ISSN 0280–5316
ISRN LUTFD2/TFRT--5477--SE

A Quantitative Feedback Theory Toolbox for Matlab 4.1

Michael Lekman

Department of Automatic Control
Lund Institute of Technology
August 1993

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 1993

Document Number
ISRN LUTFD2/TFRT--5477--SE

Author(s)
Michael Lekman

Supervisor
Kjell Gustafsson, Tore Hägglund

Sponsoring organisation

Title and subtitle
A Quantitative Feedback Theory Toolbox for Matlab 4.1

Abstract

In this master project a toolbox for Quantitative Feedback Theory (QFT) design has been implemented in Matlab 4.1. The thesis describes the theory behind QFT. It also describes the implementation as well as the man-machine interface of the toolbox.

QFT is a design method for robust design with regard to plant uncertainty and specification for the closed loop system. QFT is a graphical design method carried iteratively out in the frequency domain. Due to the intensive graphical manipulations required, the method is tedious to work with without computer support.

Derivation of bounds (boundary–curves on the loop gain) requires graphical manipulations in the Nichols chart. Derivation of template (a set of possible points [phase,amplitude] for a specific frequency) demands a lot of calculations. Those parts are well suited for computer implementation.

The QFT toolbox is not just one large program which solves control problems, it is a set of routines each solving a part of the QFT design process. This gives the user a chance to more freely use the toolbox, for example evaluate other design methods with regard to robustness. The QFT toolbox deals so far only with SISO–systems, but it is possible to expand it to MIMO–systems.

Key words
robust design, quantitative feedback theory, templates, performance bounds, stability bounds, compensator, prefilter

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
44

Recipient's notes

Security classification

Contents

1. Introduction

Quantitative Feedback Theory (QFT) is a design method for robust design with regard to plant uncertainty. QFT is difficult to work with without computer support because of the intensive graphical manipulations required. Today almost everyone however have access to a personal computer or workstation so by providing a tool, the method becomes more practical.

This master thesis aims to implement a set of routines for QFT. Matlab version 4.1 was chosen due to its new graphical features.

I wish to thank the staff at the Department of Automatic Control, Lund Institute of Technology, in particular Kjell Gustafsson and Tore Hägglund for helping me with Matlab and the theory behind QFT. I also wish to thank Leif Andersson for helping me with LaTeX and other problems I had with the computer system.

Michael Lekman

2. Quantitative feedback theory

Quantitative feedback theory (QFT) uses feedback to achieve desired dynamic performance despite plant uncertainty and plant disturbances. The main goal for any design method is to arrive at a closed loop system with desired rise time, overshoot, settling time, phase margin, gain margin, etc. These specifications are defined in either time domain or frequency domain. QFT is carried out in the frequency domain due to the fact that uncertainty is easier to deal with in the frequency domain. The fundamental steps required for QFT design are

- describe the desired system in the time domain or the frequency domain,
- derive plant templates, i.e. a set of possible frequency points for the plant,
- derive bounds on the loop gain $L(j\omega)$,
- find a compensator that makes the loop gain close to the optimum, and
- derive the prefilter.

Although QFT is applicable to different problems, e.g. SISO linear time invariant systems, SISO nonlinear systems, MIMO linear and nonlinear systems and sampled systems, only SISO Linear Time Invariant (LTI) system will be considered in this project.

The first section describes the closed loop system and its uncertainty related to the plant uncertainty. The next section shows how specifications are defined and how specifications are translated from time domain to frequency domain. Determination of templates and bounds are described in sections 2.3 and 2.4. The derivation of the compensator and the prefilter are described in sections 2.5 and 2.6.

2.1 The closed loop

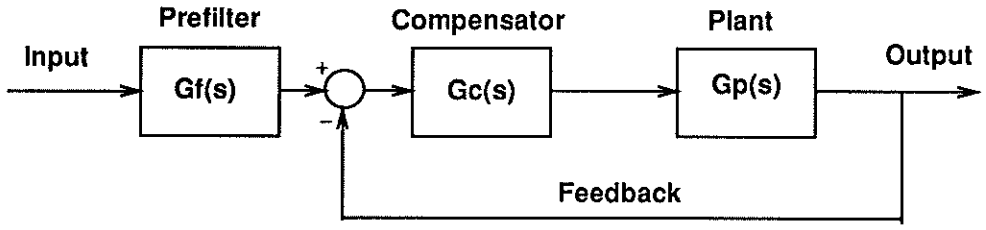

Figure 2.1 Block diagram of a two-degree-of-freedom system

Consider the block diagram in Figure 2.1. The transfer function of the system is

$$G_{cl} = G_f \frac{G_c G_p}{1 + G_c G_p}.$$

Introduce the loop gain, $L = G_c G_p$. The transfer function can then be written as

$$G_{cl} = G_f \frac{L}{1 + L}. \tag{2.1}$$

In most cases the plant is not known exactly. The basic idea is to characterize the plant with its nominal transfer function, and then determine a compensator with respect to the uncertainty in gain and phase of the plant transfer functions so that the specified limits on the closed loop behavior are obeyed. Finally, the prefilter is determined so that the transfer function from input to output behaves as specified.

To separate the design of G_c from G_f take the logarithm of the transfer function (2.1). This gives

$$\log|G_{cl}| = \log|G_f| + \log\left|\frac{L}{1+L}\right|.$$

There is no uncertainty in the prefilter. Therefore the uncertainty of the transfer function becomes

$$\Delta \log|G_{cl}| = \Delta \log\left|\frac{L}{1+L}\right|.$$

For each frequency there is a tolerance in gain of the closed loop transfer function. This is depicted in Figure 2.2. Let $a(\omega)$ denote the lower bound and $b(\omega)$ the upper bound. To obey the constraint the closed loop has to satisfy

$$a(\omega) \leq |G_{cl}(j\omega)| \leq b(\omega). \tag{2.2}$$

Introduce $\delta(\omega) = b(\omega) - a(\omega)$, then

$$\Delta\,|G_{cl}(j\omega)| \leq \delta(\omega). \tag{2.3}$$

To obey the constraint (2.3) the compensator must be designed so that the variation in loop gain satisfies

$$\Delta\left|\frac{L(j\omega)}{1+L(j\omega)}\right| < \delta(\omega).$$

Once a compensator has been found that reduces the closed loop gain variation below the specification, a prefilter can be designed to make the final loop obey (2.2).

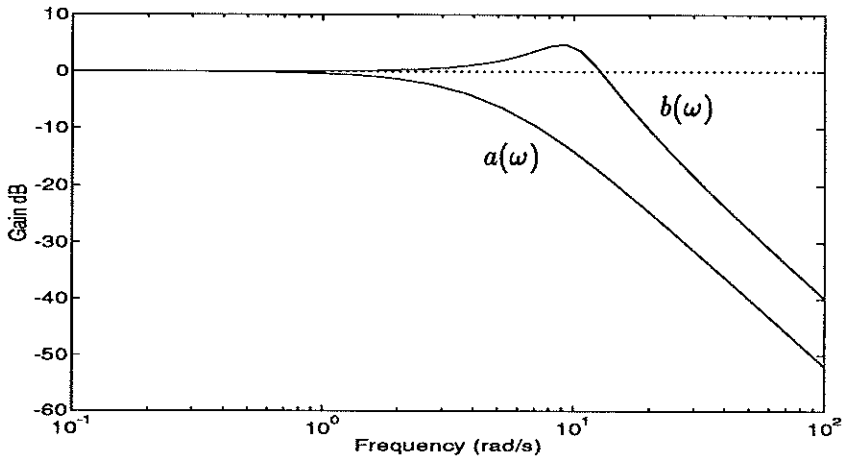

Figure 2.2 Specifications defined in the frequency domain.

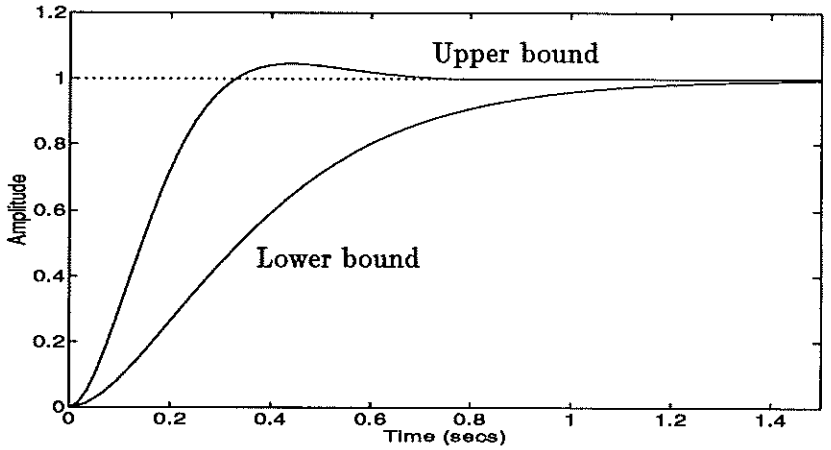

Figure 2.3 Specifications defined in the time domain.

