2. A BASIC SISO DESIGN EXAMPLE USING QSYN

In this chapter, a basic Single Input Single Output (SISO) design example will be solved using Qsyn. The user will be invited to try variations of the problem with respect to the definition of the plant, the specifications, and algorithms used for computing the templates and the bounds.

Example 2.1 A basic SISO design example

Given the uncertain plant, defined in real factored form with parametric uncertainty

$$P(s) = k \cdot \frac{s+a}{1+2\varsigma s/\omega_n + s^2/\omega_n^2}, \quad k \in [2,5],\ a \in [1,3],\ \varsigma \in [0.1, 0.6],\ \omega_n \in [4,8]. \qquad (2.1)$$

Close the loop with a feedback compensator $G(s)$, and a prefilter $F(s)$ such that the following specifications are satisfied:

$$\begin{cases} M \le 10\% & (2.2a) \\ t_s \le 15 \text{ seconds} & (2.2b) \\ S(j\omega) = \left| 1/(1+L(j\omega)) \right| \le 6 \text{ dB} & (2.2c) \end{cases}$$

where M stands for the reference step response overshoot, t_s is the reference step response time for convergence within 5 % of the output steady state value, $S(j\omega)$ is the sensitivity function (1.4), where $L(j\omega) = G(j\omega)P(j\omega)$ is the open loop frequency function whereby $P(j\omega)$ stands for any of the plant cases defined in (2.1). Moreover it is demanded that the reference step response steady state error be zero, which implies that one integrator must be included in $G(s)$ since there is none in $P(s)$, and that the complementary sensitivity function $L(s)/(1+L(s))$ have minimum bandwidth, in order to attenuate the effect of sensor noise and to reduce the effect of unmodelled plant dynamics (see Theorem 1.2).

●

2.1 Plant description

In Qsyn, the easiest way to describe the plant is with an editable m-file. Issue the command

```
plnt ex2 1a
```

and a default editor (that you may choose as the Editor Preference in the Matlab Command Window Options menu)[1] is invoked with the file `ex2_1.m` if it exists, or otherwise with a standard plant file "template" (that is present as the file `plant.m` in the Qsyn library), see Figure 2.1. After editing, the file will be saved as `ex2 1a.m` in your default directory.

[1] In Unix systems, copy the file `plant.m` to `ex2_1.m` and use the editor of your choice.

```
function   [Par,w tpl,w nom,method,P num,P den, ...
            n_dif,Uns_Par] = Plant_name

%PLANT      model of plant description file to be copied and edited by the user
%
%       plant           % evaluate the output variables
%
%       plnt new plant  % If new plant.m does not exist in your current directory:
%                       % invokes default editor with this file as a model. The file
%                       % will be saved in your current directory under the name
%                       % new_plant.m
%                       % If new plant.m exists in your current directory:
%                       % invokes default editor with new plant.m.
%
%       edit plant      % invokes default editor. Do not forget to save the edited
%                       % file under a new name in your current directory
%-----------------------------------------------------------------

% Plant_name :  Plant description


% Definition of the parameters
% ====================
   Par = [
         'p1=[p1min,p1max,p1nom,number of cases]' , ...     % uncertain parameters
         'p2=[p2min,p2max,p2nom]', ...
         'p3=[p3min,p3max,p3nom]', ...
         'c1=[c1value]',...                          % constant parameters
         'c2=[c2value]',...
         ];

% Multiplicative unstructured uncertainty:
% uncertainty circle radius m(w), in [0,1), as a function of frequency w [rad/s]
% =============
  Uns Par=[];   % Uns Par is either
                % (1) empty   --> no unstructured uncertainty;
                % (2) one real number, m, in the range [0,1)
                %      --> m(w) = m for all w;
                % (3) a matrix with two rows and at least two columns,
                % where the upper row contains the frequencies
                %      w [rad/s], and the lower row m(w), in [0,1),
                %`the  unstructured uncertainties,
                %      --> m(w tpl) is computed by linear interpolation



% Definition of the frequency vectors [rad/sec]
% =====================================
  w_tpl=[ wmin ... wmax];     % Template frequency vector.

  w nom=[ wmin ... wmax];     % Nominal frequency vector.
                % w nom will automatically include all points in w tpl

% Definition of the template computation method
% ============================================
  method = 'rff [1,1]';
                % (1) grid_dist       = Grid.
                % (2) rgrid_dist      = Random Grid.
                % (3) adgrid dist     = Recursive Grid.
                % (4) aedgrid dist    = Recursive Edge Grid.
                % (5) rff dist        = Real Factored Form.

    % dist=[Max phase distance [deg], max magnitude distance [dB]]; maximum
    %   resolution of template computation, 2-norm, in the Nichols chart for
    %   adgrid, aedgrid, rff methods
    % grid, rgrid: resolution is given by parameter grid, while dist gives the
    %    Unstructured Uncertainty resolution only, if Uns Uncertainty is present.
    %In RFF, the Max phase distance must be chosen such that 360 is a multiple integer
    %    of dist(1). Example: dist(1) = 1,2,5,8,0.8,4/3,etc are OK but not 7,0.33, etc
    % WARNING: IF RFF STRUCTURE IS USED BELOW, THEN method MUST BE 'rff', EVEN
    %    THOUGH OTHER METHODS MAY BE REQUESTED IN THE ctpl COMMAND.

% Plant definition
% ================

% Polynomial Structure
% ================
  P num ='p1..';              % numerator
  P den='(s)*(s+p2)*((s^2)/(wn^2)+2*zet*s/wn + 1)*(s+c1)..';        % denominator
    % PLEASE INCLUDE THE CERTAIN INTEGRATORS IN P den


%%%%%%%     The description of plant.m continues on the next page    %%%%
```

```
%%%%%%%     Cont'd form the previous page    %%%%

% Real Factored Form Structure
% ============================
% WARNING: AT LEAST ONE UNCERTAIN PARAMETER MUST BE PRESENT IN THE PLANT DEFINITION.
% NUMERATOR AND DENOMINATOR MUST EACH HOLD >= ONE PARAMETER (CERTAIN OR UNCERTAIN)
% IF THE PLANT IS CERTAIN, DEFINE E.G. AN UNCERTAIN GAIN 'k=[1,1+eps,1,1]' IN Par

  P num='(gain,p1)(delay,p3)..';
  P den='(hf,p2)[1 c1 0]..';
                %Certain polynomial factors: [  ] with Matlab syntax, incl integrators
                % Uncertain rff factors: (  )
                % (gain,k)=k    (dc,a)=(1+s/a)      (hf,a)=(s+a)
                % (delay,tau)=exp(-s*tau)
                % (dc,wn,zet)=(1 + 2*zet*s/wn + s^2/wn^2)
                % (hf,wn,zet)=(s^2 + 2*zet*wn*s + wn^2)
                % Note that the uncertainty in each factor will be treated as
                % independent even if the same parameter is used.
                % PLEASE INCLUDE THE CERTAIN INTEGRATORS IN P den AS [1 0], etc

% number of additional uncertain differentiators
% ======================
  n dif=[0 0]; % n dif = []; or n dif not given  <==> n dif=0 ,
               %   the nominal case is 0, and
               %   there are no uncertain differentiators in the templates.
               % In all other cases, n dif has at least two elements. The
               %   last element of n dif denotes the number of differentiators
               %   in the nominal case which may be outside the templates.
               %   The other elements of n_dif denote the uncertain differentiator
               %   cases of the templates.
               % Examples:
               % n dif = [d1 d2 d3];
               %       the nominal case has d3 differentiators, while a template is
               %       the union of T*s^d2 and T*s^d3, where T is the template without
               %       uncertain differentiators defined above.
               % n dif = [-2 -2]
               %       the nominal case has 2 integrators, while the templates
               %       also have two integrators, so indeed we have a case of a
               %       certain number of integrators, that could have been treated
               %       in P den above, with n dif = [0 0];

               % Note: negative numbers denote integrators
```

Figure 2.1. Plant description template invoked when issuing the command `plnt ex2_1a` *when the file* `ex2_1a.m` *does not exist previously.*

2.1.1 Plant description file head

The plant description file is a Matlab function file that is called by other Qsyn functions, e.g. the template computation function `ctpl`. The left hand side of the head of `ex2_1a.m` must not be changed by the user. The user may, however, freely change the name of plant definition file, the right hand side of the head (which does not necessarily have to be equal to the file name), and the values of the parameters defined in the file, following the rules given here and in the file. Comments may be inserted freely, thus enabling a complete, readable description of the plant. Let us go through the plant description line by line, ending up with an edited file ex2_1a.m that describes the plant (2.1) together with some design choices.

The line "`%  Plant_name :  Design description`" is a good place to put the plant description comments, see Figure 2.2.

2.1.2 Definition of the parameters

The vector `Par` defines the uncertain and certain parameters, in *exactly* the way given. All parameters defined in `Par` must be used in the plant transfer function (numerator or denominator) below. All *uncertain* parameters to be used in the plant transfer function must be defined in `Par`, while constant paramaters need not be defined in `Par`, see below. Of course, the *names* of the parameters may be chosen by the user, i.e. `p1, p2, ..., c1, ...` can be exchanged for other names. `p1min, p1max,` and `p1nom` stand for the minimum, maximum, and nominal values, respectively, of the parameter `p1`.

For the grid or random grid template computation methods (see below), the positive integer `number of cases` must be assigned in order to define to how many (equidistant) cases between (and including) `p1min` and `p1max` are to be considered. It is advisable always to assign `number of cases`, even if it has no significance for some template computation methods. The uncertain parameters of (2.1) are defined e.g. with the following names. Notice that there are no constant parameters.

```
Par = [               %'p1=[p1min,p1max,p1nom,number of cases]'
       'k=[2, 5, 2, 8]' , ...          % uncertain gain
       'a=[1, 3, 3, 8]', ...           % uncertian zero: s+a
       'z=[0.3, 0.6, 0.6, 8]', ...     % uncertain complex pole
       'wn=[4, 8, 4, 8]',...
       ];
```

Somewhat arbitrarily we selected to partition each parameter into 8 cases, thus having 8^4 plant cases to compute, if the grid method would be selected for the template computation. The user is invited to change the number of cases. Notice that we use comments liberally to explain the significance of the parameters.

Since there is no multiplicative unstructured uncertainty in Example 2.1 we leave its parameter empty (`Uns Par=[];`) and delete the comments.

2.1.3 Definition of the frequency vectors

The user defines two frequency vectors [rad/s], the template frequency vector, `w_tpl`, and the nominal frequency vector, `w nom`. As mentioned in Chapter 1, `w tpl` has to be chosen "wisely". It is wise to include frequencies over all the relevant frequency range, frequencies at and near plant resonances, and frequencies at and around the bandwidths of the different specifications. The penalty of including extra frequencies is a longer time to compute the plant templates and Horowitz bounds which may prove to be unnecessary to display. Note that Qsyn allows for the computation of templates and Horowitz bounds for additional frequencies, and their inclusion in already existing templates and bounds files.

Often it is sufficient to compute templates and Horowitz bounds for 8 to 10 frequencies, only. Therefore, to get a quick cut at a preliminary design, and estimate the difficulty of the design problem, we recommend to start with template frequency vector `w_tpl` of 8 to 10 frequencies, and complement the list later, if necessary.

The nominal frequency vector, , for which the nominal plant, open loop, and closed loops are computed should be sufficiently dense over the relevant frequency range such that the Bode and Nichols curves are "smooth". Twenty to forty logarithmically spaced points per decade is suitable.

In Example 2.1, the specifications seem indicate a bandwidth of not more than about 10 rad/s while the plant resonance resides between 4 and 8 rad/s. The relevant frequency range seems to be beween 0.1 and 100 rad/s. We therefore choose, somewhat arbitrarily,

```
w_tpl = [0.2 0.5 1 2 5 10 20 50]; %template frequencies [rad/s]
w_nom = logspace(-1,2);           %nominal frequencies  [rad/s]
```

2.1.4 Selection of the template computation method

Qsyn offers several template computation methods, listed in Figure 2.1, and briefly described here. Further examples of their use are found in Chapter 3.

