Inversion Impossible?

Jörg J. Buchholz (Hochschule Bremen)
Wolfgang v. Grünhagen (DLR Braunschweig)

January 23, 2008

This page intentionally left blank

Contents

1 Introduction **5**

2 Transfer Function Inversion **7**

3 State-Space Inversion **10**

4 Improper Inverse **13**

5 Proper Inversion **16**

6 The MIMO Case **18**

6.1 Example 18

6.2 Numerical Inversion 19

6.3 Analytical Inversion 20

6.4 Propering Filter 21

6.5 Proper Inversion 23

6.5.1 Analytical Proper Inversion 24

6.5.2 Inverse via Limit 25

6.5.3 Root Locus 25

6.5.4 Implementation of the Proper Inverse 26

7 Unstable Systems **27**

8 Transmission Zeros **31**

9 Inversion of a Nonlinear System **35**

10 EC-135 Inversion **37**

A Matlab Function `sym2tf` **44**

List of Figures

1 Turbulence model extraction 5
2 General inversion scheme 7
3 Transfer function inversion block diagram 8
4 Transfer function inversion result 8
5 State-space inversion block diagram 12
6 Improper system inversion block diagram 14
7 Instability resulting from inadequate solver 15
8 Impact of propering poles 15
9 Proper inversion block diagram 16
10 Proper inverse result 17
11 Pole zero map of MIMO example 19
12 State-space proper inversion block diagram 23
13 Extended root locus of proper inverse 26
14 Instability 28
15 Stabilized system block diagram 29
16 Stabilized system result 30
17 Mirrored transmissions zero block diagram 32
18 Time delay resulting from all-pass 33
19 Phase shift of system, mirrored system, and mirrored system with time delay 34
20 Nonlinear system inversion block diagram 35
21 Nonlinear system inversion result 36
22 Input and output signals of EC-135 model 37
23 EC-135 inversion block diagram 41
24 EC-135 inversion result 42
25 Zoom of EC-135 inversion result 43

1 Introduction

Under the terms of the US/German MoU (Helicopter Aeromechanics) the Task IX - *Modeling and Simulation for Rotorcraft Systems* - is defined:

> ”The overall objective of this task is to improve the modeling accuracy and understanding of helicopter dynamics and control. Improved modeling and understanding of the important issues can be used to increase the fidelity of ground-based simulations, thus allowing early pilot evaluation during the development of new control systems, compatibility checks for improved safety, decreases in experimental flight testing, and hence a reduction in costs and risks.”

One of the recent subtasks under Task IX has been a disturbance rejection study [1], resulting in a UH-60 Black Hawk control equivalent turbulence simulation model [2].

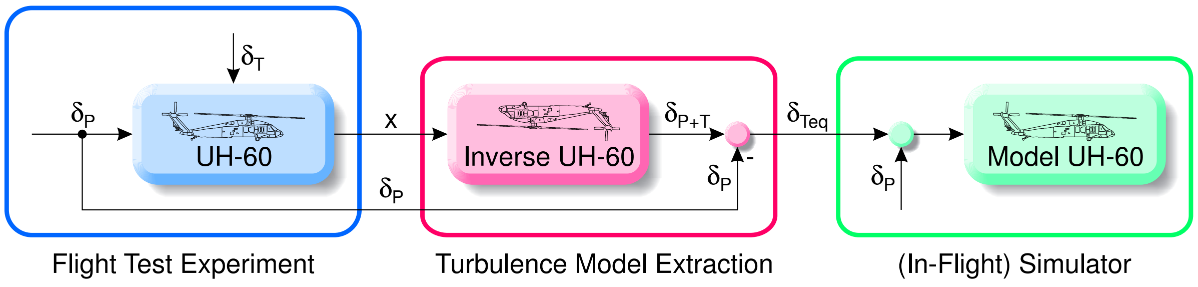

Figure 1: Turbulence model extraction

As illustrated in Figure 1 the basic idea is to have a pilot loosely stabilize a helicopter in a turbulent (input δ_T) environment (e. g. hovering on the leeward side of a high building), and measure the pilot control inputs (δ_P) and the reaction (rates, velocities, . . .) of the helicopter (x).

In the off-line extraction phase the measured reaction x (which includes the reaction of the helicopter to both the turbulence and the pilot input) is fed into an *inverse model* of the helicopter, resulting in the corresponding control input δ_{P+T} that would be necessary to produce the measured reaction. Again, δ_{P+T} includes turbulence and pilot input. If the measured pilot input δ_P is subtracted, an equivalent turbulence input δ_{Teq} remains.

This equivalent turbulence input can then directly be used as an additional control input in any kind of simulator; without the need to gain and implement more complex turbulence models.

During the actual model extraction approach [1] it became clear that

> ”Ideally, an exact numerical inverse of the coupled MIMO model would be used.”

This paper will therefore present and discuss different approaches to invert dynamical systems.

2 Transfer Function Inversion

The inversion of a general dynamic system is depicted in Figure 2

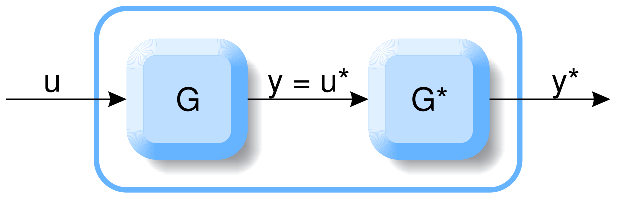

Figure 2: General inversion scheme

where u and y are the input and the output of the system G to be inverted, while $u^\star$ and $y^\star$ denote the input and the output of the inverse system $G^\star$. If $G^\star$ is a true dynamic inverse of G, a serial connection of both systems

$$u^\star = y \tag{1}$$

produces an identity system, in which the overall input and output are identical

$$y^\star = u. \tag{2}$$

If the system to be inverted is linear and time invariant (LTI) and has a single input and a single output (SISO) it can be represented as a transfer function

$$G(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0} = \frac{zeros}{poles} \tag{3}$$

where the roots of the numerator and denominator are called the **zeros** and the **poles** of the system respectively.

The inversion of a transfer function is a very straightforward exchange of numerator and denominator

$$G^\star(s) = \frac{1}{G(s)} = \frac{poles}{zeros} \tag{4}$$

Therefore, the roots of the system are the zeros of the inverted system and vice versa. Matlab carries out the inversion via the overloaded `inv` command:

```
>> G_tf = tf ([0.1 1], [1 1])

Transfer function:
0.1 s + 1
---------
  s + 1
```

```
>> G_star_tf = inv (G_tf)

Transfer function:
  s + 1
---------
0.1 s + 1
```

SIMULINK can then simulate the identity system (Figure 3) according to Figure 2.

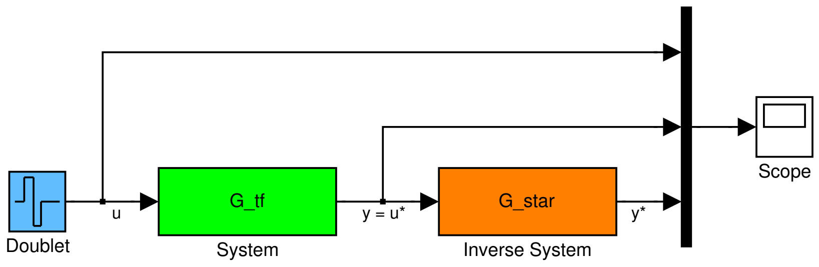

Figure 3: Transfer function inversion block diagram

Figure 4 demonstrates that, even though the doublet with its sharp edges (and the corresponding high frequency spectrum portion) poses quite a challenge for the inverter, it is regained almost perfectly.

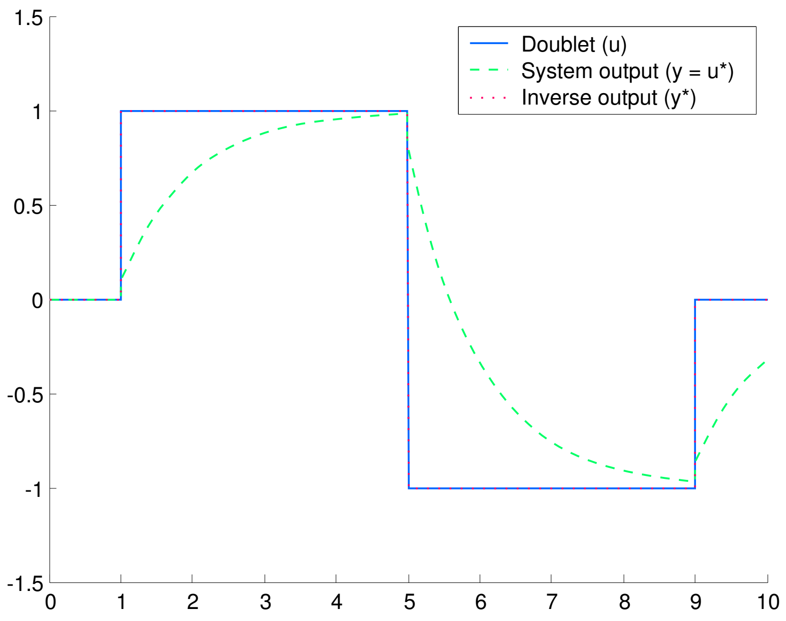

Figure 4: Transfer function inversion result

Also, the influence of the transfer function zero with the time constant of 0.1 sec can be found in the immediate ”jumping” of the system output at every doublet edge.

3 State-Space Inversion

The inversion of a dynamic system can also be performed in state-space. The state-space representation of an LTI system is given by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{5}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{6}$$

where $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^p$ are the input and the output vectors, and the state vector and its derivative are denoted by $\mathbf{x} \in \mathbb{R}^n$ and $\dot{\mathbf{x}} \in \mathbb{R}^n$. The $n \times n$ state matrix $\mathbf{A}$ holds the dynamics of the system, while the $n \times m$ input matrix $\mathbf{B}$ and the $p \times n$ output matrix $\mathbf{C}$ define the impact of every single input on the derivative and the representation of every single state in the output vector, respectively. The $p \times m$ feedthrough matrix $\mathbf{D}$ quantifies the immediate reaction of the output with respect to the input and will obtain some importance in the scope of this paper.