2.2 Specification

The first step in QFT is to specify desired performance for the system. It can be done in either the frequency domain or the time domain. In the time domain, the desired step response for the system can e.g. be characterized in terms of max and min rise time, t_r^{max}, t_r^{min} and max overshoot, see Figure 2.3. Figures 2.4 and 2.5 depict the step responses of a set of second order systems with different ζ and ω. The closed loop system should not have larger bandwidth ω_{bw} than necessary,

$$|G_{cl}(j\omega_{bw})| = \frac{G_{cl}(j0)}{\sqrt{2}}.$$

A larger bandwidth makes the system amplify high frequent noise and gives problems with control signal saturation.

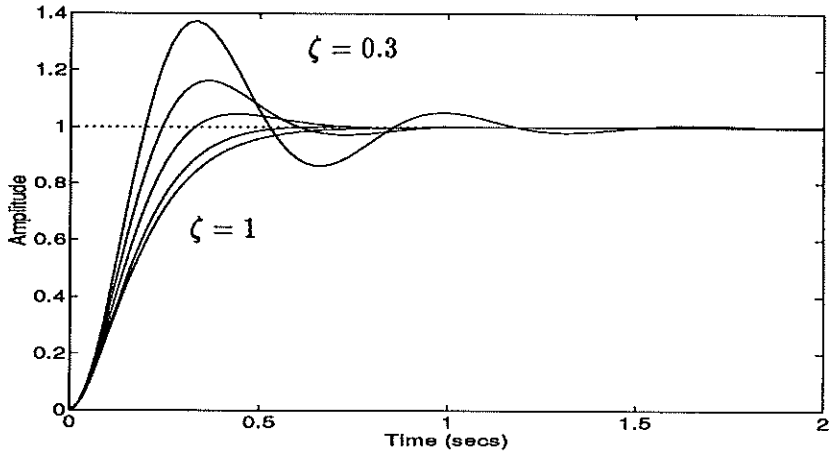

Figure 2.4 Step responses of second order systems with different values of $\zeta \in [0.3\ 0.5\ 0.7\ 0.9\ 1]$ and $\omega = 10$.

The next step is to translate the specification from the time domain to the frequency domain. In general this is an unsolved problem. An approximating translation can be done by approximating the upper bound of the step response with the response of a dominated second order system (2.4), and the lower

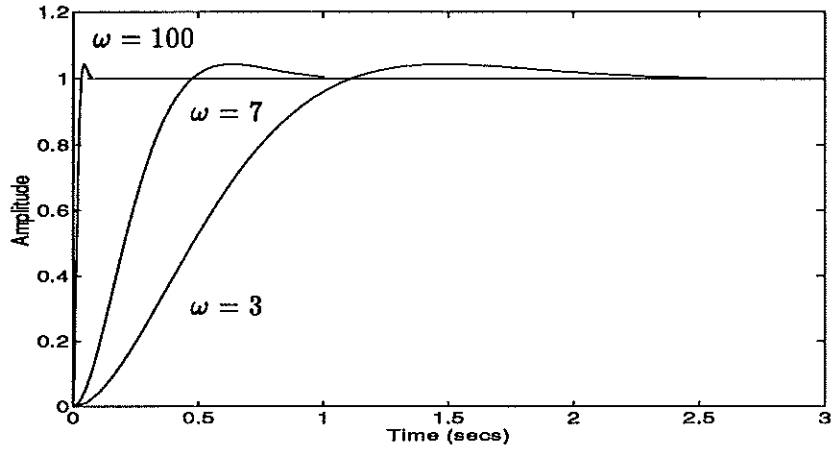

Figure 2.5 Step responses of second order systems with different values of $\omega \in [3\ 7\ 100]$ and $\zeta = 0.707$.

bound of the step response with the response of a second order system (2.5), with ζ fixed to one.

$$G_{upper}(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} \tag{2.4}$$

$$G_{lower}(s) = \frac{\omega^2}{(s+\omega)^2} \tag{2.5}$$

The corresponding step responses are

$$y_{upper}(t) = \begin{cases} 1 - e^{-\zeta\omega t}\left[\cos\left(\omega\sqrt{1-\zeta^2}t\right) + \frac{\zeta}{\sqrt{1-\zeta^2}}\sin\left(\omega\sqrt{1-\zeta^2}t\right)\right] & \zeta < 1 \\ 1 - e^{-\omega t}(1+\omega t) & \zeta = 1 \end{cases}$$

$$y_{lower}(t) = 1 - e^{-\omega t}(1+\omega t)$$

Specification can also be defined directly in the frequency domain. Typical specifications are maximum/minimum values for the bandwidth and the resonance peak. Figure 2.2 showed an example of specification defined in the frequency domain.

2.3 Templates

Suppose the plant can be modeled by a rational transfer function

$$G_p(s) = \frac{B(s)}{A(s)} = \frac{\sum_{i=o}^{m} b_i s^i}{\sum_{i=o}^{n} a_i s^i}.$$

At each frequency $s = j\omega$ the model $G_p(s)$ describes a single frequency point, $[arg(G_p(j\omega)), |G_p(j\omega)|]$, in the Nichols chart. If the plant contains unknown parameters then the parameters have to be estimated. The estimated plant

$$\hat{G}_p(s) = \frac{\sum_{i=o}^{m} \hat{b}_i s^i}{\sum_{i=o}^{n} \hat{a}_i s^i}$$

has variation in each estimated parameter, and for each frequency the plant is described by a region in the Nichols chart. This region is called the template

$\mathcal{F}G_p(j\omega)$ (a set of possible frequency points). In the case the variation of two or several parameters depend on each other, the calculation of the template becomes quite involved. We will therefore assume that the parameters vary independently of each other.

EXAMPLE—DC-motor
The problem is to design a prefilter and compensator to control a set of motors with loads. The DC-motor can be described by a differential equation

$$J\ddot{y} + D\dot{y} = Ku$$

Let $\frac{D}{J}$ and $\frac{K}{J}$ have a wide range of values, $\frac{D}{J} \in [1,5]$, $\frac{K}{J} \in [1,20]$. Then the plant can be described by the set

$$\mathcal{G}_p = \left\{ G_p(s) = \frac{b}{s(s+a)} \;\middle|\; b \in [1,20],\ a \in [1,5] \right\}.$$

The template, $\mathcal{F}G_p(j\omega)$, for $\omega = 3$ can be obtained in the following way. The line AB shows the amplitude and phase for different values of a when b is fixed, $b = b_{min} = 1$, see Figure 2.6. The variations of b moves the line AB upward by $b_{max}/b_{min} = 20(26\text{dB})$, giving A'B'. This procedure is repeated for a set of chosen frequencies. □

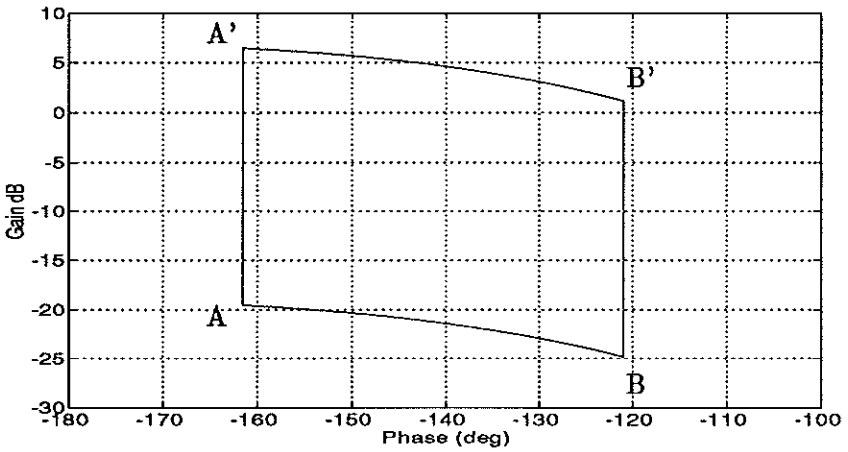

Figure 2.6 Template calculated for the DC-motor example, $\omega = 3$.

2.4 Performance bounds

The basic idea of QFT is to find a compensator so that the feedback moves the template to a position where it is completely inside an area where the closed loop specifications are met. This amounts to having a certain minimal gain for each frequency. We start by calculating bounds for this gain and then use the bound to design the compensator. The template $\mathcal{F}G_p(j\omega_1)$ can be written in complex form

$$\mathcal{F}_{polar}G_p(\omega_1) = |\mathcal{F}G_p(j\omega_1)|e^{j\arg(\mathcal{F}G_p(j\omega_1))}$$

and the template mapped by feedback as

$$\frac{\mathcal{F}_{polar}G_p(j\omega_1)}{1 + \mathcal{F}_{polar}G_p(j\omega_1)}.$$

Introduce the loop gain set $\mathcal{Z}L(j\omega_1) = \mathcal{F}_{polar} G_p(j\omega_1)G_c(j\omega_1)$. The next step is to assume a specific phase for the nominal point in the template and then calculate what magnitude the nominal point must have in order for the set of loop gains to fulfill gain for the nominal point of the loop gain set such that

$$\left|\frac{\mathcal{Z}L(j\omega_1)}{1+\mathcal{Z}L(j\omega_1)}\right| \leq \delta(\omega_1).$$

This is done for every phase in [-360,0] and the result is a lower bound $B(\omega_1, \phi)$ on the magnitude of the compensator and the nominal plant. The loop gain must lie above this bound for the closed loop system to fulfill the specifications. Figure 2.7 demonstrates the calculations. This is done for all frequencies where templates are calculated.