Grid Method

The *Grid method* is the most general method. Each uncertain parameter is equidistantly partitioned between its minimum and maximum value in a user assigned number of cases, see subsection 2.1.2. All possible parameter combinations form the plant uncertainty set

(1.2) from which the templates for are easily computed. With a sufficiently dense parameter grid, an arbitrarily good approximation of the true template may be achieved at the forbidding "curse of dimensionality" price of an immense number of frequency function evaluations.

In some texts on QFT such as Horowitz (1993) or Yaniv and Horowitz (1987) it is claimed that it suffices to partition each parameter range in a few values only. Such a grid method is also proposed in some other QFT programs. It is however easy to show simple cases when a sparse parameter grid yields a severe underestimate of the true template, see Bailey (1987) or Gutman *et al* (1994). We therefore want to specifically *warn* for the use of a sparse parameter grid without a meticulous check that the computed templates are satisfactory.

In fact, it is wise to compute the templates by *different* methods. To alleviate the curse of dimensionality, it is also wise to *decompose* the computation of a template into parameter independent parts (Ackermann, 1992), using seperate plant description files, and then combine the partial templates with the Qsyn command `tplfop`. Such a decomposition is automatic in the Qsyn Real Factored Form method, see below.

Random Grid Method

The *Random Grid method* differs from the Grid method in that each parameter range is partitioned randomly. We recommend that this method is always used as a complement to the other grid methods.

Recursive Grid Method

The *Recursive Grid method* is described in Cohen *et al* (1995). The parameter gridding is adapted recursively during the computation in such a way that a freshly computed template point lies within a given Nichols chart 2-norm distance `dist = [deg, dB]` from some previously computed template point in the same subgrid. When the template is large relative to `dist`, or the number of uncertain parameters is large, the Recursive Grid method may become forbiddingly slow.

Recursive Edge Grid Method

With *Recursive Edge Grid method* the template points are computed along the edges of the hyberbox in the parameter space: All parameters except one are kept at an extreme value and the remaining one is recurively partitioned in such a way that the computed template points lie within a given Nichols chart 2-norm distance `dist = [dB, deg]` from its nearest neighbour from the same edge. The Recursive Edge Grid method is guaranteed to give a correct approximation of the true template only when the Edge Theorem holds (Ackermann, 1992), and unfortunately it is not always easy to show that it does hold for a given uncertain transfer function. It is easy to give examples when the Recursive Edge Grid method underestimates the true template, see e.g. Ackermann (1992) or Gutman *et al* (1994).

RFF Method

For uncertain transfer functions in *Real Factored Form* (Gutman *et al* 1994),

$$P(s)=\frac{ke^{-\tau s}\prod_l(1+s/b_l)\prod_m(s+b_m)\prod_q\left(1+2\zeta_q s/\omega_q+s^2/\omega_q^2\right)\prod_r\left(s^2+2\zeta_r\omega_r s+\omega_r^2\right)}{s^n\prod_u(1+s/a_u)\prod_v(s+a_v)\prod_w\left(1+2\zeta_w s/\omega_w+s^2/\omega_w^2\right)\prod_i\left(s^2+2\zeta_i\omega_i s+\omega_i^2\right)}\left(1+M(s)\right) \tag{2.3}$$

where $k,\tau,b_l,b_m,\zeta_q,\omega_q,\zeta_r,\omega_r,n,a_u,a_v,\zeta_w,\omega_w,\zeta_i,$ and ω_i are uncertain parameters, and $M(s)$ denotes the multiplicative unstructured uncertainty. First and second order factors

whose gain equals 1 for *s*=0 are said to be given in *direct current* or *dc-form*, and the remaining first and second order factors are said to be in *high frequency* or *hf-form*.

In Gutman *et al* (1994) it is described how the real factored form makes it possible to compute, in a very fast way, approximate templates with an arbitrary small phase and gain extent error. In Qsyn, the method is implemented under the name *rff*. The goodness of approximation is controlled by the user selected distance `dist = [deg, dB]`. Note that for the rff-method, `deg` is the phase resolution of the template computation, as well as the maximum *phase extent error* contribution of *each factor* (except the gain and integrator factors whose phase error is zero). This means that for *m* error contributing factors, the phase extent error of the computed template will not exceed *m* `deg`. The gain extent error depends in a complicated way on the phase extent error, with a smaller phase extent error in general leading to a smaller gain extent error. Note that the second element of `dist` does *not* signify the gain error. Instead, when two neighbouring rff-computed points have a 2-norm distance larger than `[deg, dB]`, then points inbetween are inserted by linearly interpolation in the Nichols chart to get that maximum distance. This is in particular for template points along the constant maximum phase and minimum phase template border segments.

A clever way to exploit the computational swiftness of the rff-method, but still have good accuracy and not too many template points burdening the Horowitz bound computation, is to compute the templates with a very high phase accuracy, say `method = 'rff [0.1,1]';` followed by a reduction of of the number of points using the Qsyn command `tplfop` with the `tplreduc` option. An basic example is given in Section 2.2.3 below.

Since, in Example 2.1, the plant (2.1) is given in real factored form, we chose the rff-method, with `dist = [1, 1]`, which means that the total phase extent error will not exceed 2 degrees, since (2.1) contains two phase error contributing factors. The user is invited to test other values for `dist`.

```
method = 'rff_[1,1]';
```

This selection may be overridden when computing the templates with the command `ctpl`, see Section 2.2. Note however the restriction concerning the plant transfer function description in subsection 2.1.5.

2.1.5 Definition of the plant transfer function

The real factored form structure can be used for the rff template computation method, as well as for the other template computation methods. The polynomial structure is used exclusively for the other template computations methods, and not for rff. The rff structure must be used exactly as stated, and the uncertainty of each factor is independent of the other factors.

In the polynomial structure arbitrary order polynomial factors may be defined as well as e.g. uncertain delay in the form $\exp(-\tau s)$. The coefficients may be functions of the uncertain parameters. Dependent uncertainty is allowed in the sense that the polynomial coefficients may depend on the same uncertain parameters. Note that each non-gain factor, including `(s)` must be surrounded by brackets.

For Example 2.1, the plant descriptions are then, respectively, whereby we will use only the rff structure.

```
P_num='(gain,k)(hf,a)';        % rff structure
P_den='(dc,wn,z)';

P num='k*(s+a)';               % polynomial structure,
P den='((s^2)/(wn^2)+2*z*s/wn + 1)';
```

```
function   [Par,w tpl,w nom,method,P num,P den, ...
            n dif,Uns Par] = ex2 1a

%  Plant name :  Example 2.1. Rff method

% Definition of the parameters
% ============================

 Par = [                              %'p1=[p1min,p1max,p1nom,number of cases]'
         'k=[2, 5, 2, 8]' , ...                      % uncertain gain
         'a=[1, 3, 3, 8]', ...                       % zero: s+a
         'z=[0.3, 0.6, 0.6, 8]', ...                 % complex pole
         'wn=[4, 8, 4, 8]',...
         ];

% Multiplicative unstructured uncertainty:

Uns Par=[];

% Definition of the frequency vectors [rad/sec]
% =============================================

w tpl = [0.2 0.5 1 2 5 10 20 50];   %template frequencies [rad/s]
w nom = logspace(-1,2);             %nominal frequencies  [rad/s]

% Definition of the template computation method
% =============================================

method = 'rff [1,1]';

% Plant definition: Polynomial Structure
% ======================================
%      % not used in this example, for information only  %
%      P num='k*(s+a)';               % polynomial structure,
%      P_den='((s^2)/(wn^2)+2*z*s/wn + 1)'; %

% Plant definition: Real Factored Form Structure
% ==============================================

P num='(gain,k)(hf,a)';             % rff structure
P den='(dc,wn,z)';

% number of differentiators/integrators
% =====================================

n_dif = [0 0];
```

Figure 2.2. Plant description file `ex2_1a.m` *for Example 2.1.*

Note that if the plant has a fixed number of integrators or differentiators, they are included as factors in the plant description as e.g. `'(s)'` and `'[1 0]'` in the denominators of the polynomial and rff structures, respectively, for one certtain integrator.

In the rare case when the plant is described by an uncertain number differentiators or integrators, Qsyn offers the optional parameter `n_dif`. Note that the last element of `n_dif` denotes the nominal case which is *not* included in the template, unless it is repeated among the preceding elements. Since Example 2.1 contains no integrators, we set

```
n dif = [0 0];          %no uncertain plant integrators
```

2.1.6 Plant description files

As a summary we have a compact plant description files for Example 2.1, `ex2 1a.m` in Figure 2.2, for which all template computation methods can be used. Examples where the plant transfer function cannot be written in rff form (and hence the rff method cannot be used) are given in Chapter 3.

Before computing the templates we will display the Bode diagrams for a number of plant cases. All plant cases are plotted (after a lengthy computation) as Figure 2.3 with the command whose syntax is explained in the Qsyn Reference Guide.

```
T=cases('ex2 1a','all',[],1);
```

The nominal and two other plant cases are shown in Figure 2.4 as a result of the command

```
cases('ex2 1a',[2 5; 1 3; .3 .3; 8 4],[],1);
```

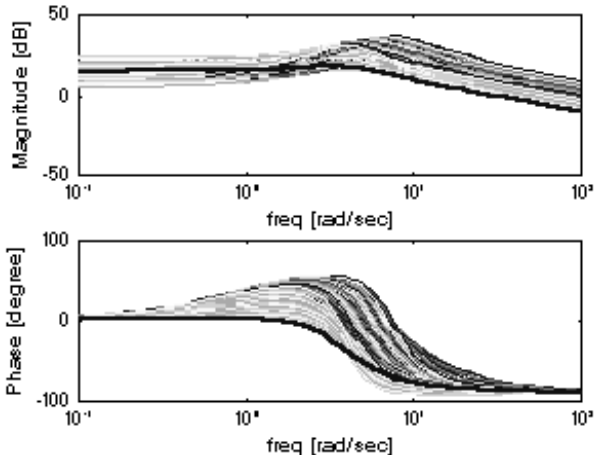

Figure 2.3. Bode diagram of all plant cases defined in `ex2_1a.m`*. Nominal is black.*

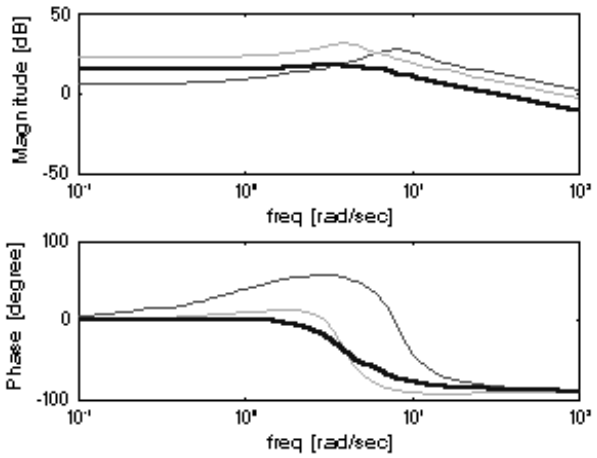

Figure 2.4. Bode diagram of two plant cases defined in `ex2_1a.m`*. Nominal is black.*

We notice a considerable spread between the plant cases which seems to make the design problem non-trivial. However, with reference to the discussion surrounding equations (1.11) and (1.12), we also notice that if the logarithmic difference between the maximum and minimum plant gains were less than the tolerance specification emanating from (2.2a,b), see Example 1.1 and (1.10), no feedback would be needed since without feedback the transfer function from output disturbance to output, $Y(s)/D_2(s)$ = 1, and hence (2.2c) would be satisfied, see Figure 1.1, (1.4) and (1.5).

2.2 Template computation

In Section 2.1 the available template computation methods were briefly described. In this section the templates for Example 2.1 will be computed. The result of some of the methods will be compared, and the user may try other parameters and methods.

With a plant definition file given (Section 2.1), the plant templates are computed with the Qsyn command `ctpl` which includes many optional parameters.

In Qsyn, templates from a `ctpl`-computation are stored in a *template file*, with suffix `tpl`, which is a special Matlab mat-file. The tpl-file structure is elucidated in Chapter 3. The contents of a tpl-file can be inspected with the command `look` or `getfrom`. A variable can be copied from a tpl-file using the command `getfrom`, and removed with `remove`. A variable can be inserted into or replaced in a tpl-file with the command `insert`. With `tpl2mat` a template file is converted to a Matlab matrix.The command `mat2tpl` converts a Matlab matrix to a tpl-file, an operation that is useful when the template is defined by a set of measured transfer functions, and made easier for the user with the command `mffd`. The templates in a tpl-file are displayed in a Nichols chart with `showtpl`.