If the feedthrough matrix is regular, Equation 6 can be solved for $\mathbf{u}$

$$\begin{aligned} \mathbf{u} &= \mathbf{D}^{-1}(\mathbf{y} - \mathbf{C}\mathbf{x}) \\ &= -\mathbf{D}^{-1}\mathbf{C}\mathbf{x} + \mathbf{D}^{-1}\mathbf{y} \end{aligned} \tag{7}$$

which can then be substituted in Equation 5

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\left(-\mathbf{D}^{-1}\mathbf{C}\mathbf{x} + \mathbf{D}^{-1}\mathbf{y}\right) \\ &= \left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)\mathbf{x} + \mathbf{B}\mathbf{D}^{-1}\mathbf{y}. \end{aligned} \tag{8}$$

For the inverse system, the input and output have to be exchanged (Equations 1 and 2) in Equations 8 and 7

$$\dot{\mathbf{x}^\star} = \left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)\mathbf{x}^\star + \mathbf{B}\mathbf{D}^{-1}\mathbf{u}^\star \tag{9}$$
$$\mathbf{y}^\star = -\mathbf{D}^{-1}\mathbf{C}\mathbf{x}^\star + \mathbf{D}^{-1}\mathbf{u}^\star. \tag{10}$$

According to Equations 9 and 10, the matrices of the inverse system are

$$\mathbf{D}^\star = \mathbf{D}^{-1} \tag{11}$$
$$\mathbf{C}^\star = -\mathbf{D}^{-1}\mathbf{C} = -\mathbf{D}^\star\mathbf{C} \tag{12}$$
$$\mathbf{B}^\star = \mathbf{B}\mathbf{D}^{-1} = \mathbf{B}\mathbf{D}^\star \tag{13}$$
$$\mathbf{A}^\star = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C} = \mathbf{A} - \mathbf{B}^\star\mathbf{C} = \mathbf{A} + \mathbf{B}\mathbf{C}^\star. \tag{14}$$

MATLAB uses Equations 11 - 14, when the overloaded `inv` command is applied to a state-space system:

```
>> G_ss = ss (G_tf);
>> [A, B, C, D] = ssdata (G_ss)

A =
    -1
B =
     1
C =
    0.9000
D =
    0.1000

>> G_star_ss = inv (G_ss);
>> [A_star, B_star, C_star, D_star] = ssdata (G_star_ss)

A_star =
   -10
B_star =
    10
C_star =
    -9
D_star =
    10
```

In the corresponding SIMULINK block diagram (Figure 5) both systems have been explicitly expanded, offering direct access to the state vectors, proving both to be identical.

The signals in the scopes are identical to those in Figure 4.

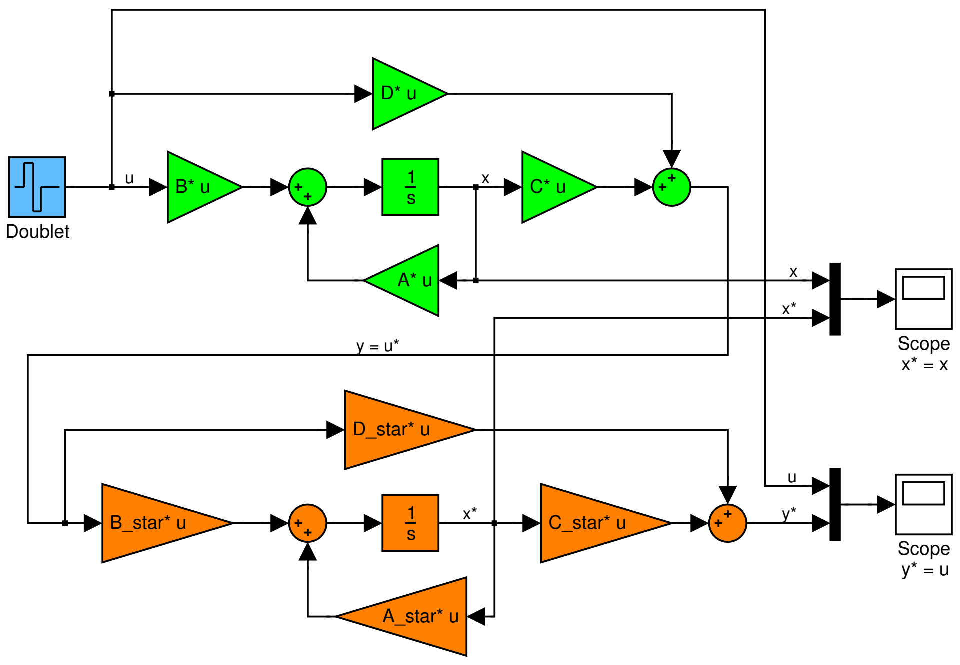

Figure 5: State-space inversion block diagram

4 Improper Inverse

The definition of the properness of a transfer function $G(s)$ depends on the relation between the degrees of its numerator (m) and denominator (n):

$n > m$: $G(s)$ is said to be **strictly proper**.

$n \geq m$: $G(s)$ is said to be **proper**.

$n = m$: Consequently, $G(s)$ is said to be proper but not strictly proper.

$n < m$: $G(s)$ is said to be (strictly) **improper**.

According to that definition, the before-used example $\frac{0.1s+1}{s+1}$ is proper but not strictly proper; its inverse having the same property. Therefore, both systems are realizable and implementable without any extra effort.

Unfortunately, many technical system representations (e. g. measuring velocities but not accelerations) are strictly proper; their inverse would therefore be improper. Improper transfer functions cannot be transformed into state-space form and they can therefore not directly be simulated in state-space-oriented simulation environments like Matlab (or Simulink):

```
>> G_tf = tf (1, [1 1])

Transfer function:
  1
-----
s + 1

>> G_star_tf = inv (G_tf)

Transfer function: s + 1

>> G_star_ss = ss (G_star_tf)
??? Error using ==> tf/ss Improper system. Conversion to
state-space is not possible

>> step (G_star_tf)
??? Error using ==> rfinputs
Not supported for non-proper models.
```

And, even though it would be possible to use a *Derivative*-, a *Gain*-, and a *Sum*-block to construct the inverse transfer function in Simulink , the usual way to circumvent the non-realizability of improper systems is to append as many high-frequency ”propering” poles as necessary to make the transfer function proper:

```
>> G_filt = tf (1, [1e-3 1])

Transfer function:
     1
-----------
0.001 s + 1

>> G_prop_star_tf = G_star_tf*G_filt

Transfer function:
   s + 1
-----------
0.001 s + 1
```

The proper inverse system is now ready to be implemented under SIMULINK to compensate the original system and regain the doublet (Figure 6).

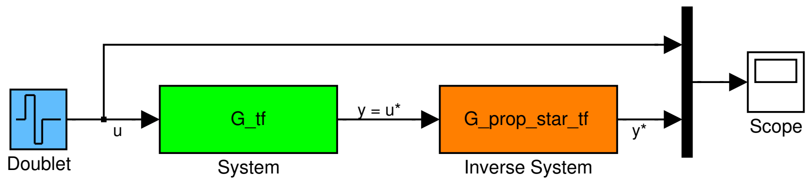

Figure 6: Improper system inversion block diagram

Using SIMULINK's default *Simulation parameters* (Solver type: Variable-step ode45, Relative tolerance: $1e-3$) the result is quite impressive (Figure 7).

This kind of instability is typical when using non-stiff solvers to solve moderately stiff systems. When appending the propering poles, a performance tradeoff has to be found: On the one hand the poles should be fast and placed far outside frequency range of interest, in order not to introduce additional significant parasitic dynamic; on the other hand, additional poles, much faster than the dynamic of the system, increase the bandwidth, make the system stiffer and therefore harder to integrate for non-stiff integration algorithms.

Fortunately, modern simulation environments supply their users with all kinds of high-sophisticated integration algorithms. For example, if the stiff solver ode15s is selected, the simulation result looks as unspectacular as Figure 4. And, if one zooms deeply in (Figure 8), the small delay caused by the propering pole becomes visible. Note that the same result could also be obtained with a fixed-step ode4 solver and a step size of 0.001.