EXAMPLE—DC-motor-Continued
We calculate bounds on the loop gain for the DC motor control system in the previous example. First choose a nominal plant

$$G_P^{nom} = \frac{1}{s(s+1)}$$

Next step is to find bounds on the loop gain such that the specifications are satisfied over $\mathcal{G}_P$. For $\omega = 3$ the maximum allowed gain variation is $\delta(3) = 3.5$ (10.9dB). To find bounds for $\omega = 3$, $L(j3)$ has to be chosen such that

$$\left|\frac{L(j3)}{1+L(j3)}\right| \leq \delta(3).$$

In Figure 2.7 the template for $\omega = 3$ is plotted at two locations. For trial one the maximum gain is 1dB and minimum gain -12dB this yields a gain variation of 13dB, which is too large. In trial two the minimum gain is -6dB which yields a gain variation of 7dB, which is acceptable but unnecessarily small. The bound line shows where the template should be at this phase. This procedure is repeated for a set of phases between 0 to -180 degrees. Observe that the template may only be translated, not rotated. □

2.5 The compensator

After the bounds on the loop gain, $L(j\omega)$, are determined, then shape the compensator so that $L(j\omega)$ fulfills the bound. This is done by adding lead and lag filters to $G_c(j\omega)$ so that $L(j\omega_i)$ lies above $B(\omega_i, \phi)$. $L(j\omega)$ must also satisfy the stability bounds. The optimum design of the compensator is obtained when the loop gain, $L(j\omega)$, lies on the bounds at low frequencies and decreases in magnitude as fast as possible in the high frequency range, i.e. $L(j\omega)$ has minimum gain and bandwidth. The reason for this is to reduce implementation problem, e.g. noise sensitivity and saturation sensitivity, etc. One way to find the compensator is to find a dominating bound, i.e. if the dominating bound is satisfied at a reasonable phase lag of $L(j\omega)$ then the other ones will automatically be satisfied. Therefore it is enough to have the loop gain, $L(j\omega_{dominating})$, on the bound, $B(\omega_{dominating}, \phi)$.

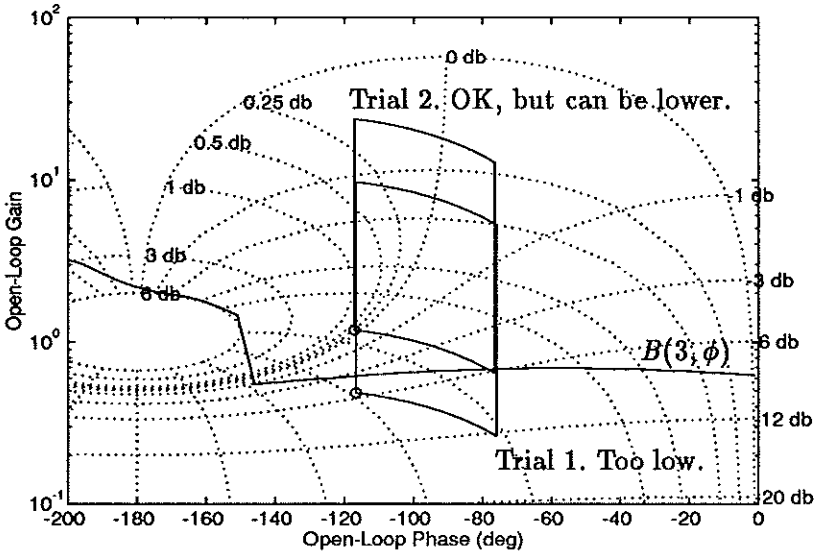

Figure 2.7 Bounds on the loop gain, $L(j3)$, and templates $\mathcal{F}G_p(j3)$, at two locations.

2.6 The prefilter

The last step is to add a prefilter so the specifications on the closed loop behavior are fulfilled. After the compensator design the closed loop will be within the limits for gain variation for each frequency, but the absolute value or the gain may not be right. Therefore a prefilter must be added to the system.

3. Implementation of QFT

Matlab version 4.1 was chosen for implementing the toolbox. Matlab is an excellent program language for implementation of this kind of graphical design methods because Matlab already has graphical visualization routines implemented and it deals also easily with matrix computations. Version 4.1 have several new features such as control buttons, menus, mouse events and figure events which are well suitable for developing a toolbox for QFT. In the following sections the implementation of the toolbox are described and it also shows how to use the toolbox.

3.1 Descriptions of some of the graphical features in Matlab 4.1.

The new version of Matlab is a substantial improvement over version 3.5. Especially the graphics have been much enhanced. In this new version it is possible to have more than one figure at one time. All objects in a figure are described with a handle (a pointer to the object). There are a lot of options for each type of object. To see which options that are available type *set(object handle)* and *get(object handle)*. Take as an example a line object:

```
>> handle=plot(1:10);
>> get(handle)
        Color = [1 1 1]
        EraseMode = normal
        LineStyle = -
        LineWidth = [0.5]
        MarkerSize = [6]
        Xdata = [1 2 3 4 5 6 7 8 9 10]
        Ydata = [1 2 3 4 5 6 7 8 9 10]
        Zdata = []

        ButtonDownFcn =
        Children = []
        Clipping = on
        Interruptible = no
        Parent = [49.0004]
        Type = line
        UserData = []
        Visible = on


>> set(handle)
        Color
        EraseMode: [ {normal} | background | xor | none ]
        LineStyle: [ {-} | -- | : | -. | + | o | * | . | x ]
        LineWidth
        MarkerSize
        Xdata
        Ydata
```

```
        Zdata

        ButtonDownFcn
        Clipping: [ {on} | off ]
        Interruptible: [ {no} | yes ]
        Parent
        UserData
        Visible: [ {on} | off ]
```

There are several new features for a graphical user interface, e.g. control buttons, menus, mouse events and figure events. Uicontrol defines a control button with a callback function connected. By pressing a mouse button when the pointer is on the uicontrol, its callback function is evaluated, i.e *eval('string')* is executed. In the following example a control button is created and a callback property is defined.

```
>> handle=uicontrol;
>> set(handle)
        CallBack
        BackgroundColor
        ForegroundColor
        HorizontalAlignment: [ left | center | {right} ]
        Max
        Min
        Position
        String
        Style: [ {pushbutton} | radiobutton | checkbox | edit |
                 text | slider | frame | popupmenu ]
        Units: [ inches | centimeters | normalized | points |
                {pixels} ]
        Value

        ButtonDownFcn
        Clipping: [ {on} | off ]
        Interruptible: [ {no} | yes ]
        Parent
        UserData
        Visible: [ {on} | off ]

>> set(handle,'callback','plot(sin(1:0.1:50))');

>> get(handle)
          BackgroundColor = [1 1 1]
          CallBack = plot(sin(1:0.1:50))
          ForegroundColor = [0 0 0]
          HorizontalAlignment = center
          Max = [1]
          Min = [0]
          Position = [20 20 60 20]
          String =
          Style = pushbutton
          Units = pixels
          Value = [0]
```

```
ButtonDownFcn =
Children = []
Clipping = on
Interruptible = no
Parent = [1]
Type = uicontrol
UserData = []
Visible = on
```

In this example a sinusoidal is plotted in the figure when the control button is pressed. When using callback functions (m-files) it could take a few seconds the first time the callback function is called, specially when there are a lot of m-files connected to the figure. There is one way to implement these features so that the callbacks are executed faster. This is done by having an input argument *action*, which is a string telling which part of the m-file should be executed, in this way Matlab do not have to store more than one m-file in the CPU-stack. This yields a faster execution of the callback.

```
function example(action)
%example Makes a figure with a control button.
%       By pressing the control button, a sinusoidal
%       is displayed.
%

if nargin < 1, % example call by the user.
  action = 'start';
end;

if strcmp(action,'start'),
  uicontrol('callback','example(''plot'')');
elseif strcmp(action.'plot'),
  plot(sin(1:0.01:100));
else,
  error('Illegal action!!');
end;
```

There are several other objects available such as text, axis, figure, uimenu etc. More about those objects are described in Matlab 4.1 reference guide [8]. The QFT-toolbox relies heavily on these features.

