ISSN 0280-5316
ISRN LUTFD2/TFRT--5658--SE

Linearization and Nonlinear Control in Flight-control and Aerolasticity for Civil Aircraft using MATRIXx

Peter Odebjer
Johan Svahn

Department of Automatic Control
Lund Institute of Technology
December 2000

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2000

Document Number
ISRN LUTFD2/TFRT—5658--SE

Author(s)
Peter Odebjer and Johan Svahn

Supervisor
R. Johansson and M. Gäfvert (LTH) G. Mai and. J. Schuler (EADS Airbus GmbH)

Sponsoring organization

Title and subtitle
Linearization and nonlinear control in flight-controls and aeroelasticity for civil aircraft using MATRIXx. (Linearisering och olinjär reglering av styrsystem och aeroelastiska modeller för civila flygplan i MATRIXx).

Abstract
This Master Thesis has been written at EADS (European Aeronautic Defence and Space Company) Airbus in Bremen.The work contains a description of a few methods when linearising SystemBuild models in MATRIXx. MATRIXx is a tool for matrix manipulations (similar to Matlab) and SystemBuild is a graphical modeling environment (similar to SimuLink). The methods examined are the linearization command “lin” and the method of attaining a linear model through system identification (subspace identification). A command scipt for frequency response analysis has also been written.
To get more familiar with the tools in MATRIXx an attempt to solve a control problem has also been made. The problem consists of a limit cycle that appears in an airplane when one control surface (outer aileron) is in damping mode. The two main ideas that have been investigated are scalar feedback and nonlinear feedback. Due to the cxomplexity of the model the outcome is that a simple feedback is the best solution of the approaches made.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
84

Recipient's notes

Security classification

Master-of-Science Thesis in Electrical Engineering

Peter Odebjer
Johan Svahn

Linearization and nonlinear control in flight-controls and aeroelasticity for civil aircraft using MATRIX$_x$

Lund Institute of Technology, Lund, Sweden
Department of Automatic Control
Prof. Rolf Johansson, MD, Ph.D.

EADS Airbus GmbH, Hamburg, Germany
Segment Flight Controls, Bremen, Germany
Segment Aeroelasticity, Bremen, Germany

Attendants:

Magnus Gäfvert, M.Sc.
Department of Automatic Control at Lund Institute of Technology, Sweden

Dipl.-Ing., Dipl.-Wirt.-Ing. Georg Mai
Flight Controls at EADS Airbus GmbH, site Bremen, Germany

Dr.-Ing. Jörg Schuler
Aeroelasticity at EADS Airbus GmbH, site Bremen, Germany

Contents

List of figures

1 Preface

1.1 Project description

Modern Airbus aircraft use Electronic Flight Control Systems (EFCS) to control the aircraft via digital computers and electrical signals (fly-by-wire technology). In a recent research program DaimlerChrysler Aerospace Airbus GmbH (DA) developed a complete fly-by-wire system including Flight Control Laws (FCLs). These flight control systems consist of highly *nonlinear control algorithms*, as well as hydraulically or electrically powered actuators, which too have *nonlinear dynamic characteristics*. Control algorithms of interest and the actuation are modeled using the tool MATRIXx/SystemBuild for the description of their functional behavior with nonlinear differential equations. For certain analysis tasks and especially for flutter calculation, which takes place in frequency domain, *linear* system descriptions are necessary and must be elaborated from the nonlinear models using state of the art linearization techniques and the control software package MATRIXx/SystemBuild.

As an extension of the work a nonlinear control case will be investigated for an actuator failure mode.

The Master Thesis shall elaborate the following subjects:

- *Theory and investigation*:
  1. Theoretical chapter to describe the technique of linearization
  2. Theoretical chapter to describe the linearization techniques using MATRIXx
  3. Investigation and analysis of the MATRIXx "`lin`" command taking into account dependencies on working point, mix of *continuous* and *discrete* models, application considerations on the usage of specific nonlinear blocks (like *Quantization*, *Dead Band*, *Saturation*, *Gain Table*, *Time Delay* etc.) and parameter settings inside the analysis commands
  4. Theoretical chapter to evaluate a given nonlinear frequency analysis algorithm
  5. Elaboration of basic techniques for nonlinear control

- *Implementation*:
  1. Implementation of the "`lin`" command applicable to nonlinear models described in SystemBuild
  2. Implementation of a given nonlinear Frequency Response Analysis (FRA) algorithm in MATRIXx/SystemBuild
  3. Implementation of an alternative linearization algorithm, if the built in functions should fail (Subspace identification)
  4. Implementation of nonlinear control laws

- *Application and analysis*:
  1. Linearization of nonlinear actuator models and an elevator flight control system
  2. Modular linearization of specific nonlinear elementary blocks
  3. Nonlinear control to reduce structural accelerations due to actuator failure mode
  4. Elimination of the limit cycle due to actuator failure mode

- *Available resources:*
  1. Actuator models and flight control axis model elaborated with SystemBuild
  2. Given algorithm for nonlinear Frequency Response Analysis (gain and phase plot) using *Advanced Continuous Simulation Language* (ACSL), described in the ACSL reference manual
  3. Generic linear aero-elastic model using SystemBuild
  4. Combined linear aero-elastic model and nonlinear actuator force leading to limit cycle

1.2 Abstract

This Master Thesis has been written at EADS (European Aeronautic Defence and Space Company) Airbus in Bremen. The work contains a description of a few methods when linearising SystemBuild models in MATRIXx. MATRIXx is a tool for matrix manipulations (similar to Matlab) and SystemBuild is a graphical modeling environment (similar to SimuLink). The methods examined are the linearization command "`lin`" and the method of attaining a linear model through system identification (subspace identification). A command scipt for frequency response analysis has also been written.

To get more familiar with the tools in MATRIXx an attempt to solve a control problem has also been made. The problem consists of a limit cycle that appears in an airplane when one control surface (outer aileron) is in damping mode. The two main ideas that have been investigated are scalar feedback and nonlinear feedback. Due to the complexity of the model the outcome is that a simple feedback is the best solution of the approaches made.

1.3 Acknowledgements

This work has been elaborated during our stay at EADS Airbus GmbH, site Bremen, Germany between July and November 2000.

We would like to show our appreciation to the following persons and organizations: Dipl.-Ing., Dipl.-Wirt.-Ing. Georg Mai, Dr.-Ing. Jörg Schuler, M.Sc. Magnus Gäfvert, Prof. Rolf Johansson, Beck's (brewery), the ladies in the Dasa breakfast cafeteria, the French guys (dear colleagues), Sebastian Lauckner (bar friend), Purnendu Sarkar (MATRIXx support), Studentenwerk (housing), Lloyd Imbiss (curry Bratwurst), Bremen Police Department (good cooperation), die Oase (health spa), and CSN (sponsoring).

1.4 Declaration

We hereby declare that we edited the presented Master Thesis independently and only using resources explicitly mentioned.

Peter Odebjer Johan Svahn

1.5 Table of abbreviations

Abbreviation	Full description
ACSL	Advanced Continuous Simulation Language (http://www.acslsim.com)
CIT	Comfort In Turbulence, control to improve passenger comfort
CPU	Central Processing Unit
DA	DaimlerChrysler Aerospace Airbus GmbH, Hamburg, Germany; Predecessor of EADS Airbus GmbH, Hamburg, Germany
Dasa	DaimlerChrysler Aerospace AG, Munich, Germany
CMU	Configuration Modification Unit
EADS	European Aeronautic Defence and Space Company (http://www.eads-nv.com)
EFCS	Electronic Flight Control System
FCL	Flight Control Laws
FCS	Flight Controls Segment at EADS Airbus GmbH
FRA	Frequency Response Analysis
GUI	Graphical User Interface
MIMO	Multiple-Input / Multiple-Output
PDM	Parameter-Dependent Matrix, storage format in MATRIXx
PRBS	Pseudo Random Binary Sequence
URL	Uniform Resource Locator

1.6 Table of terms and definitions

Term	Definition
Aliasing	Sampling interval is too low to detect frequencies uniquely. A high frequency is then detected as a lower.
Flap	High-Lift surfaces at the trailing edge of the wings
MATRIXx	MATRIXx is a tool for matrix manipulations (with its core module Xmath) especially in automatic control and also provides a graphical modeling environment for block diagramming called SystemBuild. MATRIX$_x$ is provided by WindRiver, Inc. (which took over former Integrated Systems Inc.), Internet URL: http://www.windriver.com/products/html/matrixx.html.
Slat	High-Lift surfaces at the leading edge of the wings.
SuperBlock	Well-defined block-diagram (sub-model) in SystemBuild.
SystemBuild	Graphical modeling environment of SW-tool MATRIXx.
Validation	Proving that the model is correct, e.g. comparing with a different method or with different data.
Xmath	Command oriented core module of SW-tool MATRIXx.

2 Linearization

Linearization means that nonlinear equations are approximated with linear equations around a *working point*. This is done to make the analysis of a system's dynamic behavior possible and also, many methods for system analyses only support systems in linear form.

Linear systems are made to match the original nonlinear system well around a stationary working point. Most of the time the system is controlled and this controller is designed to keep the system close to the same working point. Linear approximations are therefore usually good approximations of the system's behavior.

If a system is to be linearized, the *Taylor* expansions are taken for each equation and terms of order 2 and higher are neglected [SLOTINE, LI]. The state and input values of the working point are then inserted and a linear system is attained as follows.

Assume that the following system is nonlinear

$$\dot{x} = f(x,u)$$

The systems dynamics can be described by the *Taylor* expansion

$$f(x,u) = f(x_0,u_0) + \sum_{k=1}^{\infty} \frac{(x-x_0)^k}{k!} f^{(k)}(x_0,u_0) + \sum_{k=1}^{\infty} \frac{(u-u_0)^k}{k!} f^{(k)}(x_0,u_0)$$

Where the term $f^{(k)}(x_0, u_0)$ is the kth derivative of the function $f(x,u)$. Under the assumption that $f(x_0, u_0)=0$ the following expression can be given:

$$\dot{x} = \left(\frac{\partial f}{\partial x}\right)_{x=0} x + f_h(x) + \left(\frac{\partial f}{\partial u}\right)_{u=0} u + f_h(u)$$

Where *h* means higher order terms. This system is also valid for a MIMO (*Multiple-Input/Multiple-Output*) system, in other words a matrix- and vector-system.

When the system is of MIMO type then:

$$A = \left(\frac{\partial f}{\partial x}\right)_{x=0}, B = \left(\frac{\partial f}{\partial u}\right)_{u=0}$$

is the approximation of the nonlinear system matrix. The A matrix is called the *Jacobian*, and the linear system becomes:

$$\dot{x} = Ax + Bu$$

which is the linearization of the original nonlinear system.

A linear system is characterized by the two following equations:

$$f(ax) = a * f(x)$$
$$f(x + y) = f(x) + f(y)$$

This means that a zero input must give a zero output (no offset allowed). Outputs of positive inputs are equal to those of negative with the change of sign. Two inputs, one half the magnitude of the other, will result in the same output with the same shape, only different magnitudes. A sinus shaped input will result in a sinus shaped output.

These things are important to bear in mind when linearising. A linear model has limitations of what it can model.

3 Linearization in MATRIXx using the "`lin`" and "`trim`" commands

Linearization in MATRIXx is done with the command "`lin`". It is a very flexible command that can perform linearization of discrete-, continuous-, hybrid- and multi-rate systems around a working point. A *hybrid* system is a system that consists of both *discrete* and *continuous* sub-models, while a *multi-rate* system consists of *discrete* sub-models with different sampling rates. The command could be used to either get the linearized system in *explicit* or *implicit* form.

The *explicit* form of a system with p inputs, n states and q outputs contains the following data:

- A, the (*n, n*) Jacobian matrix
- B, the (*n, p*) input matrix
- C, the (*q, n*) output matrix
- D, the (*q, p*) direct term matrix
- u, the *p* rows vector of the inputs
- x_0, the *n* rows vector of the initial states
- x, the *n* rows vector of the states
- $\dot{x}$, the *n* rows vector of the states derivatives
- y, the *q* rows vector of the outputs

The *explicit* system can then be written as:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$
$$x(t=0) = x_0$$

When the *implicit* form is used information about the matrices A, B, C, D, and additionally E, F, Sampling Rate ("`tsamp`"), State Names, Input Names, Output Names and Implicit Output Names is needed. The implicit form of linearization is then:

$$E\dot{x} = Ax + Bu$$
$$y = F\dot{x} + Cx + Du$$

The *syntax* of the command is:

```
Sys = lin("modelname", {keywords})
```

For more information about the keywords see MATRIXx/SystemBuild user's guide [MATRIXX_SBUG].

3.1 Methods used in the "`lin`" command

The methods of linearization depend on what kind of system is to be linearized. For a pure *continuous* system or a pure *discrete* system with all SuperBlocks having the same computational timing attributes two different approaches may be used:

- *Exact* linearization
- *Finite-difference* approximation

If any or all of the perturbation vectors "`dx`, `du`" and "`dxdot`" are specified in the command, then the system is automatically linearized using the finite-difference approximation. The perturbation vectors indicate how much the states and the inputs are going to be excited around the working point. In the ideal case the perturbations are infinitely small.

Systems that are *hybrid* or *multi-rate* use the *Kalman-Bertram* method to merge the systems with different sampling rates together to a system with same sampling rate, then the linearization is done with *finite difference* approximation.

Algebraic loops that occur in a system are always resolved for continuous systems. For a discrete system it is possible to set the keyword "`algloop=0`", which means that the loops are not resolved, this makes the loops instead appear as additional states.

3.1.1 Exact linearization

Many of the SystemBuild blocks have build in exact linearisations. This does not mean that the linearization makes an exact fit to the nonlinear system, it means that the linearization is *analytically* and not *numerically* solved.

3.1.2 Finite-Difference approximation

The problem making a linearization with a computer is to calculate the Jacobian, while it can't be solved analytically it has to be solved by finite difference approximation [JORDAN]:

$$Df(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{\Delta_h f(x)}{h}$$

This gives a linear approximation of the nonlinear function. This approximation is only valid in a certain input interval, depending on the function and on the demands of the approximation.

3.1.3 Kalman-Bertram method

The Kalman-Bertram method is not a linearization method, it is used in *hybrid* (a system containing both continuous and discrete systems) and *multi-rate* systems to merge the different subsystems together [MATRIXX_SBUG]. The actual linearization is then done with finite-difference approximation.

To be able to use the Kalman-Bertram method all the different systems and subsystems must be transformed into systems having the same sampling rate. This sampling rate is called the *basic time period*. The basic time period is calculated so that all subsystems are sampled at the same time, with a maximum difference that is proportional to the "`btptol`" value. When this is done the actual merging of the systems starts.

The merging is done during one basic time period, T. The continuous system is updated between $t=0$ and $t=T^-$, and the discrete is updated between $t=T^-$ and $t=T^+$.

First a *transition matrix* for the continuous systems is made and then a *transition matrix* for the discrete systems is calculated.

When the matrix for the continuous states is calculated, two assumptions are made:

- The *discrete* states are assumed to be constant over the sample interval. This is perhaps not true since most of the subsystems have a faster sampling rate, which means that they are updated more often.
- The external input is also assumed to be constant, which implies that a zero-order hold sampler is used.

During the computation of the discrete transition matrix, the continuous states are held constant as well as the external inputs. This means that the external inputs are not updated until the discrete subsystems have been updated.

Then the overall state matrix is computed as:

$$\Phi = Ge^{FT},$$

where G is the discrete transition matrix and F is the continuous transition matrix.

3.2 How to use the keywords in the "`lin`" command

The "`u0`" and "`du`" keywords are aiming on the input, while the "`x0`" and "`dx`" aims on the states.

The keywords "`u0`" and "`x0`" are used in purpose to set the working point of the system. The working point of the non-linearity is then calculated with the system equations. This is of course a problem when setting several working points for systems containing more than one nonlinear block.

A great deal of consideration has to be done, while it is important to get the keywords "u0" and "x0" to satisfy the equilibrium points of the system, in order to achieve a good linearization. This is not a problem for small systems, but for larger systems the "trim" command can be of great help, see chapter [The "trim" command]. The keywords "du" and "dx" are used for setting the perturbation around the working point for each non-linearity. The values that are specified also have to be recalculated according to the system equations to get the correct values for each non-linearity.

The linearization is then calculated in the same way as described by the finite difference approximation. This means that the perturbation value that was achieved for a certain nonlinear block is centered around the respective working point.

3.3 Linearization of selected blocks

In this chapter it will be discussed how to linearize the nonlinear blocks in System Build. This will be done by showing some simple examples of all the blocks; all examples are based on the same dynamic system. Since the nonlinear block is only affecting the output the only matrices that will be affected by the linearization is the C and D matrices.

This means that the only difference between the examples is the nonlinear block. The dynamic system is a small system consisting of only two states. The first example in this chapter will be thoroughly and the rest of the examples will be a little more basic. While the blocks are rarely linearized one and one the purpose of this chapter is to increase the understanding of linearization strategy in MATRIXx.

At the end of this chapter a linearization of a small system containing nonlinear blocks in a way so that the non-linearities also affects the states.

3.3.1 Saturation block

Example [2-1.]:

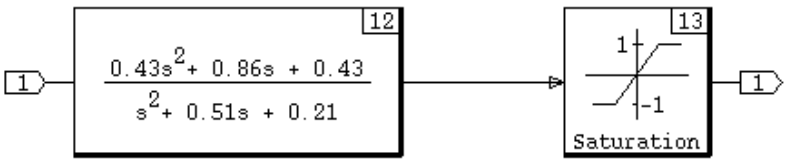


Fig. 1 System with saturation

To be able to calculate the working point and the perturbation vectors for the nonlinear block the dynamic system in front of the saturation block (see Fig. 1 above) has to be considered.

This dynamic system has the following system matrices:

$$A = \begin{bmatrix} -0.51 & -0.42 \\ 0.5 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.6407 & 0.6794 \end{bmatrix} \qquad D = \begin{bmatrix} 0.43 \end{bmatrix}$$

Since it is only the output that is affected of the non-linearity, the only equation that has to be linearized is the output equation. According to the explicit form equations the output signal is given by:

$$y = C \cdot x + D \cdot u$$

The keywords are set to:

$$x_0 = [1, \quad 1] \quad dx = [4, \quad 2]$$
$$u_0 = [1] \qquad du = [1]$$

With the above given "`x0`" and "`u0`" values and the output equation the *working point* becomes 1.7501. This is the point, which the linearization is going to be made around. The perturbation vectors are then perturbated around the working point, and the new matrices are then calculated.

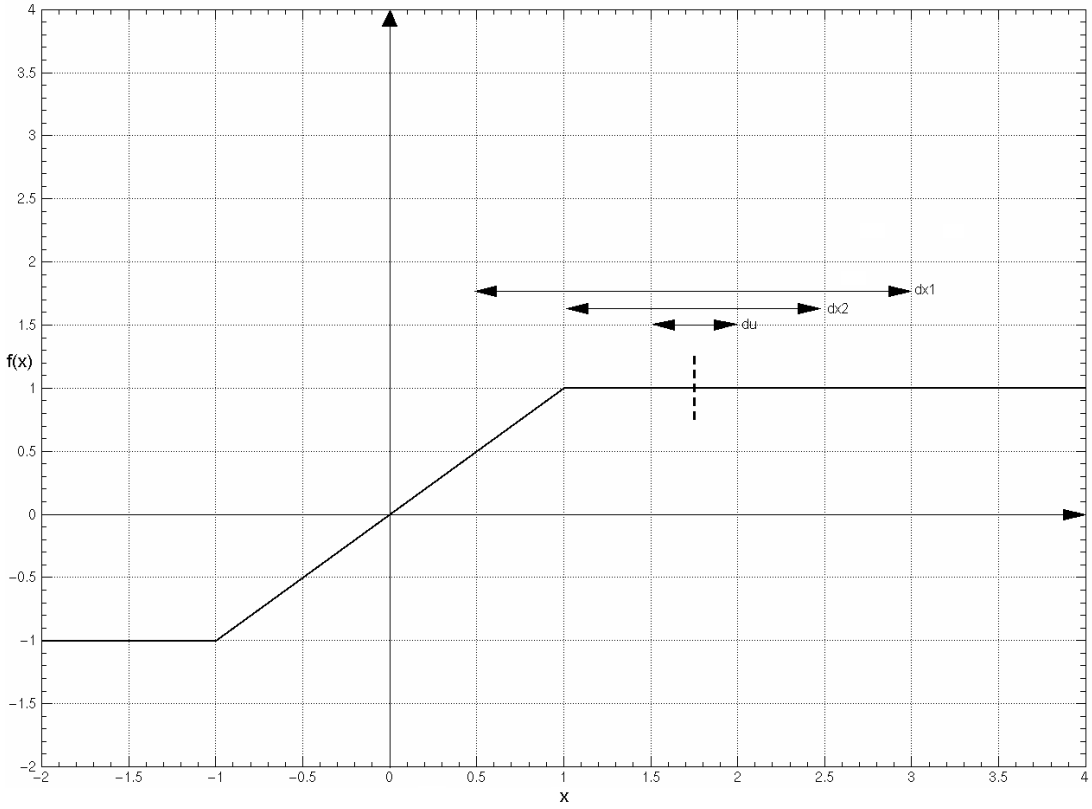


Fig. 2 Saturation system analysis

In Fig. 2 above the saturation function, the working point and the perturbation intervals can be seen. The perturbation intervals are calculated by multiplying the predefined "dx" and "du" values with its respective matrix value.

$$dx_1 = 4 \cdot 0.6407 = 2.5628$$
$$dx_2 = 2 \cdot 0.6794 = 1.3588$$
$$du = 1 \cdot 0.43 = 0.43$$

Then the derivative is estimated with the finite different approximation. The new linearized matrices are then calculated by multiplying the old matrices' values with its estimated derivative.

$$\frac{\partial f}{\partial x_1} = 0.207312 \qquad \Delta c_1 = 0.6407 \cdot 0.207312 = 0.132825$$
$$\frac{\partial f}{\partial x_2} = 0 \qquad \Delta c_2 = 0$$
$$\frac{\partial f}{\partial u} = 0 \qquad \Delta d_1 = 0$$

$$\Delta C = \begin{bmatrix} 0.1328 & 0 \end{bmatrix}$$
$$\Delta D = \begin{bmatrix} 0 \end{bmatrix}$$

This example shows a very ill conditioned linearization. To get a better linearization of this system all of the perturbation areas have to be inside the non-linearities non-zero derivative area. Of course this is not always possible. In the Bode diagram (see Fig. 3 below) the solid line shows the bode plot of an exact linearization and the dashed line gives the bode plot of the linearization calculated above.

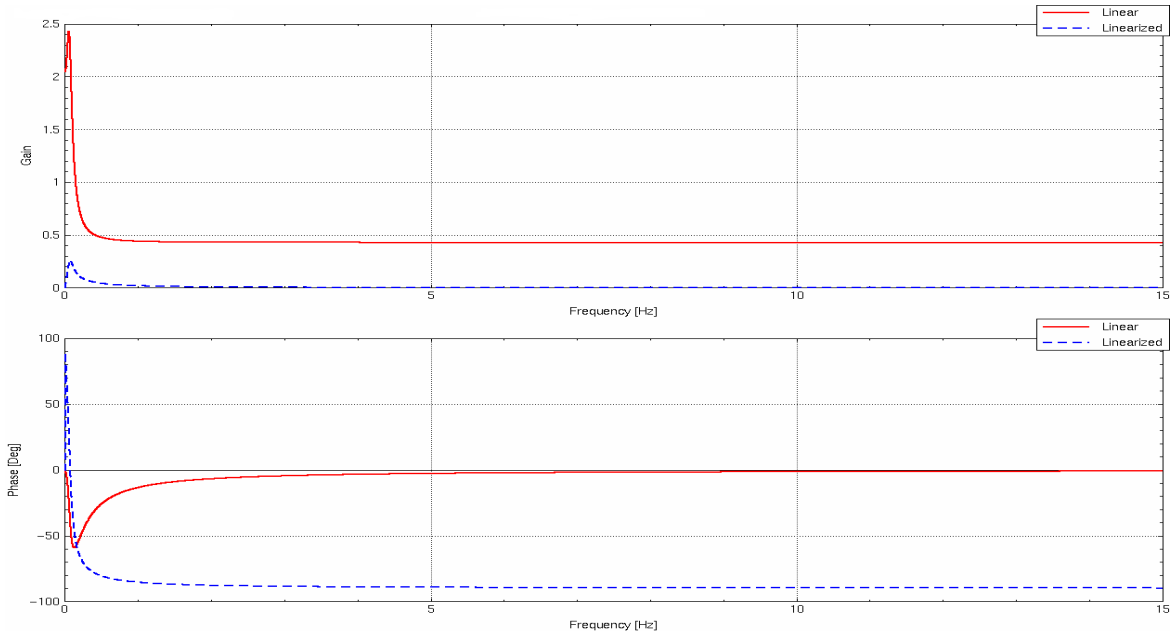


Fig. 3 Saturation system Bode plot

The exact linearization could have been achieved if the linearization parameters "du, dx, x0" and "u0" had been chosen better. For example:

$$x0 = [0 \quad 0] \quad dx = [1 \quad 1]$$
$$u0 = [0] \qquad du = [1]$$

This makes all the perturbation intervals stay inside the non-linearities non-zero derivative area and the linearized matrices C and D will make a good fit to the nonlinear system around the working point.

3.3.2 General Algebraic Expression block

In this block it is possible to enter a general algebraic expression. The block is always linearized using finite difference approximation, since there is no exact linearization available.