The syntax of all commands, with examples, are found in the Qsyn Reference Guide. The same information is displayed when typing `help` and the command name, e.g.

```
help ctpl
```

2.2.1 Templates computed with the Real Factored Form Method

The basic and simplest way to use `ctpl` is to simply invoke it with the plant description file:

```
ctpl('ex2 1a')
```

whereby all command parameters are set by default from `ex2 1a.m`, and the resulting template file stored as `ex2_1a.tpl`. While computing the following messages appear in the Matlab command window:

```
Calculating templates using the Real Factored Form method
--> for w=0.2 [rad/sec]
--> for w=0.5 [rad/sec]
--> for w=1 [rad/sec]
--> for w=2 [rad/sec]
--> for w=5 [rad/sec]
--> for w=10 [rad/sec]
--> for w=20 [rad/sec]
--> for w=50 [rad/sec]
 Computing time  :  [min]  =  0.4752
```

As shown in Figure 2.5, The templates can be inspected graphically by issuing the command

```
showtpl('ex2 1a');
```

The options of `showtpl` can be exploited to study individual templates, and the command `hzoom` can be used to zoom in on indivudual templates.

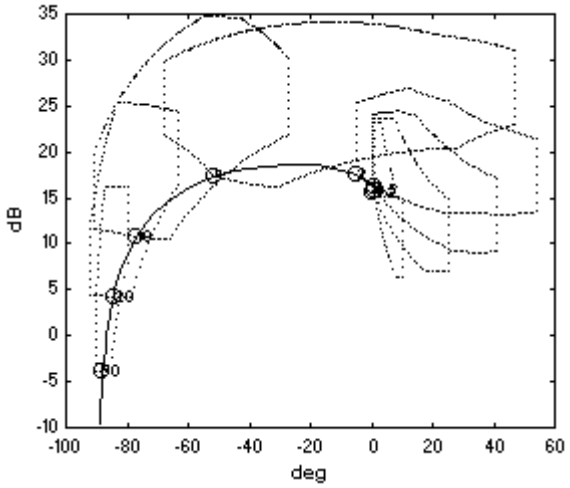

Figure 2.5. Nichols chart with templates in file `ex2_1a.tpl` *around nominal (solid line).*

2.2.2 Templates computed with the edge grid and grid methods

We wish to stress that it is always advisable to compute the templates with different methods, if possible. For Example 2.1, all methods can be used, and we will show the result of the Recursive Edge Grid Method and the regular Grid method. The results are placed in the template files `ex2 1b.tpl` and `ex2 1c.tpl`, respectively.

```
ctpl('ex2 1b','ex2 1a','aedgrid [1,1]');

Calculating Template using the Recursive Edge grid method
...
 Computing time  :  [min]  =  27.584


ctpl('ex2 1c','ex2 1a','grid',[]);

Calculating templates using the grid method
...
 Computing time  :  [min]  =  6.128
```

The templates for 5 rad/s are compared in Figure 2.6 which was produced by the following commands with the exception that Figure 2.6 shows all edges computed by the recursive edge grid method before pruning (see Section 2.2.3) while `aedgrid` prunes by default,

```
showtpl('ex2 1a',5,'nom');            % rff
hold on
showtpl('ex2 1b',5,'point',[],gcf); % rec edge grid placed by mouse
                                      % click
showtpl('ex2 1c',5,'nom',[],gcf);     % grid
```

For plant (2.1) the Edge Theorem holds, and the outer edge of the Recursive Edge Grid method is exact. A careful study reveals the the RFF template edge has an error of at most 0.2 degrees and 0.1 dB. It is clearly seen in Figure 2.6 that the Grid template with the chosen grid misses significant portions of the template.

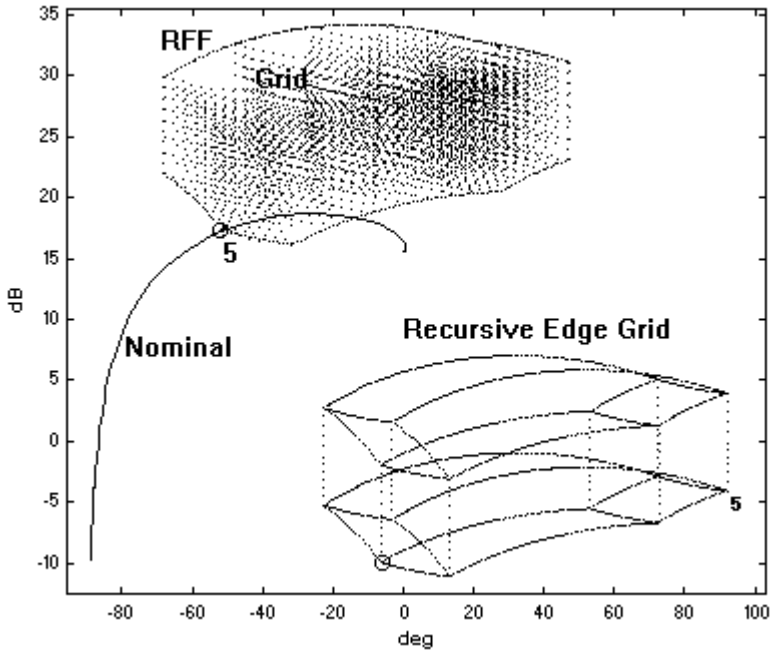

Figure 2.6. A comparison between three template computation methods for the template at 5 rad/s of (2.1) in a Nichols chart. The nominal plant transfer function, defined in Figure 2.2, is marked "Nominal". The Recursive Edge Grid template (before pruning) is moved, but not rotated. The nominal points at 5 rad/s is marked by o.

2.2.3 Pruning and reduction

For the bound computation (Section 1.5, and Chapter 3), only the outer template edge counts, and interior template points only prolong the computation time. The operation to eliminate interior points is called *pruning* (Cohen, Nordin, and Gutman 1995), and is performed by the Qsyn command `prune` on individual templates, and with the command `tplprune` on one or more templates in a template file. Note that RFF templates need no pruning. We will prune the Recursive Edge Grid template and Grid template in Figure 2.6 and compare them in Figure 2.7. We notice that the edge of the grid template does not represent the true edge very well. The command sequence is as follows:

```
redge5=gettpl('ex2 1b',5);     % extract pruned rec  edge templ
grid5=gettpl('ex2 1c',5);      % extract grid template
Tgrid5=prune(grid5,[2 2]);     % prune grid template
plot(redge5,'r'),hold on,plot(Tgrid5,'g')
xlabel('deg'),ylabel('dB')
hzoom                          % plot and zoom: Fig 2.7
```

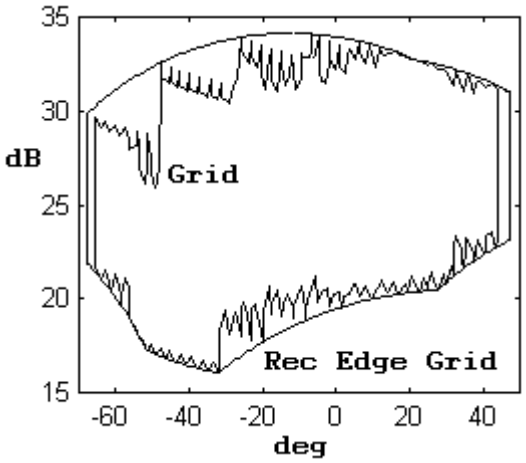

Figure 2.7. The template computed by the Grid method (jagged edge) and the template computed by the Recursive Edge Grid method (smooth edge) from Figure 2.6 after pruning.

A operation similar to pruning is *template reduction* by which an edge of an already pruned template, or rff template, is thinned out to a lower resolution by eliminating superflous template points. Thus the size of the template vector is reduced (while the template shape is still retained), causing the bound computation to be faster. This operation is performed on individual templates with the command `tplreduc` and on one or more templates in a template file with the command `tplfop` with the `tplreduc` option. Notice that `tplreduc` also adds template points by interpolation where the edge is not as dense as the desired resolution. We illustrate the operation on an rff-computed template, in Figure 2.8:

```
rff5=gettpl('ex2 1a',5); size(rff5)
 ans =
   709     1
redrff5=tplreduc(rff5,[10 2]); size(redrff5)
      TPLREDUC:size reduction ratio 91.11%
 ans =
     1   63
plot(rff5,'k.'),hold,plot(redrff5,'ko')   % Fig. 2.8
```

Notice that the resolution [10 2] is selected only to clearly show the effect of reduction in the scales of Figure 2.8; it may be too sparse to yield smooth Horowitz bounds. As usual, the user is encouraged to try her own options.

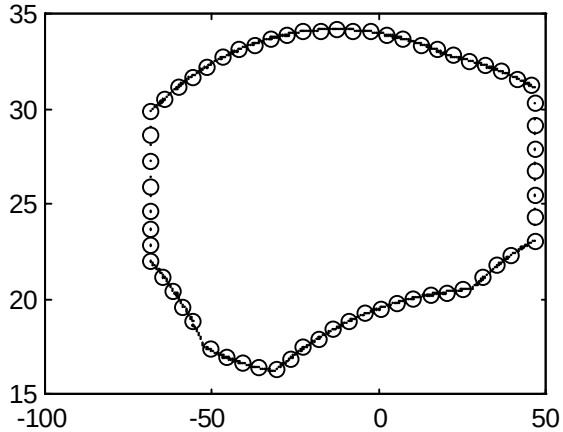

Figure 2.8. The template computed by the RFF method from Figure 2.6, as originally computed (.) and after reduction (o).

2.3 Specifications

In QFT specifications must be given in the frequency domain, as discussed in Section 1.3. In Example 2.1, (2.2a) and (2.2b) specify a servo specification in form of a reference step response in the time domain, relating to the closed loop transfer function $F(s)G(s)P(s)/(1+G(s)P(s))$. Hence (2.2a) and (2.2b) can be approximately translated to the frequency domain as in Example 1.1. Specification (2.2c) is already given in the frequency domain.

Qsyn contains three functions that aid the user to formulate the time domain specifications and translate them to the frequency domain: `rsrs` (reference step response specification) to specify the step response from *r* to *y* in Figure 1.1; `idsrs` (input disturbance step response specification) to specify the transmission from d_1 to *y* in Figure 1.1; and `odsrs` (output disturbance step response specification) to specify the transmission from d_2 to *y* in Figure 1.1.

In Qsyn, time domain and frequency domain specifications from a `rsrs`-, `idsrs`-, or `odsrs`-computation are stored in a *specification file*, with suffix `spc`, which is a special Matlab mat-file. The spc-file and variable structure is elucidated in Chapter 3. The contents of a spc-file can be inspected with the command `look` or `getfrom`. A variable can be copied from an spc-file using the command `getfrom`, and removed with `remove`. A variable can be inserted into or replaced in an spc-file with the command `insert`.

The specification variables in an spc-file are plotted with the command `showspc`, and graphically updated with `spcupd`.

The user is not restricted to the above mentioned specifications. The command `makespc` enables her to graphically define a frequency domain specification matrix. She may also produce her own frequency domain specification matrix in any way she wants, and then insert it (and its underlying time-domain specification, if it exists) into an spc-file with the command `insert`. Certain name and format conventions have to be adhered to, see Chapter 3.

2.3.1 Servo specification

As in Example 1.1, the command `rsrs` is used to translate (2.2a) and (2.2b) to a frequency domain servo specification (1.8) and a tolerance specification (1.10):

```
rsrs('ex2 1a',[],[1.2 0.2],10,1.5,[],logspace(-1,2),2.85,3.1);
```

where `'ex2 1a'` indicates that the specification should be stored in the file `ex2 1a.spc`; the next [] means that the standard specification variable names `rsrs t` and `rsrs w` should be used, `[1.2 0.2]` defines maximum and minimum rise times (90%), `10` stands for allowed overshoot, `1.5` denotes the maximum settling time, `logspace(-1,2)` gives the frequency range [rad/s], `2.85` rad/s is demanded cut-off frequency for the lower frequency domain specification in `rsrs w(:,3)`, and `3.1` is an instruction that both 2nd and 3rd order approximants of the closed loop transfer function are to be used in a way defined in Horowitz (1993), page 48. More about `rsrs` is found in the Qsyn Reference Guide, or by typing `help rsrs`. Note that all input arguments have default values.