3.2 Data structures

In Matlab the only data structure available is matrixes. To avoid a lot of input and output arguments it is possible to pack data in a matrix. This makes it possible to have a lot of data under one variable name. Data such as templates, bounds and specification need to be predefined. To obtain a well suited data structure there are a few facts to have in mind. First, if there are a lot of input arguments and output arguments with the function call, then it tends to be messy and hard to work with. This can be avoided by a well defined structure on the data handling. Second there must be a way to separate the

data structures from each other. With these points in mind the data structures were defined as follows.
Specifications are defined as:

$$spec = \begin{pmatrix} \omega_1 & \dots & \omega_n \\ mag_1^{max} & \dots & mag_n^{max} \\ mag_1^{min} & \dots & mag_n^{min} \end{pmatrix}$$

where ω_i are the frequencies for which maximum amplitude, mag_i^{max}, and minimum amplitude, mag_i^{min}, for the desired closed loop are defined.
The templates data structure is:

$$tmp = \begin{pmatrix} mag_1^1 & phase_1^1 & \omega_1 \\ mag_2^1 & phase_2^1 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_1}^1 & phase_{n_1}^1 & NaN \\ mag_1^2 & phase_1^2 & \omega_2 \\ mag_2^2 & phase_2^2 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_2}^2 & phase_{n_2}^2 & NaN \\ \vdots & \vdots & \vdots \\ mag_1^n & phase_1^n & \omega_n \\ mag_2^n & phase_2^n & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_n}^n & phase_{n_n}^n & NaN \\ mag_{G_p(\omega_1)} & phase_{G_p(\omega_1)} & \omega_1 \\ \vdots & \vdots & \vdots \\ mag_{G_p(\omega_m)} & phase_{G_p(\omega_m)} & \omega_m \\ n & m & NaN \end{pmatrix}$$

where ω_i are the frequencies for which templates are calculated. mag_j^i and $phase_j^i$ for $j = 1 \dots n$; defines the contour of the template for ω_i. NaN[1] is used to separate the templates from each other. $mag_{G_p(\omega_i)}$ and $phase_{G_p(\omega_i)}$ defines the frequency response for the plant. n is the number of templates and m is the number of frequency points.

1 NaN is the IEEE arithmetic representation for Not-a-Number (NaN)

Bounds are defined as:

$$bnd = \begin{pmatrix} mag_1^1 & phase_1^1 & \omega_1 \\ mag_2^1 & phase_2^1 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_1}^1 & phase_{n_1}^1 & NaN \\ mag_1^2 & phase_1^2 & \omega_2 \\ mag_2^2 & phase_2^2 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_2}^2 & phase_{n_2}^2 & NaN \\ \vdots & \vdots & \vdots \\ mag_1^n & phase_1^n & \omega_n \\ mag_2^n & phase_2^n & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_n}^n & phase_{n_n}^n & NaN \\ mag_{G_p(\omega_1)} & phase_{G_p(\omega_1)} & \omega_1 \\ \vdots & \vdots & \vdots \\ mag_{G_p(\omega_m)} & phase_{G_p(\omega_m)} & \omega_m \\ n & NaN & NaN \end{pmatrix}$$

where ω_i are the frequencies for which bounds are calculated. mag_j^i and $phase_j^i$ are the line for the bound, for $j = 1 \ldots n$; defines the bound for ω_i. NaN is used to separate the bounds from each other. $mag_{G_p(\omega_i)}$ and $phase_{G_p(\omega_i)}$ are the frequency response for the plant. n is the number of bounds.

There are m-files to check what type of data (specification, template, or bound) is stored in a matrix. See *isbnd* and *istmpl* on page 37 and page 38.

3.3 Defining specifications

The specifications are defined in the frequency domain with a graphical tool, i.e a Matlab figure with a set of uimenus with callback functions. To define specifications in the frequency domain just type, *frqd2spec*. Then a figure appears on the screen, see Figure 3.1. There is a set of possible ways to define the specifications, and there is a menu bar at the top of the figure.

3.4 Derivations of templates

Calculating the template is difficult when the plant transfer function coefficients vary in relation to each other. The used algorithm calculates the templates assuming that the parameters vary independently. This makes the templates bigger than necessary, which will make the design conservative. The used algorithm is described in [5].

3.5 Derivations of bounds

Derivations of bounds are the most time consuming part of QFT. It takes a lot of time when the calculations are made without computer support. Templates

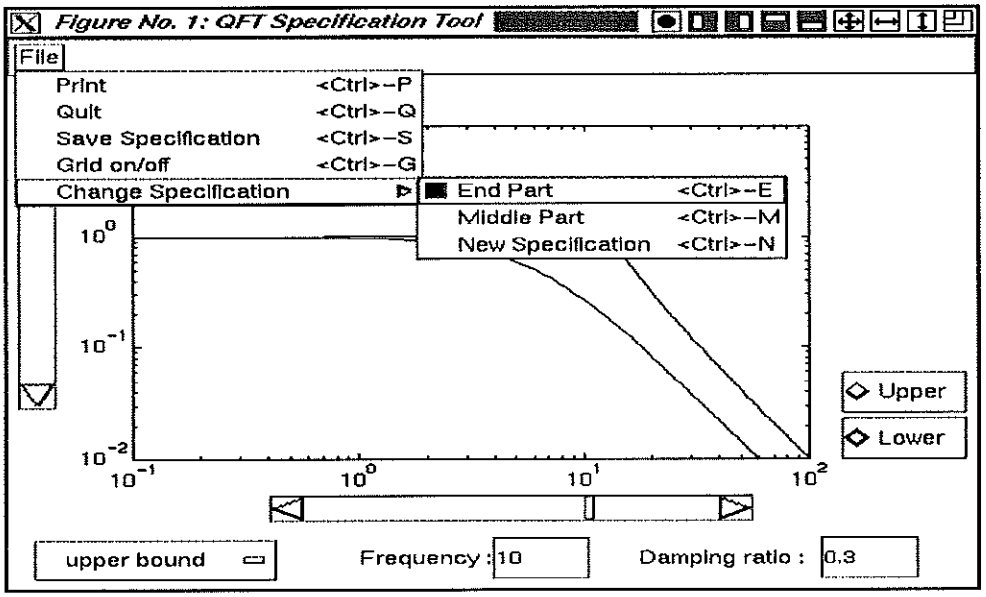

Figure 3.1 The Matlab figure which appears when *frqd2spec* is used.

are drawn on transparent paper, with a hole made for the nominal point, and then moved around on a Nichols diagram to find the bounds. The bounds are found by placing the nominal point for the template at a fixed phase and then moving the template until the template lies inside the M-circles with the desired gain variations. These steps are tedious to do manually.

The m-file for bound calculation implements the following algorithm:

Step 1. Get frequencies for which templates have been calculated.

Step 2. Get one template and map it in the complex plan and normalize with the nominal gain.

Step 3. Place the normalized template above the critical point -1 (in Nichols [-180,1 (0dB)]) and use the bisection algorithm to find the gain where the specifications are obtained.

Step 4. Move to next phase towards 0 degrees and do step 3.

Step 5. If there is not a gain where the specification is fulfilled place the template at the same phase and start reverse bisection for the gain, i.e do the bisection algorithm for lower bound.

Step 6. Do the same for -180 degrees to -360 degrees. Note that the algorithm places the template above the stability bound (under when reverse bisection is used).

Step 7. Finally multiply the bound with the nominal gain.

Step 8. Store the result in the bounds data structure.

Step 9. Go to step 2.

3.6 Visualization of templates and bounds

There are two plot routines available, *bndpl* and *tmplpl*. Templates are plotted with *tmplpl* and bounds with *bndpl*. These m-files use a figure, each with a lot of control buttons and menus, see Figures 3.2 and 3.3. In *tmppl* it is also possible to grab one template and move it around. This makes it possible to check for

locations of bounds. These control buttons and menus execute *bndpl(action)* or *tmplpl(action)* when a mouse button click appears on a control button or menu.

3.7 Compensator design tool

This is a graphical tool, i.e a Matlab figure with menus. It is possible to add and delete zeroes to the numerator and denominator polynominal for the compensator, add or delete first order systems and second order systems. For the first order system the user has to define ω, and for the second order system ω and ζ.

$$\frac{1}{\frac{1}{\omega}s+1}$$

$$\frac{1}{\frac{1}{\omega^2}s^2+\frac{2\zeta}{\omega}s+1}$$

Low frequency gain can also be changed. The new loop gain will be plotted after every change in the compensator filter. In figure 3.4 the compensator tool is plotted. After the design is completed then use save in the File-menu to save the design in the root windows user data, *set(0,'userdata',design)* is executed. Use *getcompensator* to get the compensator design.

3.8 Prefilter design tool

Prefilter is a graphical design tool like the compensator design tool, see Figure 3.5. It is possible to add or delete first order and second order systems such as in *compensator*. The design is stored in the root windows user data and can be reached by using *getprefilter* after save is done.

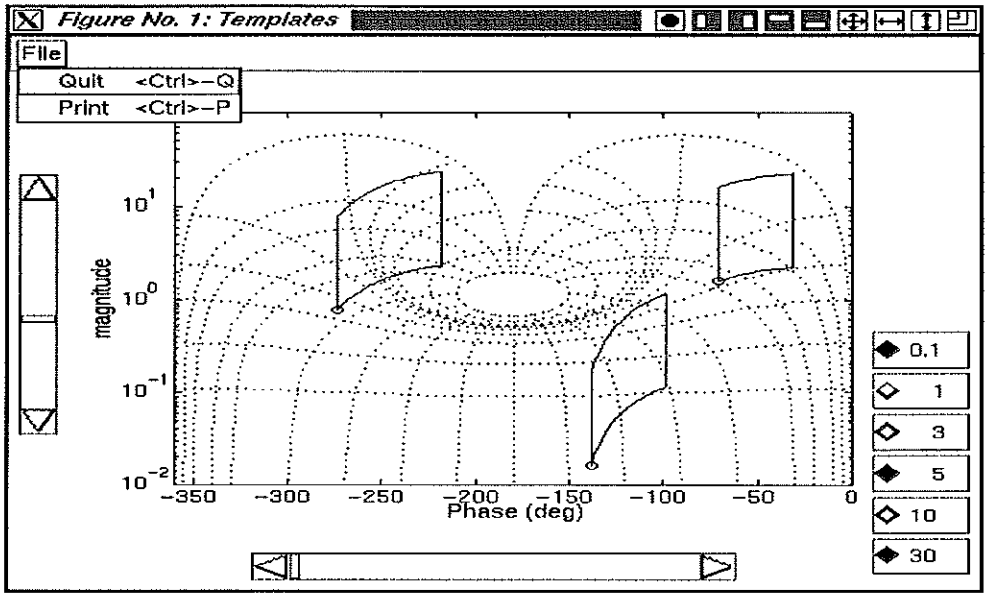

Figure 3.2 The Matlab figure which appears when *tmplpl* is used. It is also possible to grab one template and move it around.