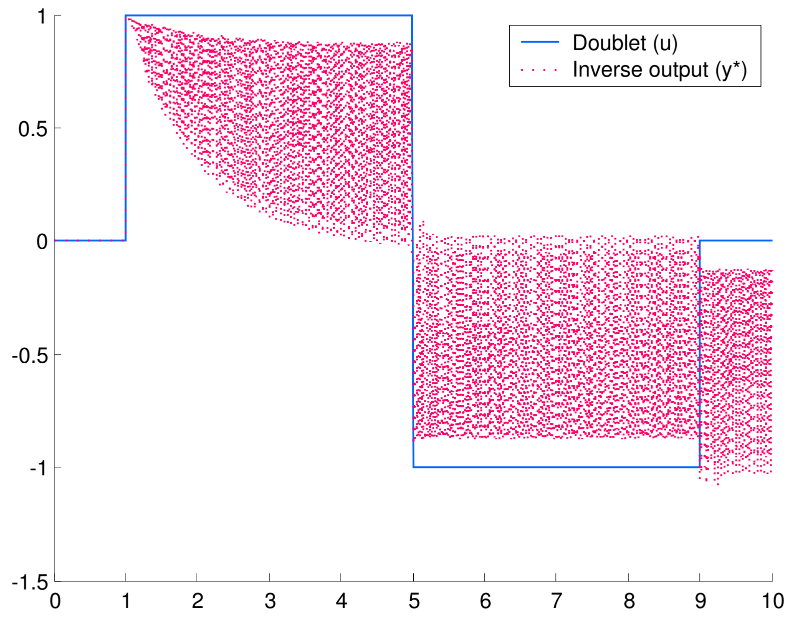

Figure 7: Instability resulting from inadequate solver

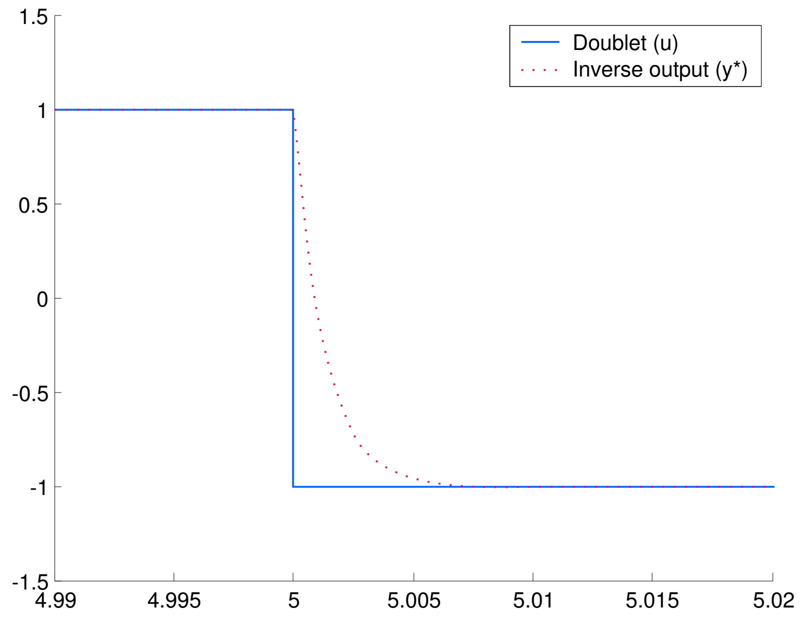

Figure 8: Impact of propering poles

5 Proper Inversion

There is another interesting approach to finding the inverse of a dynamic system, that will be proven to work with multi-input-multi-output (MIMO) systems and nonlinear systems too.

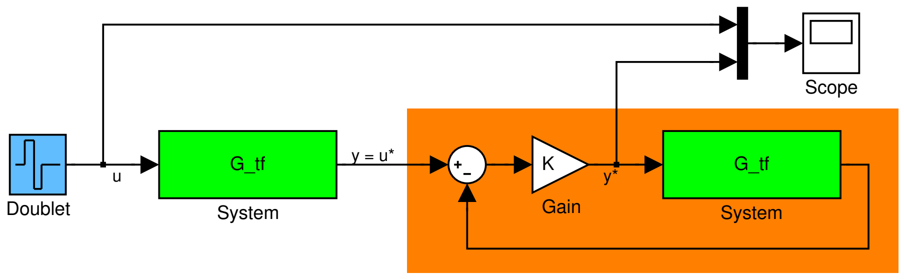

Figure 9: Proper inversion block diagram

By placing the original system to be inverted into a conventional feedback loop (Figure 9), taking the **input** of the system (`G_tf`) in the inversion block as the output of the inversion block, and using a reasonably high "controller" gain (e. g. $K = 1000$), this implicit inverter produces almost exactly the same output as the explicit inverter described in Section 4 (Figures 7 and 8).

The reason for this can be derived from the transfer function of the inverter

$$G^\star(s) = \frac{y^\star}{u^\star} = \frac{K}{1 + K \cdot G(s)} = \frac{K}{1 + K \cdot \frac{zeros}{poles}} = \frac{K \cdot poles}{poles + K \cdot zeros} \tag{15}$$

revealing that the zeros of $G^\star$ are the poles of G (always, independent of the magnitude of K) and that the poles depend on K: If K is very small, the poles of $G^\star$ "start" at the poles of G; if K increases and approaches infinity, the poles of $G^\star$ approach the zeros of G:

$$\lim_{K \to \infty} G^\star(s) = \frac{poles}{zeros} \tag{16}$$

Strictly speaking Equation 16 is a bit misleading, because it suggests that $G^\star$ has as many poles as there are zeros in G, which is not the case. For every proportional feedback gain K a system retains the same number of poles; a proper G (n number of poles $\geq$ m number of zeros) must therefore have $n - m$ poles that do not approach its zeros but move towards infinity as K increases. The additional poles added to the inverse system are highly welcome because they automatically make the system proper and therefore implementable. The position of these poles can directly be controlled by K; the before-used first order example results in almost the same proper inverse as in the explicit case of Section 4:

$$G(s) = \frac{1}{s + 1} \tag{17}$$

$$G^\star(s) = \frac{K}{1+K\cdot G(s)} = \frac{K}{1+K\frac{1}{s+1}} = \frac{K(s+1)}{s+1+K} = \frac{\frac{K}{1+K}(s+1)}{\frac{s}{1+K}+1} \tag{18}$$

For example, if K is set to 999, the proper inverse

$$G^\star(s) = \frac{999(s+1)}{s+1000} = \frac{0.999(s+1)}{0.001s+1} \tag{19}$$

has the same zero (at -1), the same propering pole (at -1000) and a slightly different stationary gain (0.999 instead of 1).

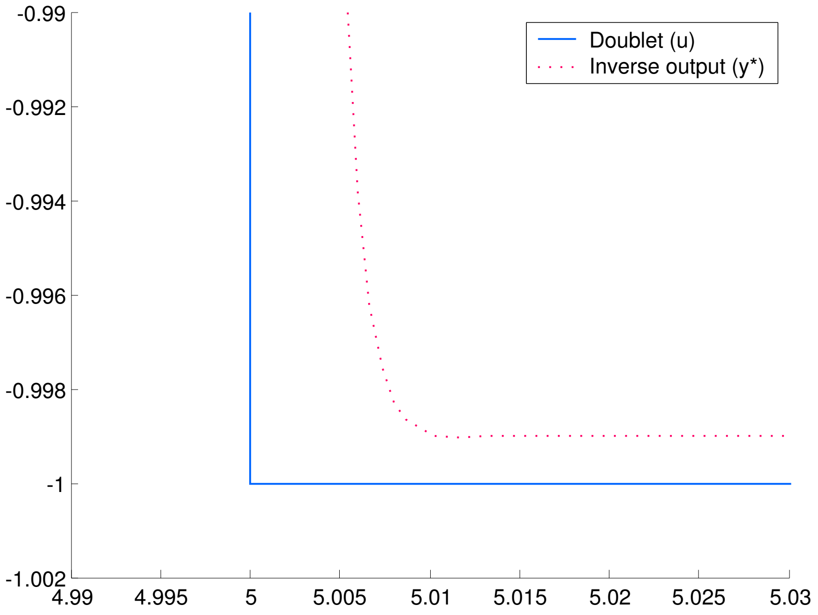

Figure 10: Proper inverse result

If an exact stationary gain is important, a static prefilter G_P

$$G(0)\cdot G_P \cdot G^\star(0) = 1 \quad \rightarrow \quad G_P = \frac{1}{G(0)\cdot G^\star(0)} = \frac{1+K\cdot G(0)}{K\cdot G(0)} \tag{20}$$

can be serialized with the inverter.

In order to simulate the details in Figure 10 correctly, not only a stiff solver (ode15s) but also a smaller relative tolerance (e. g. $1e-6$) have been chosen as simulation parameters.

Again, a tradeoff has to be found, because on the one hand high gains reduce the stationary gain errors and the dynamic impact of the propering poles, while on the other hand, they increase the bandwidth of the inverse and therefore challenge the integration algorithm.

6 The MIMO Case

If the system to be inverted has more than one input and output, its inversion becomes qualitatively more complicated.

6.1 Example

The following quadratic system, having two inputs, two outputs and three states, will serve as a MIMO example throughout this section:

$$\begin{bmatrix} \mathbf{A} & \vdots & \mathbf{B} \\ \cdots & \cdot & \cdots \\ \mathbf{C} & \vdots & \mathbf{D} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 & \vdots & 1 & 0 \\ 0 & -1 & 1 & \vdots & 1 & 0 \\ -1 & 0 & -1 & \vdots & 0 & 1 \\ \cdots & \cdots & \cdots & \cdot & \cdots & \cdots \\ 0 & 1 & 0 & \vdots & 0 & 0 \\ 0 & 0 & 1 & \vdots & 0 & 0 \end{bmatrix} \tag{21}$$

Matlab can compute the poles and transmission zeros (the MIMO-equivalent of the zeros of a transfer function) of the system:

```
>> A = [-1, 1, 0; 0, -1, 1; -1, 0, -1];
>> B = [1, 0; 1, 0; 0, 1];
>> C = [0, 1, 0; 0, 0, 1];
>> D = zeros (2,2);
>> G_ss = ss (A, B, C, D);

>> zero (G_ss)

ans =

    -1

>> pole (G_ss)

ans =

  -2.0000
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

All poles and zeros are located in the left hand side of the complex plane (Figure 11), indicating that the system as well as the inverse system are stable.

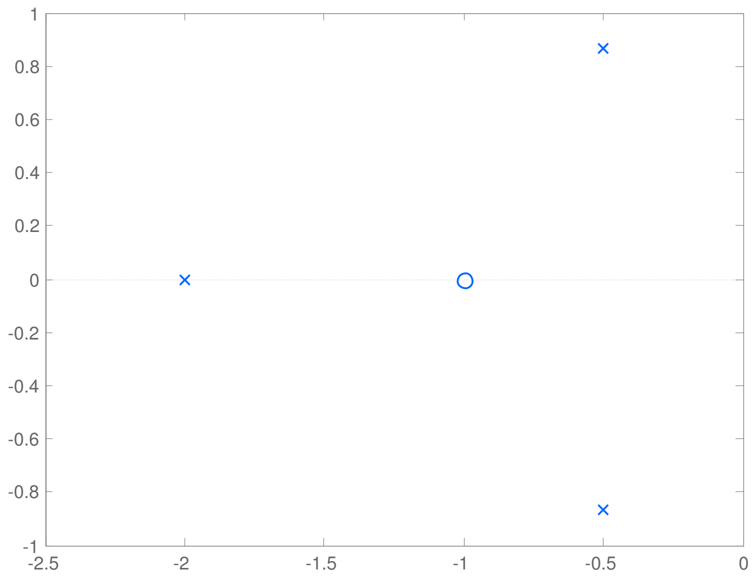

Figure 11: Pole zero map of MIMO example

6.2 Numerical Inversion

Trying to invert the system numerically via the `inv` command