During the computation, the command `rsrs` draws Figure 2.9, and includes the frequency domain envelopes $a(\omega)$ and $b(\omega)$, from which the tolerance specifiation $b(\omega)/a(\omega)$ is taken during the bound computation. Note that the cut-off frequency is not marked in Figure 2.9, but it is stored in `ex2 1a.spc`. The frequencies for which `rsrs w` is computed do not necessarily have to coincide with `w tpl` in the plant description file `ex2 1a.m` in Figure 2.2., since the bound computation algorithm will interpolate between the frequencies in the spc-file to produce tolerances for the requested template frequencies. The user may change

the command line parameters, the frequency list, and the order and number of the approximants and study the effect.

A look into the specification file `ex2 1a.spc` shows the following:

```
look('ex2 1a.spc')

Your variables are:

filename    rsrs_t      rsrs_tab    rsrs_w
```

The variable `rsrs_tab` contains the demanded time domain specification, `rsrs_t` contains the time domain specification envelope and `rsrs w` contains the frequency domain specification in a format revealed by:

```
[rsrst,rsrsw] = getfrom('ex2_1a.spc','rsrs_t','rsrs_w');
[rsrstab] = getfrom('ex2_1a.spc','rsrs_tab');
```

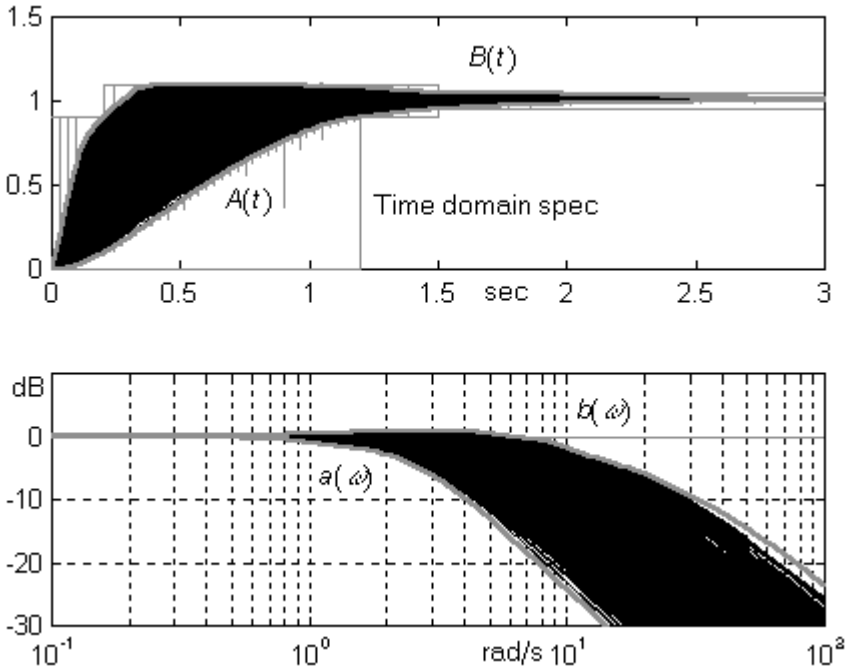

Figure 2.9. The graphical output during computation of the command

```
rsrs('ex2 1a',[],[1.2 0.2],10,1.5,[],[],2.85,3.1);
```

The heavy grey lines, $A(t)$, $B(t)$, and $a(\omega)$, $b(\omega)$, denote the time domain and frequency domain envelopes, respectively, of the tested 2nd and 3rd order cases. The envelopes constitute the resulting specifications and are saved in the variables `rsrs_t`, *and* `rsrs_w`, *respectively, in the specification file* `ex2_1a.spc`. *The specifications $a(\omega)$ and $b(\omega)$ are identified with (1.8) although $a(\omega)$ is saved in* `ex2_1a.spc` *to include the desired steep roll off. The tested cases are displayed as thin black lines within the envelopes. The thin grey lines outside the upper envelope in the time domain plot is a sketch of the user requested time domain specification of the* `rsrs` *input parameters. In addition there are some vertical grey lines in the time domain plot which denote simulated cases that did not meet the specifications and which can be neglected.*

The variable `rsrs_t` has 3 columns. With reference to Figure 2.9, the first (leftmost) column hold the time vector *t* [seconds], whose increment was defined by the user as an input argument in `rsrs`, or set by default; the second holds $B(t)$, and the third $A(t)$. `rsrs w` has 3 columns; the first holds the frequency vector ω [rad/s], which was set by the user in the `rsrs` command, or set by default; second $b(\omega)$ [dB] and the third $a(\omega)$ [dB]. The format of `rsrs tab` is the same as that of `rsrs t`.

Both time and frequency domain specifications in an existing spc-file are plotted with the command `showspc`:

```
showspc('ex2 1a','rsrs');      % Do it!
```

If only frequency domain specifications are desired for inspection, one proceeds like this

```
showspc('ex2 1a','rsrs','freq'); hzoom;   % and click to zoom
```

and we notice in the resulting Figure 2.10 that $a(\omega)$ includes the desired roll off. If one wishes to furher change the frequency domain specification, one may do it interactively using the command `spcupd`, with manual mouse operations in the figure window, .

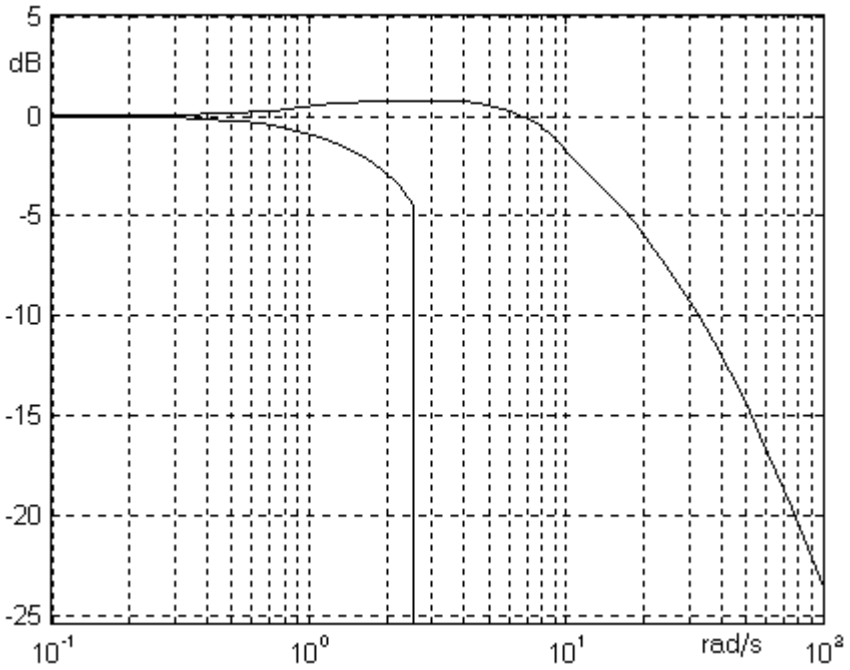

Figure 2.10. The frequency domain specification from Figure 2.9 as saved in `ex2_1a.spc`*, displayed by the command* `showspc('ex2 1a','rsrs','freq')`.

Notice that the frequency domain specification can be modified by extracting `rsrs_w` as a matrix using `getfrom`, assing new values to the relevant matrix elements, and insert the modified matrix into a specification file with the command `insert`. Alternatively, and better, the command `spcupd` enables you to graphically update the frequency domain specification. To modify `rsrs_w`, just issue the command

```
spcupd('test','rsrs');
```

and follow the instructions in the figure window, click the mouse, and update!

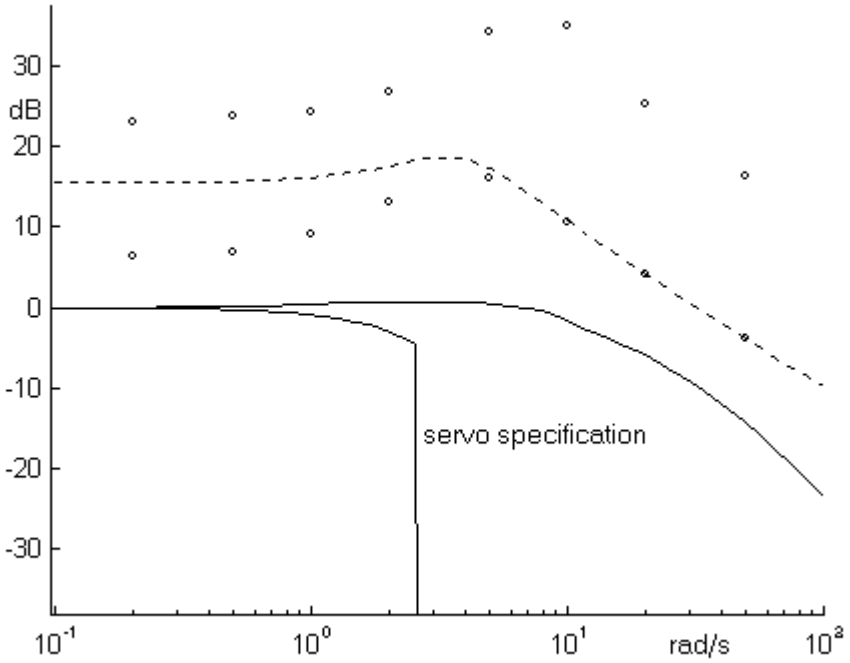

Figure 2.11. The dotted line is the plant nominal gain in `ex2_1a.tpl`*, the gain extents of the plant templates in* `ex2_1a.tpl` *are marked by* o*, and the servo specification is from* `rsrs_w` *in* `ex2_1b.spc`*.*

Before continuing the design process, it is instructive to compare the frequency domain specification in Figure 2.10, with its tolerance, with the plant gain uncertainty given e.g. in the template file `ex2 1a.tpl` (Figure 2.5). The following sequence of commands easily enables the comparison, with the zoomed result in Figure 2.11:

```
figure, fdesign('ex2 1a.tpl');      % show gain extent of templates
hold on, showspc('ex2_1a','rsrs','freq',[],gcf); % show spec
```

From Figure 2.11 it is clear that the gain extents of the templates are larger than the tolerance specification (see equation 1.10) for frequencies below the desired bandwidth of about 3 rad/s. Hence one cannot solve the control problem in open loop, with feedforward from the reference. Feedback is needed to reduce the closed loop uncertainty within the given tolerance for frequencies lower than 3 rad/s. For higher frequencies, feedback is not needed with respect to the servo specification.

2.3.2 Sensitivity specification

The easiest way to include the sensitivity specification (2.2c) into the specification file `ex2_1b.spc` is the following:

```
add2spc('ex2 1a','odsrs',logspace(-1,2),6); % create and insert
showspc('ex2 1a','odsrs','freq');           % show
```

where we note that the standard name for the sensitivity specification in the frequency domain is `odsrs_w` since it alse refers to the transmission from d_2 to y in Figure 1.1, for which the command `odsrs` creates an Output Disturbance Step Response Specification. Like in the specification variable `rsrs w`, the leftmost column of `odsrs w` contains the frequencies [rad/s]. Since in our case only an upper bound for the sensitivity is specified, the second column holds it [dB]. The plot, a straight line at 6 dB is omitted.

2.4 Bounds computation

In Section 1.4 and 1.5 the Horowitz bounds were described. In Qsyn there is one central command for the the bound compututation, called `cbnd`. The bounds are computed with respect to the nominal open loop, $L_{\text{nom}}(j\omega_k) = P_{\text{nom}}(j\omega_k)G(j\omega_k)$, where ω_k [rad/s] stands for the frequencies for which templates and specifications exist. As noted above, the specifications do not have to be defined precisely for the template frequencies since the bound computation algorithm interpolates among the specification frequencies. The user must however make sure that the range of specification frequencies covers the template frequencies since the bound computation algorithm does not extrapolate.