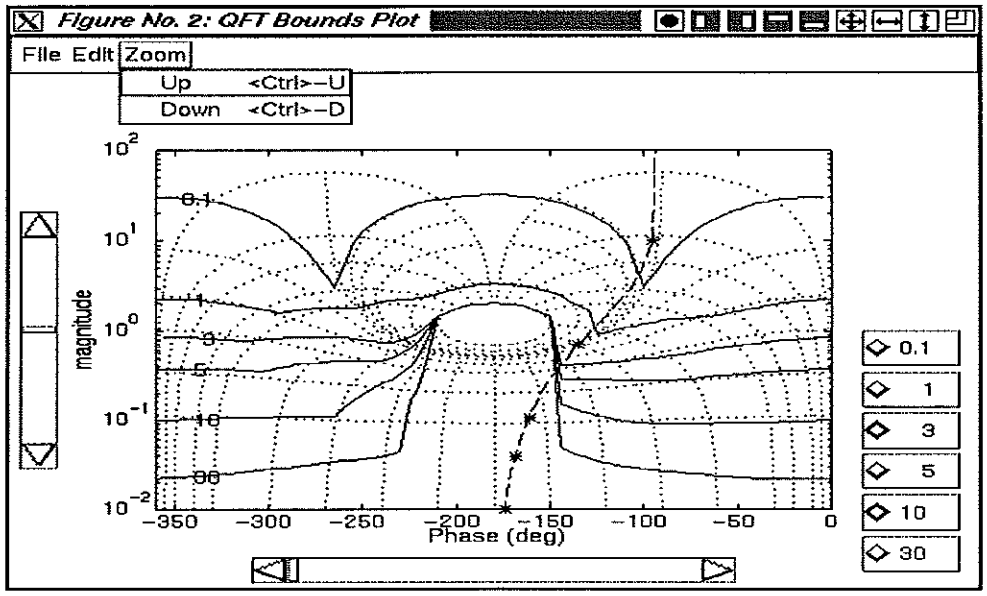

Figure 3.3 The Matlab figure which appears when *bndpl* is used.

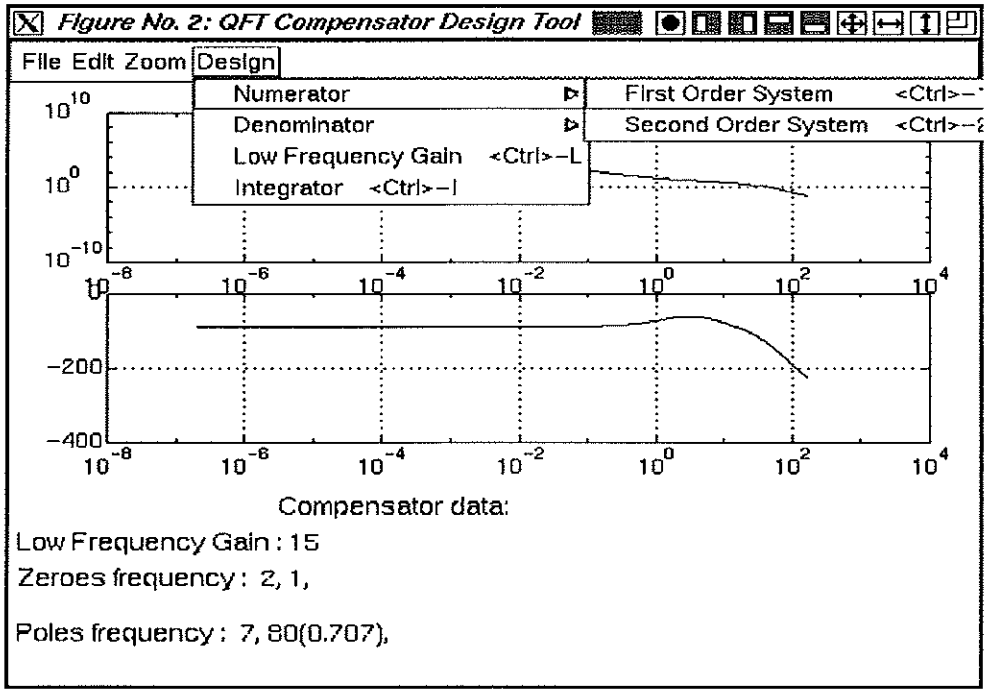

Figure 3.4 The Matlab figure which appears when *compensator* is used. Note that there is a menu bar at the top of the figure.

The input arguments are the compensator numerator and denominator polynominal, the specifications on the dynamical behavior of the closed loop and the possible values of the plant.

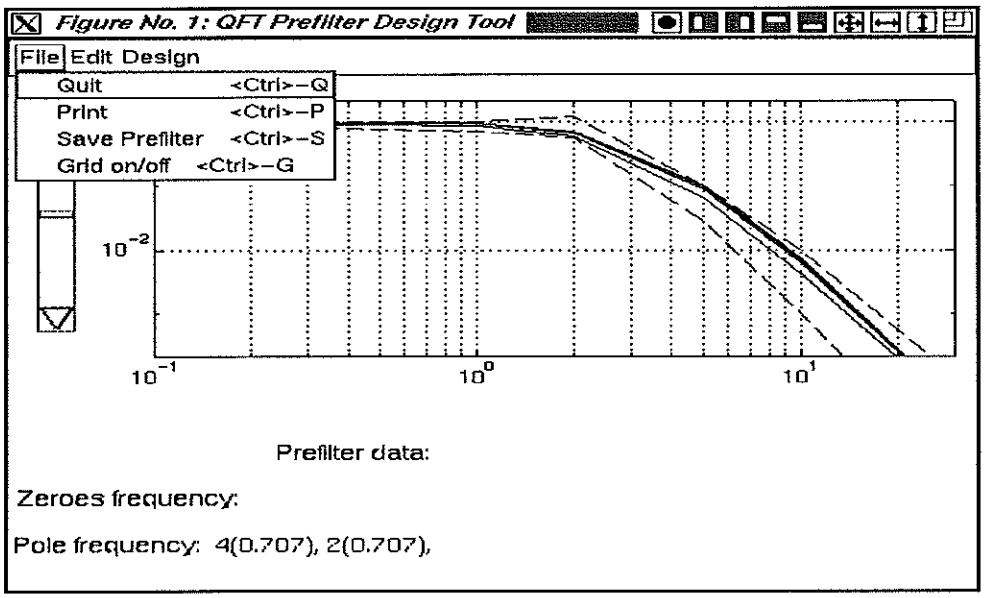

Figure 3.5 The Matlab figure which appears when *prefilter* is used. Note that there is a menu bar at the top of the figure.

3.9 Future improvements

The QFT-toolbox deals so far only with SISO systems. In a future version it would also be able to use MIMO systems and other systems, such as non minimum phase systems or digital sampled systems. Derivation of bounds should also be able to take design parameters such as phase margin, gain margin, etc in consideration.

There is a bug in the m-file for calculations of bounds,*tmpl2bnd*. When the bounds are enclosing the critical point -1 ([-180,1 (0dB)] in the Nichols chart) the reverse bisection algorithm does not function as expected. This bug is an implementation error which is not located yet. There should also be a m-file for derivation of disturbance bounds. This was not implemented due to the bug.

4. A design example

This section contains a design example solved with the QFT toolbox.

4.1 The Plant

To demonstrate the toolbox consider the plant model below, taken from Horowitz and Sidi [1].

$$G_p(s) = \frac{Ka}{s^2 + as}$$

where $a \in [1\ 10]$ and $K \in [1\ 10]$. Note that the coefficients of the numerator and the denominator do not change independently. Rewriting the plant as

$$G_p(s) = \frac{K}{\frac{1}{a}s^2 + s}$$

yields a transfer function whose coefficients change independently with $K \in [1\ 10]$, $\frac{1}{a} \in [0.1\ 1]$.