Example [2-2.]:

In this example the General Algebraic Expression Block, contains the following expression:

$$Y = 0.1 \cdot U^3 + U^2 + U + 3$$

The gain of the system without the nonlinear block is 2. To be able to linearize around this point the "x0" and "u0" parameters have to be chosen carefully. Having chosen "x0" first, "u0" can be calculated so that the desired working point is achieved.

$$x_0 = [1 \quad 1]$$
$$u_0 = \frac{2 - 0.6407 - 0.6794}{0.43} = 1.5811$$

To get an as accurate as possible derivative the perturbation vectors should be chosen as small as possible. The perturbation values after multiplication with their respective matrix values, see matrices C and D, are given below:

$$dx = [0.1 \quad 0.1] \quad du = [0.1]$$

A comparison between the linearized and the nonlinear step response is given in Fig. 4 below:

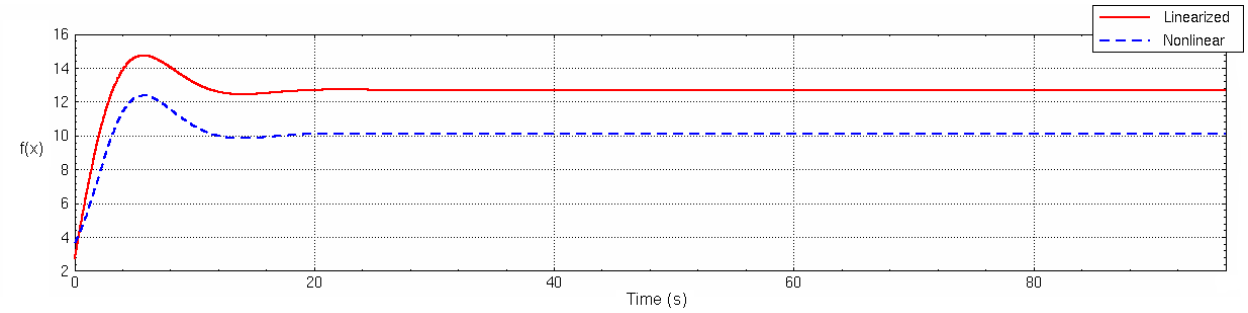


Fig. 4 Step response comparison 1

Comparison of the estimated derivative and the true value shows a good match.

$$Approximated\ value: \quad 6.2003$$
$$True\ value: \quad \frac{\partial f}{\partial u} = 6.2$$

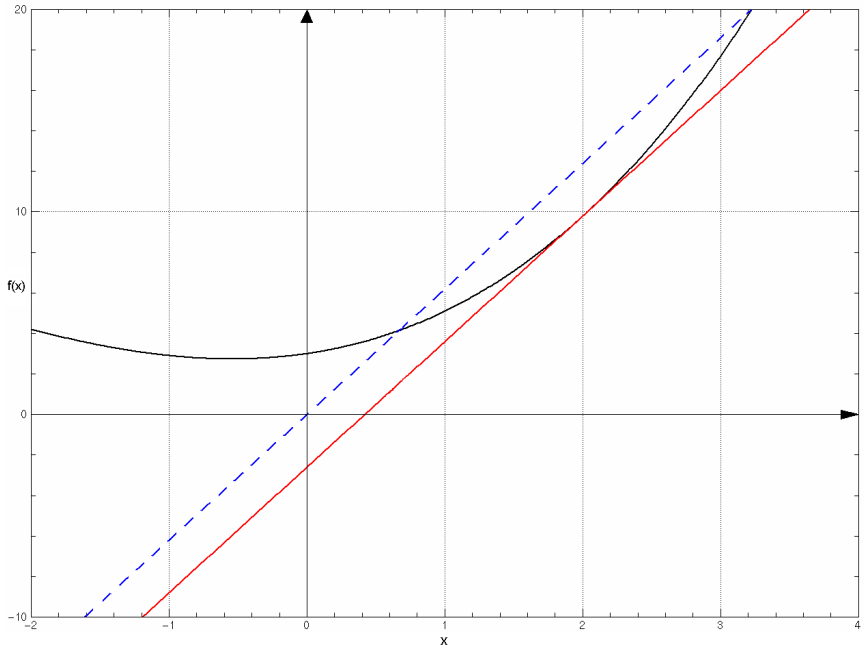


Fig. 5 Comparison of gain and derivative

As can be seen in Fig. 4 above the two systems do not match each other very well. This can be understood if figure Fig. 5 above is considered. The solid line shows the exact match to the nonlinear system when the input is 2.

The dashed line shows the linearization, and as can be seen there is an offset between the two curves that is approximately 2.6. So the source of the difference in amplitude between the different input steps has been found.

To make a better linearization it is possible to calculate the difference in gain and then find a new working point that has the desired slope and linearize around that working point instead. The difference in gain is:

$$k_{lin} - k_{non} = \frac{2.6}{2} = 1.3$$
$$The\ new\ gain: \quad k_{new} = 6.2 - 1.3 = 4.9$$

To be able to find a working point with this slope, the derivative of the function must be considered:

$$\frac{\partial f}{\partial u} = 0.3 \cdot U^2 + 2 \cdot U + 1 = 4.9$$
$$U_1 = 1.5770 \quad U_2 = -8.2436$$

U_1 is chosen to be the working point. A linearization around this point should give a gain around 4.9, depending on the perturbation vectors. U_1 can be achieved by choosing:

$$x_0 = [1 \quad 1]$$
$$u_0 = \frac{1.5770 - 0.6407 - 0.6794}{0.43} = 0.5974$$

With these values a gain of 4.9003 is achieved, which is a good approximation. The comparison of the nonlinear step response and new linearization step response can be seen in Fig. 6 below.

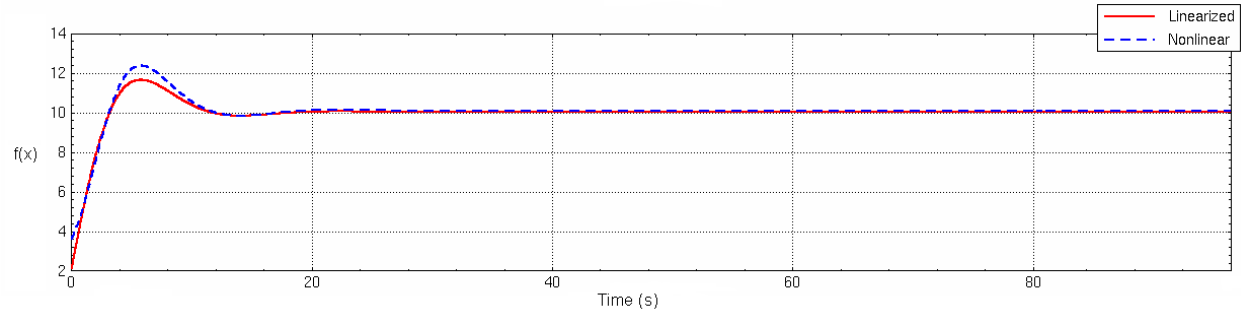


Fig. 6 Step response comparison 2

Now the linearization is much better, but since the new slope is not the true derivative of the working point, the interval where the linearization is valid is very small.

3.3.3 Dead Band block

In the Dead band block the output is zero until a certain value is reached.

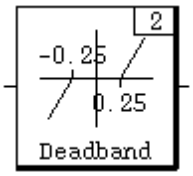


Fig. 7 Dead-band block

A dead band that has a working point close to zero will probably get a bad linearization. This depends of course also on the width of the dead band and the amplitude of the input.

Example [2-3.]:

The following parameter values are used for the linearization:

$$x_0 = [0.5 \quad 0.3] \quad dx = [1 \quad 2]$$
$$u_0 = [-2.5] \qquad du = [3]$$

This gives us the working point -0.55083. In figure Fig. 8 below it is shown how the different perturbations are located in relation to the working point (the thick line):

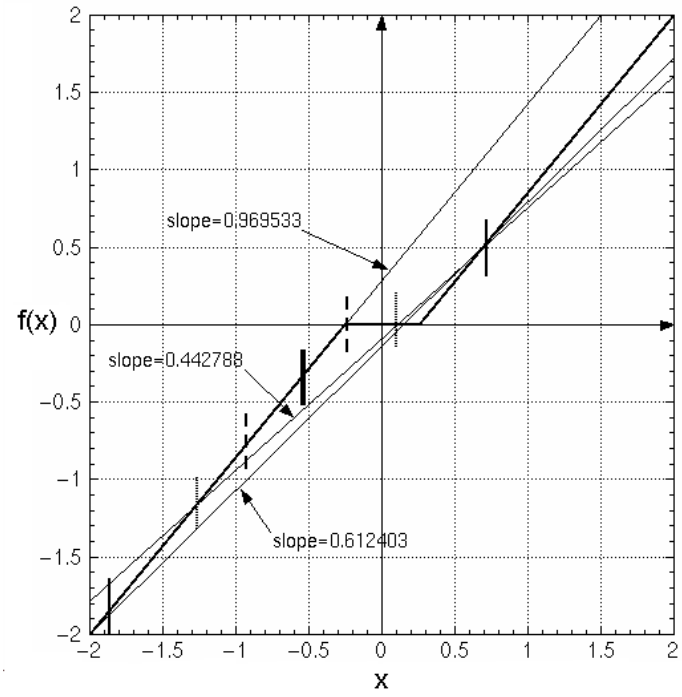


Fig. 8 Dead-band analysis

As can be seen in the above figure (Fig. 8) the only element that has a good match is the "`dx(1)`", since it is almost equal to one. But this does not mean that this linearization is bad, it all depends on how the linearization is going to be used.

The derivative vectors become:

$$dx_e = [0.969533 \quad 0.442788] \quad du_e = [0.612403]$$

And the linearized matrices will become:

$$\Delta C = [0.62118 \quad 0.490115] \quad \Delta D = [0.315277]$$

As mentioned above the linearization of the dead band is not trivial, but depending on the usage, the derivative can be altered so it makes a good approximation of the non-linearity.

3.3.4 Quantizer block

The quantizer block estimates a value with different resolution depending on the step size. A perfect linearization of a quantizer block would of course be a straight line with a slope equal to 1. A quantizer block can be seen in Fig. 9 below:

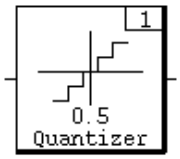


Fig. 9 Quantizer block

A linearization of the quantizer block with the "`lin`" command can either be done by choosing the perturbation vectors very carefully and by this way getting an exact linearization, or the perturbation vectors can be chosen very large with respect to the resolution.

Example [2-4.]:

Here a quantizer with resolution 0.5 is used. The working point in this example is set to zero, and the perturbation vectors are set to:

$$dx = [5 \quad 5] \quad du = [5]$$

The perturbation vectors are then multiplied with their respective system matrix:

$$dx_n = [3.2 \quad 3.4] \quad du_n = [2.15]$$

These values are then perturbated around the working point.

The resulting quantisized values are then:

$$dx_q = [3.0 \quad 3.0] \quad du_q = [2]$$

The derivative vectors are:

$$dx_e = [0.9375 \quad 0.8824] \quad du_e = [0.9302]$$

Multiplying the derivative vectors with the system matrices gives the new linearized system matrices:

$$\Delta C = [0.6 \quad 0.6] \quad \Delta D = [0.4]$$

As can be seen in the above example the choice of perturbation vectors can be rather critical. By choosing the perturbation vectors large in comparison to the quantizer's resolution, the derivative vectors will converge to 1. Of course the derivative vectors can also become 1 with smaller perturbation vectors, if they are correctly chosen.

3.3.5 Gain Scheduler block

The gain scheduler block has different gains for different inputs; this is a type of adaptive control.

Example [2-5.]:

The dynamic system is linked to the gain scheduler block as Fig. 10 shows below.

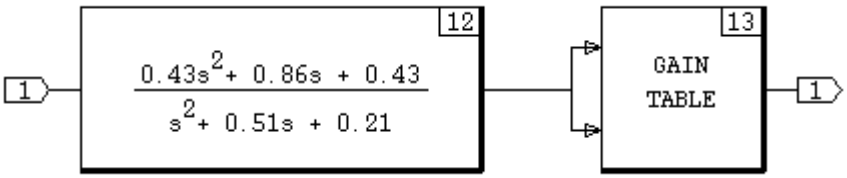


Fig. 10 Gain scheduler example

The parameters in the Gain Table are set to:

- `Break Points = [-2.0, -1.0, 0.0, 1.0]`
- `Gain Matrices = [ 0.0, 1.0, 0.0, 3.0]`

From the former examples the system matrices are known. The working point, and the perturbation vectors have been chosen as follows:

$$x0 = [1.0 \quad 1.0] \quad dx = [2 \quad 3]$$
$$u0 = [1.0] \qquad du = [10]$$

The working point and new perturbation values of the system can be calculated. The working point and the perturbation areas are shown (Fig. 11) below together with the Gain Scheduler function.

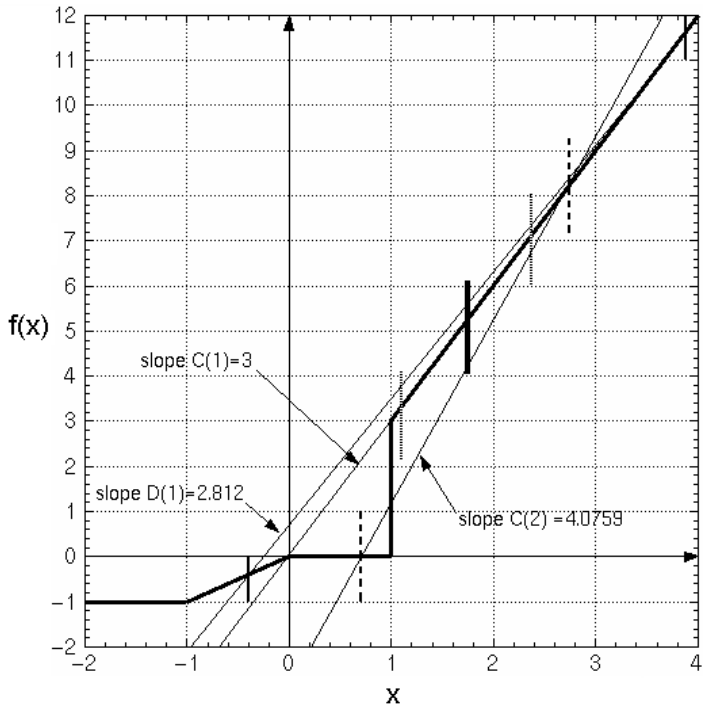


Fig. 11 Gain scheduler analysis

The derivatives are then calculated as previous and the new C and D matrices can be calculated.

$$\Delta C = \begin{bmatrix} 1.9221 & 2.7692 \end{bmatrix} \quad \Delta D = \begin{bmatrix} 1.21002 \end{bmatrix}$$

3.3.6 Hysteresis block

The hysteresis block can be seen in Fig. 12 below:

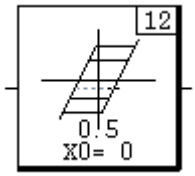


Fig. 12 Hysteresis block

It is linearized according to the thick lines in Fig. 13 below.

The working point is calculated by the following equation:

$$w_p = x_0 - u_0$$

Often a perfect linearization of a hysteresis is a line with the same slope as the slope of the hysteresis. If this is considered perfect, then the perfect linearization of a hysteresis will result in a state space system looking like following:

$$A = [-\omega_c] \quad B = [\omega_c]$$
$$C = [k] \quad D = [0]$$

The "k" is the *slope* that is chosen and the "ω_c" is the chosen *cut-off frequency*. Both choices are made in the hysteresis block.

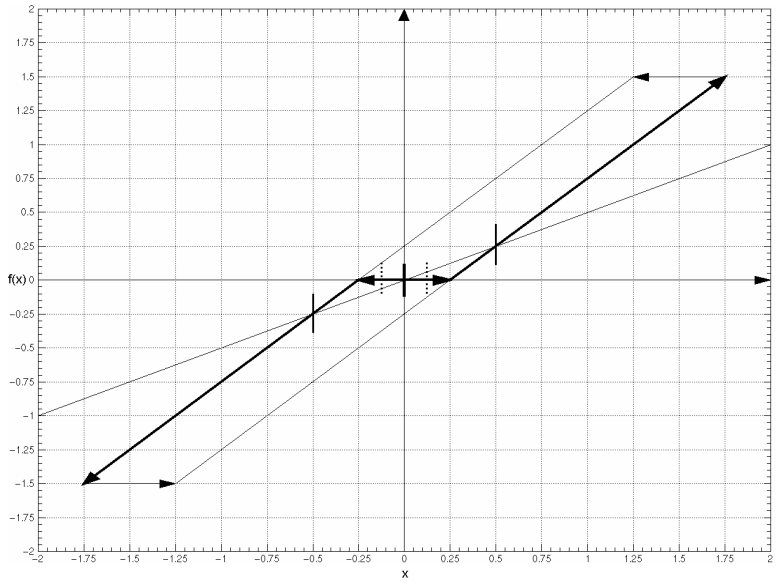


Fig. 13 Hysteresis analysis

3.3.7 Small nonlinear system

The biggest difference between this model and the others are that in this system it is not only output that is affected by the non-linearity, but also the states. The system used is a very simple system just to show how the linearization works.

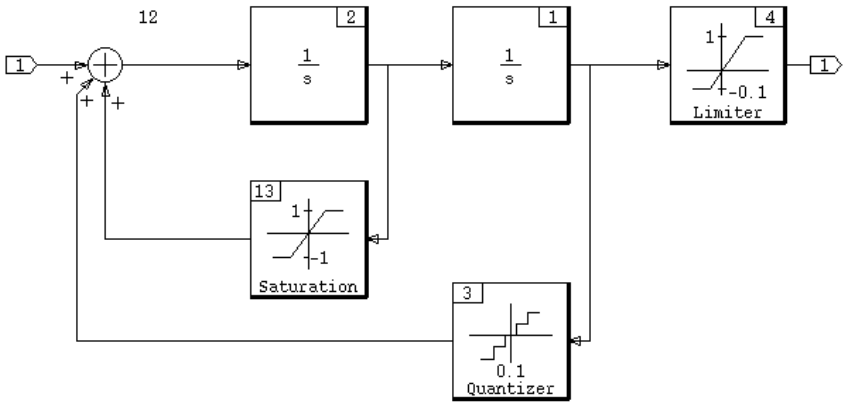


Fig. 14 System with nonlinear states and output

If the system is analyzed and put on state space form the following matrices are received.

$$A = \begin{bmatrix} \text{sat}(x_1) & \text{quant}(x_2) \\ 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & \text{limit}(x_2) \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

As can be seen the first state and the output are affected by the non-linearities. A perfect linearization of the system would be if all nonlinear blocks were replaced by a gain of one, since the linear parts of the Saturation and Limiter blocks has a slope of one and an approximation of the Quantizer is a slope of one. To be able to achieve this linearization an analysis of the system has to be done.

Since the Limiter makes the system non symmetrical the working point cannot be put in the origin. The interval of the Limiter goes from –0.1 to 1.0 this tells us that the working point should be at 0.45, since this is in the middle of the interval. If the working point is put here it gives us the possibility to use the perturbation vectors as much as possible. If "`u0=[0.45]`" and "`x0=[0,0]`" this gives a working point of 0.45 for all blocks except the Quantizer block that gets a working point around 0.5 because of the resolution.

The perturbation value of the first state is not critical as long as it is kept under 1.1. State number two's perturbation value has to be chosen a little more carefully because of the Quantizer and the Limiter, since the working point is 0.5, the nearest multiples of 0.1 is 4*0.1 and 6*0.1, this gives us a perturbation value of 0.2. The output perturbation value has no limitations and therefore it is not necessary to specify this.
If a linearization is done with these values an exact fit is achieved.

3.4 The "trim" command

The "trim" command is used to find the trimmed input, state and output vectors for an equilibrium point. These vectors can then be used for linearization purposes.
To be able to use this command the system has to have at least one state, a system without inputs can also be trimmed [MATRIXX_SBUG].

The syntax of the command is:

```
[xt, ut, yt, yimpt] = trim(model, {keywords})
```

If the "u0" keyword is not specified then it is initialized to zero, if "y0" is not specified then it is calculated with help of the "simout" function and if "x0" is not specified its values are taken from the SystemBuild catalogue.

The algorithm used can have some problems with free integrators, using the keyword "xdt_float" can sometimes solve this problem.

Often are the vectors "xt" and "ut" the only ones needed to make a linearization. To get these vectors it is often enough to specify the input vector, i.e. the working point.

When one or several equilibrium points are found a good linearization can be achieved by altering the perturbation vectors.

4 Frequency response analysis (FRA)

4.1 Background

Frequency Response Analysis (FRA) is used at EADS Airbus in Bremen when a nonlinear system needs to be analyzed. For some analysis questions in flight controls development and aeroelasticity linear models are needed, but for others a frequency response graph is sufficient. The *Flight Controls Segment* (FCS) elaborates detailed specifications of aircraft systems and equipment, e.g. the *high-lift system* moving the leading edge Slats and the trailing edge Flaps of the wing. Further, hydraulic and electrical control surface actuators are specified. These latter specifications include *gain and phase margins*, which could be evaluated with the tools described below.

Up to now a programming language called "*Advanced Continuous Simulation Language*" (ACSL) has been used. In this tool a frequency response analysis algorithm is documented [MGA]. Since the FCS more and more switched over to use MATRIXx as the general purpose simulation tool and for early Validation of required behavior, the algorithm had to be implemented in there. The frequency response analysis (FRA) worked perfect in ACSL and has therefore been missed in MATRIXx. As part of this thesis the function has been implemented in MATRIXx.

4.2 Theory

The basic idea behind frequency response analysis (FRA) is to find the *gain* $|G(i\omega)|$, and phase shift $\phi(\omega)$ of a system at different frequencies [JOHANSSON]. By doing this the output from a system fed with a certain frequency ω, and magnitude *m*, can be estimated as:

$$y(t) = |G(i\omega)|\, m \sin(\omega t + \phi(\omega)), \qquad \phi(w) = \arg G(i\omega)$$

The simplest way to do this is to feed the system with a sinusoid of a particular frequency and then plot the input and the output. From the plot it is easy to find the gain and the phase shift. This is repeated for different frequencies in an interval of interest. Finally a Bode-style diagram can be drawn. A more sophisticated way of finding phase shift and gain is to use a method based on *averaging*. One way of doing this is to multiply the output of the system with a sine and a cosine of the same frequency as the input (see Fig. 15 below):

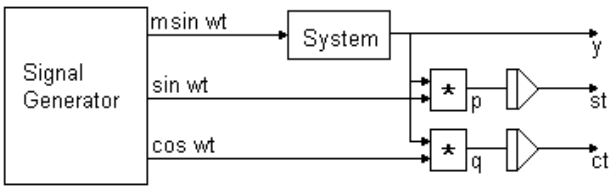


Fig. 15 Frequency Response Analysis (FRA) algorithm

The results of these multiplications are often called *in-phase*, *P*, and *quadrature*, *Q*. These are then integrated over a full number of periods of the input signal. By integrating over a full number of periods, effects of disturbance are minimized. The outputs then look like:

Sine output: $$P(\omega) = \int_0^T y(t) \sin \omega t \, dt = \frac{1}{2} T \left| G(i\omega) \right| m \cos \phi(\omega), \qquad T = \frac{2\pi}{\omega} k$$

Cosine output: $$Q(\omega) = \int_0^T y(t) \cos \omega t \, dt = \frac{1}{2} T \left| G(i\omega) \right| m \sin \phi(\omega)$$

Where *k* is an integer. From these, phase shift and gain can be estimated as:

$$\phi(\omega) = \arctan \frac{Q(\omega)}{P(\omega)} + k\pi \qquad \text{and} \qquad \left| G(i\omega) \right| = \frac{2}{T\, m} \sqrt{Q^2(\omega) + P^2(\omega)}$$

To get a good result that is not affected by non-zero initial conditions, it is recommended to let the system settle before *P* and *Q evaluation* is started.

The results from several computations with different frequencies in a frequency range of interest can then be presented in a regular *Bode*-style diagram.

When this method is applied to nonlinear time invariant systems there is a problem with the system's response depending on *input magnitudes*. A way to solve this is to feed the system with different magnitudes of the input sinus. This means that the frequency interval has to be scanned once for each magnitude. The magnitudes should be chosen so that they match the systems working environment.

4.3 Given implementation of FRA in ACSL

The method used in the ACSL implementation [MGA] is based on the theory previously explained [Theory]. To avoid effects from non-zero initial conditions the program calculates phase shift and gain for one period at a time. The results of the last period are then compared with the results of the previous period. If the difference is considered small the output calculation (phase shift and gain) is recorded.

Frequency calculations are started from the *top* end of the frequency interval and then the frequency is reduced geometrically. The start from the top means that the fast-simulated high frequency points give a fast clue of what the result will look like. The geometrical reduction of frequencies results in equal spacing on a logarithmic plot scale. When the frequency has to be changed, this is done in a way so that no "jump" occurs, i.e. the input signal is continual. This is done by adding a phase shift to the forcing sine function. By doing so, the settling time is reduced.

The *sampling rate* of the data evaluation in the FRA changes during the run of the program. For high frequencies it is set as a 10^{th} of the period length and for low frequencies it is set to an absolute time parameter, depending on the system's dynamics.

4.4 Usage of MATRIXx for FRA

4.4.1 Implementation of FRA in MATRIXx

In MATRIXx the method has been implemented [FRA function implemented in MATRIXx] in basically the same way as it is implemented in ACSL. It is however divided into sub functions for better overview (see Fig. 16 below).

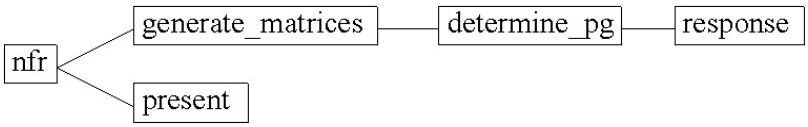


Fig. 16 Sub functions of FRA in MATRIXx

A special feature is that it also detects the *offset* in the output for an input signal with a mean of 0 for each frequency. This feature was a request from the FCS. This of course only applies to nonlinear systems.

The following *sub functions* are used:

- "**`response`**" retrieves all data needed from the examined system for one period simulation for a certain frequency. It generates a time vector and an input sine vector and then simulates the system. For each new period when the frequency is maintained it restarts the old simulation.

- "**`determine_pg`**" determines the *phase shift*, *gain*, and *offset* for a certain frequency and magnitude. It keeps calling the "`response`" function until the phase and gain have settled. When this is done, phase shift, gain, and offset are passed on to the "`generate_matrices`" function (see below). If no *settling* occurs after twenty periods, it stops. This can occur when the system is unstable or more common when there is interference between a discrete system's sampling rate and the update rate of the output of the system. It can also occur when too high accuracy is demanded or when the gain of the system is too low. Then it is possible to form the *average* phase shift and gain of the last ten periods.

- "**`generate_matrices`**" simply loops the "`determine_pg`" function over frequencies and magnifications, where magnification is the outer loop. The values are then put into matrices. It also tracks phase shift for jumps over 360˚.

- "**`present`**" plots the result in Bode form and the additional plot with offset if this keyword is set. All evaluated frequencies are marked with a circle. If the result was achieved by taking the average as mentioned earlier, the result is marked with a triangle. One should be a little careful with these.