```
>> G_inv_ss = inv (G_ss)
??? Error using ==> ss/inv Cannot invert system with
singular D matrix.
```

results in an error. As already stated, MATLAB uses Equations 11 - 14 to numerically invert a state-space system and Equation 11 wants to invert the feedthrough matrix D, which obviously is not possible if D is singular (or even "worse", not "present" at all, as in the example). Looking at the same problem from the transfer function matrix point of view

```
G_tf = minreal (tf (G_ss))

Transfer function from input 1 to output...
          s
 #1:   -----------
       s^2 + s + 1

         -1
 #2:   -----------
       s^2 + s + 1

Transfer function from input 2 to output...
```

```
               s + 1
 #1:  ---------------------
      s^3 + 3 s^2 + 3 s + 2

           s^2 + 2 s + 1
 #2:  ---------------------
      s^3 + 3 s^2 + 3 s + 2
```

reveals the effect of the vanishing feedthrough matrix. Every single transfer function is strictly proper, indicating that an inverse would be improper and therefore not realizable. Note that the `minreal` command has been used in order to cancel identical poles and zeros in `G_tf`.

6.3 Analytical Inversion

Fortunately, the *Symbolic Math Toolbox* of MATLAB offers a very elegant way of inverting the system anyway.

Defining a symbolic variable

```
>> syms s
```

and utilizing the Laplace transform of the state-space equations

$$s\mathbf{X(s)} = \mathbf{AX(s)} + \mathbf{BU(s)} \quad \Rightarrow \quad \mathbf{X(s)} = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{BU(s)} \tag{22}$$

$$\begin{aligned}\mathbf{Y(s)} &= \mathbf{CX(s)} + \mathbf{DU(s)} \\ &= \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{BU(s)} + \mathbf{DU(s)}\end{aligned} \tag{23}$$

$$\mathbf{G}(s) = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \tag{24}$$

the transfer function matrix can be computed analytically

```
>> G_sym = simple (C*inv (s*eye (3, 3) - A)*B + D);
>> pretty (G_sym)
```

```
                          [      s                  s + 1         ]
                          [ ----------     --------------------]
                          [  2                        2         ]
                          [ s  + s + 1     (s + 2) (s  + s + 1)]
                          [                                     ]
                          [                               2     ]
                          [       1              (s + 1)        ]
                          [- ----------     --------------------]
                          [   2                       2         ]
                          [  s  + s + 1    (s + 2) (s  + s + 1)]
```

Apparently, it is identical to the numerically computed transfer function matrix. The analytical inversion can then directly be performed via the overloaded symbolic `inv` command

```
>> G_star_sym = inv (G_sym);
>> pretty (G_star_sym)

                             [s + 1        -1    ]
                             [                   ]
                             [s + 2    s (s + 2)]
                             [-----    ---------]
                             [s + 1      s + 1  ]
```

Obviously, the product of the transfer function matrix and its inverse has to simplify to the identity matrix:

```
>> ident = simple (G_sym*G_star_sym)

ident =

[ 1, 0]
[ 0, 1]
```

6.4 Propering Filter

As expected, some entries of `G_star_sym` are improper transfer functions which preclude a straightforward implementation of the inverse.

Again, a fast first order system can be used to make the inverse proper. In this case the filter is defined and appended analytically.

```
>> T_filt_sym = 1e-3;
>> G_filt_sym = 1/(T_filt_sym*s + 1);
>> G_star_filt_sym = G_star_sym*G_filt_sym;
>> pretty (G_star_filt_sym)

              [        s + 1                         1           ]
              [     ------------             - ------------      ]
              [     1/1000 s + 1               1/1000 s + 1      ]
              [                                                  ]
              [        s + 2                   s (s + 2)         ]
              [----------------------    ----------------------]
              [(1/1000 s + 1) (s + 1)    (1/1000 s + 1) (s + 1)]
```

Unfortunately, MATLAB itself does not provide a function to convert a symbolic transfer function (matrix) to its numerical representation. Therefore, the corresponding function (`sym2tf`) had been coded and can be found in Appendix A.

```
G_star_filt_tf = sym2tf (G_star_filt_sym)

Transfer function from input 1 to output...
      1000 s + 1000
 #1:  -------------
        s + 1000

         1000 s + 2000
 #2:  -------------------
      s^2 + 1001 s + 1000

Transfer function from input 2 to output...
       -1000
 #1:  --------
      s + 1000

       1000 s^2 + 2000 s
 #2:  -------------------
      s^2 + 1001 s + 1000

G_star_filt_ss = ss (G_star_filt_tf)

a =
              x1       x2       x3
   x1      -1000        0        0
   x2          0        0   -31.25
   x3          0       32    -1001

b =
               u1        u2
   x1      -975.6   -0.9766
   x2     0.06104    -30.52
   x3      0.9766    -975.6

c =
            x1     x2     x3
   y1     1024      0      0
   y2        0      0   1024

d =
            u1     u2
   y1   1000      0
   y2      0   1000

Continuous-time model.
```

`G_star_filt_ss` can then be implemented in a block diagram like Figure 6 and, except for the inevitable time delay caused by the propering poles, the doublet is regained perfectly.

Additionally, it should be noted here, that a direct manual propering of the *improper* transfer function matrix entries only, would also be possible.

6.5 Proper Inversion

The proper inversion method described in Section 5 can also be used in the MIMO case to directly simulate the inverse without any prior calculation effort:

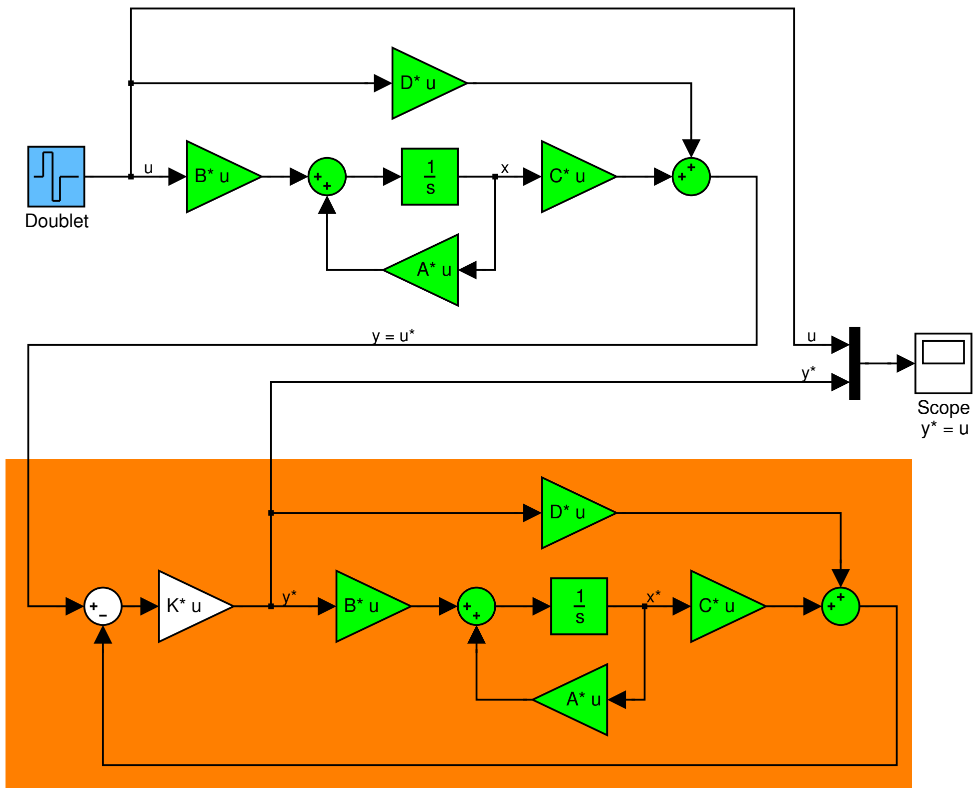

Figure 12: State-space proper inversion block diagram

6.5.1 Analytical Proper Inversion

If the system to be inverted has a nonsingular **D**-matrix (and could thus be inverted directly in state-space via Equations 11 - 14), the algebraic loop, consisting of **D**, **K**, and the two outer sums in Figure 12, can be broken (even though SIMULINK would also do that automatically during simulation) by expressing $\mathbf{y}^\star$ as a function of state, input, and itself

$$\mathbf{y}^\star = \mathbf{K}\left(\mathbf{u}^\star - \left(\mathbf{C}\mathbf{x}^\star + \mathbf{D}\mathbf{y}^\star\right)\right) \tag{25}$$

leading to the inverse output equation of

$$\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)\mathbf{y}^\star = \mathbf{K}\mathbf{u}^\star - \mathbf{K}\mathbf{C}\mathbf{x}^\star \tag{26}$$

$$\mathbf{y}^\star = -\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{C}\mathbf{x}^\star + \left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{u}^\star. \tag{27}$$

The state-space differential equation follows accordingly

$$\begin{aligned}
\dot{\mathbf{x}}^\star &= \mathbf{A}\mathbf{x}^\star + \mathbf{B}\mathbf{y}^\star & (28)\\
&= \mathbf{A}\mathbf{x}^\star + \mathbf{B}\left(-\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{C}\mathbf{x}^\star + \left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{u}^\star\right) & (29)\\
&= \left(\mathbf{A} - \mathbf{B}\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{C}\right)\mathbf{x}^\star + \mathbf{B}\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{u}^\star & (30)
\end{aligned}$$

leaving the inverse system:

$$\begin{bmatrix}\mathbf{A}^\star & \mathbf{B}^\star\\ \mathbf{C}^\star & \mathbf{D}^\star\end{bmatrix} = \begin{bmatrix}\left(\mathbf{A} - \mathbf{B}\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{C}\right) & \mathbf{B}\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\\ -\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\mathbf{C} & \left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K}\end{bmatrix}. \tag{31}$$

Two properties of this inverse system are worth mentioning:

1. If **K** approaches infinity (meaning that at least the diagonal elements of **KD** are much greater than 1), the identity matrix can be neglected, **K** can be "cancelled" and the matrix definitions become identical to those in Equations 11 - 14; e. g. :

$$\lim_{\mathbf{K}\to\infty}\mathbf{D}^\star = \lim_{\mathbf{K}\to\infty}\left(\mathbf{I} + \mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K} = \lim_{\mathbf{K}\to\infty}\left(\mathbf{K}\mathbf{D}\right)^{-1}\mathbf{K} = \lim_{\mathbf{K}\to\infty}\mathbf{D}^{-1}\mathbf{K}^{-1}\mathbf{K} = \mathbf{D}^{-1}. \tag{32}$$

2. If the system is strictly proper ($\mathbf{D} = \mathbf{0}$) the inverse system simplifies to

$$\begin{bmatrix}\mathbf{A}^\star & \mathbf{B}^\star\\ \mathbf{C}^\star & \mathbf{D}^\star\end{bmatrix} = \begin{bmatrix}\left(\mathbf{A} - \mathbf{B}\mathbf{K}\mathbf{C}\right) & \mathbf{B}\mathbf{K}\\ -\mathbf{K}\mathbf{C} & \mathbf{K}\end{bmatrix}. \tag{33}$$

   Comparing this result to Equations 11 - 14, **K** has "taken over" the function of $\mathbf{D}^{-1}$; large values of **K** have the same "propering" effect as the addition of small values to the **D**-matrices.

6.5.2 Inverse via Limit

The inverse system could therefore also analytically be computed via the limit of Equation 31:

```
>> syms k_sym
>> K_sym = [k_sym, 0; 0, k_sym];

>> D_star_sym = K_sym;
>> C_star_sym = -D_star_sym*C;
>> B_star_sym = B*D_star_sym;
>> A_star_sym = A - B_star_sym*C;

>> G_star_sym = ...
    C_star_sym* ...
    inv (s*eye (3, 3) - A_star_sym)* ...
    B_star_sym + ...
    D_star_sym;
>> pretty (limit (G_star_sym, k_sym, inf))

                                        [s + 1        -1     ]
                                        [                    ]
                                        [s + 2    (s + 2) s]
                                        [-----    ---------]
                                        [s + 1      s + 1   ]
```

Note that, for the sake of simplicity, a diagonal **K**-matrix with identical entries (`k_sym`) has been used; other regular **K**-matrices would serve the same purpose.

6.5.3 Root Locus

The extended root locus of the proper inverse system (Equation 31) can be drawn by computing the transmission zeros and poles of the system described in Section 6.5.2, and increasing `k_sym` from zero to infinity. The root locus is said to be extended because `k_sym` is not used in the single feedback loop of a SISO system but in the two entries of the MIMO feedback gain matrix **K** simultaneously. The root locus in Figure 13 therefore reveals some ”unconventional” properties, like poles ”moving to and fro” on the real axis.

As already indicated in Section 5 the transmission zeros of the inverse immediately cover the poles of the system (they are not affected by the gain and are therefore not plotted in Figure 13), while the poles of the inverse move depending on the gain. Their journey begins at the poles of the system (for `k_sym = 0`) and ends at the system's transmission

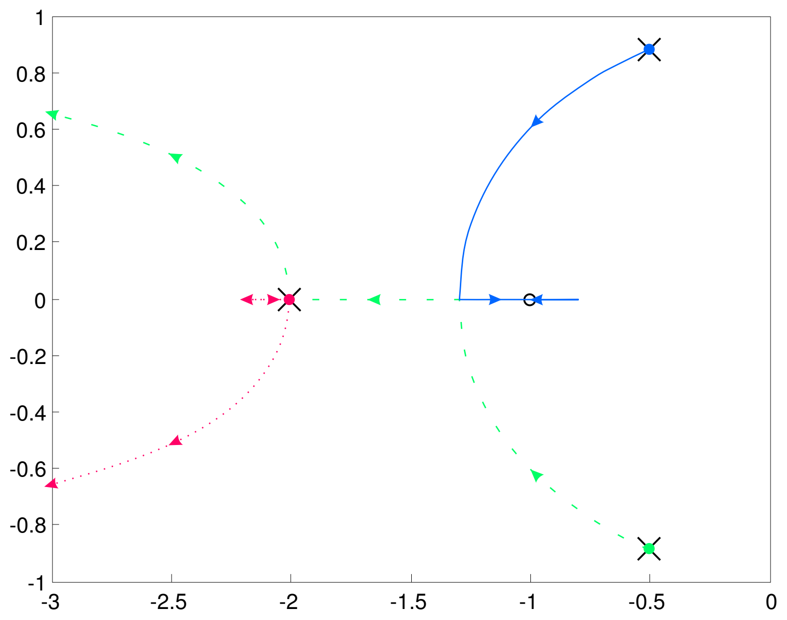

Figure 13: Extended root locus of proper inverse

zero(s) with the surplus ones approaching infinity. In the case of the example, one pole finally (for large values of `k_sym`) covers the transmission zero of the system, while the other two poles serve as the fast propering poles described in Section 5.

6.5.4 Implementation of the Proper Inverse

For the simulation of proper MIMO inverse systems no analytical or numerical precalculation is necessary. Figures 9 or 12 can be used to implement the MIMO system to be inverted directly. If a reasonable gain matrix is used, e. g.

$$\mathbf{K} = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix} \tag{34}$$

the results are almost identical to the ones in the case of the analytical inverse with additional propering poles (Section 6.4); except for the 0.1 % stationary gain error already discussed in Section 5.

7 Unstable Systems

The following example system will be used in the scope of this section to demonstrate the impact of instabilities:

```
>> A = [-1, 4, 0; 0, 0, 1; -1, 0, 0];
>> B = [0, 0; 1, 1; 0, 1];
>> C = [0, 1, 0; 0, 0, 1];
>> D = zeros (2,2);
>> G_ss = ss (A, B, C, D);

>> zero (G_ss)

ans =

   -1.0000

>> pole (G_ss)

ans =

  -2.0000
   0.5000 + 1.3229i
   0.5000 - 1.3229i
```

It has the same transmission zero and the same stable real pole as the previous example but the complex conjugate pair of poles has positive real parts now, making the system unstable.

Nevertheless, the inverse can be computed analytically:

```
>> G_sym = simple (C*inv (s*eye (3, 3) - A)*B + D);
>> pretty (G_sym)
```

```
[                    2          ]
[  s (s + 1)        s  + 2 s + 1]
[ -----------      ------------]
[  3    2           3    2      ]
[ s  + s  + 4      s  + s  + 4 ]
[                               ]
[                         2     ]
[       4          -4 + s  + s ]
[- -----------     ----------- ]
[   3    2          3    2      ]
[  s  + s  + 4     s  + s  + 4 ]
```

```
>> pretty (inv (G_sym));