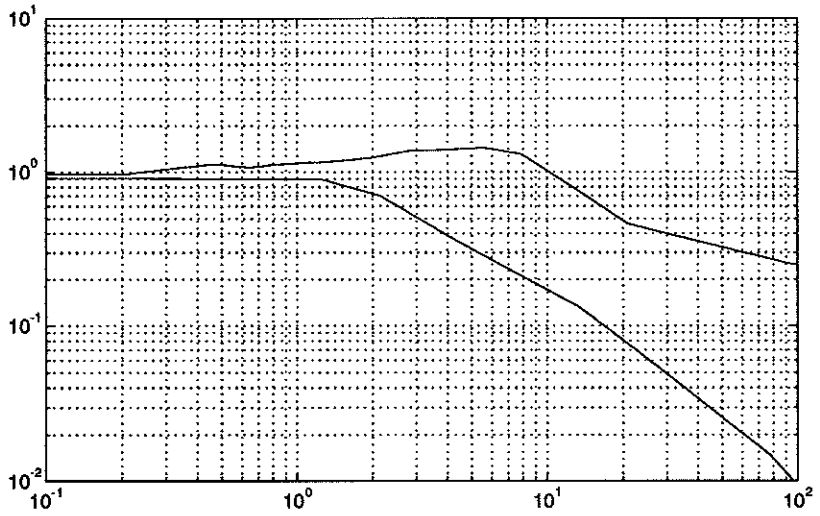

Figure 4.1 Specifications of the closed loop behavior.

4.2 Defining specifications

The first step is to define specifications on the closed loop system. This is done in the frequency domain. By calling the QFT-function *frqd2spec* a graphical tool for defining specifications appears on the screen.

```
>> frqd2spec;
```

Here it is possible to choose a couple of ways to define the specifications. The bounds can be described with two second order models. For the lower bound, ζ is fixed to one, only ω has to be defined. It is also possible to change parts

of the bound lines, end parts or middle parts, by clicking on the bound line with the mouse and move the pointer to change points, the new points are the current pointer locations. To save the specifications choose save in the menu and use *getspec* to get the specifications.

```
>> specmatrix = getspec;
```

4.3 Derivation of templates and bounds

Use *tf2tmpl* to determine the templates for a set of selected frequency points, $\omega \in [0.1\ 1\ 3\ 5\ 10\ 30]$ rad/s.

```
>> nummatrix = [1;10;1]

nummatrix

     1
    10
     1

>> denmatrix = [1 1 0;1 1 0;0.1 1 0]

denmatrix

    1.0000    1.0000         0
    1.0000    1.0000         0
    0.1000    1.0000         0

>> omegavect = [0.1 1 3 5 10 30]

omegavect =

    0.1000    1.0000    3.0000    5.0000   10.0000   30.0000

>> tmpl=tf2tmpl(nummatrix,denmatrix,omegavect);

>> tmplpl(tmpl); % Generates a plot of templates
```

In Figure 4.2 the templates are plotted for a set of selected frequencies from ω. The next step is to compute the bounds on the loop gain, $L(j\omega)$. This is done by using *tmpl2bnd*.

```
>> bnd=tmpl2bnd(tmpl,specmatrix);

>> bndpl(bnd); % Generates a plot of the bounds
```

In Figure 4.3 the bounds are plotted. Note that the *bndpl* makes a plot of the loop gain in the same Nichols chart as for the bounds.

4.4 Compensator and prefilter design

The last two steps are to design the compensator and the prefilter to obtain the specified behavior of the closed loop. This is done by using *compensator*

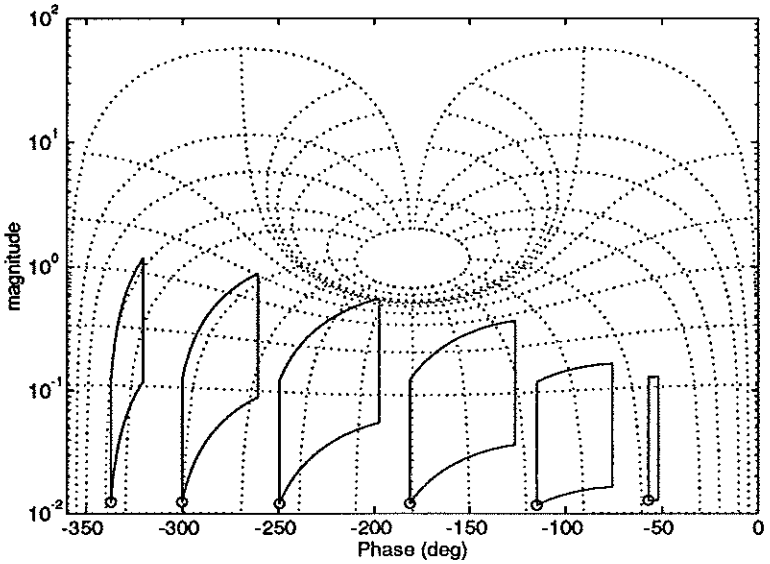

Figure 4.2 Templates for $\omega \in [0.1, 1, 3, 5, 10, 30]$ rad/s.

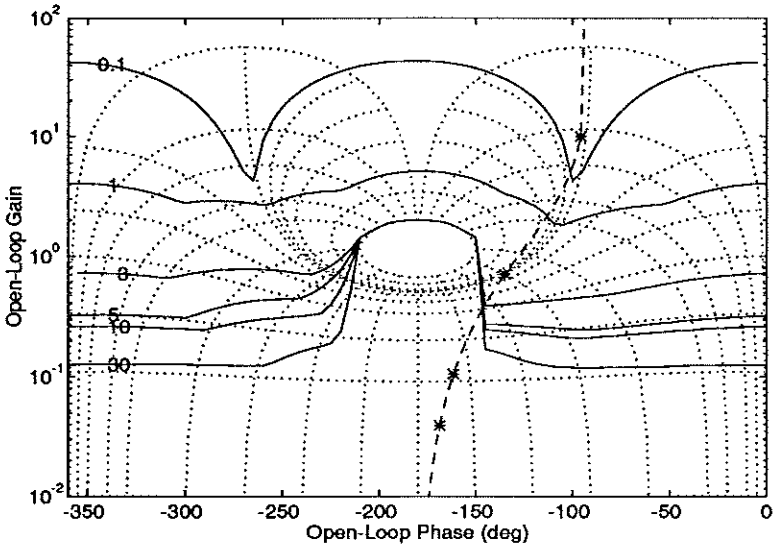

Figure 4.3 Bounds on the loop gain. Bounds are plotted with solid lines while the loop gain is plotted with dashed line. The stars marks the frequencies for which the bounds are computed.

and *prefilter*. The compensator filter and the prefilter are then chosen by the user.

```
>> compensator;
```

The design is done iterative by adding first and second order models to the numerator and denominator polynominal. In this example the chosen compensator is

$$\frac{12800(s+1)(s+7)}{(s+2)(s^2+\frac{2}{\sqrt{2}}80s+80^2)}$$

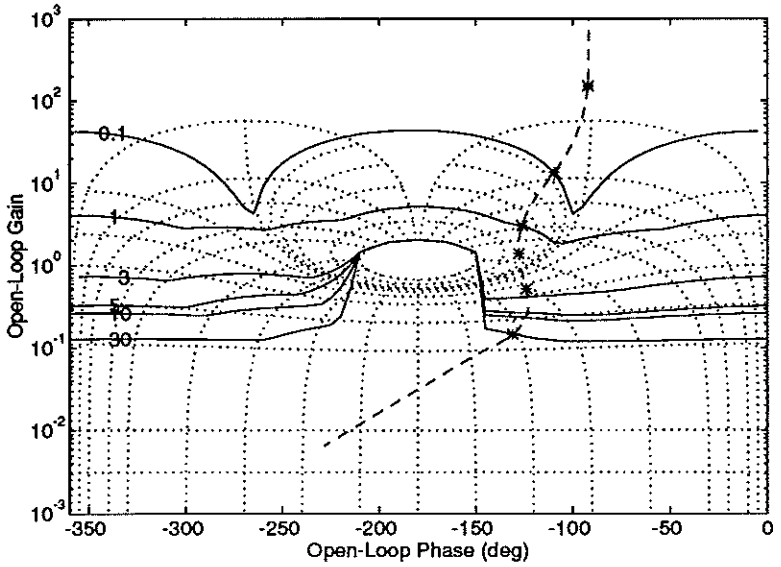

Figure 4.4 The bounds on the loop gain and the final design on the compensator.

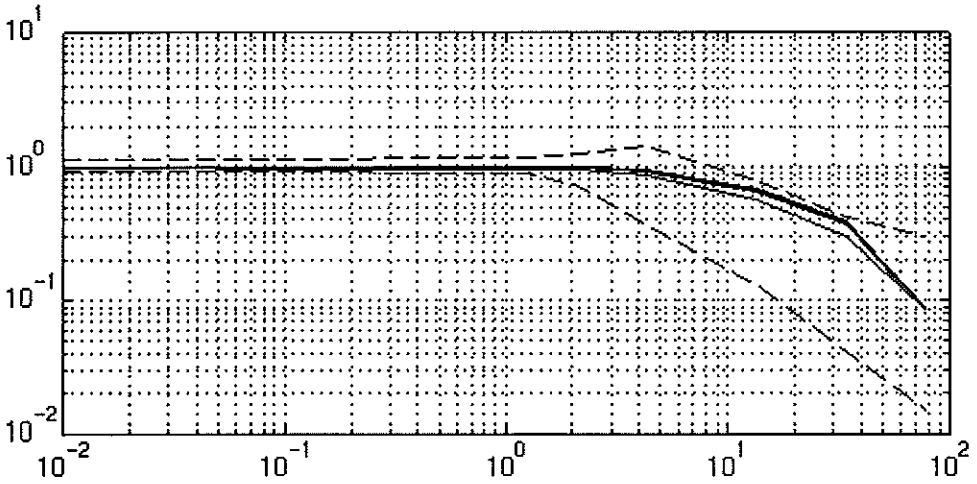

Figure 4.5 A set of possible plants with the prefilter and the compensator.