The bound computation in `cbnd` is straight-forward: the value of the specification is computed for all $L_{\text{nom}}(j\omega_k)$ in a user defined grid in a Nichols chart, and then those complex numbers $L_{\text{nom}}(j\omega_k)$ for which the specification is satisfied with equality (e.g. equation 1.5 or equation 1.10 with an equality sign) are saved in a *bounds file*, with suffix `bnd`, which is a special Matlab mat-file. The command `bndupd` enables the user to recalculate a bound for a finer grid in a selected area of the Nichols chart. An example is given in Chapter 5.

The bnd-file structure is elucidated in Chapter 3 The contents of a bnd-file can be inspected with the command `look` or `getfrom`. A variable can be copied from a bnd-file using the command `getfrom` or `getbnd`, and removed with `remove`. A variable can be inserted into or replaced in a bnd-file with the command `insert`. The bounds in a bnd-file are displayed in a Nichols chart with `showbnd`.

2.4.1 Tolerance bounds

The tolerance bounds (1.10) emanating from the variable `rsrs w` in the specifications file `ex2 1b.spc` (Figure 2.10) and the templates in the template file `ex2 1a.tpl` (Figure 2.5) are computed by

```
cbnd('ex2 1a','rsrs');
```

with plots of the bounds appearing in a figure window as they are computed. The bound marked by o denotes the strict *stability bound*, i.e. the bound outside which the compensated nominal open loop has to be so that no compensated open loop case intersects the instability point -1. The stability bound always belongs to the forbidden part of the complex, and hence will not be operative during the design phase. The stability bound is not saved in the bnd-file, but is shown only as a reference to the user. The following messages appear in the Matlab Command Window during the computation:

```
Calculating bounds for rsrs specification, with the templatefile
ex2_1a.tpl
w---> 0.2 [rad/s]
GETFROM: The file ex2 1a.bnd does not exist
INSERT: New datafile ex2 1a.bnd created
w---> 0.5 [rad/s]
w---> 1 [rad/s]
w---> 2 [rad/s]
w---> 5 [rad/s]
MAKEBND: Large problem, divided into 110 subproblems
No bound found, use larger search area, or higher accuracy
w---> 10 [rad/s]
No bound found, use larger search area, or higher accuracy
w---> 20 [rad/s]
No bound found, use larger search area, or higher accuracy
w---> 50 [rad/s]
No bound found, use larger search area, or higher accuracy
```

Not surprisingly are there no bounds above the default search limit of -50 dB for frequencies higher than 2 rad/s, since the specification gives a very large tolerance after the cut-off frequency of 3 rad/s, see Figure 2.11 and the last paragraph of Section 2.2.3. The tolerance bounds are displayed by the command

```
figure, hngrid, hold on, showbnd('ex2 1a',gcf,[],'rsrs');
```

The tolerance bounds should be interpreted such that the compensated nominal open loop frequency function, for each frequency, must be above its respective bound.

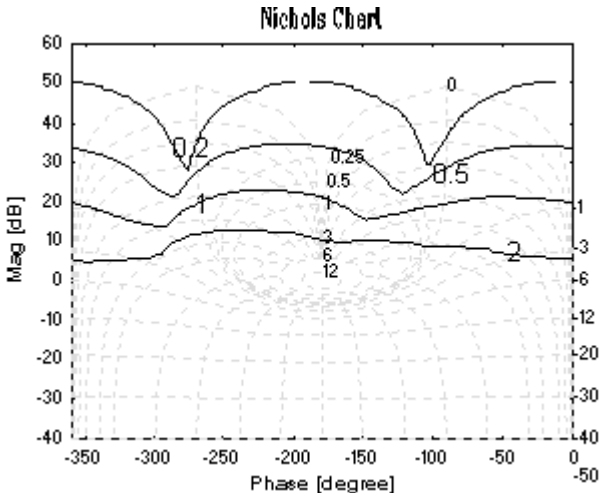

Figure 2.12. Tolerance bounds, $B_{TL}(\omega_k)$, for Example 2.1, computed by the command `cbnd('ex2_1a','rsrs');`. *The bounds are parametrized by the frequency ω_k.*

2.4.2 Sensitivity bounds

The sensitivity specification (2.2c) coded as `odsrs w` in the specification file `ex2 1b.spc` (Section 2.3.2) and the templates of the template file `ex2_1a.tpl` give the sensitivity bounds for Example 2.1,

```
cbnd('ex2 1a','odsrs');
```

with the following messages appearing in the command window during the computation,

```
Calculating bounds for odsrs specification, with the templatefile
ex2 1a.tpl
w---> 0.2 [rad/s]
w---> 0.5 [rad/s]
w---> 1 [rad/s]
w---> 2 [rad/s]
w---> 5 [rad/s]
MAKEBND: Large problem, divided into 110 subproblems
w---> 10 [rad/s]
w---> 20 [rad/s]
w---> 50 [rad/s]
```

During the computation of $B_{SL}(5)$, the sensitivity bound for 5 rad/s relating to the nominal open loop, the accompanying figure displays three curves; the true bound relative to the current specification, the stability bound, and a third curve interior to the stability bound. The third curve is a "dummy bound", emanating from the fact that the only the template edges

are used. It should be neglected, although it is stored in `ex2 1a.bnd`. It can be removed with the command `bndupd`. A look into `ex2 1a.bnd`, where all the bounds are saved,

```
look('ex2 1a.bnd')
```

reveals that your variables are:

```
ans         odsrs_4     odsrs_w     rsrs_5
filename    odsrs_5     rsrs_1      rsrs_6
odsrs_1     odsrs_6     rsrs_2      rsrs_7
odsrs_2     odsrs_7     rsrs_3      rsrs_8
odsrs_3     odsrs_8     rsrs_4      rsrs_w
```

The variables can be inspected with the command `getfrom` or `getbnd`. The first column of `rsrs_w` and `odsrs_w` hold the frequencies for which, respectively, the tolerance and sensitivity specifications have been computed. The second columns hold the indeces referring to the bound variables themselves, `rsrs_1, ..., rsrs_8` and `odsrs_1, ..., odsrs_8`, respectively. Actually the four last rsrs bound variables are "non-existent" as stated above in Section 2.4.1: the tolerance after the cut-off frequency of 3 rad/s is so large that the gain extent of the templates are smaller than the tolerance and the bounds would fall at $-\infty$ dB, i.e. outside the search area of the bound computation algorithm. Therefore, each of `rsrs_5`, `rsrs_6`, `rsrs_7`, and `rsrs_8` contain each a `NaN`, as revealed e.g. by the command

```
getbnd('ex2 1a','rsrs', 20)
  ans =
         NaN
```

Quite appropriately, the sensitivity bounds may be displayed in an inverse Nichols chart, i.e. a Nichols chart with the closed loop loci $|1/(1+L)|$ = constant, and $\arg(1/(1+L))$=constant. Figure 2.13 is generated by the following command:

```
figure, hngrid([],[],1),, hold on, showbnd('ex2 1a',gcf,[],'odsrs');
```

The sensitivy bounds are interpreted such that the compensated nominal open loop frequency function, for each frequency, must be outside its respective bound.

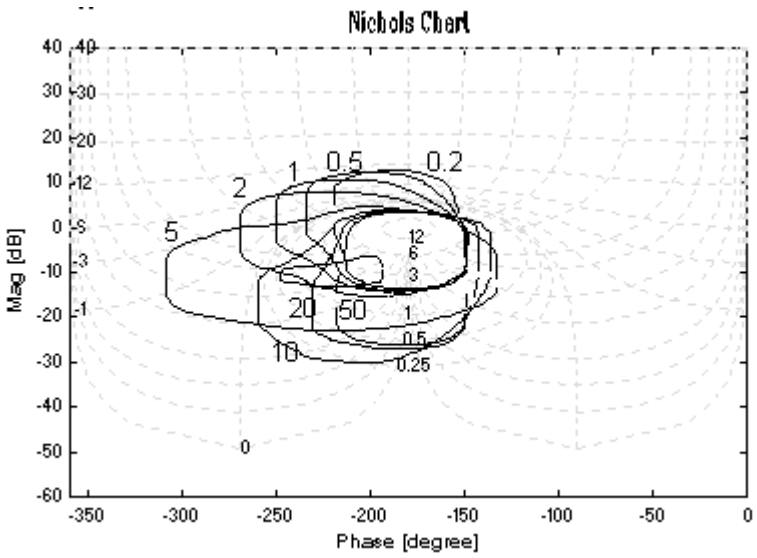

Figure 2.13. Sensitivity bounds, $B_{SL}(\omega_k)$, for Example 2.1, computed by the command `cbnd('ex2_1a','odsrs');`. *The bounds are parametrized by the frequency ω_k. Notice the interior dummy bound for 5 rad/s that should be neglected and can be removed with the help of the command* `bndupd`.

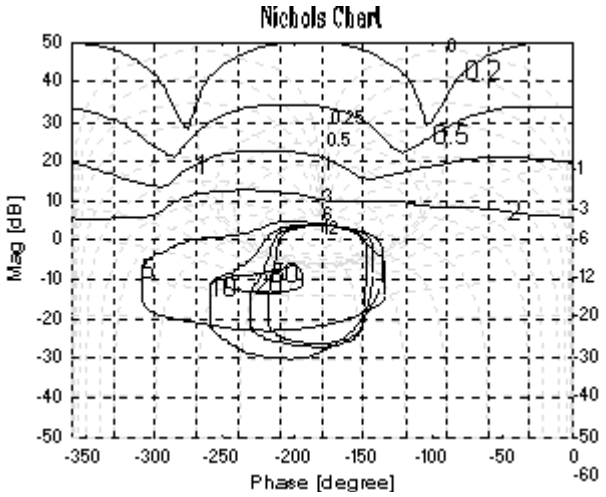

Figure 2.14. Dominant bounds, $B_L(\omega_k)$, for Example 2.1, selected by the user from the tolerance and sensitivity bounds in Figures 2.12 and 2.13, and generated by the command hngrid, hold on, `showbnd('ex2_1a',gcf,[.2 .5 1 2],'rsrs','r',[5 10 20 50],'odsrs','r');`*. Notice the interior dummy bound for 5 rad/s that should be neglected.*

2.4.3 Dominant bounds

A comparison between Figures 2.11 and 2.12 reveals that the tolerance bounds are dominant for 0.2, 0.5, 1, and 2 rad/s, while the sensitivity bounds are dominant for the frequencies 5, 10, 20, and 50 rad/s. We may then display, in Figure 2.14, a user chosen composite set of bounds which will form the Nichols chart on which we will design the nominal open loop.

```
clg,hold off, hngrid, axis([-360 0 -50 50]), mgrid(12,10), hold on
showbnd('ex2 1a',gcf,[0.2 0.5 1 2],'rsrs',[], [5 10 20 50],'odsrs')
```

For many problems, however, a composite bound for a particular frequency would consist of segments of bounds from different specifications. For clarity it is then often better to leave the original bounds. See the example in Chapter 5.

2.5 Feedback compensator design

Referring to Section 1.6, we are now in a position to design the feedback compensator $G(s)$, such that, for ω_k = 0.2, 0.5, 1, 2, 5, 10, 20, and 50 rad/s, the nominal open loop, $L_{\text{nom}}(j\omega_k) = P_{\text{nom}}(j\omega_k)G(j\omega_k)$, lies in the permitted side of the Horowitz bound $B_L(\omega_k)$ in Figure 2.14.

In Qsyn, the user defines her feedback compensator as an m-file with a certain structure. A fully commented "template" or "model" of a feedback compensator in Real Factored Form is found in the Qsyn library as `fbcomp.m` and is given in Figure 2.15 without its help text. In `fbcomp.m`, the transfer function $G(s)=1$ is realized.

The user should copy `fbcomp.m` into the work directory, change its file name, and edit it to reflect the desired feedback compensator. Since the loop shaping process is interactive and developmental, the user may keep a sequence of feedback compensator files in his work directory, e.g. named `g1.m`, `g2.m`, `g3.m`, ... etc, to trace (fundamental) changes of the compensator.