                              [      2               ]
                              [-4 + s   + s          ]
                              [-----------    -s - 1]
                              [   s + 1              ]
                              [                      ]
                              [     4                ]
                              [   -----          s   ]
                              [   s + 1              ]
```

Again, propering poles could be appended to the inverse and the proper inverse could then be used in a block diagram like Figure 6 to analyze the performance of the inverse; or the instable system could directly be implemented in Figure 9, without prior inversion.

Either way, the result is devastating:

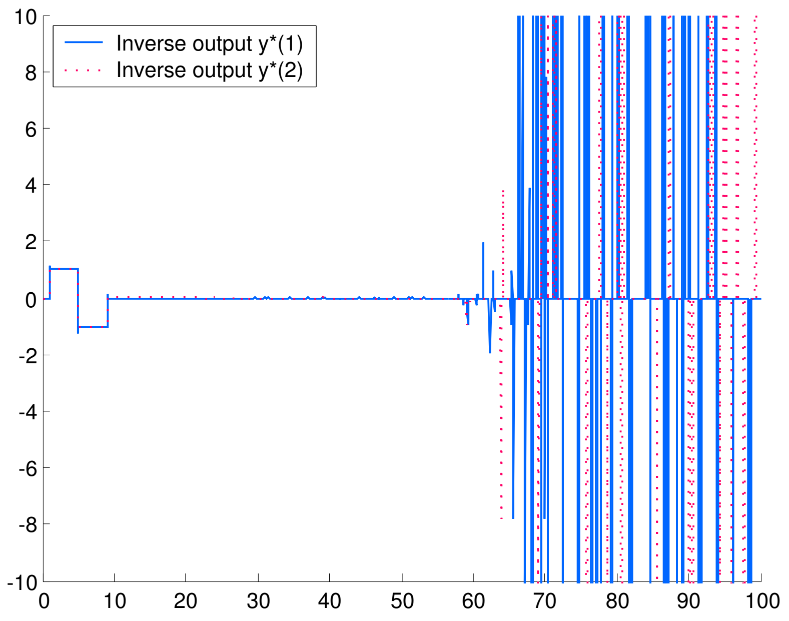

Figure 14: Instability

For a few seconds the inverse in Figure 14 seems to regain the doublet pretty well, but then small spikes grow into large ones, and around 60 sec the system explodes. It is very important to understand that this obvious instability is *not* a property of the inverse - the pole of the inverse covers the transmission zero of the system in the ”stable” left hand side half plane - but that the unstable system itself produces signals of such large amplitude (1e13) that numerical computation errors occur.

In real-world applications unstable systems are always integrated in some kind of control loop. In the flight test described in Section 1 the pilot loosely stabilizes the unstable modes of the helicopter, hovering inside a predefined target area. Figure 15 illustrates this situation by introducing an additional (not optimized) feedback gain matrix, stabilizing the system.

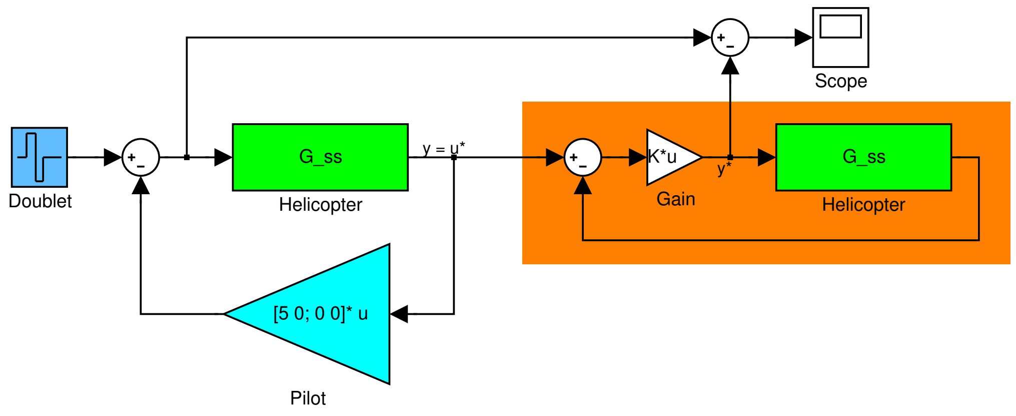

Figure 15: Stabilized system block diagram

The inverse can now prove its functionality by trying to regain the input of the system, including the feedback signal.

Figure 16 shows the error between the input of the system and the output of the inverse (Figure 15); This difference should be zero for a perfect inverse. In order to demonstrate the achievable performance of the proper inverse, a very high gain matrix

$$\mathbf{K} = \begin{bmatrix} 1e8 & 0 \\ 0 & 1e8 \end{bmatrix} \tag{35}$$

has been used, keeping the error well in the order of magnitude of 1e − 7; except for ”one sample interval long” error spikes where the doublet has its unrealistic vertical edges, that cannot numerically be regained.

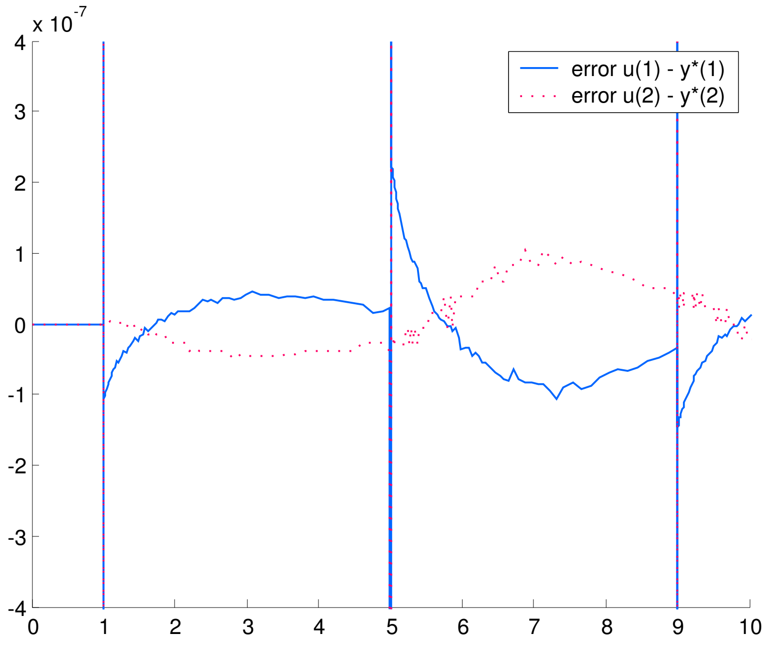

Figure 16: Stabilized system result

8 Transmission Zeros

By modifying one element of the **C**-matrix (based on the example of Section 6.1) the transmission zero moves from the left half plane to the right half plane:

```
>> A = [-1, 1, 0; 0, -1, 1; -1, 0, -1];
>> B = [1, 0; 1, 0; 0, 1];
>> C = [0, 1, 0; 11, 0, 1];
>> D = zeros (2,2);
>> G_ss = ss (A, B, C, D);

>> z = zero (G_ss)

z =

   10.0000

>> pole (G_ss)

ans =

  -2.0000
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

Since the transmission zeros of the system become the poles of the inverse, the inverse would have an unstable pole at +10 and could therefore not be simulated over a longer time period.

One interesting way to circumvent the problem with right hand side transmission zeros (RHSTZ) has been proposed in [4]. An all-pass filter with poles at the RHSTZs (and zeros at their mirror images with respect to the imaginary axis) can be defined

```
>> G_all = zpk (-z, z, -1)

Zero/pole/gain:
- (s+10)
--------
 (s-10)
```

and used as a pre-filter, in order to ”mirror” the transmission zero to the left hand side of the complex plane.

```
>> G_mirr_ss = minreal (G_all*G_ss);
1 state removed.
```

```
>> zero(G_mirr_ss)

ans =

 -10.0000 + 0.0000i
 -10.0000 - 0.0000i

>> pole(G_mirr_ss)

ans =

  10.0000
  -2.0000
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

Note that `minreal` has been used to cancel the transmission zero of the system (at $+10$) with the pole of the all-pass filter. The transmission zero has effectively been mirrored to -10. Unfortunately, a second, unnecessary pair of zero at -10 and pole at $+10$ has been introduced by the fact that the all-pass filter has been serialized with both inputs of the system. But fortunately, the inverse will therefore just have an additional stable pole at -10 and an additional transmission zero at $+10$ and will thus be stable.

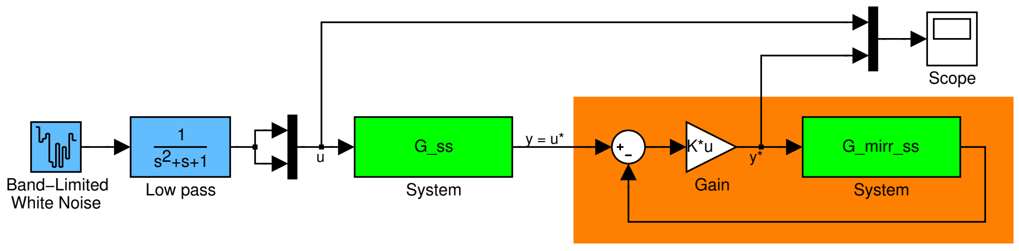

Figure 17: Mirrored transmissions zero block diagram

In Figure 17 a random (low-pass filtered) input signal has been utilized in order to demonstrate the impact of the all-pass filter more clearly.

Figure 18 illustrates that the difference between the input signal of the system and the output signal of the inverse can mainly be described as a time delay of 0.2 sec. In an off-line application this time delay could easily be compensated by delaying the input or shifting the output signal back in time by the corresponding time interval; a method definitely not suitable for real-time applications.

The time delay compensation of the all-pass can also be demonstrated in the frequency domain. By appending an appropriate time delay to the system (with the transmission

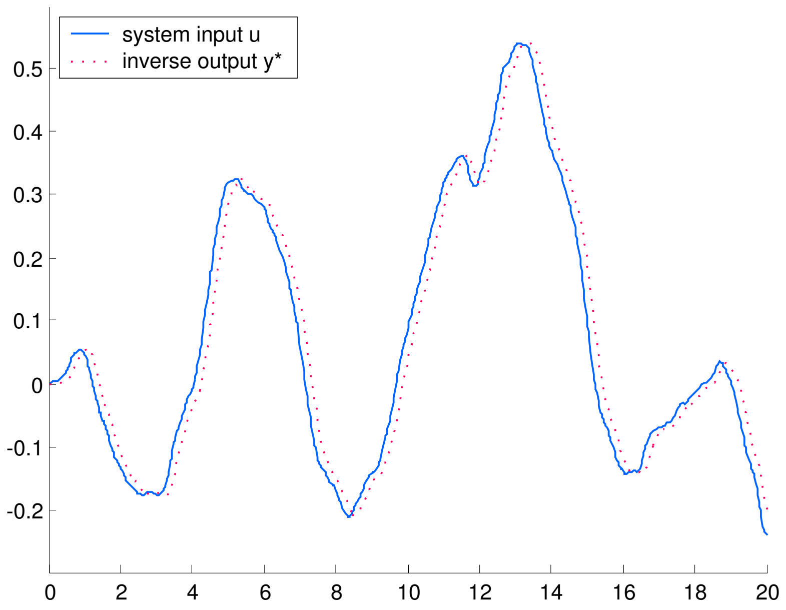

Figure 18: Time delay resulting from all-pass

zero already all-pass-mirrored to the left side)