To save the design choose save in the menu and then use *getcompensator* to get the compensator design.

```
>> [cnum,cden] = getcompensator

cnum =

    2.1429   17.1429   15.0000


cden =

    0.0001    0.0090    0.5177    1.0000
```

Next step is to find the prefilter this is done by using *prefilter*

```
>> prefilter(cnum,cden,nummatrix,denmatrix,specmatrix);
```

The input arguments are the compensator numerator and denominator, the possible plants and the specifications on the closed loop behavior. In this design tool it is possible add and delete first order and second order models to the numerator and denominator polynominal like in *compensator*. In this example the prefilter was chosen as

$$\frac{10}{s+10}$$

To save the current prefilter choose save in the menu and use *getprefilter* to get the prefilter.

```
>> [fnum,fden] = getprefilter

fnum =

     1


fden =

    0.1000    1.0000
```

In Figure 4.4 the loop gain with the final design of the compensator and the bounds on the loop gain are plotted. The final transfer functions from input to output and specifications are plotted in Figure 4.5.

4.5 Simulations

In Figure 4.6 there are several step responses for the plant with different parameters

$$G_p(s) = \frac{K}{\frac{1}{a}s^2+s} \quad K \in [1, 3.25, 7.75, 10] \quad \frac{1}{a} \in [0.1, 0.325, 0.55, 0.775, 1]$$

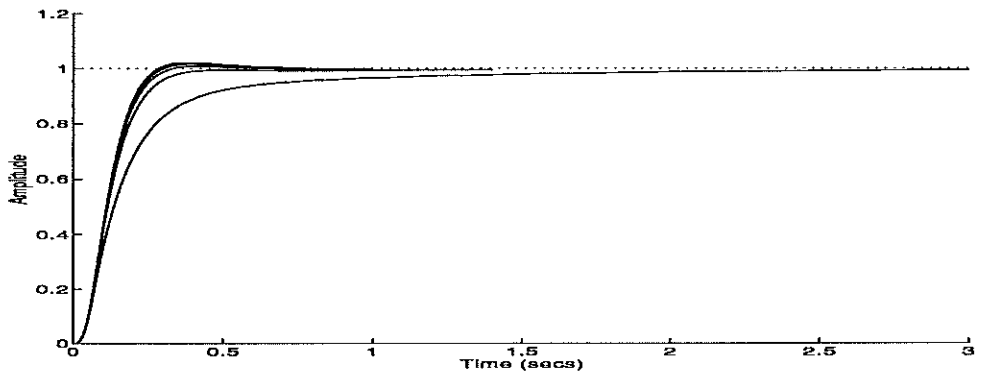

Figure 4.6 Step responses for a set of plants.

5. References

[1] Isaac M. Horowitz and Marcel Sidi. Synthesis of feedback systems with large plant ignorance for prescribed time-domain tolerance. Int.J.Control, 1972, vol. 16, no. 2, 287-309

[2] Isaac M. Horowitz and Uri Shaked. Superiority of transfer function over state-variable methods in linear time-invariant feedback system design. IEEE Transaction On Automatic Control, vol. AC-20, no. 1, February 1975.

[3] Isaac M. Horowitz. A synthesis theory for linear time-varying feedback systems with plant uncertainty. IEEE Transaction On Automatic Control, vol. AC-20, no. 4, August 1975.

[4] Per-Olof Gutman . Robust and adaptive control of a beam deflector. IEEE Transaction On Automatic Control, vol. AC-33, no 7, July 1988.

[5] F.N. Baliey, D. Panzer and G. Gu. Two algorithms for frequency domain design of robust control system. Int. J. Control, 1988, vol. 48, no. 5, 1787-1806.

[6] J.M. Maciejowski. Multivariable feedback design. Addison-Wesley, 1989.

[7] John J. D'Azzo and Constantine H. Houpis. Linear control system analysis and design. McGrawHill, third Edition.

[8] The MathWorks, Inc. 1992. Reference Guide, New Features Guide, User's Guide.

A. Keywords

Compensator	A transfer function that obtains the desired closed loop behavior for the plant.
Performance bounds	Bounds on the loop gain which shows possible locations for the loop gain with regard to the performance specifications.
Prefilter	A transfer function which places the transfer function from input to output within the desired dynamically behavior for the system.
Specifications	Specifications for the desired dynamically behavior for the plant.
Stability bounds	Bounds on the loop gain with regard to stability.
Templates	A set of possible frequency points for the plant at a fix frequency.

B. QFT reference guide

This appendix contains detailed descriptions of all QFT-TOOLBOX functions. Some of the internal functions that are used by the main functions are not described in detail.

Main Functions	
frqd2spec	Define specification in the frequency domain
tf2tmpl	Transfer function to template
tmpl2bnd	Performance bounds on the loop gain
tmpl2stbnd	Stability bounds on the loop gain
compensator	Tool for compensator design
prefilter	Tool for prefilter design

Graphs Routines	
nichlpl	Plots a Nichols chart in the current figure
bndpl	Plot of bounds
tmplpl	Plot of templates

Logical Functions	
isbnd	True for bound matrix
istmpl	True for template matrix

Matrix Manipulation	
get1bnd	Returns one bound at a specified frequency
get1tmpl	Returns one template at a specified frequency
getomega	Returns frequencies from templates or bounds
getspec	Returns specification
getcompensator	Returns compensator filter
getprefilter	Returns prefilter

BNDPL

Purpose

Visualization of loop gain, $L(j\omega)$, and bounds on $L(j\omega)$.

Synopsis

bndpl(bnd)
bndpl(bnd,frc)

Description

bndpl(bnd) plots the bounds calculated by *tmpl2bnd* in a figure. There are control buttons for each bound which makes the bounds visible or not visible. There are also sliders for the frequency axis and the magnitude axis. In the top of the figure there is a menu bar with the following items.

- File
  - Quit <Ctrl>-Q
  - Print <Ctrl>-P
- Edit
  - Compensator <Ctrl>-C
  - Delete design <Ctrl>-N
- Zoom
  - Up <Ctrl>-U
  - Down <Ctrl>-D

bndpl(bnd,frc) takes the input argument *frc*, which is a frequency response chosen by the user, actual loop gain with compensator designed by the user. Note that the plant should be the nominal plant. This makes it possible to evaluate other designs than QFT with respect to robustness.

See Also

tmplpl, *nichlpl*

COMPENSATOR

Purpose

Tool for design of compensator filter.

Synopsis

compensator

Description

compensator displays a Matlab window with a bode plot of the loop gain from the current bounds plot. Observe that *compensator* can not be used without *bndpl*. At the top of the Matlab figure there is a menu with a couple of submenus.

- File
  - Quit <Ctrl>-Q
  - Print <Ctrl>-P
- Edit
  - Delete
    - Current Design
    - Numerator
      - First Order System
      - Second Order System
    - Denominator
      - First Order System
      - Second Order System
    - Integrator
- Zoom
  - Up <Ctrl>-U
  - Down <Ctrl>-D
- Design
  - Numerator
    - First Order System <Ctrl>-1
    - Second Order System <Ctrl>-2
  - Denominator
    - First Order System <Ctrl>-3
    - Second Order System <Ctrl>-4
  - Low Frequency Gain <Ctrl>-G
  - Integrator <Ctrl>-I

There is also a tabular of the current compensator shown in the figure.

See Also

bndpl

FRQD2SPEC

Purpose

Define specification in frequency domain.

Synopsis

frqd2spec(spec)
frqd2spec

Description

frqd2spec creates a new figure with a set of menus and control buttons. There is a popup menu to select whether the upper or lower bound should be changed. The bounds can be described with two second order models. For the lower bound, ζ is fixed to one, only ω has to be defined. The menu bar contains:

- File
  - Quit <Ctrl-Q>
  - Print <Ctrl-P>
  - Save Specifications <Ctrl-S>
  - Grid on/off <Ctrl-G>
  - Change Specifications
    - End Part <Ctrl-E>
    - Middle Part <Ctrl-M>
    - New Specification <Ctrl-N>

It is possible to change a line by moving the pointer to the line to be changed and press one of the mouse buttons. The new points are the pointer locations and new points are registered as long as a mouse button is pressed and moved around. It is possible to change end parts or middle parts of a line. This is chosen in the menu.

See Also

getspec

GET1BND

Purpose

Returns one bound from the bounds storage class.

Synopsis

onebnd = get1bnd(bnd,omega)

Description

get1bnd returns the bound for *omega* from *bnd*. *onebnd* has the following structure

$$onebnd = \begin{pmatrix} mag_1 & phase_1 \\ \vdots & \vdots \\ mag_n & phase_n \end{pmatrix}$$

See Also

get1tmpl

GET1TMPL

Purpose

Returns one template from the template storage class.