It should be noted that the user is allowed to define the frequency function in any form in his feedback compensator m-file, as long as elementwise vector operation notation is used, e.g. `.*`, `./`, `.^`, etc. In the help text of `fbcomp.m`, it is also suggested how a discrete time feedback compensator is defined.

```
function [G] = fbcomp(s)

% fbcomp.m     Qsyn feedback compensator model file
% ...
% (help text omitted, to see it, issue the command: help fbcomp)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User's comment: feedback compensator no. for plant
%
% DC-gain
% =======
  k = 1;

% Number of integrators
% =====================
  n = 0;

% Real Poles
% ==========
  p1 = -1/eps;         p2 = -1/eps;   p3 = -1/eps;   p4 = -1/eps;   p5 = -1/eps;

% Real Zeros
% ==========
  z1 = -1/eps;         z2 = -1/eps;   z3 = -1/eps;   z4 = -1/eps;   z5 = -1/eps;

% Complex Poles
% =============
  zp1 = 0;             zp2 = 0;       zp3 = 0;       zp4 = 0;       zp5 = 0;
  wp1 = 1/eps;         wp2 = 1/eps;   wp3 = 1/eps;   wp4 = 1/eps;   wp5 = 1/eps;

% Complex Zeros
% =============
  zz1 = 0;             zz2 = 0;       zz3 = 0;       zz4 = 0;       zz5 = 0;
  wz1 = 1/eps;         wz2 = 1/eps;   wz3 = 1/eps;   wz4 = 1/eps;   wz5 = 1/eps;
%
% ------------------------------------------------------------
  pole1 = 1./(1-s/p1);        pole2 = 1./(1-s/p2);          pole3 = 1./(1-s/p3);
                              pole4 = 1./(1-s/p4);          pole5 = 1./(1-s/p5);

  zero1 = 1-s/z1;             zero2 = 1-s/z2;               zero3 = 1-s/z3;
                              zero4 = 1-s/z4;               zero5 = 1-s/z5;

  cpole1 = 1./(1 + (2*zp1 + s/wp1).*s/wp1);
       cpole2 = 1./(1 + (2*zp2 + s/wp2).*s/wp2);
       cpole3 = 1./(1 + (2*zp3 + s/wp3).*s/wp3);
       cpole4 = 1./(1 + (2*zp4 + s/wp4).*s/wp4);
       cpole5 = 1./(1 + (2*zp5 + s/wp5).*s/wp5);

  czero1 = 1 + (2*zz1 + s/wz1).*s/wz1;
       czero2 = 1 + (2*zz2 + s/wz2).*s/wz2;
       czero3 = 1 + (2*zz3 + s/wz3).*s/wz3;
       czero4 = 1 + (2*zz4 + s/wz4).*s/wz4;
       czero5 = 1 + (2*zz5 + s/wz5).*s/wz5;

  G = (k./s.^n) ...
       .*pole1.*pole2.*pole3.*pole4.*pole5 ...
       .*zero1.*zero2.*zero3.*zero4.*zero5 ...
       .*cpole1.*cpole2.*cpole3.*cpole4.*cpole5 ...
       .*czero1.*czero2.*czero3.*czero4.*czero5;
```

Figure 2.15. Feedback compensator template `fbcomp.m` *for the user to copy, rename, and edit in order to create the appropriate feedback compensator. As it stands,* `fbcomp.m` *realizes the transfer function G(s)=1. Some parameters are pre-set to* `1/eps`, *where* `eps` *is the Matlab machine precision constant, and others to* `0`, *to make their respective factor = 1. The user should change the parameter values as appropriate, see Figure 2.16 and 2.18. If needed, the user may add more parameters and factors, such as* `p6` *and* `pole6`, *etc.*

With a feedback compensator file ready, one first draws a Nichols diagram with appropriate bounds, using the commands `hngrid`, `hold on`, and `showbnd`. The commands `mgrid` and `hzoom` are also useful. Then `cdesign` is used to display the current nominal open loop, with or without a previously designed nominal open loop. The procedure is now illustrated for Example 2.1.

First `fbcomp.m` is copied into the file `g1.m` in the workspace. Clearly `g1.m` realizes $G(s)=1$, and hence we will first study the nominal plant transfer function in relation to the bounds.

```
clg,hold off, hngrid, axis([-360 0 -50 50]), mgrid(12,10), hold on
showbnd('ex2_1a',gcf,[.2 .5 1 2],'rsrs','roll', ...
  [5 10 20 50],'odsrs','roll' );    % roll refers to rolling colors
h1=cdesign('ex2 1a.m','g1',[], [],[],[],logspace(-2,3));
```

The output variable `h1` of `cdesign` is a handle that allows the user to erase or keep old nominal plots during subsequent design attempts.

Clearly we need to have a PI or PID, and in order to get a straight, nice Nichols plot, we place the zeros and pole of the PID in such a way as to cancel the nominal plant dynamics. The controller file `g1.m` is edited to realize the desired compensator, see Figure 2.16.

In the above command sequence `showbnd` was issued before `cdesign`. It is often smarter to do the opposite, in particular if the nominal open loop spans over several Riemann surfaces, since `showbnd` adapts its plot to the existing figure if the option `gcf` is used.

The command

```
h2=cdesign('ex2_1a.m','g1',[], [],[],[],logspace(-2,3));
```

shows both the nominal plant, and the compensated open loop which now equals $2/s$. In fact, it would now be sufficient to adjust the gain, in order satisfy the Horowitz bounds constraints. That would however constitute a dirty design, with an excess of phase margin and hence of bandwidth, and lacking a high frequency low pass filter for sensor noise attenuation. So, we do adjust the gain, but also include a lag, and a high frequency low pass filter, editing `g1.m` accordingly. The new nominal open loop is displayed together with the previous two open loops by the command

```
h3=cdesign('ex2 1a.m','g1',[], [],[],[],logspace(-2,3));
```

It turns out, however that the first attempt was not a complete success. One has to tune the parameters, including the location of the pole at -3. After each iteration one issues the command

```
h3=cdesign('ex2 1a.m','g1',h3,logspace(-2,3));
```

which leaves the first two curves untouched (A and B in Figure 2.17a) but erasing the previous tuning attempt. The final feedback compensator is recorded in Figure 2.18, and the final nominal open loop is displayed as curve C in the zoomed Figure 2.17a. We see that for 2, 5, and 10 rad/s the nominal touches or almost touches its bounds.

```
function [G] = g1(s)

% g1.m Feedback compensator no. 2 for Example 2.1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DC-gain
% =======
  k = 1;

% Number of integrators
% =====================
  n = 1;

% Real Poles
% ==========
  p1 = -3;            p2 = -1/eps;   p3 = -1/eps;   p4 = -1/eps;   p5 = -1/eps;

% Real Zeros
% ==========
  z1 = -1/eps;        z2 = -1/eps;   z3 = -1/eps;   z4 = -1/eps;   z5 = -1/eps;

% Complex Poles
% =============
  zp1 = 0;            zp2 = 0;       zp3 = 0;       zp4 = 0;       zp5 = 0;
  wp1 = 1/eps;        wp2 = 1/eps;   wp3 = 1/eps;   wp4 = 1/eps;   wp5 = 1/eps;

% Complex Zeros
% =============
  zz1 = 0.6;          zz2 = 0;       zz3 = 0;       zz4 = 0;       zz5 = 0;
  wz1 = 4;            wz2 = 1/eps;   wz3 = 1/eps;   wz4 = 1/eps;   wz5 = 1/eps;
%
% ------------------------------------------------------------
  pole1 = 1./(1-s/p1);        pole2 = 1./(1-s/p2);          pole3 = 1./(1-s/p3);
                              pole4 = 1./(1-s/p4);          pole5 = 1./(1-s/p5);

  zero1 = 1-s/z1;             zero2 = 1-s/z2;               zero3 = 1-s/z3;
                              zero4 = 1-s/z4;               zero5 = 1-s/z5;

  cpole1 = 1./(1 + (2*zp1 + s/wp1).*s/wp1);
       cpole2 = 1./(1 + (2*zp2 + s/wp2).*s/wp2);
       cpole3 = 1./(1 + (2*zp3 + s/wp3).*s/wp3);
       cpole4 = 1./(1 + (2*zp4 + s/wp4).*s/wp4);
       cpole5 = 1./(1 + (2*zp5 + s/wp5).*s/wp5);

  czero1 = 1 + (2*zz1 + s/wz1).*s/wz1;
       czero2 = 1 + (2*zz2 + s/wz2).*s/wz2;
       czero3 = 1 + (2*zz3 + s/wz3).*s/wz3;
       czero4 = 1 + (2*zz4 + s/wz4).*s/wz4;
       czero5 = 1 + (2*zz5 + s/wz5).*s/wz5;

  G = (k./s.^n) ...
       .*pole1.*pole2.*pole3.*pole4.*pole5 ...
       .*zero1.*zero2.*zero3.*zero4.*zero5 ...
       .*cpole1.*cpole2.*cpole3.*cpole4.*cpole5 ...
       .*czero1.*czero2.*czero3.*czero4.*czero5;
```

Figure 2.16. The edited feedback compensator file `g1.m` *realizing the transfer function*

$$G(s)=\frac{\left(1+2\cdot 0.6s/4+s^2/16\right)}{s(1+s/3)}.$$

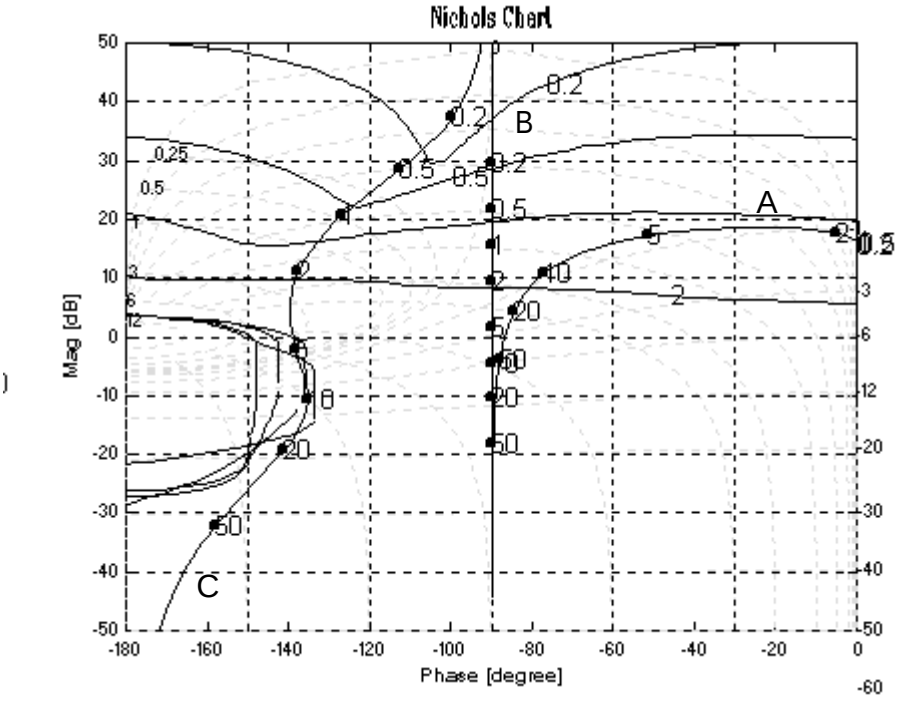

Figure 2.17a. Design of the nominal open loop, $L_{nom}(s)=P_{nom}(s)G(s)$, in a Nichols chart for Example 2.1. The dominant Horowitz bounds, $B_L(\omega_k)$ from Figure 2.14, and $L_{nom}(j\omega)$ are parametrized by frequency [rad/s]*. The nominal plant is $P_{nom}(s)=2(1+s/3)\Big/\left(1+2\cdot 0.6s/4+s^2/16\right)$. $L_{nom}(j\omega)$ is displayed with $G(s)=1$ from Figure 2.15 (curve A), with $G(s)=2\Big/\left(sP_{nom}(s)\right)$ from Figure 2.16 (curve B), and in curve C with the final compensator from Figure 2.18,*

$$G(s)=\frac{2.5(1+s/6)\left(1+2\cdot 0.6s/4+s^2/16\right)}{s(1+s)(1+s/3.2)(1+s/26)} \qquad (2.4)$$

The open loop templates are simply the plant templates multipled by $G(j\omega_k)$. As an interesting illustration we proceed to display in an inverse Nichols diagram, in Figure 2.17b, the final nominal open loop, together with the open loop templates (note that the order between `showtpl` and `hngrid, hold` on could be reversed if the `gcf` option is `showtpl` were used):

```
tplfop('oex2_1a','*',[],'ex2_1a',1,'g1');%create open loop tpl file
showtpl('oex2 1a',[],[],'-');,hold on, hngrid([],[],1)
```

It is easy to check in Figure 2.17b that the sensitivity specifications are satisfied for the template frequencies. However one might suspect that that for some frequency between 5 and 10 rad/s a violation occurs. The way that the Horowitz bounds for these frequencies are touched by the nominal (Figure 2.17a) also points in this direction. The alternatives for the user are then either *i)* to neglect the potential problem since it is deemed to be insignificant, *ii)* to redesign the nominal open loop with some spare "air" between the nominal points and the bounds for 5 and 10 rad/s, or *iii)* compute an additional template for e.g. 7 rad/s and check if a violation occurs. In Chapter 4 an example is given how an additional template is created and inserted.