```
>> G_mirr_del_ss = G_mirr_ss;
>> G_mirr_del_ss.iodelay = 2/z;
```

the phase plot of the mirrored and delayed system matches the one of the original system up to about $3\,{}^{rad}\!/\!_{\text{sec}}$ (Figure 19), which seems to be high enough not to interfere with the dynamics of the system. If, on the other hand, the transmission zero was located within the bandwidth of the system (e. g. $z = 1$), this ”mirror and compensate” method would not be applicable.

The magnitudes of the all-pass and the time delay both equal 1 for all frequencies and therefore do not alter the magnitude of the system or the inverse.

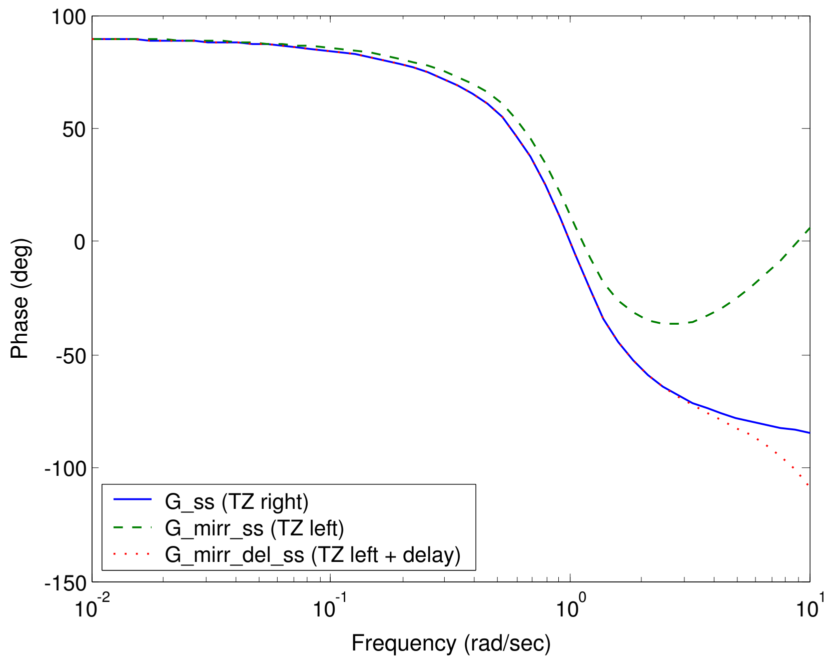

Figure 19: Phase shift of system, mirrored system, and mirrored system with time delay

9 Inversion of a Nonlinear System

The analytical inversion of nonlinear dynamic systems is rarely possible. Even the solution of a seemingly simple system like a forced mathematical pendulum

$$\ddot{y} + \sin(y) = u \tag{36}$$

where y is the displacement angle and u is the normalized thrust, involves Jacobi elliptic functions [5] and its analytical inversion is not really the daily bread of an average engineer.

On the other hand the simulation of the inverse of the pendulum is very straightforward, utilizing the approach described in Section 5.

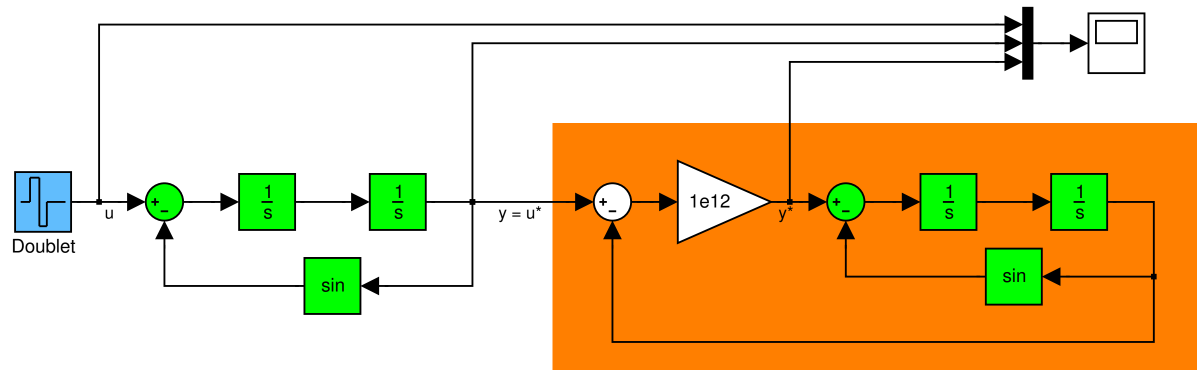

Figure 20: Nonlinear system inversion block diagram

The extremely high gain of 1e12 (Figure 20) provides an almost perfect reconstruction of the input doublet.

For the pendulum inversion in Figure 21 a variable-step stiff solver (ode15s) has been used, with all parameters (max step size, ..., absolute tolerance) set to "auto".

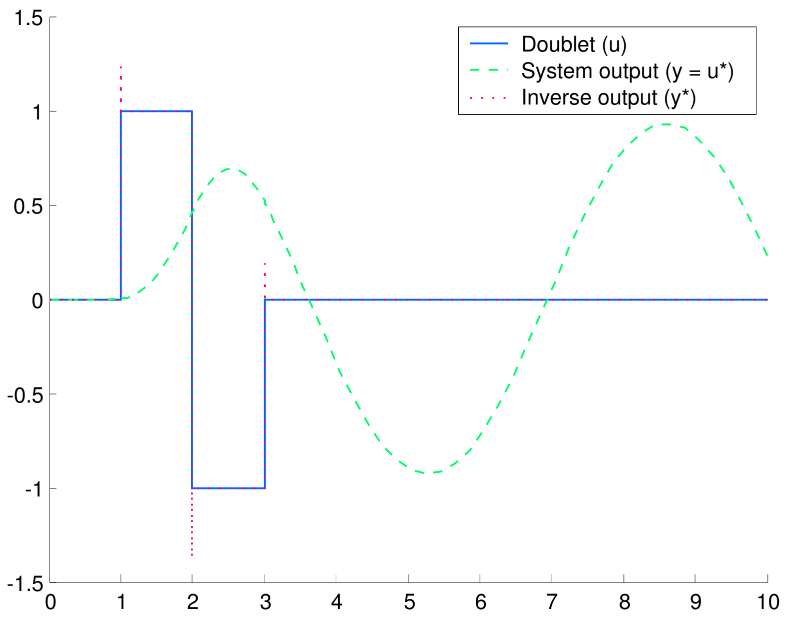

Figure 21: Nonlinear system inversion result

10 EC-135 Inversion

Both inversion methods (Analytic Inversion with Propering Filters (Section 6.3 and 6.4) and Proper Inversion (Section 6.5)) have successfully been applied to three different helicopters (UH-60, BO-105, and EC-135) using identified linear models and turbulent flight test data. This section exemplarily concentrates on the EC-135 inversion.

The reduced linear model (Figure 22) has 10 states, 4 primary control inputs (`lat`, `lon`, `ped`, and `col`) and 4 corresponding measured outputs (`roll`, `pitch`, `yaw`, and `vert`).

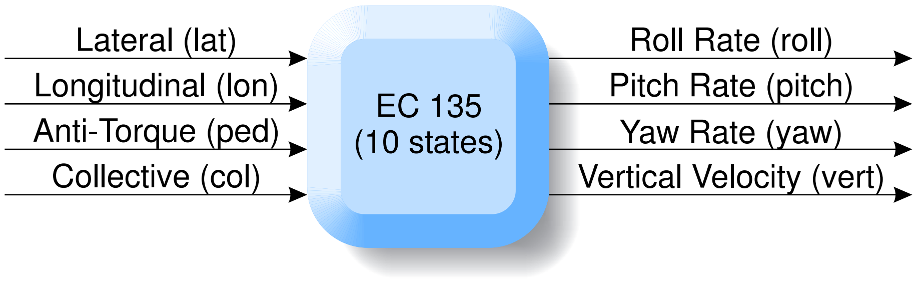

Figure 22: Input and output signals of EC-135 model

The predefined model matrices can be loaded from files and 4 transmission zeros and 10 poles can be found.

```
>> load a_mat_ec135
:
:
>> zero (G_ss)

ans =

  -0.0015 + 0.0166i
  -0.0015 - 0.0166i
  -0.0667
  -0.0359

>> pole (G_ss)

ans =

 -26.6099
  -5.1409
  -2.0776 + 2.4047i
  -2.0776 - 2.4047i
  -0.3197 + 1.6480i
  -0.3197 - 1.6480i
  -0.4733
   0.0314 + 0.3311i
   0.0314 - 0.3311i
   0.0724
```

All 4 transmission zeros are located in the left hand side complex half plane, implying a stable inverse; 3 relatively slow unstable poles forbid an open-loop simulation of the EC 135 for more than "a few" seconds.

Using the a/c matrices and Equation 24 a symbolic ("exact") transfer function matrix can be computed.

```
>> G_sym = simple (C*inv (s*eye (size (A)) - A)*B + D)

G_sym =

[625/128*(15079952533572558540421038318O...*s^6+...]
:
:
```

The computation of `G_sym` takes only a few seconds on a current computer, even though the result is a bit lengthy (involving large integer numbers) and not meant to be analyzed by a human user.

Nevertheless, the system can be inverted analytically.

```
>> G_star_sym = inv (G_sym);
>> G_star_tf = sym2tf(G_star_sym)

Transfer function from input 1 to output...

      0.2421 s^6 + 7.615 s^5 + 27.06 s^4 + 4.66 s^3 + 7.375 s^2 + 0.2548 s - 3.916e-005
 #1:  ---------------------------------------------------------------------------------
                  s^4 + 0.1057 s^3 + 0.002987 s^2 + 3.58e-005 s + 6.65e-007
:
:
```

The single transfer functions are improper (as expected) and their poles cover the transmission zeros of the original system (ditto as expected).

```
>> pole (G_star_tf(1,1))

ans =

  -0.0667
  -0.0359
  -0.0015 + 0.0166i
  -0.0015 - 0.0166i
```

Unfortunately, the transmission zeros of the improper inverse cannot be computed numerically.

```
>> zero (G_star_tf)
??? Error using ==> tf/zero
Not supported for MIMO improper transfer functions.
```

But - the product of system and inverse proves to be the 4-by-4 identity matrix.

```
>> ident = simple (G_sym*G_star_sym)

ident =

[ 1, 0, 0, 0]
[ 0, 1, 0, 0]
[ 0, 0, 1, 0]
[ 0, 0, 0, 1]
```

Again, to be implementable, the inverse has to be ”propered” by a fast second order low pass filter (the order of the filter resulting from the maximum inverse transfer function numerator denominator order difference).

```
>> T_filt_sym = 1e-3;
>> G_filt_sym = 1/(T_filt_sym*s + 1)^2;
>> pretty (G_filt_sym)

                                        1
                                 ---------------
                                               2
                                 (1/1000 s + 1)
```

The filter has been defined symbolically and can therefore directly be multiplied to the symbolic inverse.

```
>> G_star_filt_sym = simple (G_filt_sym*G_star_sym);
```

Again, the coefficients of the exact proper symbolic inverse are ratios of large integers, not suitable for direct use. In order to be used numerically, the symbolic inverse is transformed to its minimal numeric state-space representation.