- "**`nfr`**" is the main function called by the user.

The inputs to "nfr" are: SystemBuild model name, frequency vector or interval, and vector of magnifications to be evaluated.

Outputs from "nfr" are: estimated frequencies, matrices with phase shifts in radians and degrees, matrices with gains and gains in decibel, and matrices showing which values that have been estimated with the average. These are marked with a "1".

Keywords are: "makesteps", "errcrit", "step", "ialg", and "offset" and control the following:

- "**makesteps**" generates frequency steps in the input frequency interval if this is used as an input. Default is 0. Step size can be changed with keyword "step".

- "**errcrit**" is the difference in phase shift and gain compared with the previous period that determines if the system has settled enough or not. Default is 0.1, which equals 0.1 rad for phase shift, and 0.01 for gain. The value for gain is always a tenth of that for phase shift.

- "**step**" (step size) can be used if a frequency interval is used as an input. "step" is then the multiplication factor in the geometrical increment of frequencies.

- "**ialg**" is the integration algorithm used in the simulation of the system. Default is Variable-step Kutta-Merson Method. This is a good and reliable method for most cases. It has however shown difficulties when applied to stiff nonlinear systems. Then Gear's method is recommended.

- "**offset**" toggles the presentation of offset on (1) and off (0). Default is off.

4.4.2 Comments on the implementation in MATRIXx

The expression for the phase shift does not give a unique solution. The frequency is therefore started from the *bottom*. Most systems used at the FCS, which the function is written for, are of *low pass* type.

It is known that phase shift is close to 0 for low pass systems at low frequencies. This solves the problem with the non-unique solution in most cases. The phase shift is also tracked so that no jumps of 360˚ are made.

The time consuming calculations are the calculations made for low frequencies, therefore it is good to make those first to get a hint of how long it will take to perform the full evaluation. Maybe the frequency span could be changed.

In the ACSL implementation frequency evaluation is started with the highest frequency. With this implementation a lot of work might have been done before the user realizes it will take to long to evaluate the selected frequency interval.

4.4.3 Validation of FRA implemented in MATRIXx

For Validation the code has been run on a few systems. The first system is a simple transfer function, which can be seen in Fig. 17 below.

Example [4-1.]:

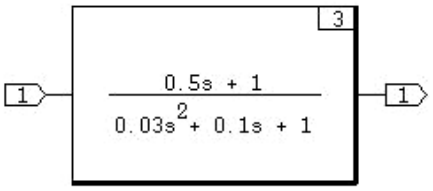


Fig. 17 Transfer function used for validation of FRA

The result in the frequency range between 0.1 and 10 Hz with the "`makesteps`" keyword is presented in a bode style plot (see Fig. 18 below). For comparison the theoretical values are also plotted in the same plot. Since a linear system has been analyzed, only one magnitude is used for the input. A problem with the implementation is to determine when the system is settled. When the gain is low, the forcing sine signal dominates in the integrated sine and cosine outputs. This could be the reason for the jump in phase shift that is seen at 6 Hz. This is a problem related to deciding when *settlement has occurred*. Theoretically the plot should be smooth and monotonically negative as the phase approaches –90˚.

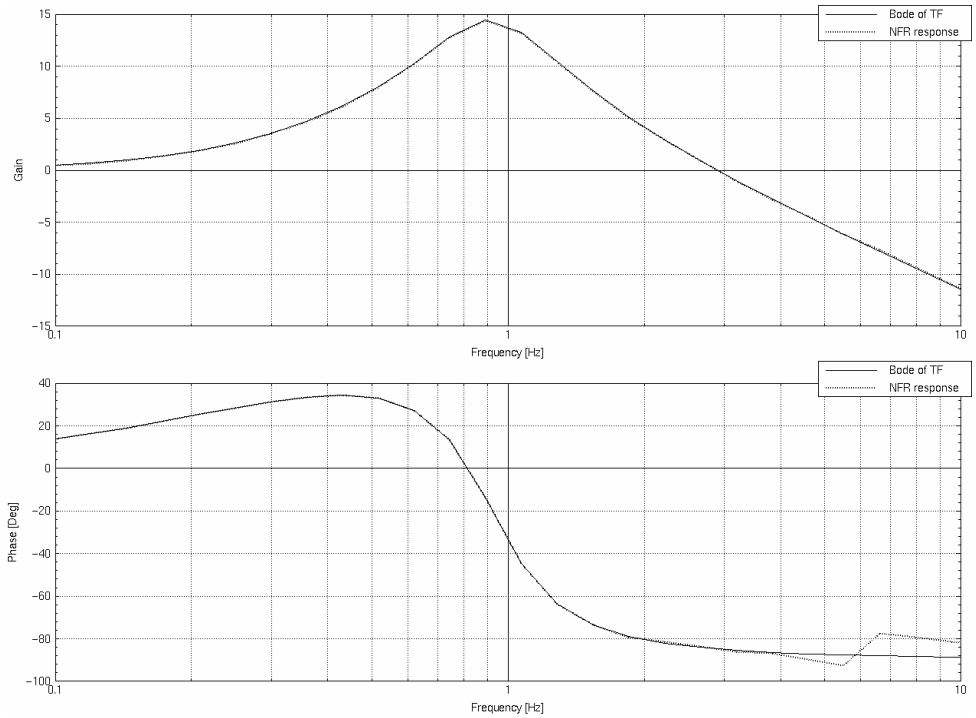


Fig. 18 FRA of transfer function with default error criterion 0.1 rad

An attempt to lessen the effect of the *driving sine* is to set the "`errcrit`" keyword value to 0.01. As can be seen (see Fig. 19 below) the graph becomes more accurate (theoretical values are also plotted in the same graph). The elapsed CPU time for the run goes up from 10 seconds to 17 seconds.

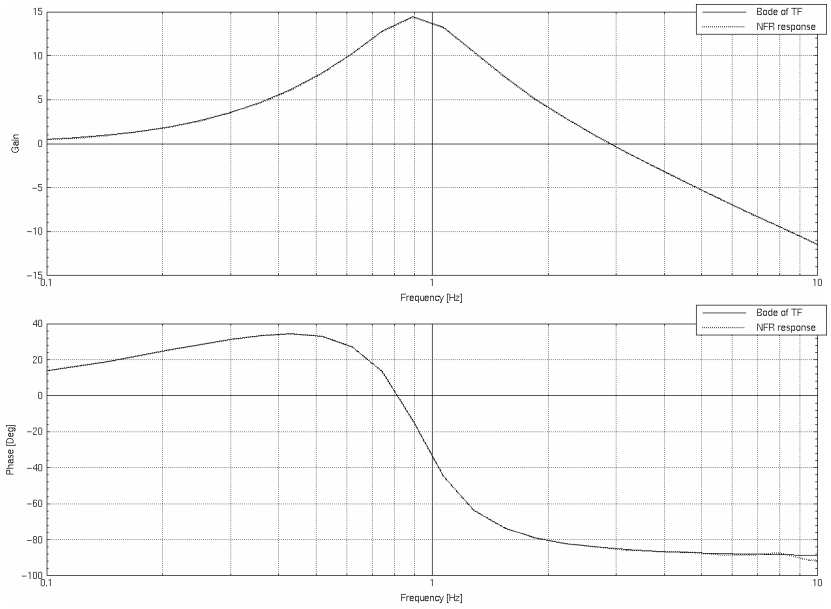


Fig. 19 FRA of transfer func. with "`errcrit`" 0.01 rad and theoretical values

There is not so much to gain in gain accuracy by changing the error criterion from 0.1 to 0.01. For *phase shift* however there is a quite a difference for the highest frequencies as can be seen in the previous plots. By setting the error criteria to 0.001 this is reduced to less than 1˚. The low frequencies are not affected since settling is achieved after the first periods anyway.

Example [4-2.]:

The second system illustrates the averaging function well (see Fig. 20 below):

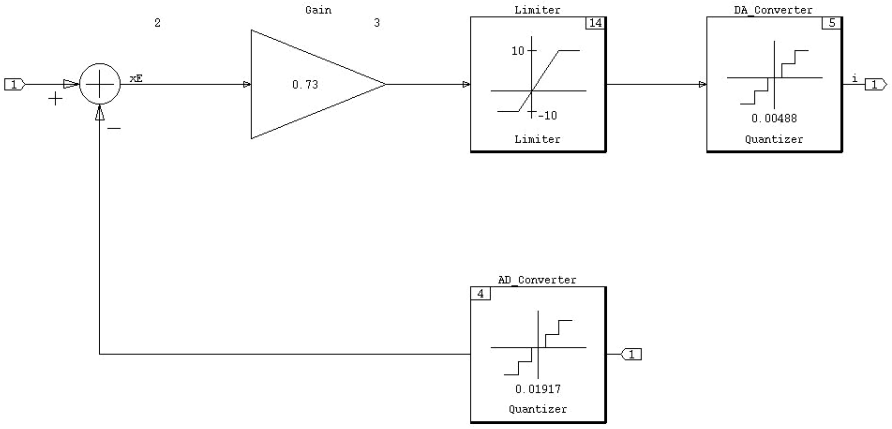


Fig. 20 System used for Validation of the average function

It has the same value on the two input ports. This is a system which in reality has no application but thanks to the two different quantizations shows the task well.

When the program is run on it the following result is achieved (see Fig. 21 below).

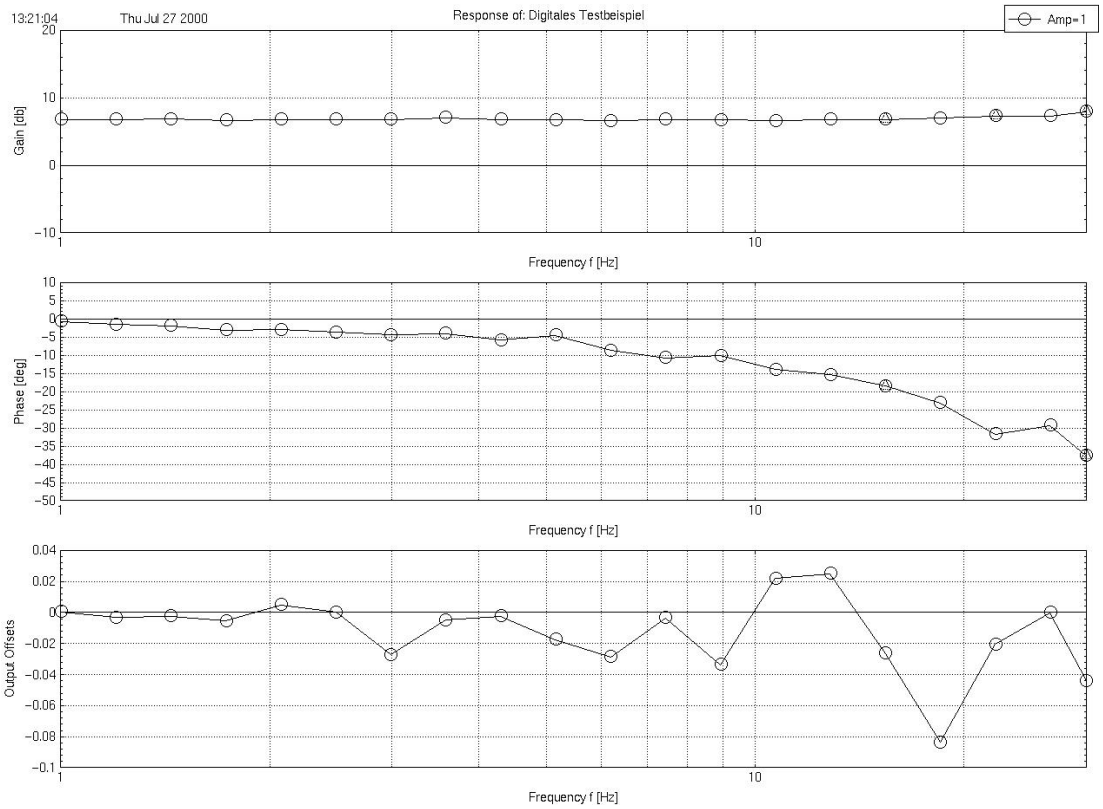


Fig. 21 FRA of system with "`errcrit`" 0.08 rad

Magnification has very little or no effect on the system, despite the non-linearities, and only one magnification is therefore examined. The fifth examined frequency from the right is 15.407 Hz and it is marked with a triangle in both the gain and the phase plot. This means that no settling was reached within the first 20 periods. The average is then taken for the last ten periods. (The first ten are not used since the system should be somewhat stationary before the measurements are taken.).

In this case phase shift and gain never reaches stationary values due to the interference between sample period and input frequency (see Fig. 22 below). This is also the reason for the returned offsets. Since the system obviously have problems in getting accurate values the offsets have to be dealt with care. Since the offsets that should not at all be there are quite small and irregular this could be suspected.

If a poorly damped system that never settles within the first 20 periods is examined, it is possible to give it more time by adding a keyword. This is not done since the systems at the FCS are well damped and the program should be kept simple.

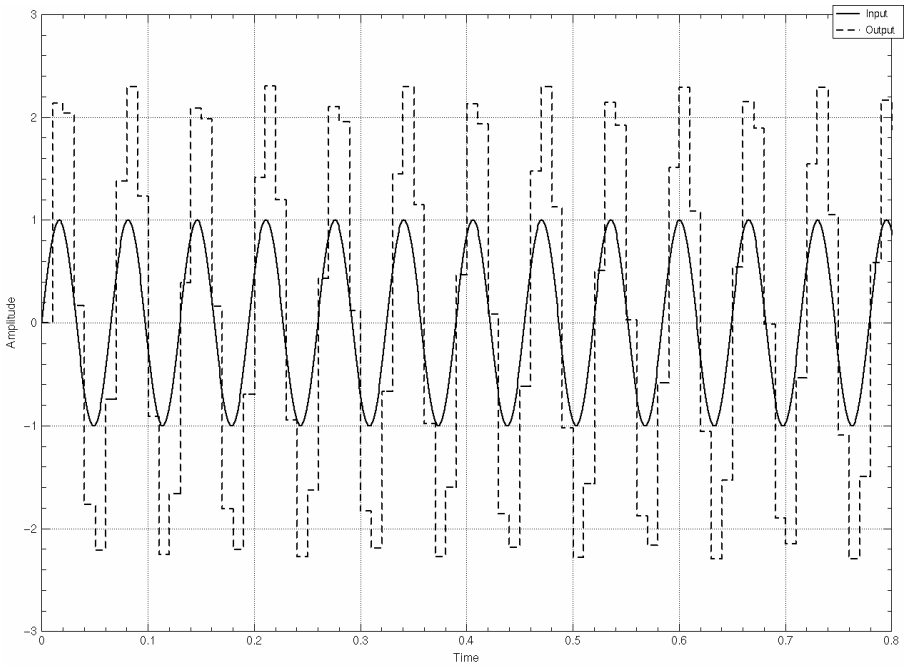


Fig. 22 Simulation of system with an input of 15.407 Hz and amplitude 1

To verify that the estimation by taking the average gives a fairly good result when applied to interference between sample period and frequency, two frequency points on each side of the frequency 15.407 Hz that do reach stationary values are examined (see Fig. 23 below). These have the frequencies 15.39 Hz and 15.42 Hz.

No problems were encountered when the evaluation of these frequencies was examined.

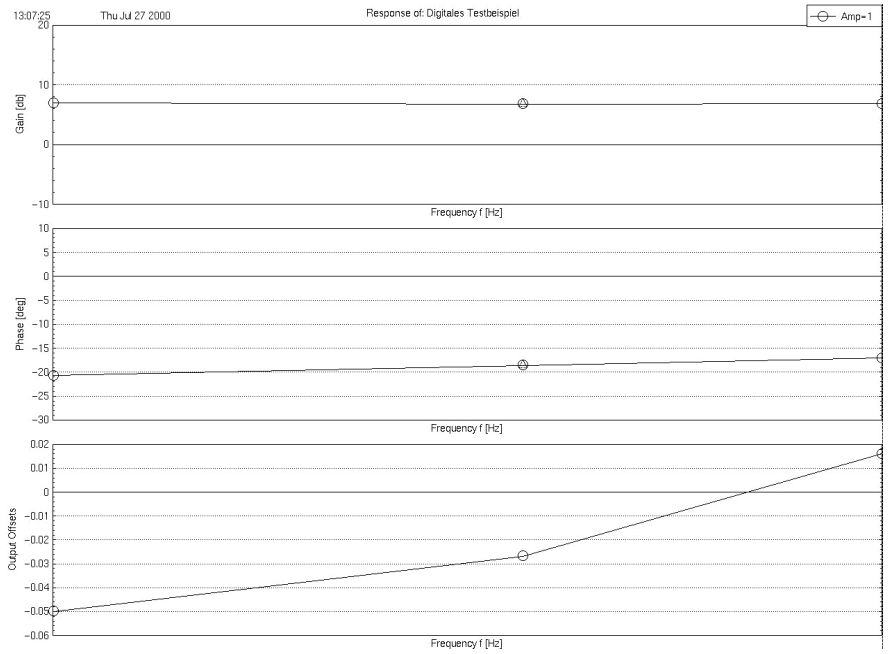


Fig. 23 Examination of frequencies close to system model critical frequency

Example [4-3.]:

To evaluate the function for a larger system it was ran on a version of a rudder model (by EADS also known as Actuator Transfer Function) for an aircraft (see Fig. 24 below).

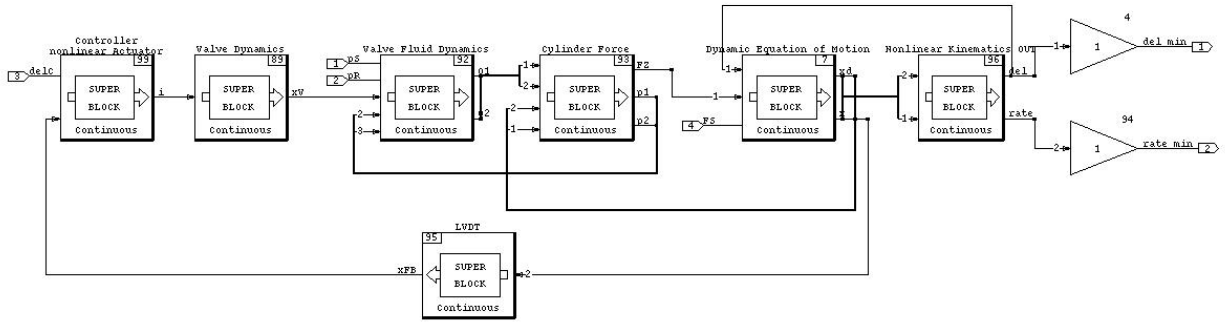


Fig. 24 Rudder model (input "`delC`", output "`del`")

This system has many non-linearities and is therefore fed with a few magnitudes (see Fig. 25 below). These could be seen in the upper right corner legend. By changing the error criterion from the default 0.1 to 0.01 *phase shift* becomes a little smoother for high frequencies whereas *gain* is not affected at all. The reason for this is that the systems used are very well damped due to their applications. A further change of error criterion down to 0.001 does not have any effect. The only result from this is that the numbers get so small so that no settling is reached and the result gets averaged instead. This only makes the run time longer.

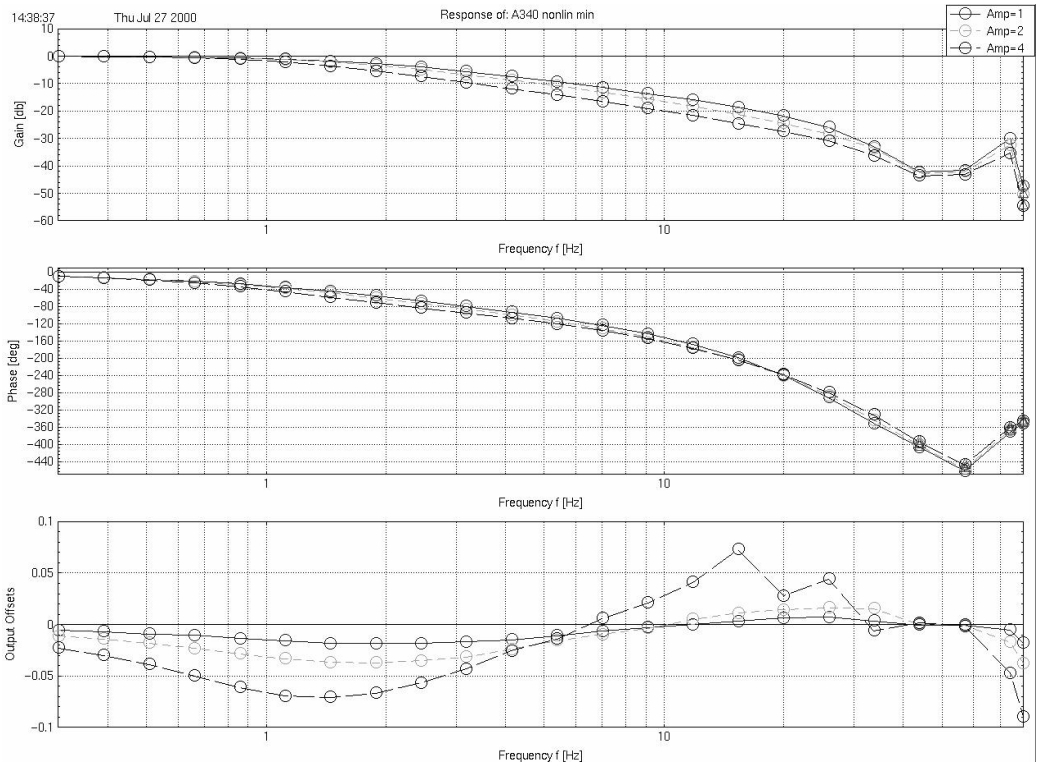


Fig. 25 FRA of rudder model

The values in the diagrams can be verified by looking at the simulation of the system for each frequency and magnitude.

For example it could be seen that for 10 Hz and 1 in magnitude the gain should be around -14 dB, which equals 0.20 absolute and the phase shift should be just below 160˚.

This could be verified by plotting the simulation of the system for this frequency, see Fig. 26 below:

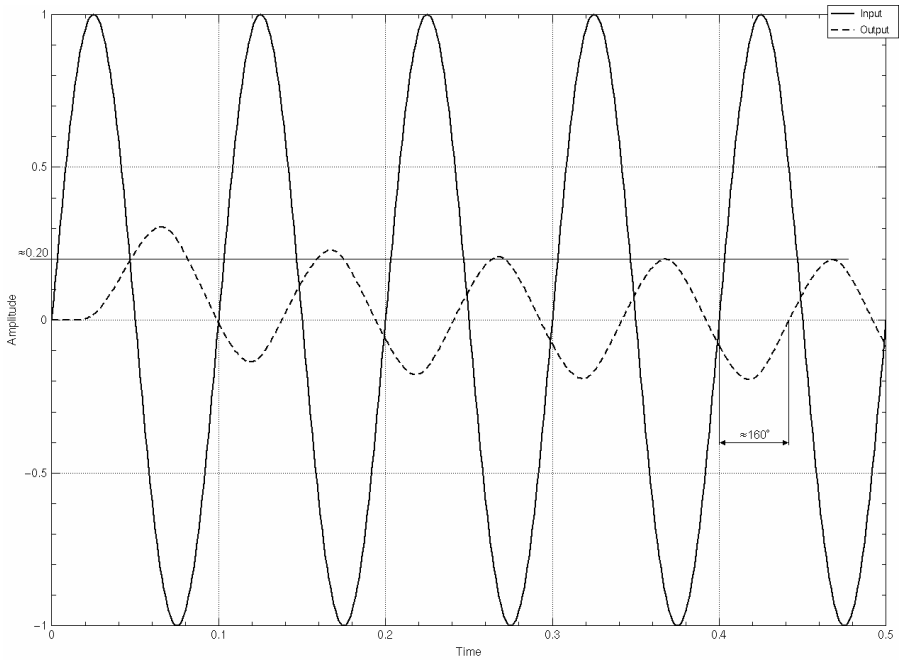


Fig. 26 Simulation of the model with an input sine of 10 Hz and amplitude 1º

A common problem for these big systems is that the default integration algorithm (*Variable-step Kutta-Merson Method*) takes so small steps that they become too many. If the message “more than 1000 steps needed” appears this has happened.

Then “*Gear’s method*” is a good option. When Gear’s method is used on this system no difference in results were found.

4.4.4 Discussion of the keywords in the "`nfr`" command for FRA function

The "**`step`**" keyword is good for initial analysis on a system. Default it is set to 1.2, but can for initial evaluation be set for example to 3 or any other number. Larger numbers give less plot density in the frequency domain, which result in faster execution time.

"**`errcrit`**" gives more accurate results when changed from default 0.1 to lower decimal numbers. There is however a limit for how little it can be set for further accuracy. This can be examined by observing in the Xmath window the result matrices that are generated after each run. Higher accuracy means that more periods have to be run for each frequency and that means longer execution time. Eventually settling is not reached at all for the requested accuracy and an averaged result is returned instead.

"**`ialg`**" changes the integration algorithm in the simulation. If the default "VKM" (*Variable-step Kutta-Merson*) does not work, try "*GEAR*" (*Gear's method)*.

"**`makesteps`**" generates steps if an interval is used as frequency input.

5 Linearization using subspace identification in MATRIXx

5.1 The identification process

The identification of a system is an *iterative* process (see Fig. 27 below). As more information about the system is attained it might be necessary to go back in the procedure and change values or methods.

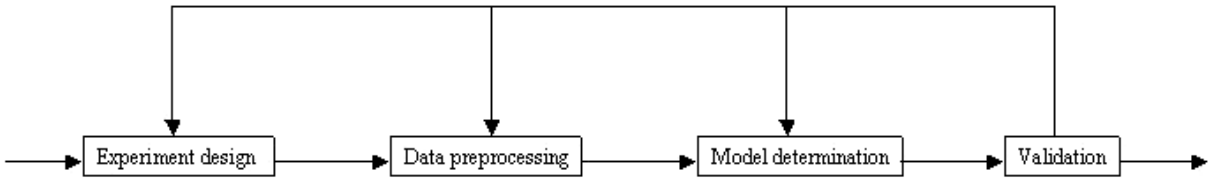


Fig. 27 The identification process

An example of the *identification procedure* described in general in this section can be found in the [Linearization example using subspace identification]. All commands used and mentioned can be found in the System Identification Toolbox of MATRIXx.