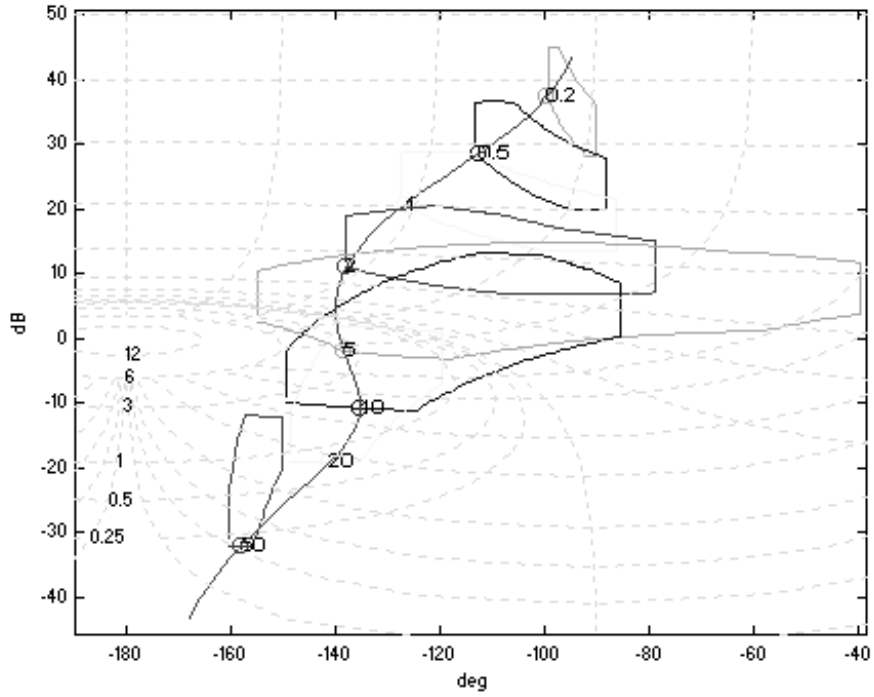

Figure 2.17b. The nominal open loop, $L_{\text{nom}}(s) = P_{\text{nom}}(s)G(j\omega_k)$, and the open loop templates in an inverse Nichols chart for Example 2.1. $P_{\text{nom}}(s)$ and $G(s)$ are given in the caption of Figure 2.17a.

The user may try to improve the design. in a new compensator file `g2.m`, in three directions: *i)* better tuning of the present compensator; *ii)* better adherance to the bounds by smarter compensator elements, like for instance a high frequency lowpass filter consisting of a complex pole and a real zero, a so called Slor filter; and *iii)* designing a lower order compensator that still meets the Horowitz bound constraints in a reasonable way. In such a way one gets a feeling for the trade off between compensator complexity and specifications.

As a closing remark for this section we would like to point out that Example 2.1 is a synthetic example, not only because it satisfies Theorem 1.2, but also because the pole excess of $P(s)$ is one, only. Such systems do not exist in reality. For analog implementation, we always recommend to design strictly proper controllers, because otherwise the electronics will decide the high frequecy cut-off frequency and roll-off for you. We followed this advice, and hence the compensated open loop has a pole excess of two. As already noted, a realistic plant description includes high frequency dynamics, or unstructured uncertainty. Examples of such are given in Chapter 4.

```
function [G] = g1(s)

% g1.m Feedback compensator (final) for Example 2.1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DC-gain
% =======
  k = 2.5;

% Number of integrators
% =====================
  n = 1;

% Real Poles
% ==========
  p1 = -3.2;          p2 = -1;       p3 = -26;      p4 = -1/eps;   p5 = -1/eps;

% Real Zeros
% ==========
  z1 = -1/eps;        z2 = -6;       z3 = -1/eps;   z4 = -1/eps;   z5 = -1/eps;

% Complex Poles
% =============
  zp1 = 0;            zp2 = 0;       zp3 = 0;       zp4 = 0;       zp5 = 0;
  wp1 = 1/eps;        wp2 = 1/eps;   wp3 = 1/eps;   wp4 = 1/eps;   wp5 = 1/eps;

% Complex Zeros
% =============
  zz1 = 0.6;          zz2 = 0;       zz3 = 0;       zz4 = 0;       zz5 = 0;
  wz1 = 4;            wz2 = 1/eps;   wz3 = 1/eps;   wz4 = 1/eps;   wz5 = 1/eps;
%
% ------------------------------------------------------------
  pole1 = 1./(1-s/p1);        pole2 = 1./(1-s/p2);          pole3 = 1./(1-s/p3);
                              pole4 = 1./(1-s/p4);          pole5 = 1./(1-s/p5);

  zero1 = 1-s/z1;             zero2 = 1-s/z2;               zero3 = 1-s/z3;
                              zero4 = 1-s/z4;               zero5 = 1-s/z5;

  cpole1 = 1./(1 + (2*zp1 + s/wp1).*s/wp1);
       cpole2 = 1./(1 + (2*zp2 + s/wp2).*s/wp2);
       cpole3 = 1./(1 + (2*zp3 + s/wp3).*s/wp3);
       cpole4 = 1./(1 + (2*zp4 + s/wp4).*s/wp4);
       cpole5 = 1./(1 + (2*zp5 + s/wp5).*s/wp5);

  czero1 = 1 + (2*zz1 + s/wz1).*s/wz1;
       czero2 = 1 + (2*zz2 + s/wz2).*s/wz2;
       czero3 = 1 + (2*zz3 + s/wz3).*s/wz3;
       czero4 = 1 + (2*zz4 + s/wz4).*s/wz4;
       czero5 = 1 + (2*zz5 + s/wz5).*s/wz5;

  G = (k./s.^n) ...
       .*pole1.*pole2.*pole3.*pole4.*pole5 ...
       .*zero1.*zero2.*zero3.*zero4.*zero5 ...
       .*cpole1.*cpole2.*cpole3.*cpole4.*cpole5 ...
       .*czero1.*czero2.*czero3.*czero4.*czero5;
```

Figure 2.18.The edited feedback compensator file `g1.m` *realizing the final compensator* (2.4).

2.6 Closing the loop

As pointed out in Section 1.7, stability of the closed loop system must be ascertained independently. Our example is however covered by Theorem 1.1, and hence closed loop stability is ensured.

The closed loop template file `cex2 1a.tpl` is produced with the command

```
tplfop('cex2 1a','iosrs',[],'ex2 1a',1,'g1');
```

The user may study the closed loop templates with `showtpl`, and conclude that for the four lowest frequencies where there is significant feedback, the templates are smaller than the open loop templates.

We are however interested in the closed loop gain extent in comparison with the servo specification. As in Section 2.3.1 we issue the command

```
figure, fdesign('cex2_1a.tpl');
hold on, showspc('ex2 1a','rsrs','freq',[],gcf);
```

and get Figure 2.19 which should be compared with Figure 2. 11. The gain extent satisfies the tolerance specification, but not within the envelope of the servo specification. A prefilter is needed to get the the closed loop *FPG/*(1+*PG*) within the specification.

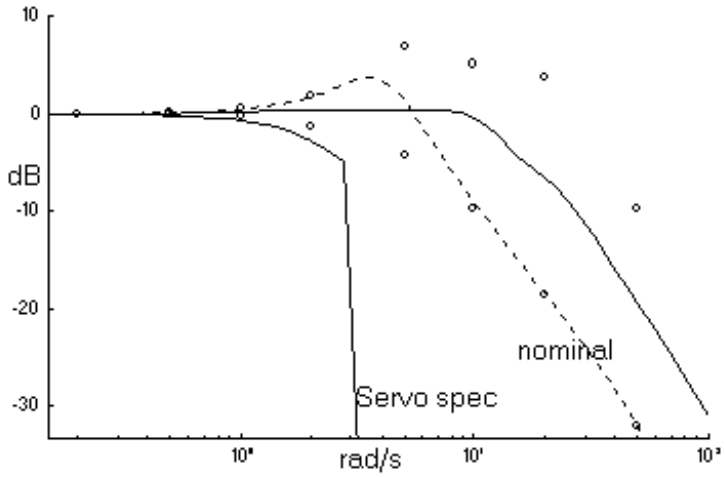

Figure 2.19. The dotted line is the gain of the nominal complementary sensitivity function $P_{nom}(s)G(s)/\left(1+P_{nom}(s)G(s)\right)$ *in* `cex2_1a.tpl`, *where* $G(s)$ *is given in (2.4) and* $P_{nom}(s)=2\left(1+s/3\right)/\left(1+2\cdot 0.6s/4+s^2/16\right)$. *The gain extents of the complementary sensitivity function templates in* `cex2_1a.tpl` *are marked by* o, *and the servo specification is from* `rsrs_w` *in* `ex2_1a.spc`.

2.7 Prefilter design

The prefilter is designed with the help of the command `fdesign` which has the same syntax as `cdesign`. The prefilter is a Matlab m-file of exactly the same structure as the feedback controller, and it is recommended that the standard file `prefil.m` is used as a "template" for editing.

From Figure 2.19 it seems as if t $F(s)$ should be a low pass filter with a bandwidth of about 2 rad/s. After a few attemps, the prefilter file `f1.m` got its final form of Figure 2.20, i.e. including a well damped second order low pass filter whose bandwidths is about 4 rad/s. The gain of the final closed loop in Figure 2.21 is displayed the result of the command

```
figure, fdesign('cex2_1a.tpl','f1');
hold on, showspc('ex2_1a','rsrs','freq',[],gcf);
```

Figure 2.21 shows a limited frequency range around the bandwidth. The gain extent of closed loop transfer function, $\max\left|FL/(1+L)\right|$ and $\min\left|FL/(1+L)\right|$, is within specifcations. It now remains to simulate a number of plant cases in the frequency and time domains to ensure that there is no wobbling (secondary resonances, see Horowitz and Sidi 1972), and that the time domain specifications are satisfied.

```
function [F] = f1(s)

% f1.m Prefilter (final) for Example 2.1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DC-gain
% =======
  k = 1;

% Number of integrators
% =====================
  n = 0;

% Real Poles
% ==========
  p1 = -1/eps;         p2 = -1/eps;   p3 = -1/eps;   p4 = -1/eps;   p5 = -1/eps;

% Real Zeros
% ==========
  z1 = -1/eps;         z2 = -1/eps;   z3 = -1/eps;   z4 = -1/eps;   z5 = -1/eps;

% Complex Poles
% =============
  zp1 = 0.83;          zp2 = 0;       zp3 = 0;       zp4 = 0;       zp5 = 0;
  wp1 = 3.4;           wp2 = 1/eps;   wp3 = 1/eps;   wp4 = 1/eps;   wp5 = 1/eps;

% Complex Zeros
% =============
  zz1 = 0;             zz2 = 0;       zz3 = 0;       zz4 = 0;       zz5 = 0;
  wz1 = -1/eps;        wz2 = 1/eps;   wz3 = 1/eps;   wz4 = 1/eps;   wz5 = 1/eps;
%
% ------------------------------------------------------------
  pole1 = 1./(1-s/p1);        pole2 = 1./(1-s/p2);          pole3 = 1./(1-s/p3);
                              pole4 = 1./(1-s/p4);          pole5 = 1./(1-s/p5);

  zero1 = 1-s/z1;             zero2 = 1-s/z2;               zero3 = 1-s/z3;
                              zero4 = 1-s/z4;               zero5 = 1-s/z5;

  cpole1 = 1./(1 + (2*zp1 + s/wp1).*s/wp1);
       cpole2 = 1./(1 + (2*zp2 + s/wp2).*s/wp2);
       cpole3 = 1./(1 + (2*zp3 + s/wp3).*s/wp3);
       cpole4 = 1./(1 + (2*zp4 + s/wp4).*s/wp4);
       cpole5 = 1./(1 + (2*zp5 + s/wp5).*s/wp5);

  czero1 = 1 + (2*zz1 + s/wz1).*s/wz1;
       czero2 = 1 + (2*zz2 + s/wz2).*s/wz2;
       czero3 = 1 + (2*zz3 + s/wz3).*s/wz3;
       czero4 = 1 + (2*zz4 + s/wz4).*s/wz4;
       czero5 = 1 + (2*zz5 + s/wz5).*s/wz5;

  F = (k./s.^n) ...
       .*pole1.*pole2.*pole3.*pole4.*pole5 ...
       .*zero1.*zero2.*zero3.*zero4.*zero5 ...
       .*cpole1.*cpole2.*cpole3.*cpole4.*cpole5 ...
       .*czero1.*czero2.*czero3.*czero4.*czero5;
```