```
>> G_star_filt_ss = minreal (ss (sym2tf (G_star_filt_sym)))
10 states removed.

a =
                 x1          x2          x12
   x1        -321.9      -492.5 ...  -494.1
   x2        -493.7      -755.6 ...    -758
   :            :          :           :
   :            :          :           :
   x12     -575.5     -880.7 ...  -908.2

:
:

d =
               u1           u2           u3           u4
   y1   2.421e+005   8.403e+005            0            0
   y2  -5.225e+004   3.839e+006            0            0
   y3  -4.243e+005   3.851e+006            0            0
   y4   5.774e+004   -1.86e+006            0            0

Continuous-time model.
```

Now the transmission zeros of the (proper) inverse can be computed

```
>> zero (G_star_filt_ss)

ans =

 -26.6099
  -5.1409
  -2.0776 + 2.4047i
  -2.0776 - 2.4047i
  -0.3197 + 1.6480i
  -0.3197 - 1.6480i
  -0.4733
   0.0724
   0.0314 + 0.3311i
   0.0314 - 0.3311i
```

as expected, covering all EC-135 poles. The poles of the proper inverse can be separated into two blocks,

```
>> p = pole (G_star_filt_ss);
```

the first block covering the 4 EC-135 transmission zeros

```
>> p(1:4)

ans =

  -0.0015 + 0.0166i
  -0.0015 - 0.0166i
```

```
   -0.0359
   -0.0667
```

and the second block representing the 8 poles of the propering filters (4 inputs $*$ second order filters = 8 poles).

```
>> format long
>> p(5:12)

ans =

   1.0e+003 *

  -0.99999997882045
  -0.99999998120528
  -1.00000000000000 + 0.00000004492552i
  -1.00000000000000 - 0.00000004492552i
  -1.00000000000000 + 0.00000003424624i
  -1.00000000000000 - 0.00000003424624i
  -1.00000001879472
  -1.00000002117956
```

Note that due to numerical errors the propering poles are "not exactly" located at -1000.

The inverse can now be implemented in Figure 23, using raw real-world flight test data (`lat`, `lon`, `ped`, and `col`) as inputs to the EC-135 model.

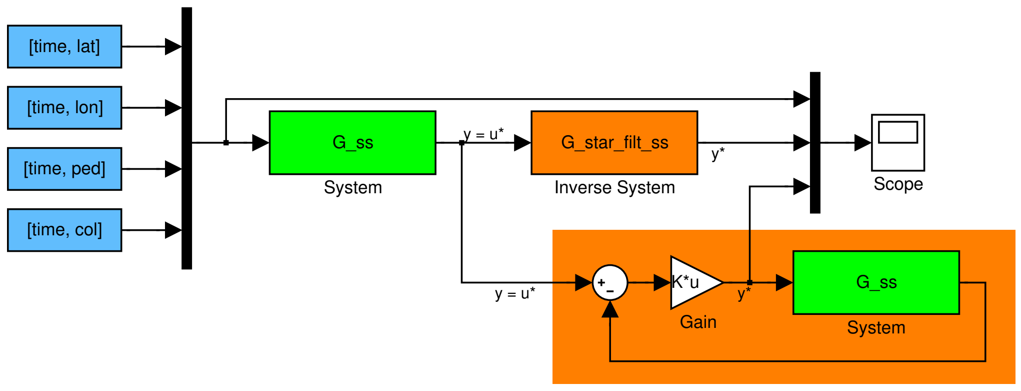

Figure 23: EC-135 inversion block diagram

Additionally the direct proper inverse (Section 5) has been implemented in Figure 23 for comparison.

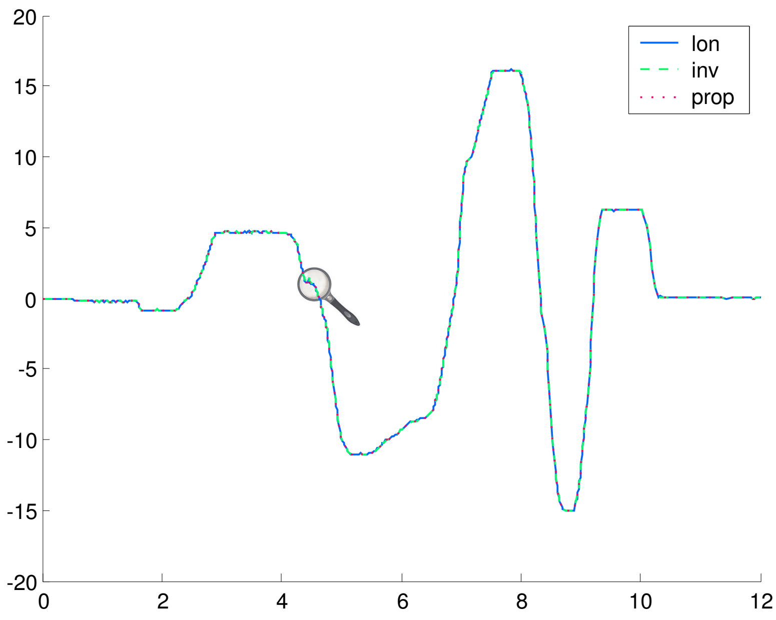

Figure 24: EC-135 inversion result

Figure 24 shows the longitudinal input (`lon`), the result of the analytical inverse (`inv`) and the output of the direct proper inverse (`prop`). Both inverses regain the overall input signal with satisfactory precision.

A closer zoom at an arbitrary location (Figure 25) reveals the loss of precision of the analytical inverse, while the direct proper inverse is still doing a perfect job.

This result is not really astonishing since quite a high gain of

```
>> k = 1e9;
>> K = [  1  0  0  0; ...
          0  1  0  0; ...
          0  0  1  0; ...
          0  0  0 -1];
>> K = k*K;
```

has been used for the proper inverse, while the propering poles of the analytic inverse have a magnitude of ”only” 1000. One has to keep in mind that, while the precision of both methods can be increased by increasing the gain and the propering poles magnitudes, there are different numerical computation limits for each method. While, for this EC 135 example, the maximum magnitude of the propering poles of the analytic inverse can be found somewhere around 1e6, the gain of the direct proper inverse can be raised up to 1e12 before numerical errors sweep off the high gain precision benefits, making this method clearly preferable if high precision is needed.

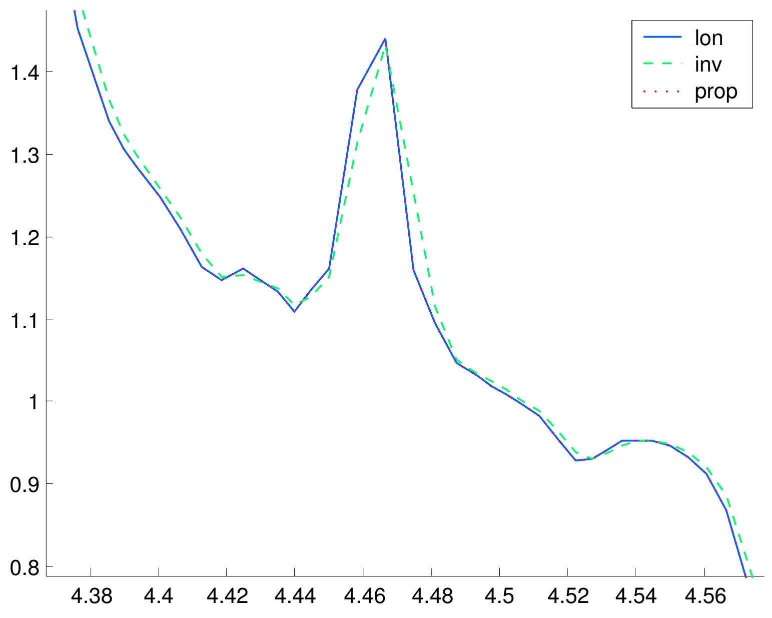

Figure 25: Zoom of EC-135 inversion result

A Matlab Function sym2tf

```
function G_tf = sym2tf (G_sym)

%SYM2TF Symbolic transfer function matrix to numerical transfer function matrix.
%
%   SYM2TF (G_SYM) returns the normalized numerical transfer function matrix
%   representation of the symbolic transfer function matrix G_SYM.
%
%   Example:
%
%      sym2tf ([s/(s+1), (s+2)/(2*s+1)])
%
%      returns
%
%      Transfer function from input 1 to output:
%        s
%      -----
%      s + 1
%
%      Transfer function from input 2 to output:
%      0.5 s + 1
%      ---------
%       s + 0.5
%

% Copyright Joerg J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

% Determine the numbers of rows and columns of the symbolic transfer function matrix
[n_rows, n_cols] = size (G_sym);

% Disassemble every single symbolic transfer function into numerator and denominator
[num_sym, den_sym] = numden (G_sym);

% Loop over all rows
for i_row = 1 : n_rows

    % Loop over all columns
    for i_col = 1 : n_cols

        % Transform the symbolic numerator of the current transfer function
        % to numerical (coefficients of the polynomial)
        num_tf{i_row, i_col} = sym2poly (num_sym(i_row, i_col));

        % Transform the symbolic denominator of the current transfer function
        % to numerical (coefficients of the polynomial)
        den_tf{i_row, i_col} = sym2poly (den_sym(i_row, i_col));

        % Normalize, so that leading denominator coefficient equals 1
        num_tf{i_row, i_col} = num_tf{i_row, i_col}/den_tf{i_row, i_col}(1);
        den_tf{i_row, i_col} = den_tf{i_row, i_col}/den_tf{i_row, i_col}(1);

    end

end

% Assemble the numerical transfer function matrix
G_tf = tf (num_tf, den_tf);
```

References

[1] Labows, S. J., Blanken, C. L., and Tischler M. B., "UH-60 Black Hawk Disturbance Rejection Study for Hover/Low Speed Handling Qualities Criteria and Turbulence Modeling", *American Helicopter Society* 56th *Annual Forum*, Virgina Beach, Virginia, May 2-4, 2000.

[2] Lusardi, J. A., Blanken, C. L., and Tischler M. B., *Piloted Evaluation of a Mixer Equivalent Turbulence Simulation Model*, *American Helicopter Society* 59th *Annual Forum*, Phoenix, Arizona, May 6-8, 2003.

[3] Levine, W. S., *The Control Handbook*, CRC Press, Inc., 1996.

[4] Hess, R. A. and Siwakosit, W., *Assessment of Flight Simulator Fidelity in Multiaxis Tasks Including Visual Cue Quality*, *Journal of Aircraft*, Vol. 38, No. 4, 2001, pp. 607-614.

[5] Lawden, D. F., *Elliptic Functions and Applications*, Applied Mathematical Sciences, Vol 80, Springer, 1989.