5.2 Experiment design

The procedure starts with the design of the experiment. This includes the choice of *input*, *sampling interval* and *duration* of the experiment. As more knowledge about the system is achieved the input could be made more optimal for the given system.

A PRBS (*Pseudo Random Binary Sequence*) is a suitable input to choose [JOHANSSON]. This is available in MATRIXx. The periodicity of the signal is dependant on the number of states used and should be chosen so that it does not repeat during the experiment. The sampling of the PRBS could also be set so that it contains the bandwidth of interest. This can be investigated through the auto-spectra which is available through the command "`sdf`" in MATRIXx.

The offset and amplitude of the PRBS is of most interest since the models often are nonlinear. The input *offset* should be the same as the operating point of the input to the system since the identified model should be well modeled around this point. The *amplitude* should be set to match the working interval of the system. The result will vary depending on if the input enters non-linearities or not. It should also be big enough to give a good signal to noise ratio. A good choice of input should reflect the inputs under normal use of the system.

The *sampling interval* should be set so that no Aliasing or interference is experienced with discrete parts within the system. If a discrete system is to be identified the sampling rate is chosen to equal the sampling rate of the system.

If the sampling rate is chosen to high, the poles of the model will appear as integrators (close to the real number 1 in a pole-zero plot). Experiment duration is also proportional

to the accuracy of the model. The longer duration the more accurate model. A rule of thumb is to choose it larger than 5 to 10 times the largest time constant to be considered by the identified model [JOHANSSON].

5.3 Data pre-processing

By just looking at the data it is easy to see if there are any major errors with the recorded set. Things that are easy to spot are *outliers*, areas of *saturation*, *quantization* effects and *sampling interference*. By plotting the spectral density function it is easy to spot periodic disturbances. This is done with the "`sdf`" command.

Since system identification results in a model, which is a linear approximation of the true system around its operating point, the operating point has to be subtracted from the data. "`detrend`" is a command useful for this.

The *coherence spectrum* is a correlation analysis made for each frequency. It indicates the degree of linear dependence between input and output. Good correlation is indicated with a value close to 1. It also indicates in what frequency interval a model could be expected to be valid.

Scaling is necessary if the identification involves the MIMO case. Then magnitude 1 is requested. It could also be good to remove the initial values of a data set to avoid the problem of initialization and non-settlement.

5.4 Model determination

The implementation in MATRIXx identifies systems by creating estimates of states by calculating the states as intersections between past and future input-output [MATRIXX_ISIM]. This is done with the function "`sds`".

Singular values or *principal angles* of the system are then calculated, based on whether a *dependent* scaled or an *independent* scaled method is used. This is chosen by setting the keyword {`basis`} to either "`combined`" or "`unscaled`". Default it is set to "`combined`". The dependent scaled method will turn the system into *frequency weighted* balanced form (frequency weighting by the spectrum of the input). Scaling the input-output data will not affect the result. The independent scaled method will just turn the system into balanced form without frequency weighting. This method will result in a slightly different algorithm in MATRIXx that might give a slightly different result.

Singular values or principal angles are used for determining the *order of the system*, which has to be done by the user. A good model order is chosen by examining the number of dominant singular values or by seeing how many principal angels that are significantly different from 90˚. A singular value that is lower than another shows that this state has less effect on the system's behavior than the other.

The same applies to principal angles except that an angle close to 90˚ is less significant than a value closer to 0˚. A singular value equal to 0˚ and an angle equal to 90˚ shows that the state has no effect on the system's behavior.

After that model order has been determined it is time for determining a state space model. This is done by using the input-output data and the chosen model order. There are two methods implemented in MATRIXx for this. One is an *asymptotically biased* method and one an *unbiased* method. The biased method often gives better results on real-life data [MATRIXX_ISIM]. The keyword {`bias`} is used for choosing method and can be set to 0 for the unbiased method and 1 for the biased. 0 is default.

The "`gui`" keyword is very helpful when using the "`sds`" function since it gives the user a Graphical User Interface (GUI) box containing all the tools available and related to the "`sds`" function, also the Validation methods.

5.5 Validation

There are several tools for Validation purposes. Some useful are the following:

- *Knowledge* of the system suggests a certain model order.
- *Prediction* – the systems real output is plotted together with the output of the model fed with the same input. This is very revealing since the two should match well. If possible it is to prefer to use a different data set than the one used for identification.
- *Prediction errors* – the difference between the results in "prediction" is plotted. This should look like white noise. This is also called "*residuals*".
- *Covariance prediction errors* – gives the covariance of the residuals. There should be a spike for zero and in the rest of the interval it should stay within the 95% confidence interval.
- *Cross correlation input <-> prediction error* – this is a test to see the correlation between input and residuals. As low correlation as possible is demanded. There is a 95% confidence interval drawn in the plots when they are presented. This is a good limit to aim for. If the plot is within the limits for positive values the model is not under modeled (too small order). If they are outside the model order is too small. For negative values the plot indicates presence of feedback in the system. This means that residuals affect later inputs. It could also indicate on non-causality. In MATRIXx "lags" is the number of samples cross correlation is estimated for.
- *Pole-zero plots* give a hint if the model has too high order. If this is the case there should be pole zero cancellations. This means that there are poles and zeros close to each other.

5.6 The final model

The final model given by MATRIXx is in state space form and discrete. To get the numerator and denominator for a transfer function the command "`numden`" can be used. To make the system *continuous* the command "`makecontinuous`" is helpful.

6 Linearization of an elevator flight control model

6.1 Description of the elevator flight control model (pitch control)

The elevator flight control model describes how the inputs from the side stick should be translated to outputs to the elevator (see Fig. 28 below).

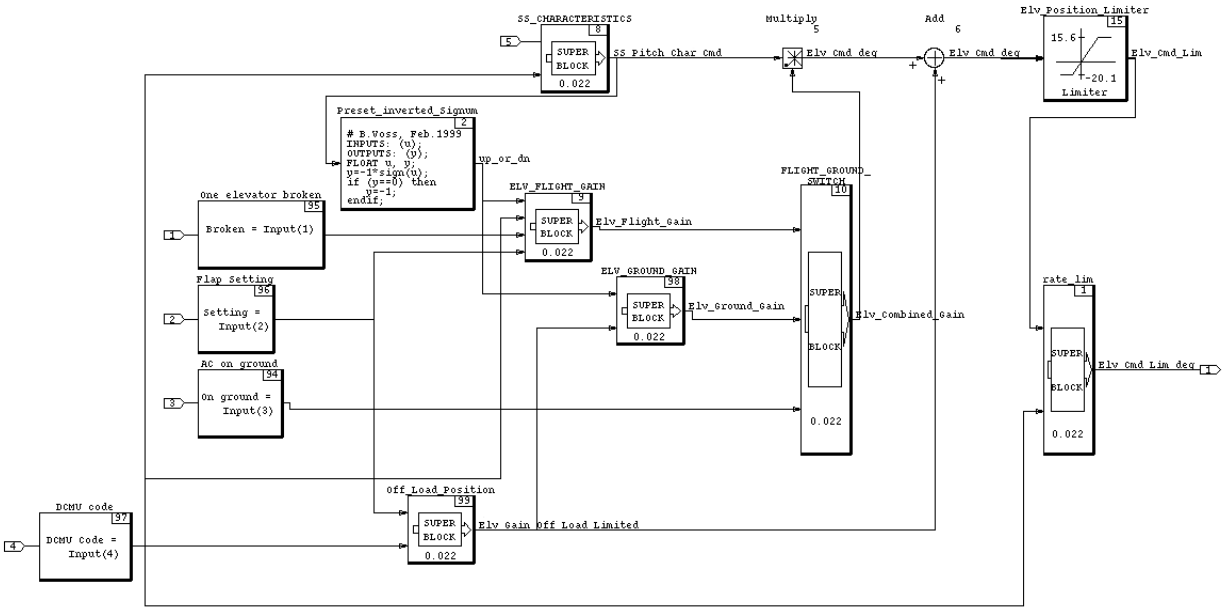


Fig. 28 Elevator flight control (side stick input is no. 5)

The main input, which is a *side stick command* in the range of ±1, enters the system as input no. 5 in block no. 8. The signal then continues through blocks no. 5, 6, 15, and 1. The output is a command to the elevator.

The other blocks set the values in blocks no. 5 and 6 depending on the configuration of the aircraft. Descriptions for each block:

- Block no. 97 – CMU code is the choice of control laws. Only the normal law has been examined. This means that the CMU code is 0 and the CMU code input is neglected in the description of the following blocks.
- Block no. 8 – Contains a *saturation* with limits ±1. A gain scheduler follows the saturation. The inputs are between –1 and 1 and the outputs are also between –1 and 1. The shape is close to linear, only a slight shift.
- Block no. 5 – A regular *multiplication* with a factor given by the flap setting. The factor also depends on if the aircraft is on ground or in the air and if both elevators are functional or if only one is. The factor also changes if the side stick has a positive or negative deflection (positive is forward). The values for normal flight are fixed defined parameters for positive and negative deflection.

- Block no. 6 – Adds an *offset* to the elevator deflection. The value is dependent on the flap setting only. This is done to compensate for the nose down momentum when a higher flap setting is set.
- Block no. 15 – A *saturation* given by the maximum deflection of the elevator. The signal is never saturated in normal flight configuration (flap 0, both elevators working, and aircraft flying).
- Block no. 1 – A *rate limiter* for the elevator. The rate limiter can be seen as a type of low pass filter. It limits the derivative of the input and always strives to reach the same value as the input. For example, when it is fed with a relatively slow sinus wave, the output is the same sinus. When either amplitude is made larger or the frequency is set higher the rate limiter will be activated and transform the sinus shaped input to a triangular shaped output (see Fig. 29 below). At 0.5 Hz and 7 in amplitude (far left), the output (dotted) is not affected. As frequency is made higher the shape is turned more and more into a triangular wave (1 Hz middle, and 2 Hz far right).

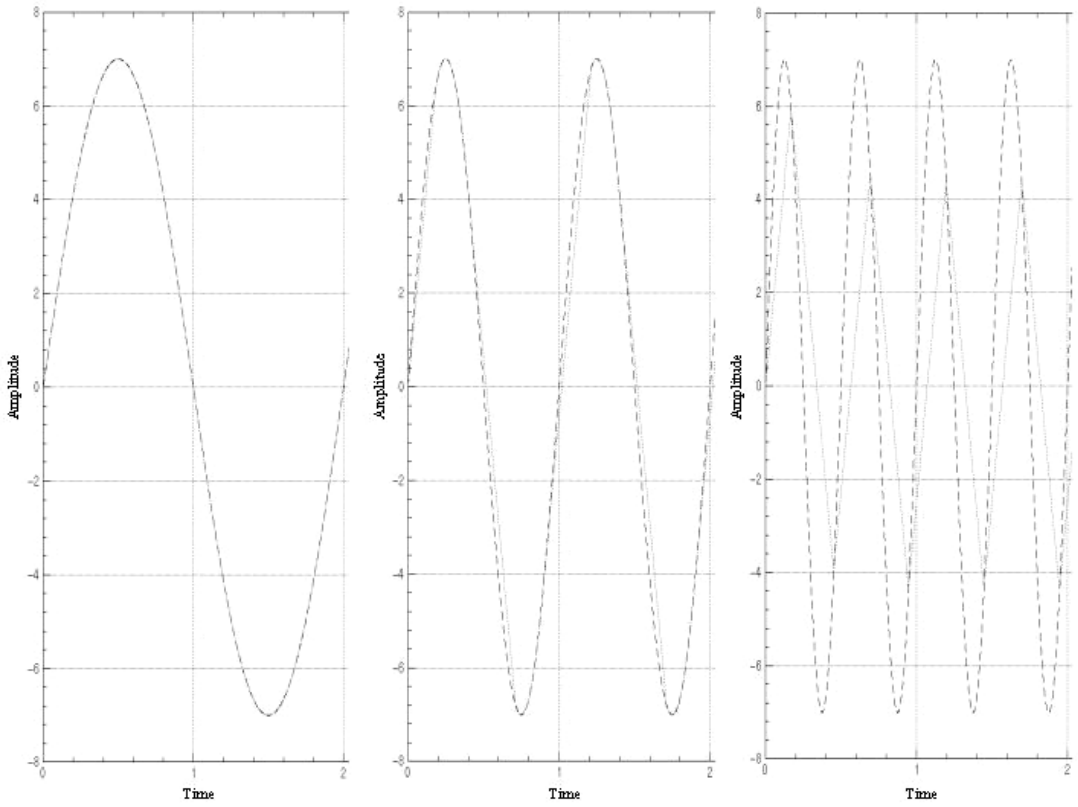


Fig. 29 Effect of rate limiter

- Block no. 2 – Determines if the side stick has positive or negative deflection.
- Block no. 99 – Sets the *offset* in block no. 6 depending on the flap setting. Flap settings are: 0, 1, 2, 3, or 4.

- Block no. 9 – Sets *different gains* in block no. 5 for positive and negative side stick deflection depending on flap setting and if both elevators work or not. The outputs ranges between 5.8 and 10.5 depending on flap setting, lower values for lower flap settings. The value for positive deflection is always a little (15-20%) higher than the one for negative. If one elevator is broken the value is doubled.
- Block no. 98 – Sets the *gains* in block no. 5 when the aircraft is on ground. The gains are typically between 12 and 20.
- Block no. 10 – Contains two *integrators* to ensure "smooth" switch between different gains in block no. 5.
- Blocks 94–96 – Are used to set different configurations.

6.2 Linearization example using "`lin`" & "`trim`" commands

An examination of the model is done using the "`trim`" command. All the inputs are frozen except input number five since this is the only input that is going to change during the later simulations. By then inserting different initial states it is possible to get an idea of where the *equilibrium points* are located.

This shows that the system has an infinite numbers of equilibrium points. This is confirmed by simulating the system with the different equilibrium points as initial states.

The simulation shows that there are no transitions, which indicates that the states are in fact equilibrium points.

The only state that cannot be chosen arbitrary is the fourth state that is 1.3546e13. This means that the only keywords that need to be used are the *state*, *input perturbation* vectors and the *initial states*, "`dx`", "`du`" and "`x0`".

In the first linearization all perturbation vectors are set to one and then each state and input are gradually decreased and then also increased. This is done just to get a hint of how the system behaves after different linearisations.

If the model that is retained is investigated after each change, it is easy to notice that the only parameters that are changing are the parameters connected to the fourth state and the fifth input. This is not so surprising if it is considered, that the only state that could not be chosen arbitrary was the fourth, and that all the other states could be chosen arbitrary indicates that these three states are already linear.

That only input five makes a difference in the linearization could perhaps be explained by the fact, that this is the only input, that was not frozen during the "`trim`" command. So concentrating on these two elements it is possible to get a reasonably good linearization, it is though only valid for a specific amplitude and frequency.

In the figures below (see Fig. 30a-d) it is shown how the linearization is affected during the examination of the dependency of the perturbation vectors (see Fig. 31 below). The dotted line is the linearization and the solid is the simulation of the system.

The plots show the amplitude as a function of time.

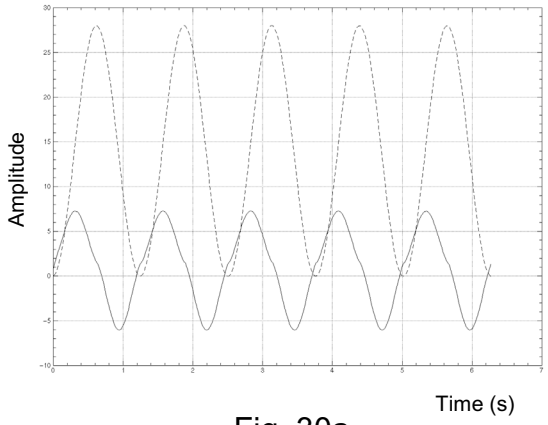


Fig. 30a

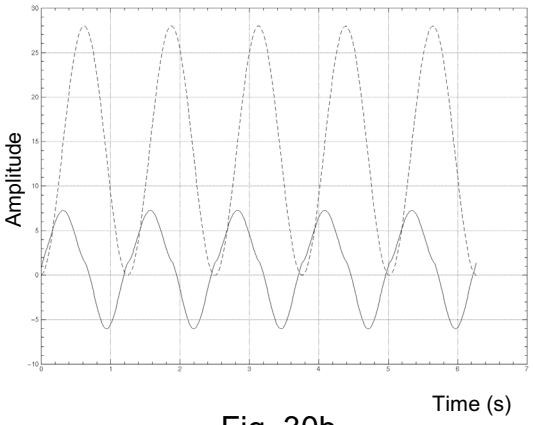


Fig. 30b

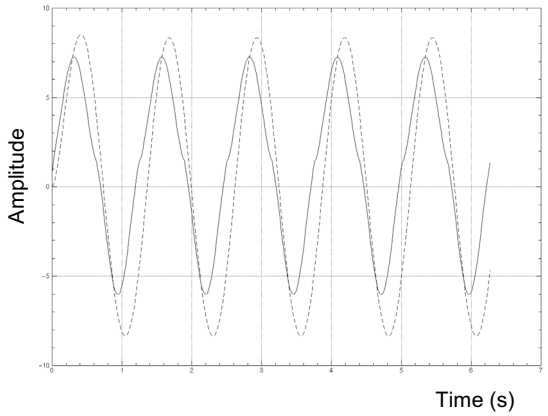


Fig. 30c

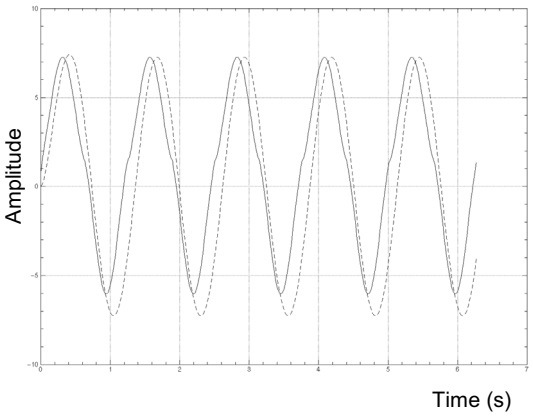


Fig. 30d

Fig. 30 Linearization effects due to perturbation vectors

Figure number	Vector dx	Vector du
Fig. 30a	[1, 1, 1, 1]	[1, 1, 1, 1, 1]
Fig. 30b	[100, 100, 100, 1]	[100, 100, 100, 100, 1]
Fig. 30c	[100, 100, 100, 10]	[100, 100, 100, 100, 1]
Fig. 30d	[100, 100, 100, 10]	[100, 100, 100, 100, 1.5]

Fig. 31 Choice of “`dx`“ and “`du`“

This shows how it is possible to achieve a reasonable good result. The frequency of the sine wave in this example is 5 Hz. For this frequency it is, as visible in figure Fig. 30 above, not possible to get a linearization without a certain phase shift.

Investigating linearization for smaller frequencies it is shown that a linearization with a much smaller phase shift can be obtained.

6.3 Linearization example using subspace identification

6.3.1 General

The method used here follows the procedure described in the chapter [Linearization using subspace identification in MATRIXx]. All the used code can be seen in the chapter [List of commands used in the subspace identification] of the appendix below.

6.3.2 Experiment design

The first step is to design the experiment. A PRBS is chosen as input. The PRBS function implemented in MATRIXx initiates a change for every sample and therefore emphasizes on quite high frequencies. This is not interesting in our case, so the rate of change is decreased. This is done by keeping the value in the output for a few samples and then take the next PRBS value. The chosen interval stretches up to 23 Hz. Higher frequencies are not used since the sampling rate of the system is 0.022, which equals 46 Hz. The *Nyquist frequency* is therefore 23 Hz. By Nyquist frequency is meant the largest frequency the system can handle where no Aliasing will appear.

The amplitude is set to 1 since there is a saturation with limits ±1 as first component in the investigated system. Offset is initially set to zero to match the input interval of the saturation. These levels of input also match the input under normal use, and are therefore natural to choose. The sampling interval is set to the same as the system's, to be able to get as much information as possible from the system.

The input is plotted, to confirm that it looks as requested (see Fig. 32a below). The spectral density is also examined to confirm that the frequency distribution is OK (see Fig. 32b below). This is done with the command "`sdf`" and a "PDM" as input. By using a "PDM" the correct values for frequency is attained.

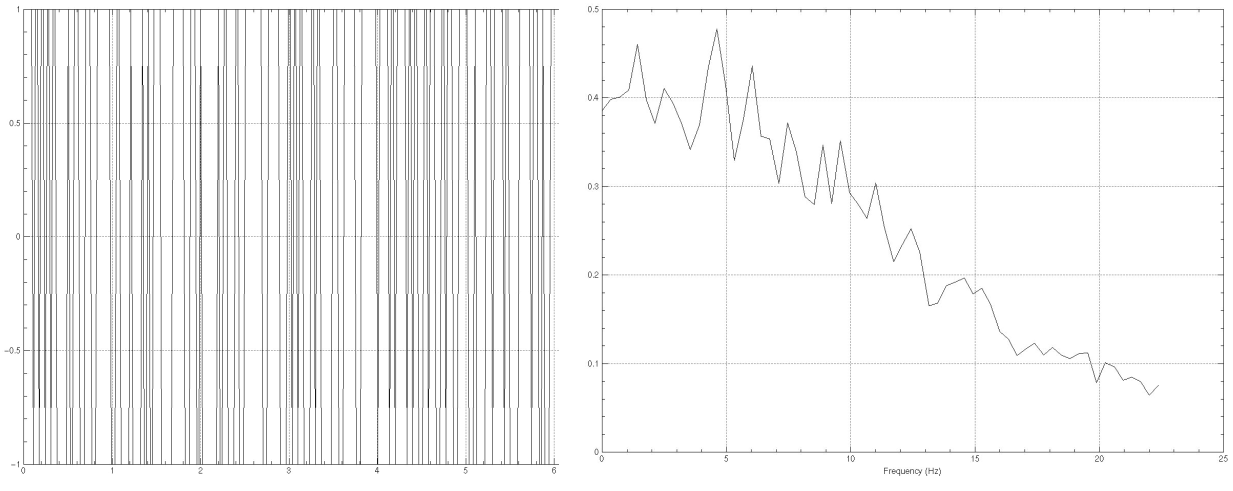


Fig. 32 a) Part of PRBS input, b) Spectral density of input

Two simulations are run with different input series. This is to get one data set for identification, and one for validation.

6.3.3 Data pre-processing

The simulations are then run. The output is plotted for confirmation and to get an impression of what happens in the system (see Fig. 33 below). It is easy to see the different effects of the system, the offset due to the different gains for positive and negative deflection and the added offset.

The change from square shapes to ramps due to the rate limiter is also possible to spot. It also looks like it saturates, but it does not. This is due to the shape of the input and that the system basically only multiplies it with a factor.

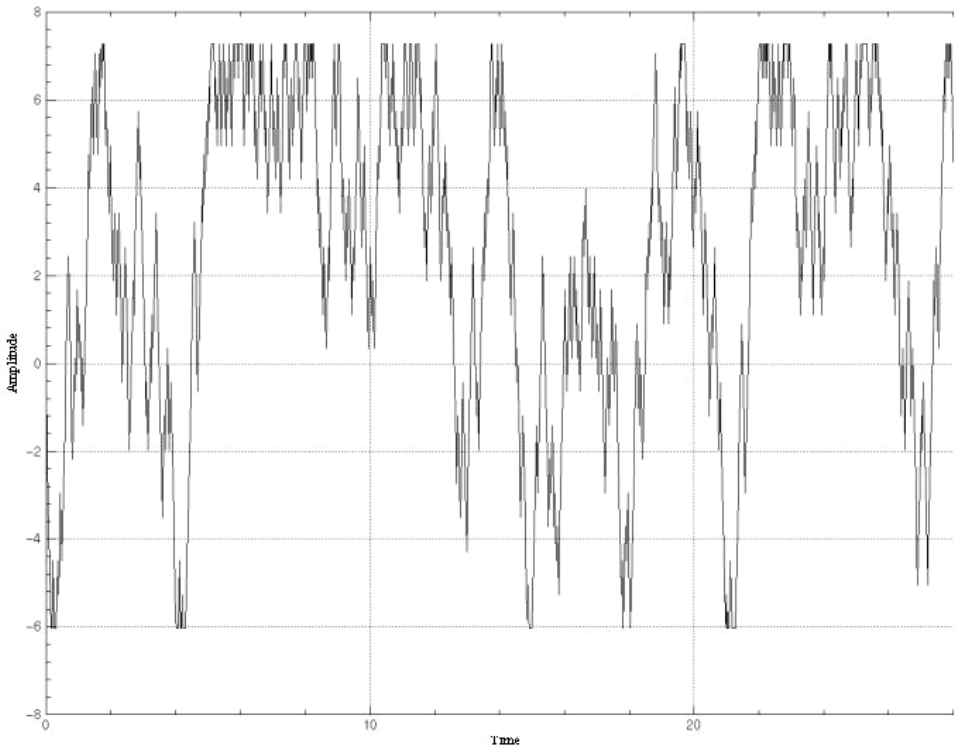


Fig. 33 Output from system when fed with the mentioned PRBS signal

The *spectral density* can also be plotted to get an impression of which frequencies are let through (see Fig. 34 below).

In this case it is easy to see that only low frequencies pass through the system.

This is due to the rate limiter that prohibits fast change.

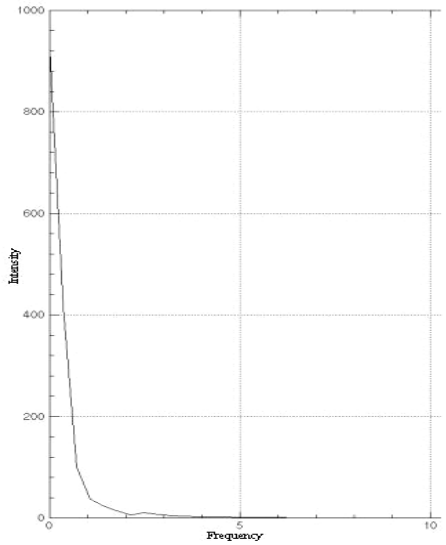


Fig. 34 Spectral density of output

Plotting the *coherence spectrum* gives a good impression of in which frequency interval the linearized model could be expected to be valid (see Fig. 35 below). The spectrum is always mirrored in the *Nyquist* frequency. Only the left hand side should therefore be considered. A value close to one is a sign of good dependence (linear dependence) between input and output for the given frequency.

