial x} \frac{\partial}{\partial y} g(x, y) \right)^T$$

$$y^T \left( \frac{\partial}{\partial y} \frac{\partial}{\partial x} g(x, y) \right) x = x^T \left( \frac{\partial}{\partial x} \frac{\partial}{\partial y} g(x, y) \right) y$$
Chain and product rules
Derivative of $z^T A x = (z^T A x)^T = x^T A^T z$:

$$\frac{\partial}{\partial x} z^T A x = z^T A$$

Derivative of $F^T(x)$:

$$\frac{\partial}{\partial x} F^T(x) = \left( \frac{\partial}{\partial x} F(x) \right)^T$$

Derivative of $H(F(x)) \in \mathbb{R}^p$:

$$\frac{\partial}{\partial x} H(F(x)) = \left( \frac{\partial}{\partial F} H(F) \right) \left( \frac{\partial}{\partial x} F(x) \right) \in \mathbb{R}^{p \times n}$$

Derivative of $G^T(x) F(x)$ with $G(x) \in \mathbb{R}^m$:

$$\frac{\partial}{\partial x} G^T(x) F(x) = \left( \frac{\partial}{\partial G} (G^T(x) F(x)) \right) \frac{\partial}{\partial x} G(x) + \left( \frac{\partial}{\partial F} (G^T(x) F(x)) \right) \frac{\partial}{\partial x} F(x)$$

$$= \left( \frac{\partial}{\partial G} (F^T(x) G(x)) \right) \frac{\partial}{\partial x} G(x) + G^T(x) \frac{\partial}{\partial x} F(x)$$

$$= F^T(x) \frac{\partial}{\partial x} G(x) + G^T(x) \frac{\partial}{\partial x} F(x)$$

Derivative of $y^T F(x)$ with respect to $x$ and $y$:

$$y^T \left( \frac{\partial}{\partial y \partial x} y^T F(x) \right) = \left( \frac{\partial}{\partial x} \frac{\partial}{\partial y} F^T(x) y \right) y$$

$$= \left( \frac{\partial}{\partial x} F^T(x) \right) y$$

$$= \left( \frac{\partial}{\partial x} F(x) \right)^T y$$

rather than: $y^T \left( \frac{\partial}{\partial y \partial x} y^T F(x) \right) = y^T \left( \frac{\partial}{\partial y} \left( y^T \frac{\partial}{\partial x} F(x) \right) \right) = y^T \left( \frac{\partial}{\partial x} F(x) \right)^T$, which has incompatible dimensions
References