Synopsis

onetmpl = get1tmpl(tmpl,omega)

Description

get1tmpl returns the template for *omega* from *tmpl*. *onetmpl* has the following structure

$$onetmpl = \begin{pmatrix} mag_1 & phase_1 \\ \vdots & \vdots \\ mag_n & phase_n \end{pmatrix}$$

See Also

get1tmpl

GETCOMPENSATOR

Purpose

Returns the compensator filter design.

Synopsis

[num,den] = getcompensator

Description

After finishing the design the compensator is stored in the root windows user data. *getcompensator* returns the stored values from root windows user data. Note that after save is selected in the compensator design tool the user should use *getcompensator* directly before another save is used. This is done in order not to mix data.

GETOMEGA

Purpose

Returns frequencies in a vector for which templates or bounds exists.

Synopsis

[omega,ind] = getomega(x);

Description

getomega finds the frequencies in the template or bounds storage class. The index is also an output argument.

GETPREFILTER

Purpose

Returns the prefilter design.

Synopsis

[num,den] = getprefilter

Description

After finishing the design the prefilter is stored in the root windows user data. *getprefilter* returns the prefilter from root windows user data. Note that after save is selected in the prefilter design tool the user should use *getprefilter* directly before another save is used. This is done in order not to mix data.

GETSPEC

Purpose

Returns the specifications.

Synopsis

spec = getspec

Description

After the specifications are defined the specifications are stored in the root windows user data. *getspec* returns the specifications from root windows user data. Note that after save is selected in specification defining tool the user should use *getspec* directly before another save is used. This is done in order not to mix data.

ISBND

Purpose

Check for bounds matrix storage class.

Synopsis

flag = *isbnd(x)*

Description

isbnd(x) is 1 if the storage class of *x* is bounds type and 0 otherwise.

Examples

```
if isbnd(x)
  bndpl(x);
end;
```

See Also

tmpl2bnd, tmpl2stbnd, bndpl

ISTMPL

Purpose

Check for template matrix storage class.

Synopsis

flag = *istmpl(x)*

Description

istmpl(x) is 1 if the storage class of x is template type and 0 otherwise.

Examples

```
if istmpl(x)
  tmplpl(x);
end;
```

See Also

tf2tmpl, tmplpl

NICHLPL

Purpose

Plot a graph with Nichols M-contours.

Synopsis

nichlpl

Description

nichlpl plots the Nichols chart with a M-contour grid. M-circles are the descriptions of feedback.

$$\left|\frac{L}{1+L}\right| = const.$$

PREFILTER

Purpose

Tool for prefilter design.

Synopsis

prefilter(Cnum,Cden,nummatrix,denmatrix,specmatrix)

Description

prefilter displays a Matlab figure with a menu bar at the top of the figure. It is possible to add and delete zeroes to the numerator polynominal and the denominator polynominal. After finishing the design use save in the menu to save the prefilter in the root windows user data. See *getprefilter*. The menu contains:

- File
  - Quit <Ctrl>-Q
  - Print <Ctrl>-P
- Edit
  - Delete
    - Current Design
    - Numerator
      - First Order System
      - Second Order System
    - Denominator
      - First Order System
      - Second Order System
- Design
  - Numerator
    - First Order System <Ctrl>-1
    - Second Order System <Ctrl>-2
  - Denominator
    - First Order System <Ctrl>-3
    - Second Order System <Ctrl>-4

There is also a tabular of the current prefilter shown in the figure.

See Also

getprefilter

TF2TMPL

Purpose

Calculates templates for an uncertain transfer function.

Synopsis

tmpl = tf2tmpl(nummatrix,denmatrix,omegavect)

Description

tf2tmpl computes templates from a transfer function with a set of parameters. Template is a set of the possible values [phase,magnitude] for the specified frequency.

nummatrix contains the possible values for the numerator.

$$nummatrix = \begin{pmatrix} nominal\ values \\ maximum\ values \\ minimum\ values \end{pmatrix}$$

denmatrix contains the possible values for the denominator.

$$denmatrix = \begin{pmatrix} nominal\ values \\ maximum\ values \\ minimum\ values \end{pmatrix}$$

omegavect contains the frequencies for which templates will be calculated.

Algorithm

Calculates subtemplates for numerator and denominator (template in complex form). Next step is to calculate ϕ_{max} and ϕ_{min}, where $\phi_{max} = \angle_{max}(num) - \angle_{min}(den)$ and $\phi_{min} = \angle_{min}(num) - \angle_{max}(den)$. Define a set $\Phi = [\phi_1, \phi_n]$ where $\phi_1 = \phi_{min}$ and $\phi_n = \phi_{max}$. Find the intersection of the sub-rectangle bounds with the line $\phi_{num} = \phi_{den} + \phi_i$, where $\phi_i \in \Phi$. Find max magnitude and min magnitude for numerator and denominator. Then proceed to next ϕ_i, $i = 1, \ldots, n$. For more details about the algorithm see reference below.

References

[1] F.N. Baliey, D. Panzer and G. Gu.
Two algorithms for frequency domain design of robust control system.
Int. J. Control, 1988, vol. 48, no. 5, 1787-1806.

TMPL2BND

Purpose

Calculates bounds on the loop gain, $L(j\omega)$.

Synopsis

bnd = tmpl2bnd(tmpl,spec)
bnd = tmpl2bnd(tmpl,spec,stabilityspec)

Description

tmpl2bnd(tmpl,spec) finds the contours which satisfy given performance specification in the frequency domain for each template. Stability bounds are also calculated for $\left|\frac{L}{1+L}\right| < 2$ $(6db)$, see *tmpl2stbnd*.
tmpl contains template for each frequency chosen by the user and it also contains the system response for a set of frequencies.
spec is the desired performance specifications:

$$spec = \begin{pmatrix} \omega_1 & \dots & \omega_n \\ mag_1^{max} & \dots & mag_n^{max} \\ mag_1^{min} & \dots & mag_n^{min} \end{pmatrix}$$

tmpl2bnd(tmpl,spec,stabilityspec) have one extra input argument, *stabilityspec* containing stability specification for each template:

$$stabilityspec = \begin{pmatrix} \omega_1 & \dots & \omega_n \\ mag_1 & \dots & mag_n \end{pmatrix}$$

Algorithm

Places each templates above -1 (in Nichols above [-180,1]) and uses bisections algorithm to find a magnitude so that the desired specification is fulfilled for the template and then moves the template to the next phase. First from -180 to 0 in increasing order and then from -180 to -360 in decreasing order. If the bisections algorithm fails then the template is placed under -1 and reverse binary search for lower bound is used. The stability bound is added to the performance bound.

See Also

tmpl2stbnd

TMPL2STBND

Purpose

Generates the contours of the stability bound.

Synopsis

[sbound,sbnd] = tmpl2stbnd(z,znom,stspec)
[sbound,sbnd] = tmpl2stbnd(tmpl,stspec)

Description

tmpl2stbnd(z,znom,stspec) computes the stability bound for one template in complex form. *z* is the template in complex form and *znom* the complex nominal value. *stspec* contains the requested specification for stability. Note that *tmpl2stbnd(z,znom,stspec)* is used by *tmpl2bnd*.
tmpl2stbnd(tmpl,stspec) can use all templates(*tmpl*) as argument and *stspec* contains desired stability specifications for each template.

$$stspec = \begin{pmatrix} \omega_1 & \dots & \omega_n \\ spec_{\omega 1} & \dots & spec_{\omega 2} \end{pmatrix}$$

The resulting bound is stored in two different data structures, *sbound* and *sbnd*. *tmpl2bnd* uses both structures, the actual structure which should be used by the user is *sbound* because it is the same as the standard structure.

$$sbound = \begin{pmatrix} mag_1^1 & phase_1^1 & \omega_1 \\ mag_2^1 & phase_2^1 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_1}^1 & phase_{n_1}^1 & NaN \\ mag_1^2 & phase_1^2 & \omega_2 \\ mag_2^2 & phase_2^2 & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_2}^2 & phase_{n_2}^2 & NaN \\ mag_1^n & phase_1^n & \omega_n \\ mag_2^n & phase_2^n & NaN \\ \vdots & \vdots & \vdots \\ mag_{n_n}^n & phase_{n_n}^n & NaN \\ n & NaN & NaN \end{pmatrix}$$

Algorithm

The main demand for stability is that $\left|\frac{L}{1+L}\right| < spec_i$ is fulfilled for every possible value of L (Loop gain in complex form) at ω_i. First map templates in to the complex plane and then find the loop gain, $L(j\omega)$, where the gain $|L(j\omega)| = spec$ for all phases. Take then the normalized template and divide all values of L with all values of the template and find the contour.

TMPLPL

Purpose

Visualization of templates in a Nichols chart.

Synopsis

tmplpl(tmpl)

Description

tmplpl(tmpl) plots templates in the Nichols chart. There is also a set of control buttons such as on/off buttons for each template and sliders for the frequency axis and the magnitude axis. There are also mouse button events connected to the figure. It is possible to move the pointer to a template and by pressing the mouse button and grab and move the template around. This makes it possible to check for possible places for the bounds on the loop gain. In the top of the figure there is a menu.

- File
  - Quit <Ctrl>-Q
  - Print <Ctrl>-P

See Also

bndpl, *nichlpl*