Normally a result lies very close to one for lower frequencies and then decreases, as the frequency gets higher. In this case the best dependence reaches just over 0.8. This indicates that the resulting linear model will not be as good as wanted. This is due to the many strong non-linearities mentioned in the chapter [Description of the elevator flight control model (pitch control)].

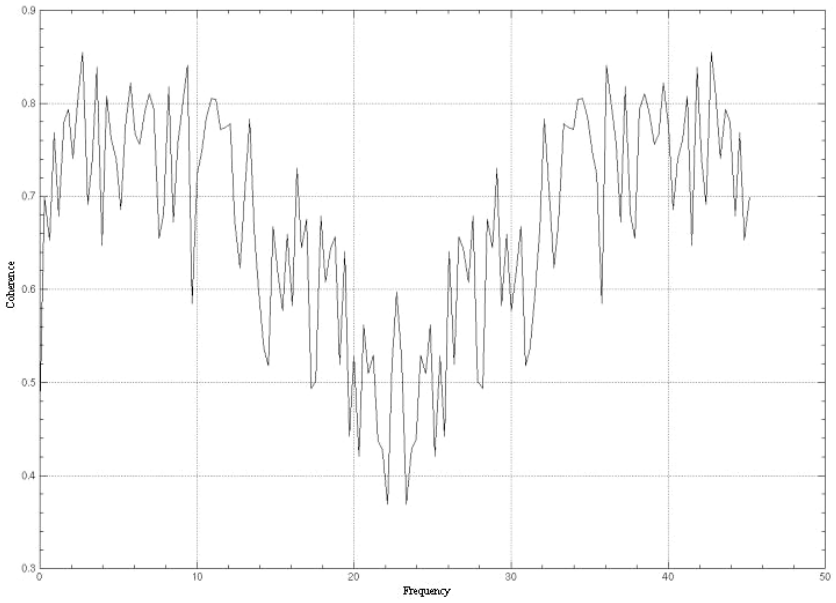


Fig. 35 Coherence spectrum between input and output

Detrending is done on the output to remove the offset, which cannot be modeled anyway. Removal of the initial settling of the system is not done since it settles so fast so that it has no effect on the retrieved model.

6.3.4 Model determination

The subsystem identification is then started with the command "`sds`" and the keyword "`gui`". A bar plot is showed (see Fig. 36 below). It is easy to see that a first order model should be chosen. This is also the conclusion from the examination of the system, where the rate limiter is seen as a low pas filter and the rest as a constant. This is done and a model is calculated.

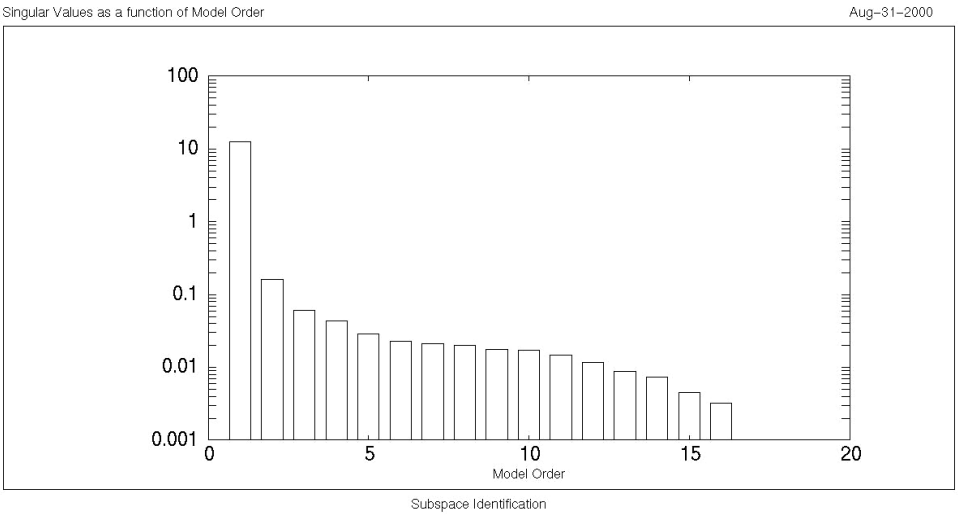


Fig. 36 Bar plot showing singular values

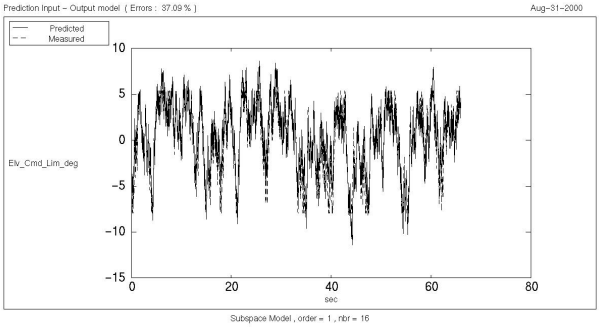
It is also possible to identify the system with the unscaled or unbiased method explained in chapter [Linearization using subspace identification in MATRIXx]. This has no effect on this system. The result is the same.

6.3.5 Validation

The Validation of the model is not a pretty sight. The prediction (see Fig. 37a below) follows the basic shapes fairly well thanks to the low pass type characteristics, but that is about it. The error is calculated to be 37.09 %. The *residual plot* (see Fig. 37b below) is clearly not white noise.

The *covariance of residuals* (see Fig. 38a below) is rather a rectangular distribution than a spike for value 0. The only thing that looks satisfactory is the *cross correlation* between input and residuals (see Fig. 38b below).

This confirms that there are no dynamics that have not been modeled between the two. This is an indication that the model order is not estimated to low which could have been the reason for the other results.



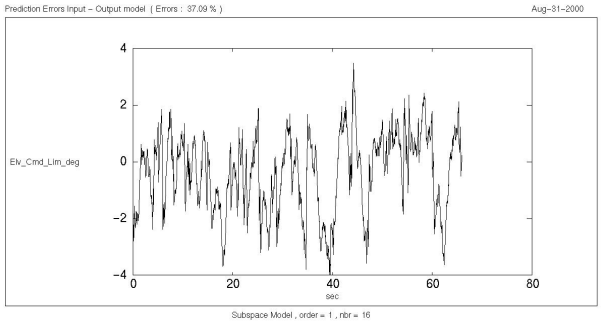
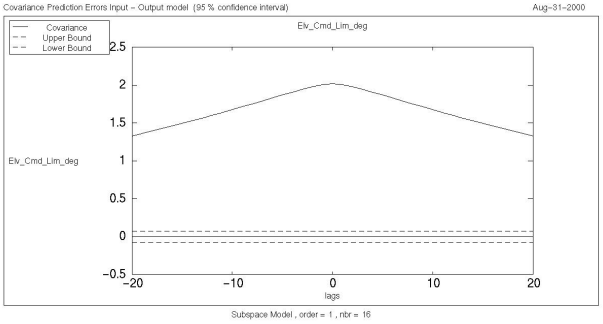


Fig. 37 a) Prediction plot, b) Residual plot



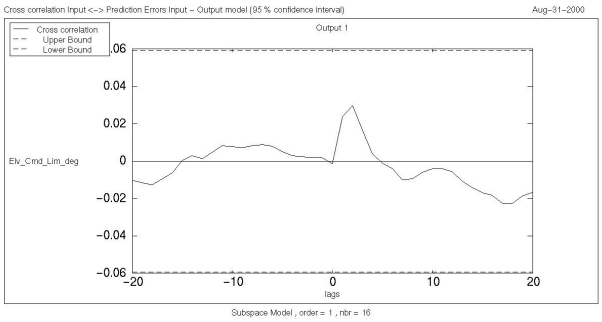


Fig. 38 a) Covariance of residuals, b) Cross correlation

Since the result is so poor when validated against identification data, the validation against the validation data is not done. This could be done in the same way as with the identification data but the result would be even worse.

6.3.6 The input's effect on final model

When the *magnitude* of the input is changed to a smaller, so that the effects of the rate limiter almost disappear, the result will be radically different. Then all results are close to perfect but the *pole zero plot* (see Fig. 39 below) indicates that the model might be over-modeled. The pole-zero pair is quite close together and could be considered to maybe cancelled.

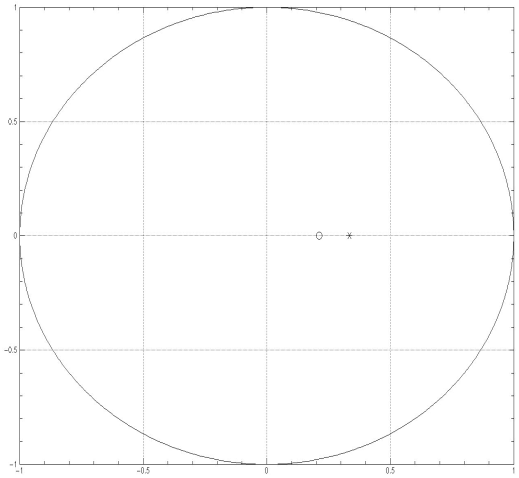


Fig. 39 Pole-zero plot of above mentioned system

Actually a model with only a gain could be considered when modeling the system and when the rate limiter is not in effect.

When an input with a magnitude that saturates in the first block is inserted the system will experience the input as the input used in the first identification with magnitude 1. This will result in the same model as in the first identification but with a lower gain, since the input is immediately scaled to be smaller.

If the input is given an offset and a magnitude so it corresponds to only positive deflection of the side stick, a better corresponding model will be attained. The reason for this is that the non-linearity in block no. 5 (see Fig. 28 above) will become linear in the interval. The model will however only be valid for positive deflection.

If the configuration of the aircraft in the model is changed, the linearized model will be changed. The main difference is that the gains and the offset are changed according to gain schedulers. Due to the *gain schedulers* it is wise to make a new linearization for each case that will be studied.

What will be discovered is that when one elevator is broken, the signal will saturate for certain magnitudes at a later stage within the model (Block no. 15 in Fig. 28) due to the doubled gain. This will result in a model that for low magnitudes and frequencies is equal to almost a *constant* since the rate limiter or the saturation have no effect. For higher amplitudes and frequencies the shape will turn into *triangular* due to the effect of the rate limiter. Then for higher amplitudes with lower frequency when the rate limiter has no effect but the saturation does, the saturation will set in as a gain with a factor less than 1. The shapes will then be close to square.

The result of all the identification of this system is, that a linearized model valid for more than one configuration is virtually impossible to make. It is also impossible to make a model valid for a wider frequency or magnitude interval within each configuration. This is the sad fact that sometimes has to be realized and accepted when dealing with linearisations.

6.4 Discussion of results

The application of the "`lin`" command and *subspace identification* on the elevator flight control give limited results.

The attained linear models are only valid in very *limited* regions of frequency and amplitude. The linearisations are also quite poor in the regions where they actually should be valid. This is due to the many components in the system that are hard non-linearities.

Here follows an explanation to the poor results when trying to linearize the model.

If the input is kept between ±1, the initial saturation in block no. 8 does not affect the system. If the input is larger, the signal will be transformed to square shape by the saturation. This is not possible to model good with a linear model. This is normally not the case so this saturation should not affect the model. The gain scheduler in the same block will affect a little if the input has different amplitudes in the interval of ±1.

The gain in block no. 5 is a major problem since it varies with positive and negative inputs. This is not possible to do with a linear system according to the definition of a linear system, see chapter [Linearization]. Therefore some sort of *average* has to be taken. This will result in that the gain will not be correct. The gain may differ as much as 20% for certain configurations.

The addition of an *offset* in block no. 6 also causes trouble. Offsets do not work for the same reason as in the previous block. The dynamics will be the same independent of the offset, but the output will be shifted.

Under normal configuration the *saturation* in block no. 15 has no effect, but when higher flaps are set and especially when one elevator fails, the saturation's will be reached. This will affect the transfer function's gain and the shape of the output, which cannot be modeled good with a linear model.

The *rate limiter* in block no. 1 is probably the largest problem. As seen in the general description of the block, it is not only of low pass type; it also changes the shape of the signal depending on amplitude and frequency (from sinus shape to triangular, see Fig. 29 below). This cannot be achieved with a linear function.

The many gain schedulers used to give different gains for different configurations motivate that a linearization is done for *each* combination of interesting configuration to make the best of the linearisations.

Due to the lack of dynamics in the path of the signal a simple constant is the best type of linearization that is possible to achieve before the signal enters the rate limiter. *The best linearization of the rate limiter is a filter of low pass type*. This will however not give the triangular shape of the output that the system gives.

7 Linearization of a rudder model

7.1 Linearization using "`lin`" and "`trim`" commands

Since this rudder model (by EADS also known as Actuator Transfer Function) contains several nonlinear elements it is necessary to use the trim command. The model has four different inputs, all these inputs can be used in the trim command, but since three of them are constant it is only necessary to enter one.
The command used to get the working point:

```
[xt, ut] = trim ("Rudder", {u0=[2],  xdt_float = [1,2]})
```

When the working point is calculated the perturbation vectors must also be chosen, this is a little harder. Often it is good to choose the parameters in the same size as the working point. To achieve a good working point it is often necessary to try several different combinations of the perturbation vectors. The linearization command used:

```
lin_sys = lin("Rudder",{u0=ut, x0=xt, dx = [3e7, 3e7, 0.0001, 0.01, 0.01, 0.01, 0.01,
0.01], du = [1]});
```

The result of the linearization compared to the original model when they are simulated with a step is showed below (see Fig. 40 below). As can be seen they do not make an exact match and it is probably possible to get a better match if the perturbation vectors are studied and evaluated more.

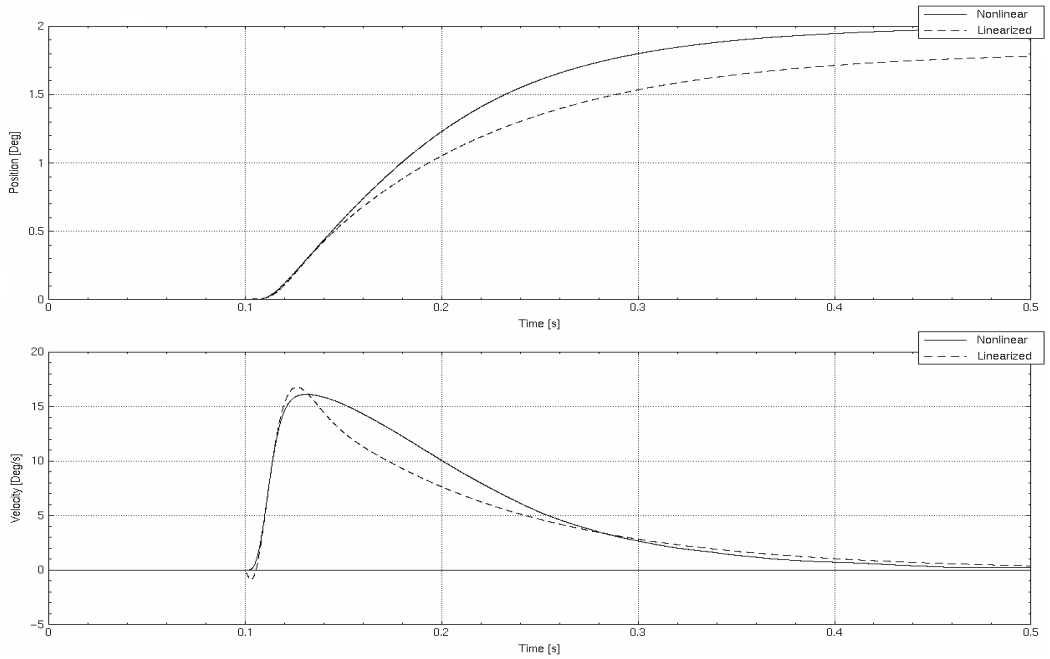


Fig. 40 Comparison between the nonlinear- and the linearized model

7.2 Linearization using subspace identification

The same steps as presented before are used and the code used is adjusted just a bit [Linearization example using subspace identification]. The proper sampling interval is set and the frequency distribution is adjusted. Amplitude is set to 2 to simulate level flight when deflection is not so large. Input and output are both detrended before inserted and the "`sds`" function is invoked.

A sixth order model is chosen and the result from a simulation in time domain with a validation input series indicate that the model is good (see Fig. 41 below).

As a reference the given *manually linearized* model is also plotted with the same input. As can be seen the original nonlinear model (True system output in graph) and the identified model follow each other closely.

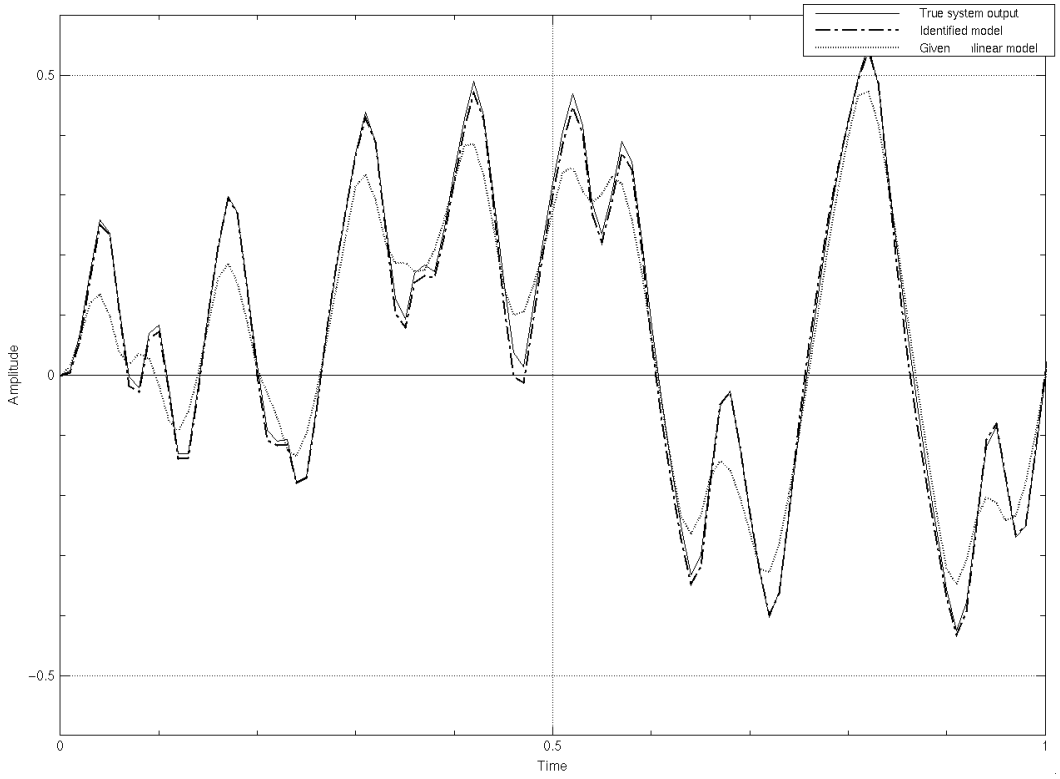


Fig. 41 Validation output

7.3 Comparison between results from all presented procedures

A comparison between the two different linearization methods and the true output shows well matching results.

The comparison is made in both time and frequency domain. In all three cases input amplitude is set to two, to enable comparison between the three.

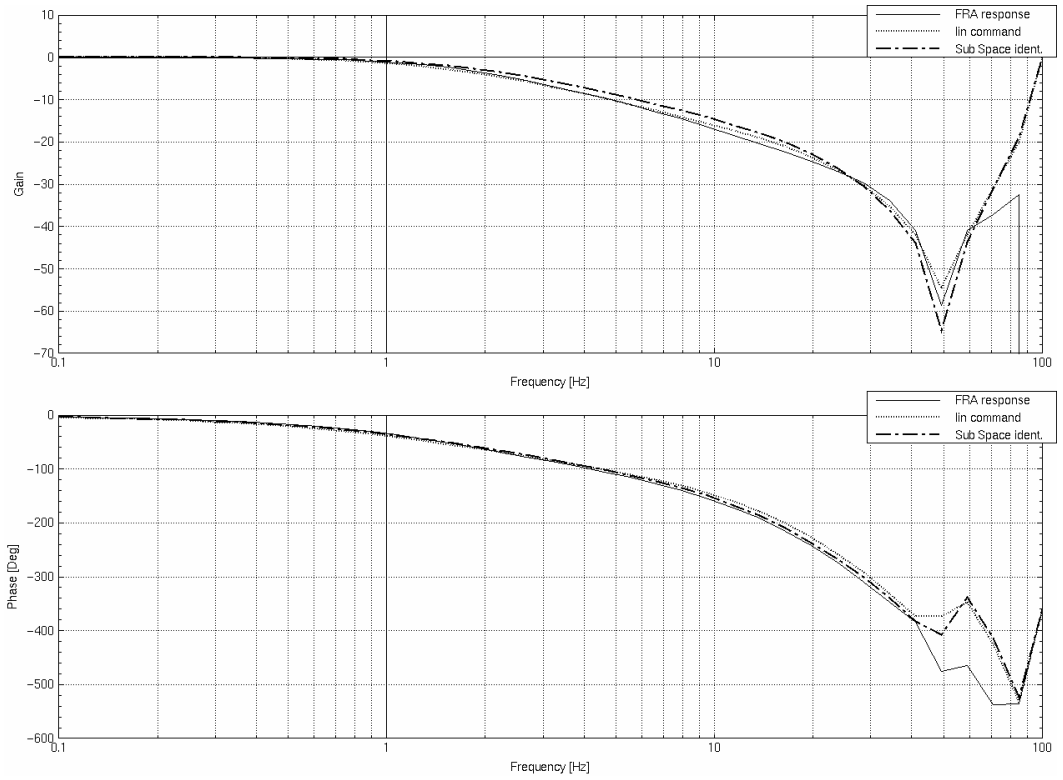


Fig. 42 Comparing Bode plots from the different methods

As can be seen the result is very satisfactory. The plots match very well up to 45 Hz. This is because of the Nyquist frequency that is 50 Hz.

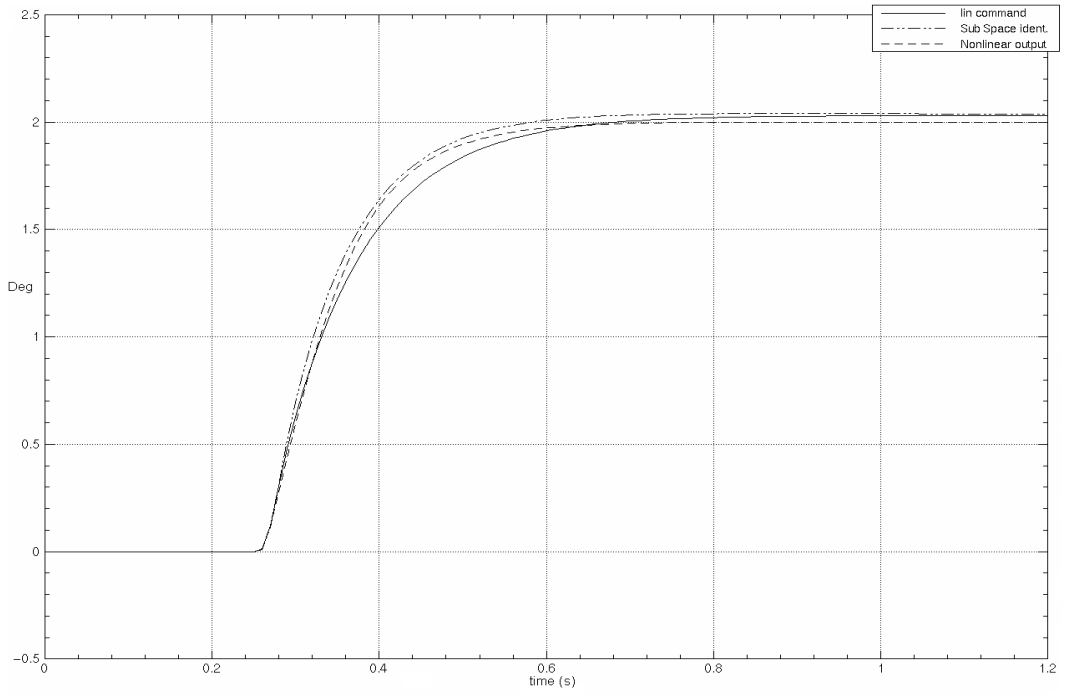


Fig. 43 Comparing Step Responses from the different methods

8 Nonlinear control

8.1 Aircraft model

The generic elastic aircraft model (see Fig. 44 below) is based on a finite element approximation of the structure in combination with unsteady aerodynamics [SCHULER]. The model is described by about 100 000 degrees of freedom. Using modal analysis techniques the model is reduced to 33 modes. The resulting model consists of 3 rigid body modes and 30 elastic and their cross coupling. It is possible to measure positions, velocities and accelerations at different points of the aircraft. The model represents cruise flight at a specific velocity and at constant altitude. The working point can be changed by assigning the matrices different velocity and altitude values.

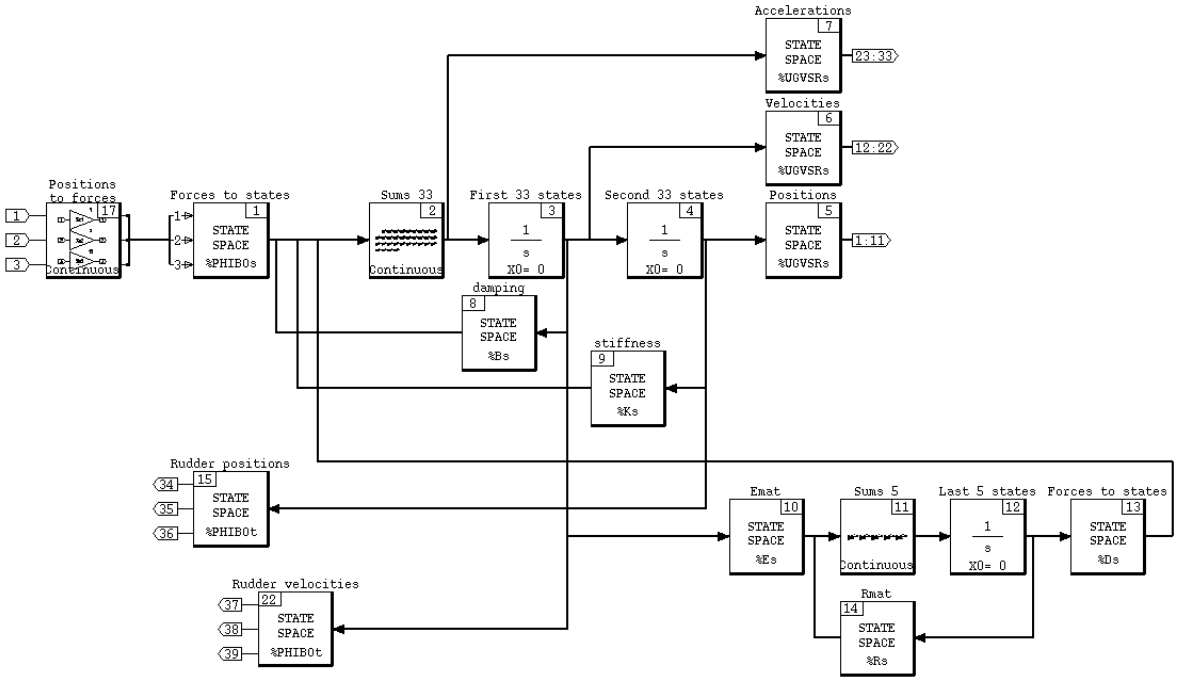


Fig. 44 Generic aircraft model (linear)

The original finite element approximation is made as a “half model” in order to save computation time. This means that only one side of the aircraft is modeled and then the aircraft is assumed to be symmetric. It is possible to model symmetrical or anti-symmetrical movements of the aircraft. The longitudinal model, which will be regarded here, has symmetrical movements of control surfaces on the left and right side of the aircraft.

The natural (no control) aircraft is unstable due to the Phygoide mode for the regarded working point. Phygoide mode oscillations is a phenomena when an aircraft starts to oscillate up and down with a low frequency depending on variations in distribution of energy between speed and altitude. Since the investigated control problem in this case has nothing to do with these, a stable model is chosen through elimination of the Phygoide mode.

Control surfaces can be deflected by affecting the implemented control inputs. The controllable surfaces are:

- Elevator – surface positioned at the tail of the aircraft. It is positioned horizontally and used primarily for changing the pitch of the aircraft. Maximum deflection is +15° -30° (+0.26 rad –0.52 rad) and maximum rate of change is 30°/s (0.52 rad/s).

- Ailerons – (inner and outer) are positioned at the outer end of the wings along the rear edges. The primary use for the surfaces is to control the aircraft in the roll plane. Maximum deflection is ±25° (±0.44 rad) and maximum rate of change is 40°/s (0.70 rad/s). Thanks to fly by wire technique it is possible to deflect the ailerons symmetrically.

The accessible inputs are the three surfaces mentioned above, and in the above order (1 = elevator, 2 = inner aileron, 3 = outer aileron). The outputs are position and velocity of the same surfaces (outputs 34 – 39). Further accessible outputs are a number of points of the body where position, velocity, and acceleration can be measured (outputs 1 – 33, see chapter [Inputs and outputs on the aircraft model]).

8.2 Problem description

In the investigated model the outer ailerons are assumed to be in failure mode. This results in that one control input is lost and the dynamics of the aileron is changed. The dynamics now represent the case when one of two actuators on each aileron is disconnected and the other goes into damping mode. For a functioning actuator the stiffness is modeled with a spring. In the model this is done by “subtracting” the dynamics of the functioning actuators by introducing a linear feedback (see Fig. 45 below). This corresponds to a free aileron and it results in an unstable linear model. Instead a nonlinear actuator force is inserted. The force is a function of the velocity of the aileron and is of quadratic character.

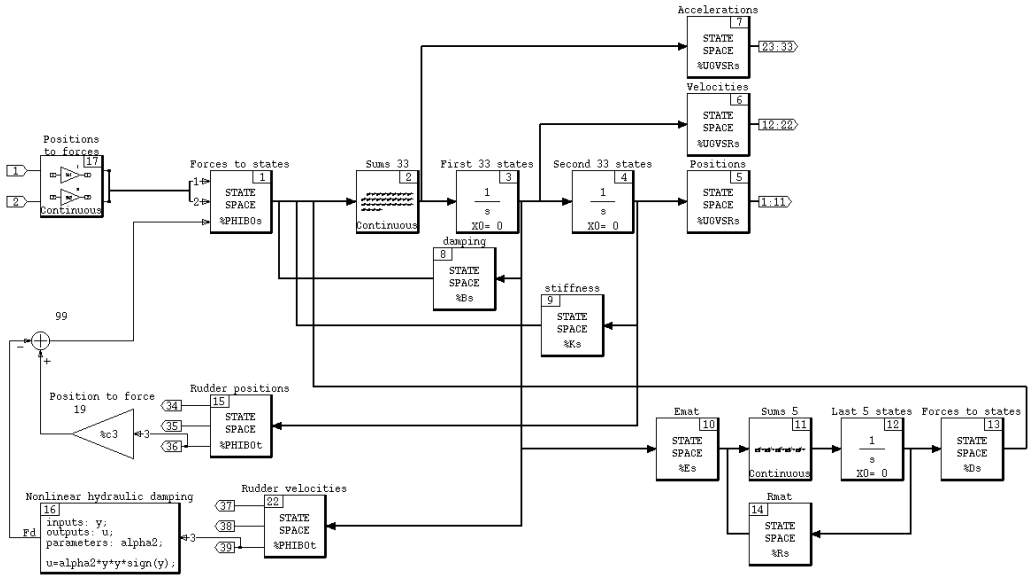


Fig. 45 Investigated model (nonlinear)

A limit cycle appears due to the above-mentioned actuator failure mode. For the chosen working point the cycles have a frequency of about 2.64 Hz and the amplitude is about 13 centimeters at the wing tip and the accelerations there are about of 3.5 g. These cycles of course spread through the aircraft and will be experienced rather uncomfortable. In the cockpit and for those seated directly behind (1st class) the cycles will be experienced with an amplitude of only 2 centimeters but the accelerations have a maximum value of 0.6 g.

Normally both structural problems and passenger comfort is considered when dealing with failure cases. The following investigation is however focused on passenger comfort. Today there is no specific control for this carried out because the probability of this failure mode is very low. The comfort control to improve the comfort in turbulence (CIT) has however some mitigating effect on the limit cycles. They are not cancelled, but the amplitude is decreased a bit. This controller is fed with the accelerations in the body close to the cockpit. The controller moves the phase of the critical gain peaks, giving them a 180° phase shift. A gain is tuned and a low pass filter is added. The controller then feeds the elevator with control signals to minimize these accelerations.

8.3 Analysis through describing function analysis

The examined generic aircraft model has been put on a form (see Fig. 46 below) recognizable from the theory behind describing function analysis (ref. SLOTINE, LI). The added input is solely used for perturbating the system. The reason for not using the standard input for this is to maintain the appearance of the original system where the initiating pulse perturbates the system after the non-linearity.

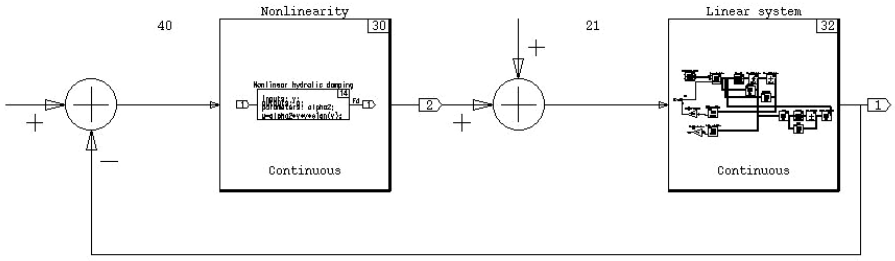


Fig. 46 The examined nonlinear system

The nonlinear block contains the following function:

$u = alpha2 * y^2 * sign(y)$ where $alpha2$ is a constant, y is the output (velocity of outer aileron) of the linear system and u is the input to the same.

The linear system contains the 71-state original system (see below Fig. 47). The constant $alpha2$ has been kept in the nonlinear block despite its linear characteristics.

This is done just to keep the original nonlinear block intact. It has no impact on the final result of the describing function analysis.

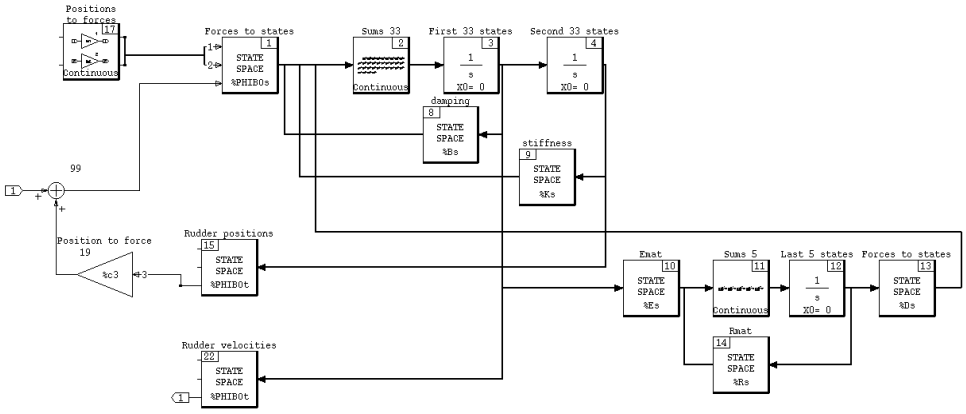


Fig. 47 The linear part of the examined system

The computation of the describing function for the non-linearity is quite basic thanks to its odd and frequency independent nature.

The $N(A)$ part of the describing function $-1/N(A)$ can therefore be written on the form

$$N(A) = \frac{b_1}{A}$$

where

$$b_1 = \frac{1}{\pi}\int_0^{2\pi} u(\phi) * \sin\phi \; d\phi$$

where $u(\phi)$ is the output of the non-linearity when fed with $A * \sin\phi$. In our case

$u(\phi) = alpha2 * A^2 * \sin^2\phi * sign(A * \sin\phi)$.

The b_1 integral finally looks like

$$b_1 = \frac{1}{\pi}\int_0^{2\pi} alpha2 * A^2 * \sin^3\phi * sign(A * \sin\phi)\, d\phi = \frac{4}{\pi} * alpha2 * A^2 * \int_0^{\pi/2} \sin^3\phi \; d\phi = \frac{4}{\pi} * 7000 * \frac{2}{3} * A^2$$

$$b_1 \approx \frac{18667}{\pi} \approx 5942 * A^2$$

The describing function becomes

$$-1/N(A) = \frac{1}{5942 * A}$$

The Nyquist plot of the linear system can be seen in Fig. 48 below.

The first intersection (Re= –0.000775, f=2.64 Hz) between the Nyquist plot and the describing function on the negative real axis represents a stable intersection. The second intersection is unstable (Re=-0.0025, f=3.31 Hz).

The third intersection is also unstable (Re=-0.014, f=2.74 Hz). The theory behind which intersections are stable and which are not will be discussed in chapter [Investigation and optimization of scalar feedback].

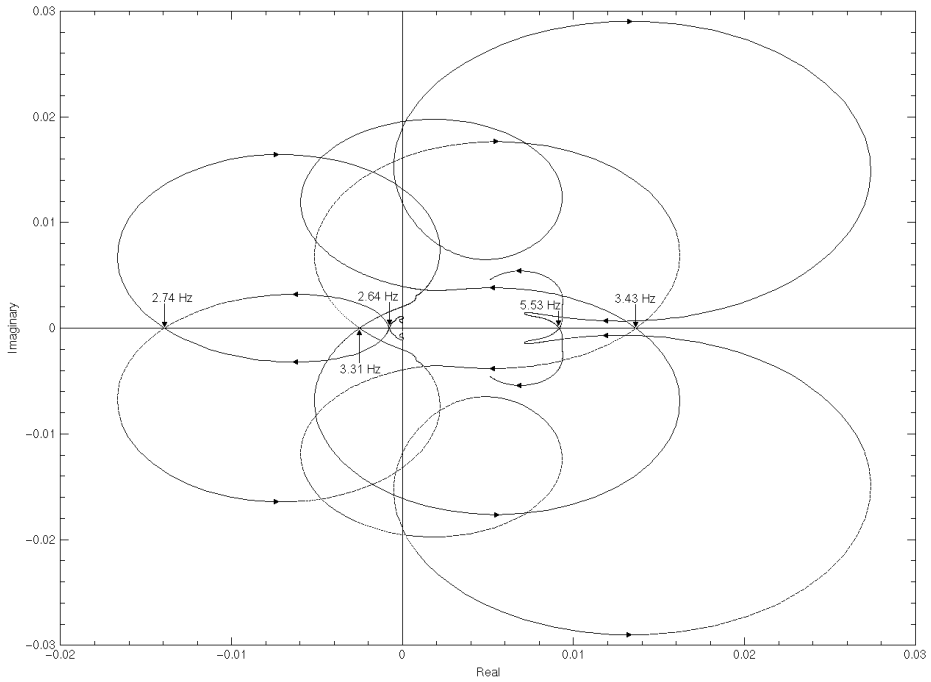


Fig. 48 Nyquist plot of linear system between 0.1 Hz and 6 Hz

Therefore it is natural to expect a limit cycle to appear for the first stable intersection. The frequency is known to be 2.64 Hz and the amplitude can be estimated as:

$$-1/N(A) = f(iw) = -0.000775 \quad \Rightarrow \quad N(A) = 1/0.000775 = 5942 * A \quad \Rightarrow \quad A \approx 0.217$$

When the system is simulated (see Fig. 49 below) the above-achieved values match the result very well.

The output from the linear part has an amplitude of just above 0.2 and the frequency content is 3.45 Hz below 75 seconds (no limit cycle) and 2.64 Hz thereafter.

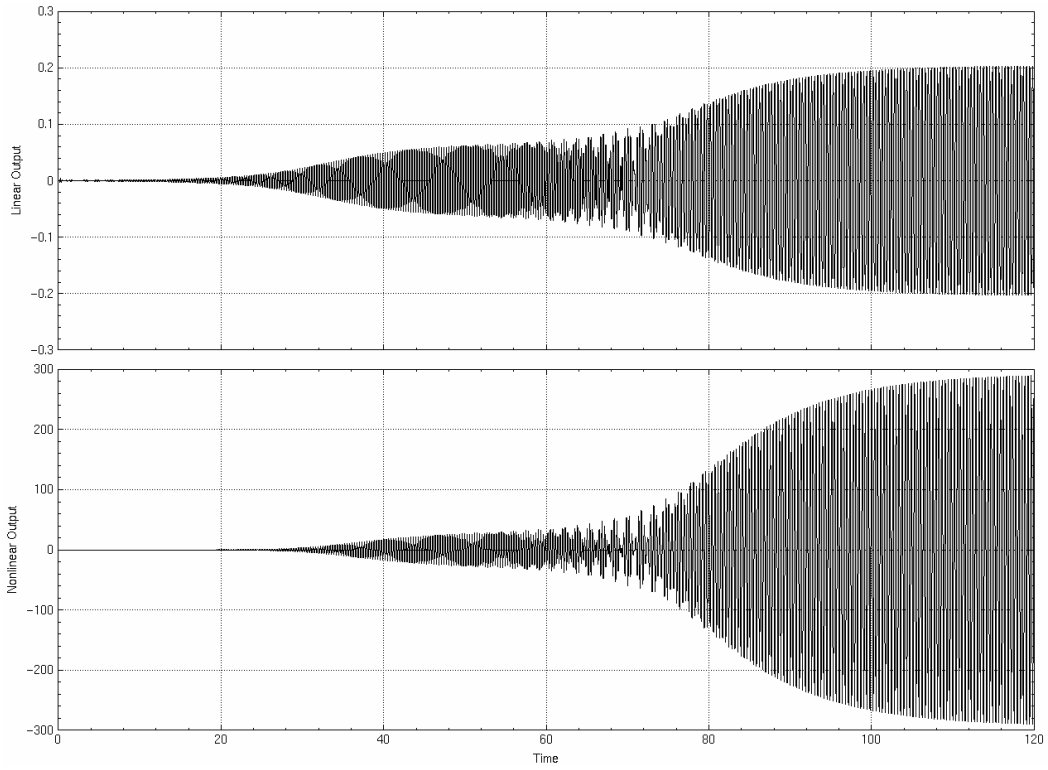


Fig. 49 The output containing the limit cycles

8.4 Initial approach to eliminate the limit cycles

As an initial approach to eliminate the limit cycles the inner aileron is fed with a control signal corresponding to the deflection of the outer broken aileron but with opposite sign. This means that when the outer aileron in failure mode goes down, the inner will go up with the same deflection. The “sum” of these will then result in a zero change of lift on the wing and it will therefore not start to oscillate. It is implemented by just connecting the output representing the position of the outer aileron to the control input to the inner aileron with negative sign and unit gain.

This sort of individual steering of single control surfaces is possible on Airbus aircraft thanks to the fly-by-wire system. On traditional aircraft the rudders are usually cross-coupled with other rudders, which result in that when one surface is moved one or more other surfaces move with it. For example with the movement of an aileron on a wing on a traditional aircraft the aileron on the other opposite wing will move in the counter direction. Sometimes the rudder in the back is also connected to the two.

The result from a realization where this feedback is connected after 140 seconds can be seen in Fig. 50 below:

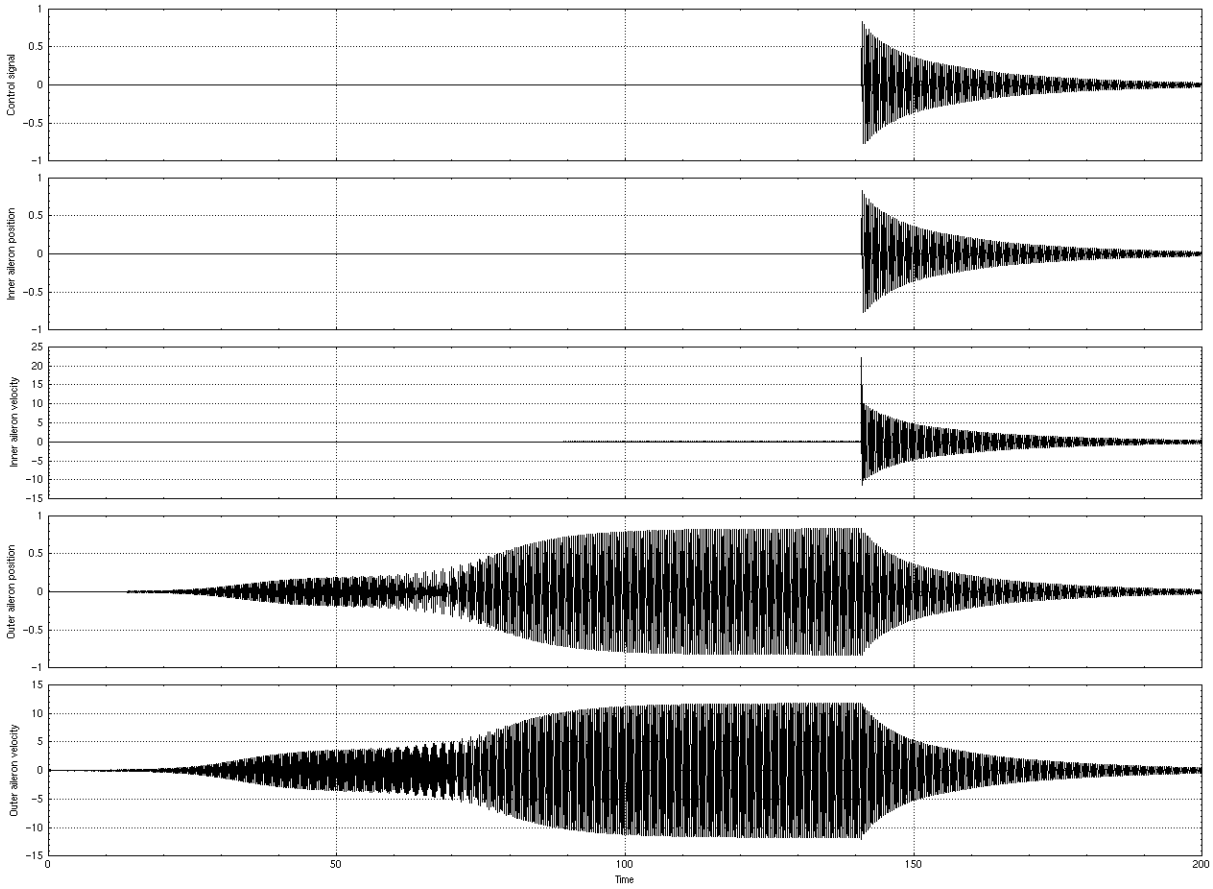


Fig. 50 Plot of output from feedback system (outputs in deg and deg/sec)

The amplitude of the limit cycle on the outer broken aileron is 0.75° and the velocity is 12°/sec. When the feedback is engaged the control signal damps the limit cycles

effectively. Control signals of the same magnitude as the magnitude of the limit cycles are generated. It is also satisfactory to see that all signals are within the limits of deflection and rate of change. The peak in inner aileron velocity comes from the sudden engagement of the feedback.

8.5 Investigation and optimization of scalar feedback

In one approach to remove the limit cycles a scalar feedback has been investigated. A feedback from the outer ailerons velocity to inner aileron position was tried. The Nyquist diagram was investigated and this showed that it was not possible to find a scalar feedback that will keep the system stable.

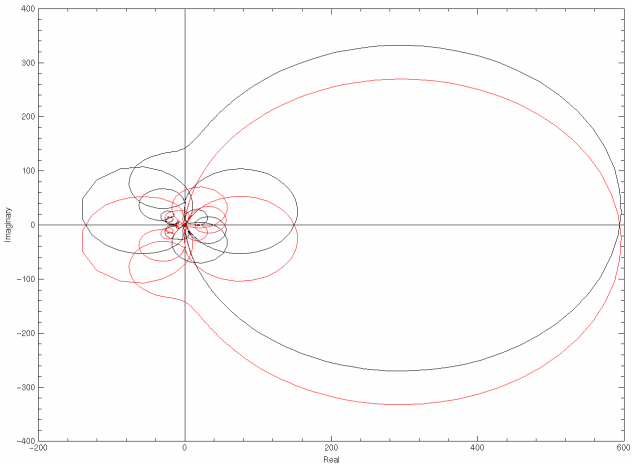


Fig 51a. 0.01Hz – 10000Hz

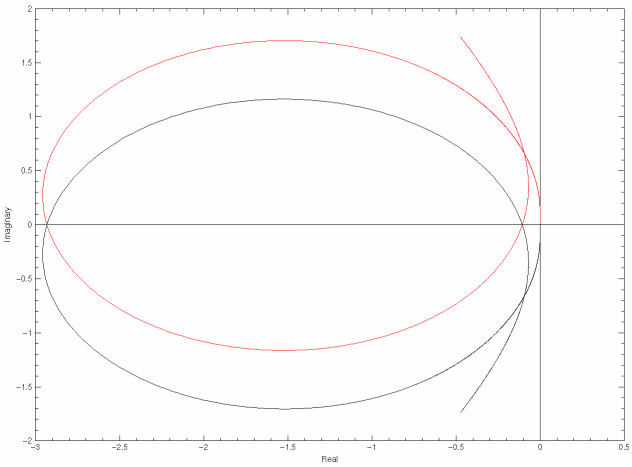


Fig 51b. 0.01Hz – 2Hz

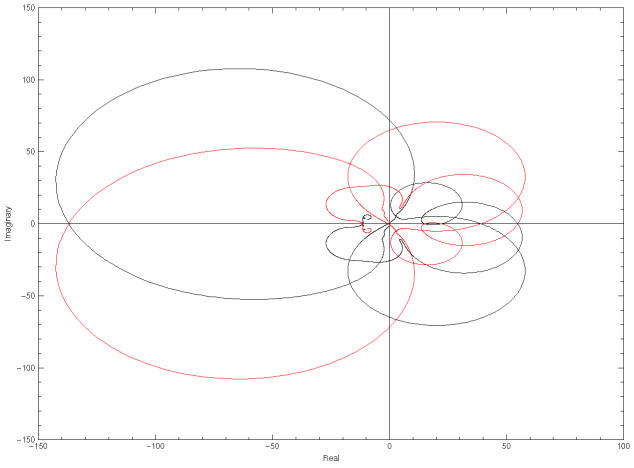


Fig 51c. 2Hz – 10Hz

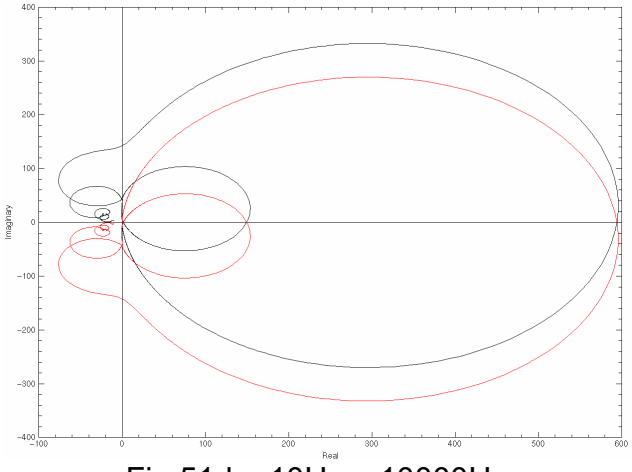


Fig 51d. 10Hz – 10000Hz

Fig. 51 Nyquist plot - outer aileron velocity due to inner aileron

In the Nyquist diagram (see Fig. 51a – Fig. 51d above) it can be shown that there does not exist any stable intersections. Since the Nyquist diagram is quite complex an alternative approach was tried when investigating the intersections. Since the diagram consists of five different intersections on the negative real axis six different feedback gains have to be tried in order to decide intersections are stable and which are not.

By investigating the eigenvalues of the A matrix in the system equations for different feedback gains it can be decided whether the system is stable or not in the investigated region. By investigation of all regions the intersections can be determined if they are stable or not.

Feedback gains from all of the six intervals have been tried and all six give unstable results, from this it is possible to draw the conclusion that it is not possible to make a feedback from outer ailerons velocity to inner aileron position that keeps the system stable.

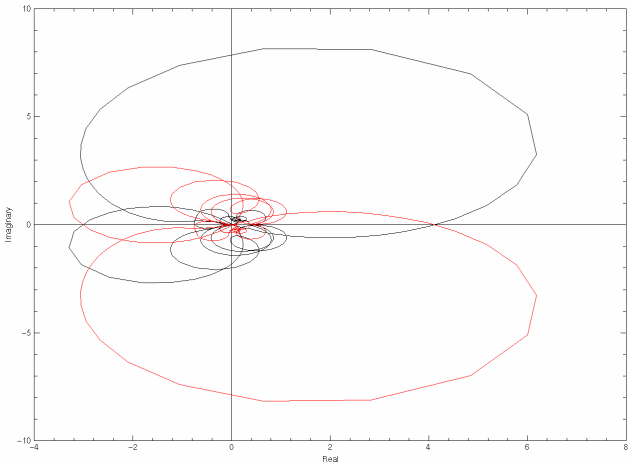


Fig 52a. 0.01Hz – 10000Hz

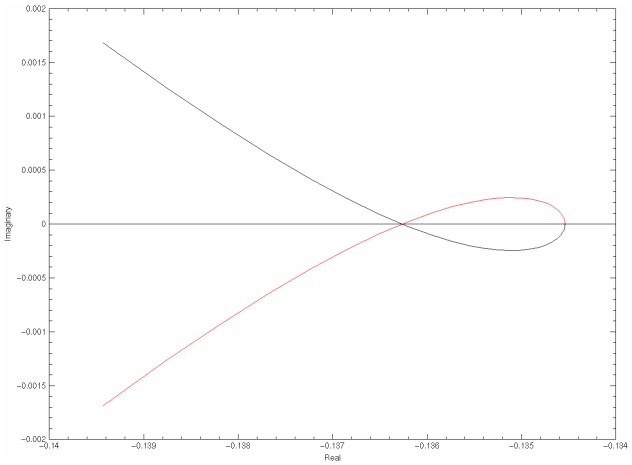


Fig 52b. 1e-5Hz – 0.2Hz

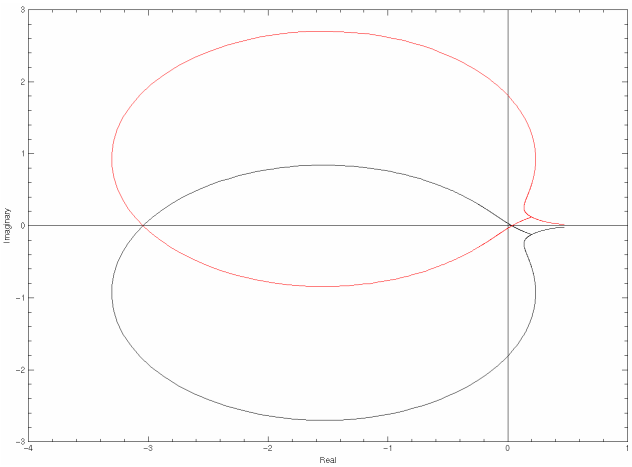


Fig 52c. 3Hz – 4Hz

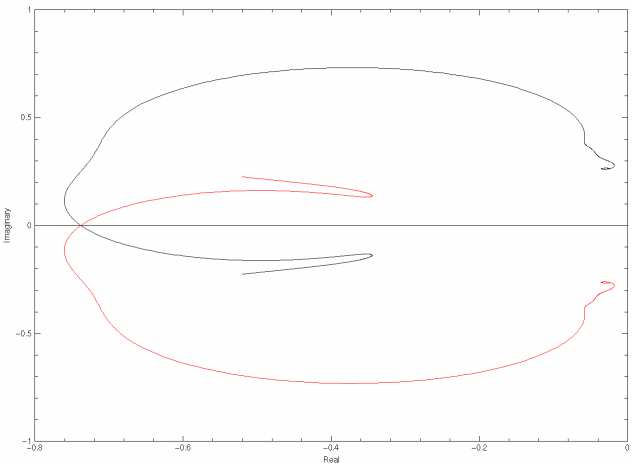


Fig 52d. 4.5Hz – 7Hz

Fig. 52 Nyquist plot - outer aileron deflection due to inner aileron position

While the previous approach did not work a second attempt was made using the output from outer aileron deflection to inner aileron position. The investigation for this feedback was made in the same way as in the previous attempt.

This investigation shows that the system is stable in the interval -0.7619 to -3.1398. These values give that the system is stable for gain between 1/(-0.7619) and 1/(-3.1398), that is between -1.3125 and -0.3185.

So now when it is shown that there is a feedback that makes the system stable it is also necessary to show that this feedback eliminates the limit cycles. The Nyquist diagram for the linear part of the system with four different feedback gains is plotted. For some of the feedback gains there are intersections with the negative real axis, but when these are investigated it is shown that they do not initiate a limit cycle. The feedback gains that do not give any intersections at all with the negative real axis initiate of course no limit cycles.

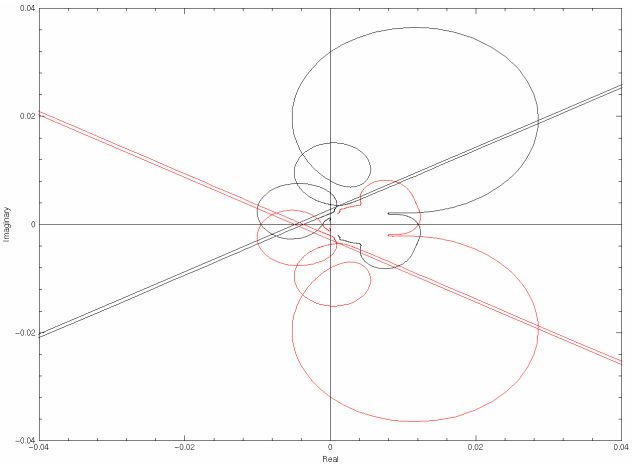


Fig 53a. Feedback –0.3185

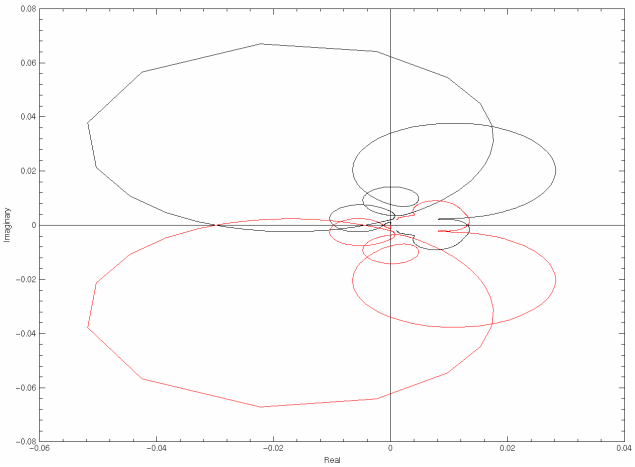


Fig 53b. Feedback –0.4

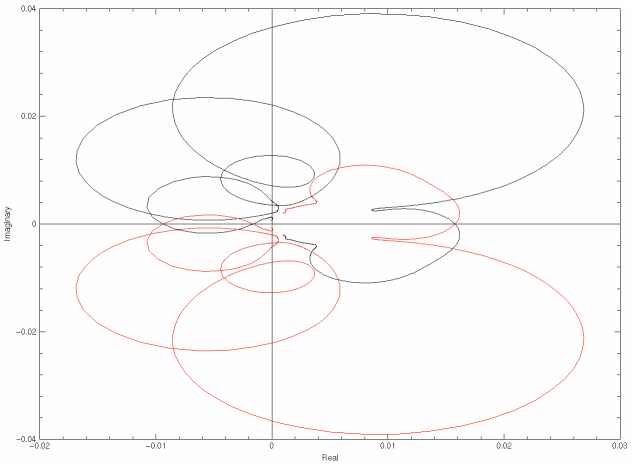


Fig 53c. Feedback –0.55

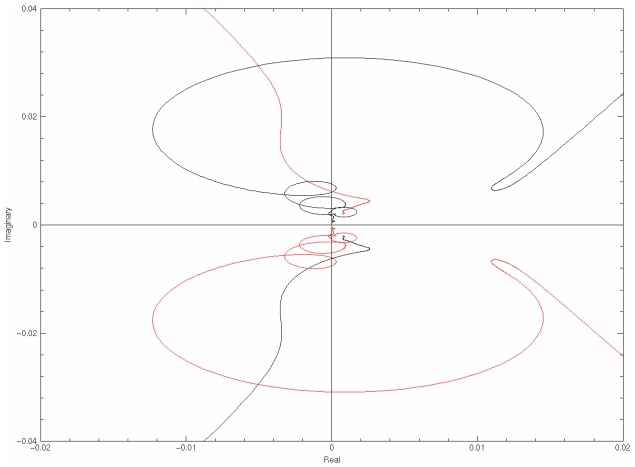


Fig 53d. Feedback –1.3125

Fig. 53 Nyquist diagrams for different feedback gains

Now when it is indicated that the feedback gains in the interval –0.3185 and –1.3125 are all useful to optimize the feedback gain with respect to the time it takes for the oscillations to settle is done. A somewhat different optimization was done, a small Xmath program was made that measures the time for the oscillation to reach a certain lower mean amplitude. This program was run for 15 different gains in the stable interval. When these times were measured a polynomial was fitted to the measure values. The optimal feedback gain could then be seen in the plot, -1.12. Of course the fitted polynomial could be derived and the minimum could be calculated, but since this method only is approximate this is not necessary.

8.6 A nonlinear approach to eliminate the non-linearity

The central idea is to eliminate the nonlinear characteristics in the system. Seeing the non-linearity as a known disturbance could do this. The non-linearity is a function of the velocity of the outer broken aileron. This velocity is measurable and therefore used as input to a feed forward block. The input to the inner aileron is used as input to the system for the feed forward block (see Fig. 54 below).

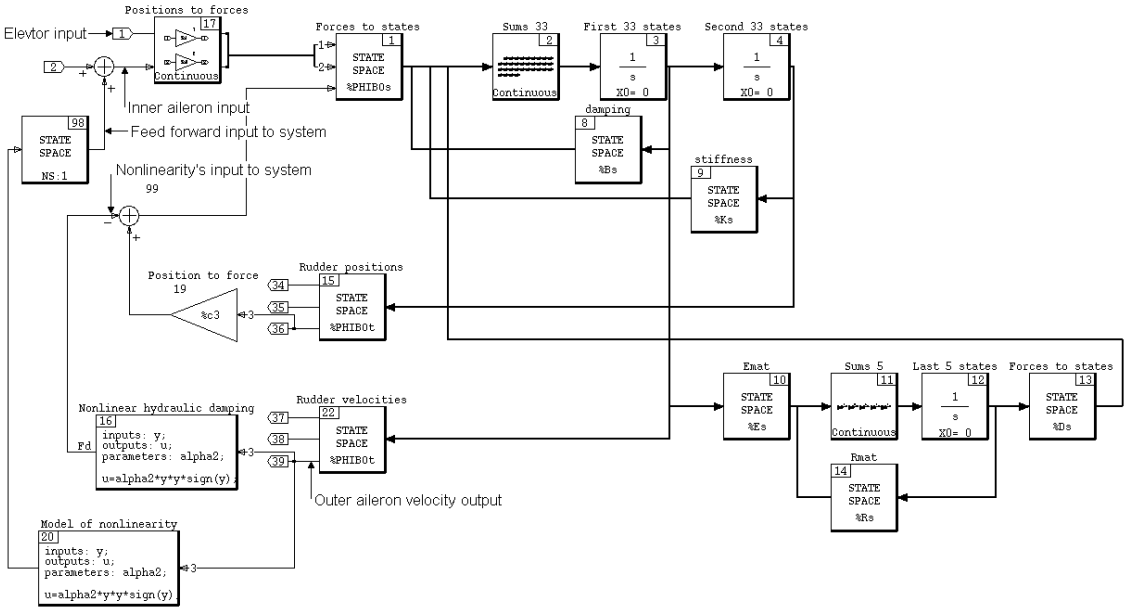


Fig. 54 System with elimination of non-linearity

What is desired is that the dynamics from the inner aileron input to the velocity of the outer aileron gets the same dynamics as the input for the non-linearity to the same output. This should be possible with a feed forward filter inserted as in the Fig. 54 above. The feed forward is given a positive sign on its input to the system to eliminate the negative sign of the non-linearities' system input.

A linear combination of the two remaining inputs (elevator and inner aileron) has been considered to get characteristics that equal the transfer function from the input of the non-linearity to the velocity of the outer aileron better. The weighting could be done with some minimization method and the result typically ends up with about 87-97% weighting on the inner aileron input, depending on the method used.

This feels right since the coupling between inner and outer aileron is larger than the one between elevator and outer aileron. Since the weighting is so dominated by the inner aileron and the improvement of the weighting is barely noticeable, the inner aileron is chosen as the only input for the feed forward block. Expression to be minimized,

$$C_1 \cdot x - C_2 \cdot (1 - x) - C_3$$

The problem can be expressed with transfer functions where y is the output from the linear part of the system (velocity of outer aileron) and u is the inner aileron control signal. The system can then be seen as

$$y = G(s) * u - H(s) * z$$

$$z = f(y) = alpha2 * y^2 * sign(y)$$

where z is the output from the non-linearity, $H(s)$ is the transfer function from the non-linearities' input (input positive) to y and $G(s)$ is the transfer function from u to y.

If u is chosen

$$u = \frac{H(s)}{G(s)} * z = F(s) * z$$

the non-linearity is eliminated. We are interested in eliminating oscillations below 6 Hz and we can therefore approximate the filter $F(s)$ with a reduced filter $\hat{F}(s)$. Since the system is not of minimum phase character there will also be a problem of instability in the filter. This also has to be considered and corrected during design and the filter $\hat{F}(s)$ will be even more an approximation of the ideal filter $F(s)$.

A problem with the whole approach is that the linear part without the non-linearity is unstable. The linear feedback that is introduced with the failure mode moves two stable poles to the right half plane. These poles will initiate oscillations with a frequency of 3.4 Hz.

Several attempts to make the system stable without also stabilizing the limit cycles have been made without success. Therefore an approach with the scalar feedback earlier mentioned is attempted [Initial approach to eliminate the limit cycles]. This feedback also cancelled the limit cycles so instead it will be investigated how much the effect of the non-linearity can be minimized.

The two transfer functions, $F(s)$ and $G(s)$, both have 70 zeroes and 71 poles. To give the filter a lower order a model reduction is performed. The method used is balanced model reduction [JOHANSSON].

The two systems are non-minimum phase and therefore reduced separately, and not when they are put together to the filter $F(s)$ since this is instable. From the Gramian it is possible to see which states that it is possible to eliminate when the system is put on balanced form. $G(s)$ can be reduced to an order of 36 and $F(s)$ to order 29.

The two reduced functions are then put together accordingly to the formula above. This system, 65 states, is then compared to the original system, 70 states (denominators cancelled) (see Fig. 55 below).

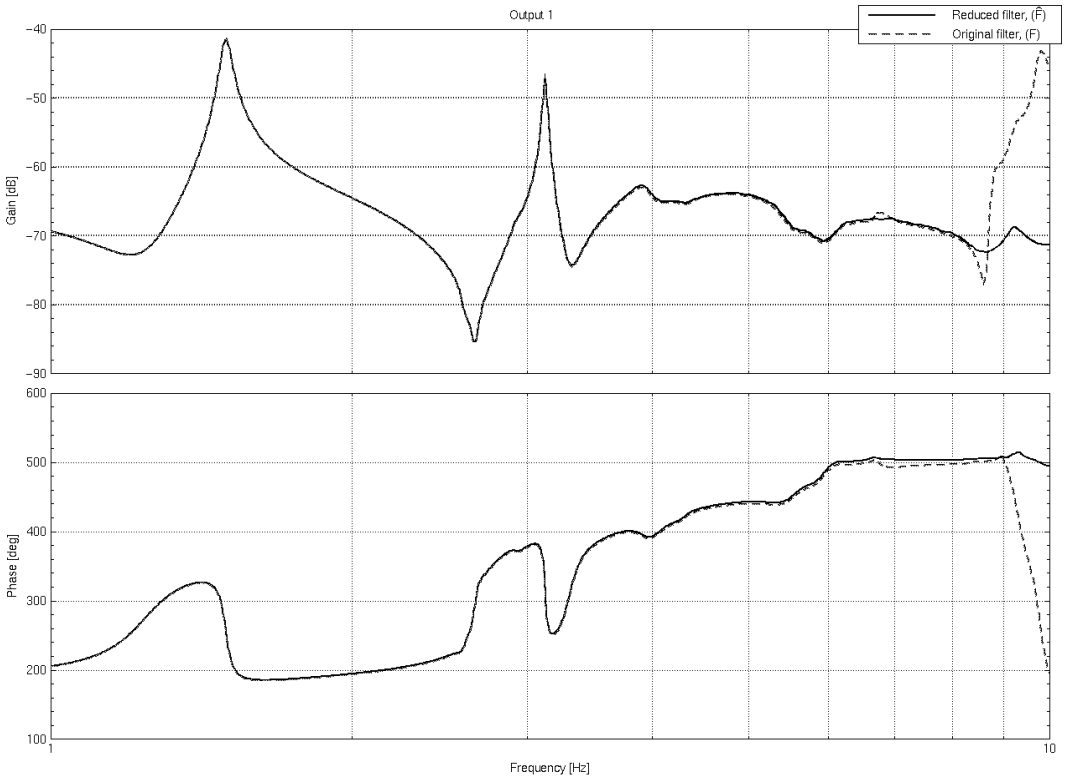


Fig. 55 Comparison between original filter $F(s)$ and reduced filter $\hat{F}(s)$

A pole zero plot is made and in this it is possible to see that many pole zero cancellation can be made. With a pole zero cancellation, with tolerance 1e-5, the system is reduced to order 42. This cancellation has marginal effect on the filter.

There is a peek at and around 10 Hz on the gain plot of the original filter that shall not be there. It comes from numerical errors when converting the system from state space form to transfer function form. It has however been shown that the Bode plot is correct up to about 8.5 Hz and the problem has not been further analyzed.

The problem with the instability has to be solved. But first a test to verify the results so far is carried out. By inserting u into y the following is attained

$$y = G(s) * \hat{F}(s) * z - H(s) * z$$

This shall then equal zero in the interval up to 5 Hz where we set the cutoff frequency for the filter. For this to happen the terms $G(s) * \hat{F}(s)$ and $H(s)$ have to equal each other. The result is quite satisfactory when the two bode plots are compared (see Fig. 56 below).

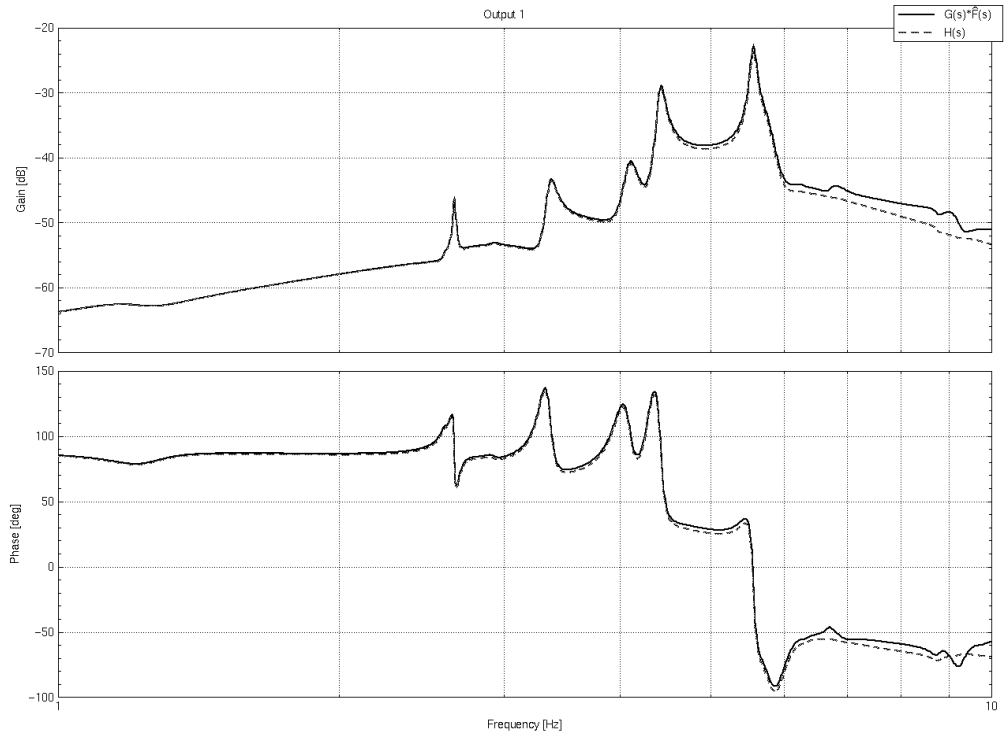


Fig. 56 Comparison between $G(s) * \hat{F}(s)$ and $H(s)$ in a Bode plot

To make the system stable the poles in the right half plane are mirrored (Lemma 4.1, [ÅSTRÖM]). There are four poles that have to be mirrored. The result of the mirroring is that the gain is maintained but the phase is changed (see Fig. 57 below).

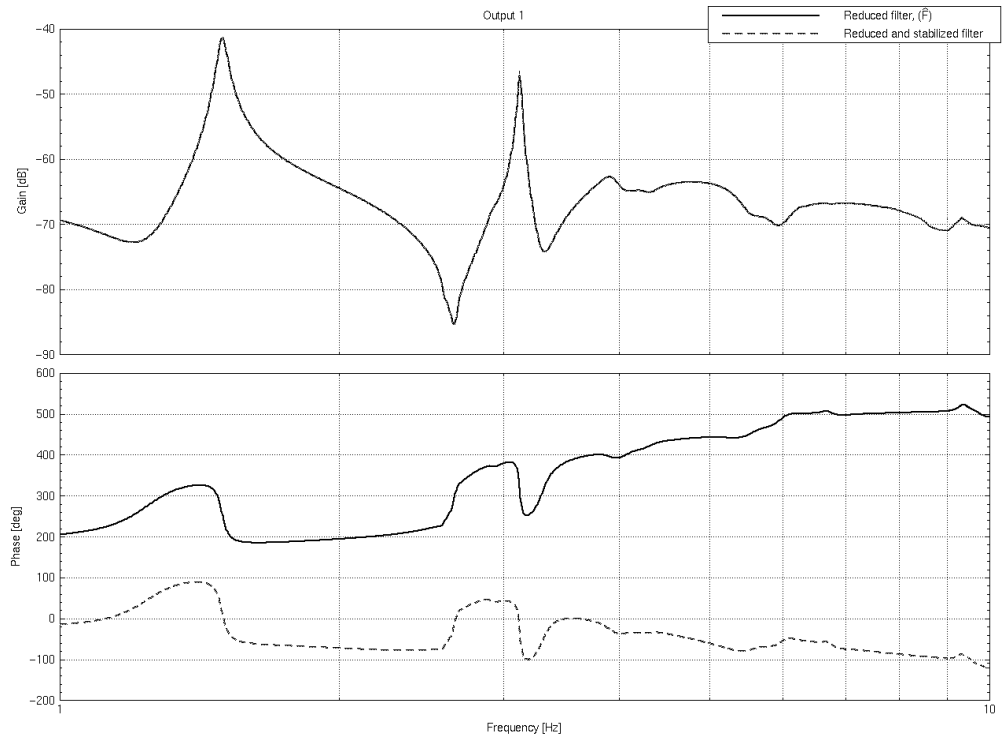


Fig. 57 Comparison between the reduced filter before and after it is stabilized

The phase is shifted 180° for low frequencies when the mirroring is performed. Therefore the sign on the input for the filter is changed.

When the system then is simulated with the filter connected the result looks quite satisfactory (see Fig. 58 below). However, when the filter solution is compared to the scalar feedback alone, it can be seen that this solution (see Fig. 58 below) gives a better result.

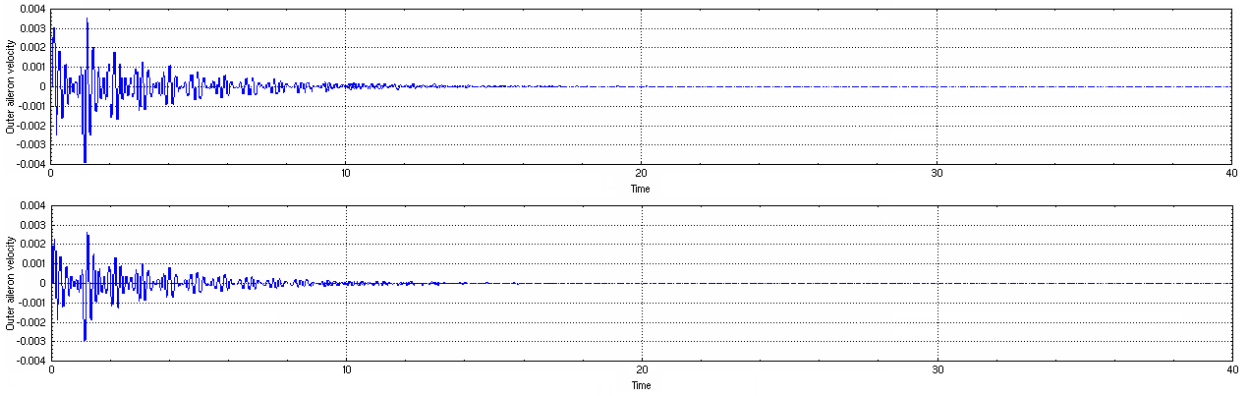


Fig. 58 Comparison between filter solution (top) and scalar feedback (bottom)

A lot of other filters were also tried, generated by different less sophisticated methods. For example were all poles and zeros of higher frequency than 10 Hz removed manually.

The results from these filters were often very poor, but results similar to the results from the filter generated by balanced realization were also received.

The problem was that the trajectory escaped to infinity in finite time. This problem can however be explained by the phrase "*finite escape time*".

The finite escape time is explained by the term,

$$y = G(s) * \hat{F}(s) * z - H(s) * z = \left(G(s) * \frac{H(s)}{\hat{G}(s)} - H(s) \right) * z = \left(\frac{G(s)}{\hat{G}(s)} - 1 \right) * H(s) * z$$

This never will equal zero when the filter $F(s)$ is approximated with a filter $\hat{F}(s)$. The error will have characteristics looking like

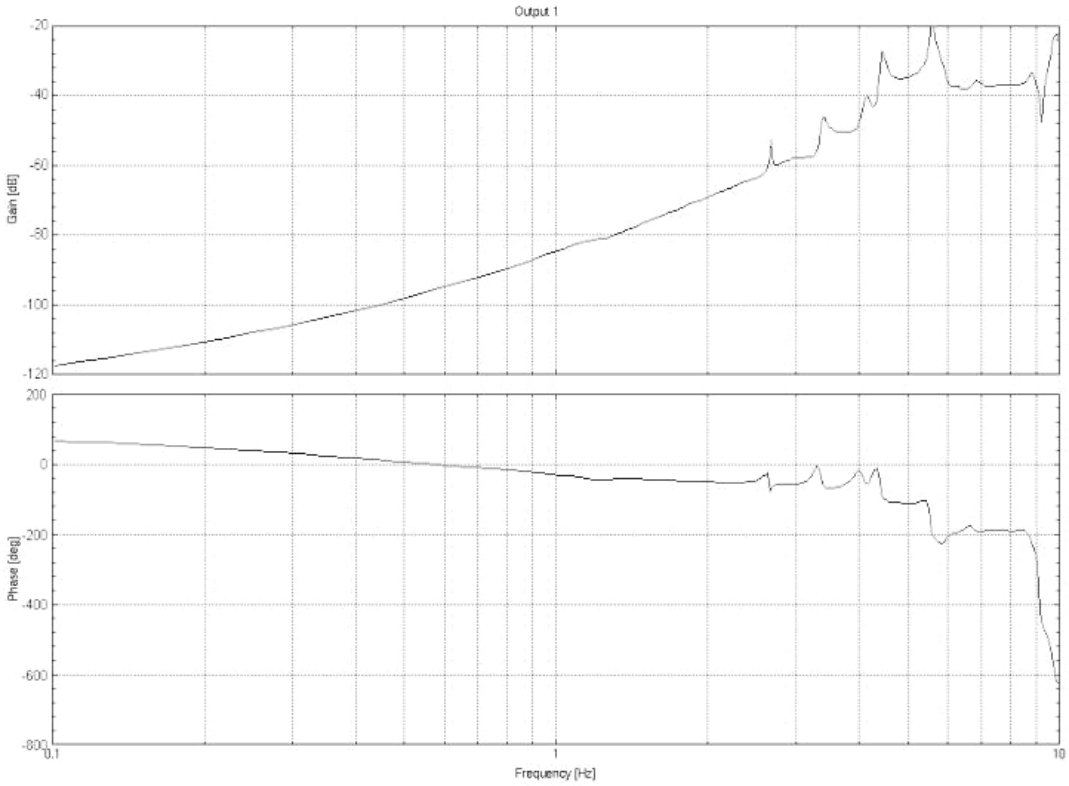


Fig. 59 Bode plot of error

8.7 Further control approaches

Additional approaches that have been considered are control through input-output and input-state linearization [SLOTINE, LI]. Due to the complex characteristics of the investigated system with the non-linearity entering all states directly this approach would result in too complex controllers.

Both mentioned control methods build on the fact that the non-linearity is accessible in the control signal. This will in this case result in that the 71 state equations will have to be derived up to 71 times and then a control signal based on this. The controllers would not only be sensitive to model variations, they would also become too big and complex. Both crew and passengers would have second thoughts flying with an airplane with such control implementations.

9 Summary

Both main methods used for linearising systems give good results. They do however show different difficulty in application on certain systems.

Linearization by using the "`lin`" command is generally a good approach for small systems. The "`trim`" command solves the problem of finding an equilibrium point but the problem of finding the perturbation vectors still exists. This is where the limitation of the system's size enters; the perturbation vectors have to be found through trial and error.

Linearization by using subspace identification requires some work with preparing data sets and validation of the models, but after a few identifications it is pretty straightforward. The attained model is not usually a theoretically correct linearization of the original system, but it usually models its dynamics well. An advantage with the identification approach is that model size is not usually a problem, and it is in most cases possible to get a reduced order model of the original system. This is sometimes requested since it makes computer simulations faster in execution time.

The attempt to solve the control problem resulted in something unexpected. The first simple approach with scalar feedback ended up being the most successful. All other attempts with more advanced filters and linearisations of the non-linearity resulted in either less damped systems with longer settling time or systems with finite escape time.

Gratifying to hear is that the approach with scalar feedback gives so good results, despite its simplicity, so that it will be further investigated at EADS and maybe implemented in the control system of future aircraft models.

10 Table of references

Reference	Full reference information
[ÅSTRÖM]	Åström, Karl Johan: Reglerteori. Stockholm: Almqvist & Wiksell Förlag AB, 1968
[JOHANSSON]	Johansson, Rolf: System Modeling and Identification. Englewood Cliffs: Prentice Hall, New Jersey, 1993.
[JORDAN]	Jordan, Charles: Calculus of Finite Differences. New York, NY: Chelsea Publishing Company, 1965. Library of congress Catalog Card No. 58-27786
[MATRIXX_ISIM]	Integrated Systems, Inc. (ed.): MATRIXx product family, Xmath Interactive System Identification Module, Part number 000-0027-002, Santa Clara, CA: Integrated Systems, Inc., 1996.
[MATRIXX_SBUG]	Integrated Systems, Inc. (ed.): MATRIXx product family, SystemBuild User's Guide, Part number 000-0051-005, Santa Clara, CA: Integrated Systems, Inc., 1996, chapter 5.6.
[MGA]	MGA Software (ed.): ACSL Reference Manual, Edition 11.1. Concord, MA: MGA Software, 1995, p. A62-A69.
[SCHULER]	Schuler, Jörg: Flugregelung und aktive Schwingungsdämpfung für flexible Großraumflugzeuge. Dűsseldorf: VDI Verlag GmbH, 1998 ISBN 3-18-368808-5
[SLOTINE, LI]	Slotine, Jean-Jacques E.; Li, Weiping: Applied Nonlinear Control. New Jersey 007632: Prentice Hall, 1991 ISBN 0-13-040890-5

11 Appendix

11.1 Inputs and outputs on the aircraft model

Inputs to control surfaces

	Position [rad]
Elevator deflection	1
Inner aileron deflection	2
Outer aileron deflection	3

Outputs from points on fixed body

	Position [m]	Velocity [m/s]	Acceleration [m/s^2]
z front fuselage	1	12	23
x center of gravity	2	13	24
z center of gravity	3	14	25
theta center of gravity	4	15	26
z rear fuselage	5	16	27
x wing tip	6	17	28
z wing tip	7	18	29
y inner engine	8	19	30
z inner engine	9	20	31
y outer engine	10	21	32
z outer engine	11	22	33

Outputs from control surfaces

	Position [rad]	Velocity [rad/s]
Elevator deflection	34	37
Inner aileron deflection	35	38
Outer aileron deflection	36	39

11.2 FRA function implemented in MATRIXx

---"nfr" function----------

```
Function [SortedFrequency, PhaseMatrix, PhaseMatrixDEG, GainMatrix, ...
GainMatrixDB, PhaseAvMatrix, GainAvMatrix]= nfr(Model, Freq, Magnitudes, ...
{makesteps, errcrit, step, ialg, offset})

//----------gives the frequency responce of a model -------
//Inputs: Model name (Model), Frequency vector (Freq), ...
//    Magnitude vector (Magnitudes)
//Keywords: Generate a frequency vector in the assigned interval, ...
//    (makesteps=1) default (=0), Error criteria (errcrit) sets the ...
//    allowed loop error (default=0.1), Step size (step) sets the ...
//    frequency increment (default=1.2), Integration algorithm (ialg) is ...
//    default "VKM" but "GEAR" is also good
//Outputs: plot and matrices
//Comment: Frequency vector can either be a vector with frequencies or ...
//    an intervall if the makesteps keyword is used
//

clock({cpu});                       //resets timer
DEFAULT step=1.2;                   //sets keyword defaults
DEFAULT errcrit=0.1;
DEFAULT ialg="VKM";
DEFAULT makesteps=0;
DEFAULT offset=0;

if makesteps==1 then                //if makesteps keyword is assigned 1, ...
  ind=1;                            //   generate frequency vector
  Frequencies(ind)=Freq(1);
  while Frequencies(ind)<Freq(2) do
    ind=ind+1;
    Frequencies(ind)=step*Frequencies(ind-1);
    if Frequencies(ind)>Freq(2) then
      Frequencies(ind) = Freq(2);
    endif;
  endwhile
else
  Frequencies=Freq;
endif

[SortedFrequency]=sort(Frequencies');     //sort frequency vector
SortedFrequency=SortedFrequency';

[SortedAmplitude]=sort(Magnitudes');      //sort amplitude vector
SortedAmplitude=SortedAmplitude';

[PhaseMatrix, GainMatrix, OffsetMatrix, PhaseAvMatrix, GainAvMatrix]= ...
      Generate_Matrices(Model, SortedFrequency, SortedAmplitude, errcrit, ...
ialg);                        //Generate the phase and gain matrixes

PhaseMatrixDEG=PhaseMatrix*180/pi; //Transform phase shift values into
degrees

GainMatrixDB=20*log10(GainMatrix); //Transform gain values into Decibel

[]=present(Model, SortedFrequency, ...    //present the result in neat diagram
      SortedAmplitude, GainMatrix, ...
      GainMatrixDB, PhaseMatrixDEG, OffsetMatrix, ...
      PhaseAvMatrix, GainAvMatrix, offset);

CPUtime=clock({cpu})?               //show elapsed CPU time

EndFunction
```

```
---"generate_matrices" function-------------------------------------

Function [PhaseMatrix, GainMatrix, OffsetMatrix, PhaseAvMatrix, ...
GainAvMatrix] = generate_matrices(Model, Freq, Mag, ErrCrit, Int);

//----------Gives matrixes for phase shift, gain, and offset for input ...
//    frequency and magnitudes--
//Inputs: Model name (Model), Frequency vector (Freq), Magnitude vector ...
//    (Mag), Loop Criteria (ErrCrit), Integration algorithm (Int)
//Outputs: Matrix containing phase shifts (PhaseMatrix), Matrix ...
//    containing gains (GainMatrix), Matrix containing offsets ...
//    (OffsetMatrix), Matrixes showing if values are averaged or not ...
//    (PhaseAv, GainAv)
//

For MagIndex=1:length(Mag)          //loop over magnitudes
  error("Setting Amplitude to ...   //gives feed back to user that next ...
"+ string(Mag(MagIndex)))     //    magnitude is evaluated

  OldPhase=0;                       //initializes phase shift
  For FreqIndex=1:length(Freq)      //loop over frequencies
    error("Running Frequency "+ string(Freq(FreqIndex))+ " Hz");
//gives feed back to user that next frequency is evaluated
    FrequencyInvestigated=Freq(FreqIndex)?
//showing on the screen which frequency is calculated
    [Phase, Gain, Offset, PhaseAv, GainAv] = determine_pg(Model, ...
Freq(FreqIndex), Mag(MagIndex), ErrCrit, Int); //Retrieve phase and gain
    If Phase>(OldPhase+1.74533) then
//attempt to avoid phase jumps on 360 dergrees
      Phase=Phase-2*pi;       //if phase shift makes a big possitive jump ...
    endIf;                    //   360 degrees are subtracted to make it cont.

    OldPhase=Phase;           //save old phase for next comparison

    PhaseMatrix(FreqIndex,MagIndex)=Phase;     //store data in matrix
    GainMatrix(FreqIndex,MagIndex)=Gain;       //store data in matrix
    OffsetMatrix(FreqIndex,MagIndex)=Offset;   //store data in matrix
    PhaseAvMatrix(FreqIndex,MagIndex)=PhaseAv; //stores a 1 if phase was ...
//    averaged
    GainAvMatrix(FreqIndex,MagIndex)=GainAv;   //stores a 1 if gain was ...
//    averaged
  endFor
endFor

endFunction

---"determine_pg" function-------------------------------------

Function [Phase, Gain, Offset, PhaseAv, GainAv] = determine_pg(Model, ...
Freq, Mag, ErrCrit, Int);

//----------Gives phase shift, gain, and offset for input frequency and ...
//    magnitude--------
//Inputs: Model name (Model), Frequency (Freq), Magnification (Mag), ...
//    Loop Criteria (ErrCrit), Integration algorittm (Int)
//Outputs: Phase shift (Phase), Gain (Gain), Offset (Offset), Matrixes ...
//    showing if values are averaged or not (PhaseAv, GainAv)
//



Period=1;               //First period for present frequency
MaxPeriods=20;          //Maximum number of periods for each frequency ...
//    before stoped and averaged
Average=11;             //MaxPeriods-Average+1 = nr of periods used when ...
//averaging when no settlement
```

```
PhaseAv=0;
GainAv=0;
DPhase=1;               //Initialize difference in phase and ...
DGain=1;                //    gain to guarantee at least two cycles
[St,Ct, Offset]=response(Model, Freq, 30, Mag, Period, Int);
//Generate the first values for the integrals ...
//(30 is resolution in timevector)
Phase=atan2(Ct,St);                 //Generate the first values for phase ...
Gain=2*Freq*sqrt(St**2+Ct**2)/Mag; //    and gain from the integrals
While ((DPhase>ErrCrit | DGain>ErrCrit*0.1) & ...    //Until phase & gain ...
      Period<MaxPeriods) do                          //    have settled loop
  Period=Period+1;   //Increment to get the next period for present frequency
  OldPhase=Phase;       //Save old values of phase
  OldGain=Gain;         //Save old values of gain
  [St,Ct, Offset]=response(Model, Freq, 30, Mag, Period, Int);
//Generate the next values for the integrals
  If Period >= Average then         //If more than "Average" number of ...
//periods start saving values
    StMV(Period)=St;    //St Memory Vector for averaging St when no setteling
    CtMV(Period)=Ct;    //Ct Memory Vector for averaging Ct when no setteling
    OffsetMV(Period)=Offset         //Offset Memory Vector for averaging ...
//    Offset when no setteling
  endIf
  Phase=atan2(Ct,St);               //Generate the next values for phase & ...
  Gain=2*Freq/Mag*sqrt(St**2+Ct**2);      //   gain from the integrals
  if Period == MaxPeriods then
    error("Max runs reached: averaging");//Giving feed back from analyze ...
//    (could not find a better way)
    StAverage=mean(StMV(Average:MaxPeriods));  //Average of St
    CtAverage=mean(CtMV(Average:MaxPeriods));  //Average of Ct
    Offset=mean(OffsetMV(Average:MaxPeriods)); //Average of Offset
    if (DPhase>ErrCrit) then
      Phase=atan2(CtAverage,StAverage);  //Calculating average of ...
//    phase if phase is not settled
      PhaseAv=1;                         //Mark in "PhaseAv" that value is...
//    averaged for later plotting
    endif;
    if (DGain>ErrCrit) then
      Gain=2*Freq/Mag*sqrt(StAverage**2+CtAverage**2);     //Calculating ...
//    average of gain if gain is not settled
      GainAv=1;              //Mark in "GainAv" that value is averaged ...
//    for later plotting
    endif;
  endif
  DPhase=abs(Phase-OldPhase); //Determine the difference in phase and gain ...
  DGain=abs(Gain-OldGain);    //   between the last two periods
endWhile

endFunction

---"response" function-----------------------------------

Function [St,Ct, Offset] = Response(ModelName, Freq, Res, Mag, Run, Int);

//----------Produces output from one period input----------
//Inputs: Model name (ModelName), Frequency (Freq), Resolution (Res), ...
//    Magnification (Mag), Nbr. of periods run (Run), Integration method (Int)
//Outputs: Integral of sine times the Output (St) and cosine times the ...
//    Output (Ct), Offset in output (Offset)
//

W=2*pi*Freq;                                //Hz to rad/sec
PeriodLength=1/Freq;                        //length of period in sec
StepSize=PeriodLength/Res;                  //stepsize in time vector
InputTime=(((Run-1)*PeriodLength): ...      //generate input time vector
      StepSize:(Run*PeriodLength))';
InputTime(1)=InputTime(1)+1e-10;            //guarantee resume time is ...
```

```
//    larger than previous end time
InputSignal=(Mag*sin(W*InputTime));      //generate input sine signal

If Run == 1 then                         //If first period of this Frequency
  [WasteTime, SystemOutput]=sim(ModelName, ... //generate output from model,
      InputTime, InputSignal, {ialg = Int}); //WasteTime is never used
Else                                           //Else
[WasteTime, SystemOutput]=sim(ModelName, ...   //generate output from ...
  InputTime, InputSignal, {Resume, ialg = Int});//   resumed model
endIf

Outputsin=sin(W*InputTime).*SystemOutput;//multiplication of output ...
//    with sine
Outputcos=cos(W*InputTime).*SystemOutput;      //multiplication of output ...
//    with cosine
Outpdm=pdm([Outputsin, Outputcos], ...      //creation of PDM for integration
      InputTime, {domainName="Time", ...
      ColumnNames=["Outputsine";"Outputcosine"]});

IntOutputs = integrate(Outpdm);                //Integration of outputs ...
//    (in-phase, quadrature)
Dim=size(IntOutputs);                          //for finding the last ...
//    integration element
St=IntOutputs(1,1,Dim(3));                     //the value of the ...
//    integrated sine channel
Ct=IntOutputs(1,2,Dim(3));                     //the value of the ...
//   integrated cosine channel
Offset=mean(SystemOutput)                      //Compute offset

endFunctionFunction [St,Ct, Offset] = Response(ModelName, Freq, Res, Mag, ...
Run, Int);

---"present" function-----------------------------------

Function [Waste] = present(Model, SortedFrequency, SortedMagnitude, ...
GainMatrix, GainMatrixDB, PhaseMatrixDEG, OffsetMatrix, ...
PhaseAvMatrix, GainAvMatrix, offset)

//----------Plots the result in bode form-------------

//Inputs: Model name (Model), Frequency vector (SortedFrequency), ...
//    Magnitude vector (SortedMagnitude), Matrix containing gains ...
//    (GainMatrix), Matrix containing gains in Db (GainMatrixDB), Matrix ...
//    containing phase shifts in degrees (PhaseMatrixDEG), Matrix ...
//    containing offsets (OffsetMatrix), Matrixes showing if values are ...
//    averaged or not (PhaseAvMatrix, GainAvMatrix), Boolean indicating if ...
//    offset should be drawn (offset)
//Outputs: Waste, which is not used for anything
//

display "----- Final result summery :"    //Show results in Xmath window
display " "
display "----- Analyzed superblock = ", Model
display " "
display "----- Column domain of InputOutput-MATRICES is the Frequency !"
display "----- Row domain of InputOutput-MATRICES is the Amplitude !"
InputFrequencies = SortedFrequency?
InputMagnitudes = SortedMagnitude?
GainMatrix?
GainMatrixDB?
PhaseMatrixDEG?

GainDB_PlotMinimum_Default =  -10; //Set graph axis defaults
GainDB_PlotMaximum_Default =   10;
Phase_PlotMinimum_Default  =  -10;
Phase_PlotMaximum_Default  =   10;
```

```
If (min(GainMatrixDB) > -Inf)       //Use default or other axel numbering
  GainDB_PlotMinimum = min(GainDB_PlotMinimum_Default, ...
  10*round(min(GainMatrixDB)/10)-10);
else
  GainDB_PlotMinimum = GainDB_PlotMinimum_Default;
endIf;
If (max(GainMatrixDB) < Inf)
  GainDB_PlotMaximum = max(GainDB_PlotMaximum_Default, ...
  10*round(max(GainMatrixDB)/10)+10);
else
  GainDB_PlotMaximum = GainDB_PlotMaximum_Default;
endIf;
GainDB_PlotIncrement = 10


Phase_PlotMinimum    = min(Phase_PlotMinimum_Default, ...
  10*round(min(PhaseMatrixDEG)/10)-10);
Phase_PlotMaximum    = max(Phase_PlotMaximum_Default, ...
  10*round(max(PhaseMatrixDEG)/10)+10);
Phase_PlotIncrement  = 20;


k=0;                     //Generate plot data for averaged gain values
SizeMatrix=size(GainMatrixDB);
For i=1:SizeMatrix(2)
  For j=1:SizeMatrix(1)
    if GainAvMatrix(j,i) == 1 then
      k=k+1;
      PlotGainAvMatrix(k)=GainMatrixDB(j,i);
      SortedFrequencyGainAv(k)=SortedFrequency(j);
    endif
  endFor
endFor

l=0;                     //Generate plot data for averaged phase shift values
SizeMatrix=size(GainMatrixDB);
For i=1:SizeMatrix(2)
  For j=1:SizeMatrix(1)
    if PhaseAvMatrix(j,i) == 1 then
      l=l+1;
      PlotPhaseAvMatrix(l)=PhaseMatrixDEG(j,i);
      SortedFrequencyPhaseAv(l)=SortedFrequency(j);
    endif
  endFor
endFor

For amp=1:length(SortedMagnitude)         //Create legend
  LegendText(amp) = "Amp=" + string(SortedMagnitude(amp))
endFor

if offset==1 then
  NFR_Plot = plot({rows=3, columns=1});  //Generate 3 or 2 by 1 plot window
else
  NFR_Plot = plot({rows=2, columns=1});
endif;

NFR_GainDB = ...                          //Generate gain plot
      plot(SortedFrequency, GainMatrixDB, ...
      {row=1, column=1, ...
      x_log, yzero_line, ...
      marker_style=[4,4,4,4,4], marker_color=[5,11,1,3,6], marker=1, ...
      line_style  =[1,2,7,4,5], line_color  =[5,11,1,3,6], ...
      xmin=SortedFrequency(1), ...
      xmax=SortedFrequency(length(SortedFrequency)), ...
      xinc=10, xlab="Frequency f [Hz]", ...

      ymin=GainDB_PlotMinimum, ...
```

```
      ymax=GainDB_PlotMaximum, ...
      yinc=GainDB_PlotIncrement, ...
      ylab="Gain [db]", ...
      title="Response of: " + Model, ...
      legend=LegendText, date, time, keep=NFR_Plot});

If k>0 then                                 //Mark the averaged dots, if any
NFR_GainDBAv = ...
      plot(SortedFrequencyGainAv, PlotGainAvMatrix, ...
      {row=1, column=1, line_style  =" ", legend="Averaged", ...
      marker_style=[8,8,8,8,8], marker_color=[2,2,2,2,2], marker=1,
keep=NFR_Plot});
endIf

NFR_Phase = ...                             //Generate phase shift plot
      plot(SortedFrequency, PhaseMatrixDEG, ...
      {row=2, column=1, ...
      x_log, yzero_line, ...
      marker_style=[4,4,4,4,4], marker_color=[5,11,1,3,6], marker=1, ...
      line_style  =[1,2,7,4,5], line_color  =[5,11,1,3,6], ...
      xmin=SortedFrequency(1), ...
      xmax=SortedFrequency(length(SortedFrequency)), ...
      xinc=10, xlab="Frequency f [Hz]", ...

      ymin=Phase_PlotMinimum, ...
      ymax=Phase_PlotMaximum, ...
      yinc=Phase_PlotIncrement, ...
      ylab="Phase [deg]", ...
      !legend, keep=NFR_Plot});

if l>0 then                                 //Mark the averaged dots, if any
NFR_PhaseDBAv = ...
      plot(SortedFrequencyPhaseAv, PlotPhaseAvMatrix, ...
      {row=2, column=1, line_style  =" ", legend="Averaged", ...
      marker_style=[8,8,8,8,8], marker_color=[2,2,2,2,2], marker=1,
keep=NFR_Plot});
endIf

if offset ==1 then
NFR_Offset = ...                            //Generate offset plot
      plot(SortedFrequency, OffsetMatrix, ...
      {row=3, column=1, x_log, yzero_line, ...
      marker_style=[4,4,4,4,4], marker_color=[5,11,1,3,6], marker=1, ...
      line_style  =[1,2,7,4,5], line_color  =[5,11,1,3,6], ...
      xmin=SortedFrequency(1), xmax=SortedFrequency(length(SortedFrequency)),
xinc=10, ...
      xlab="Frequency f [Hz]", ylab="Output Offsets", ...
      !legend, keep=NFR_Plot});
endIf
NFR_Plot?


EndFunction
```

11.3 List of commands used in the subspace identification

```
//-----code to generate a PRBS with lower rate of change----

t=(0:0.022:140)';
p=prbs(15);
f=30;
i=1;
delete u
gk=0;
for k=1:length(t) do
  u(k)=p(i);
  if mod(t(k),1/f)<gk then
    i=i+1;
  endif
  gk=mod(t(k),1/f);
endfor
delete p
u=u-0.5;
u=2*u;


//-----generation of two input series and plotting of them--

inputpdm1 = pdm(u(1:3000), t(1:3000))';
inputpdm2 = pdm(u(3001:6000), t(1:3000))';

plot(inputpdm1)
plot(inputpdm2)
plot(sdf(inputpdm1,inputpdm1,128))
plot(sdf(inputpdm2,inputpdm2,128))


//-----system simulation and pre processing-----------------

outputpdm1=sim("Pitch_Gain_Direct_Law_SIM",inputpdm1);
outputpdm2=sim("Pitch_Gain_Direct_Law_SIM",inputpdm2);

plot(outputpdm1)
plot(sdf(outputpdm1,outputpdm1,128))

coh=coherence (inputpdm1,outputpdm1,20);
plot(coh)

outputpdm1=detrend(outputpdm1);

//-----model determination----------------------------------

[sys, sr]=sds(outputpdm1, inputpdm1, {gui});

//the model is saved  as "test6" and validated through the...
// use of the pull down menus in the "gui" user interface

//-----MATRIXX buggs quite a bit using the "gui" interface ...
//-----therefore a few validation methods are implemented ...
//-----manually. They can be found below.

//-----pole-zero plot---------------------------------------

[num,den]=numden(test)

ip=imag(roots(den));
rp=real(roots(den));
plot(rp,ip, {line=0, marker_style=1})
```

```
in=imag(roots(num));
rn=real(roots(num));
plot(rn,in, {keep, line=0, marker_style=4})

pt=(0:0.01:3.2);
s=sin(pt);
c=cos(pt);
plot(c,s,{keep, linecolor=1})
plot(-c,-s,{keep, linecolor=1})

//-----prediction plots-----------------------------------

plot(outputpdm1)

valoutputpdm1=sdsmodel*inputpdm1;
plot(valoutputpdm1, {keep})

plot(outputpdm2)

valoutputpdm2=sdsmodel*inputpdm2;
plot(valoutputpdm2, {keep})

//-----to plot what is on the screen-----------------------

hardcopy; oscmd("lp -onb");

//-----to generate a superblock with result----------------

[Num,Den]=numden(test6);
NumCoef=makematrix(Num);
NumCoef=NumCoef';
DenCoef=makematrix(Den);
NumOrder = size(Num);
DenOrder = size(Den);
dT = period(test6);
CREATESUPERBLOCK "test", {type = "discrete", sampleperiod = dT };
CREATEBLOCK "NumDen", {DenominatorOrder = DenOrder, NumeratorOrder=NumOrder,
Denominator = DenCoef,  Numerator = NumCoef};
```