Figure 2.20. The edited prefilter file `f1.m` *realizing the final prefilter*

$$F(s) = \frac{1}{\left(1 + 2 \cdot 0.83 s/3.4 + s^2/3.4^2\right)} \tag{2.5}$$

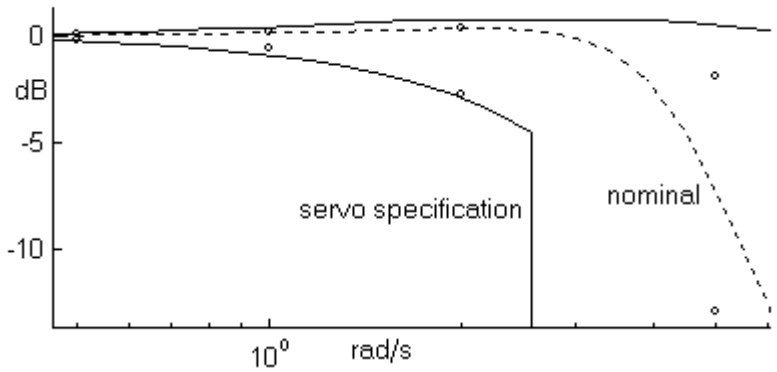

Figure 2.21. The dotted line is the gain of the nominal closed loop transfer function $F(s)P_{\text{nom}}(s)G(s)/\left(1+P_{\text{nom}}(s)G(s)\right)$, *where the complimentary sensitivity function nominal* $P_{\text{nom}}(s)G(s)/\left(1+P_{\text{nom}}(s)G(s)\right)$ *and templates* $P(s)G(s)/\left(1+P(s)G(s)\right)$ *are found in* `cex2_1a.tpl`. $G(s)$ *is given in (2.4),* $F(s)$ *in (2.5) and* $P_{\text{nom}}(s)=2\left(1+s/3\right)/\left(1+2\cdot 0.6s/4+s^2/16\right)$. *The gain extents of the templates are marked by* o*, and the servo specification is from* `rsrs_w` *in* `ex2_1a.spc`

2.8 Simulations

One efficient way to select plant cases for simulation is to use those parameter combinations from which the edes of one or more templates emanate. Some template computation methods give this information, e.g. the Recursive Edge Grid method or one of the other grid methods after pruning. For each template in the template file, a parameter matrix is stored whose columns contain the plant cases. Since the rff method does not give this information, we take it from one of the other template files if we were wise enough to save it (Section 2.2.2), or recompute a template for some critical frequency.

Based on Figure 2.17b we select the template for 5 rad/s, and compute its edge with the Recursive Edge Grid method. Then the command `gettpl` is used to get out the parameter matrix `par` which includes 84 plant cases. We use the command `ccases`, to compute the closed loop frequency functions and sensitivity functions for the selected cases, on the dense frequency vector `logspace(-1,2,120)`. Compare the use of the command `cases` in Section 2.1.6. The sensitivity and closed loop frequency functions are plotted in Figures 2.22 and 2.23, respectively.

```
ctpl('ex2_1b','ex2_1a','aedgrid_[5,5]',5);  % 5 rad/s template
[tpl,par]=gettpl('ex2_1b',5);              % par covers 84 cases
figure, showspc('ex2_1a','odsrs','freq');
ccases('ex2 1a',par,'odsrs','g1','f1',logspace(-1,2,120),'mag');
hzoom                                      % Fig. 2.22
figure, showspc('ex2 1a','rsrs','freq');
ccases('ex2 1a',par,'rsrs','g1','f1',logspace(-1,2,120),'mag');
hzoom                                      % Fig. 2.23
```

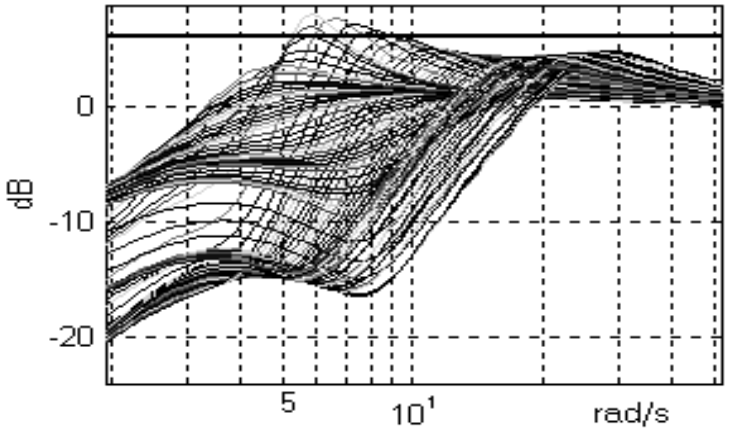

Figure 2.22. Selected sensitivity function cases for the the plant (2.1) controlled by the feedback compensator (2.4). For some plant cases, the sensitivity specification (2.2c), drawn in heavy black, is violated for the frequencies [6, 9] rad/s.

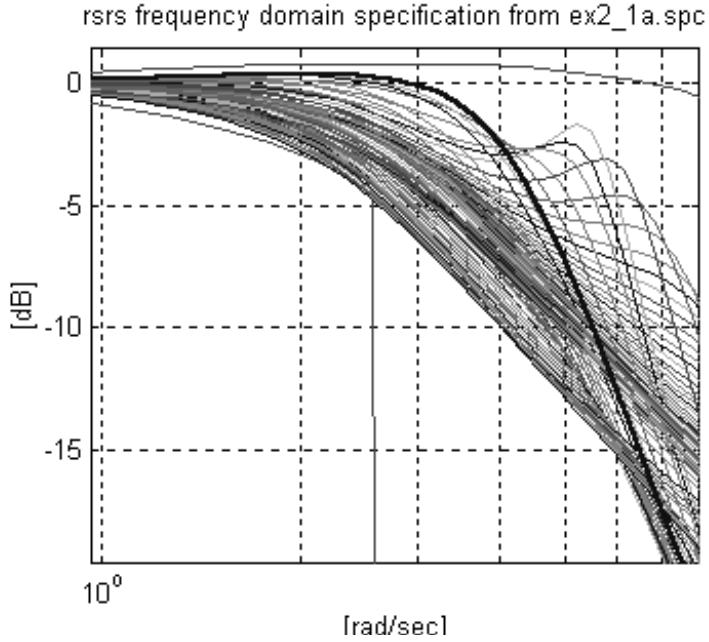

Figure 2.23. Selected closed loop frequency function cases for the the plant (2.1) controlled by the feedback compensator (2.4) and the prefilter (2.5).The servo specification in the frequency domain (Figure 2.10), is also drawn, and is seen to be almost satisfied. A better prefilter design would have satisfied it completely.

We notice (with consternation) in Figure 2.22 that the sensitivity specification is not satisfied, exactly between those frequencies in Figure 2.17a for which we had proudly designed the feedback compensator such that the nominal open loop sat right on the Horowitz bounds. As seen in Figure 2.23, a slight violation of the servo specification seems to occur for some plant cases at 3 rad/s — although the prefilter design at the design frequencies (Figure 2.22) gave an indication that all is in order. As remarked in Section 2.6 in the paragraphs following equation (2.4), we should redesign the nominal open loop either by including bounds for more frequencies, or by an "overdesign" for the current Horowitz bounds in the sense that the nominal open loop keeps a small distance to all the bounds.

Qsyn does not provide any tool for time domain simulation. *As a service to the the user who has access to the Matlab Control Systems Toolbox*, we provide the following commands that will yield the 84 closed loop responses to a unit reference step, shown in Figure 2.24 together with the time domain specification from file `ex2_1a.spc`.

```
ctpl('ex2 1b','ex2 1a','aedgrid [5,5]',5);  % 5 rad/s template
[tpl,par]=gettpl('ex2 1b',5);              % par has 86 columns
[n,m]=size(par);
Gnum = 2.5*conv([1/6  1],[1/16  2*0.6/4 1]);          % equ (2.4)
Gden  = conv(conv([1 0],[1 1]),conv([1/3.2  1],[1/26  1]));
Fnum = [1];                                           % equ (2.5)
Fden  = [1/(3.4*3.4)  2*0.83/3.4   1];
t = 0:0.01:3;
y = zeros(length(t),m);
for i = 1:m,
  k=par(1,i); a=par(2,i); z=par(3,i); wn=par (4,i);   % Fig. 2.2
  Pnum = k*[1 a];                                     % equ (2.1)
  Pden = [1/(wn*wn)  2*z/wn  1];
   [Onum, Oden] = series(Gnum, Gden, Pnum, Pden);      % open loop
   [Sbarnum, Sbarden] = feedback(Onum,Oden,[1],[1],-1);  % compl sens
   [Cnum, Cden] = series(Fnum, Fden, Sbarnum, Sbarden);  % cl loop
  y(:,i) = step(Cnum, Cden, t);     % closed loop step response
end
showspc('ex2 1a','rsrs','time'),plot(t,y);
plot([1.5 3],[.95 .95],'r',[1.5 3],[1.05 1.05],'r','linewidth',2)
```

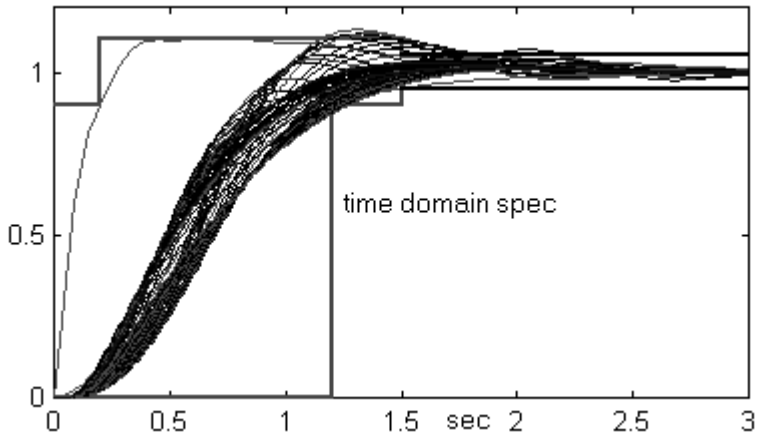

Figure 2.24. Selected closed loop reference step response cases for the the plant (2.1) controlled by the feedback compensator (2.4) and the prefilter (2.5). For some plant cases, the servo specification in the time domain (Figure 2.9) complemented with the settling time specification (2.2b), drawn in thick grey and black, is violated both with respect to the overshoot (2.2a) and settling time (2.2b) specifications.

In Figure 2.24 the time domain specifications are violated, too. We notice that the envelope of the model responses (drawn in thin grey in Figure 2.24) is also violated. The reason seems to be that the time domain specification in Section 2.3.1 is specified with too large a difference between maximim and minimum rise time ([1.2, 0.2] seconds), which causes the tolerance specification (Figure 2.9 and equation 1.10) to be to large around the bandwidth. Figure 2.24 reveals that the closed loop rise time varies between 0.8 and 1.3 seconds, a factor of 1.6, which is much less than the specified span of [0.2, 1.2] seconds. The maximum rise time should be lowered, but also the minimum rise time should be increased. It is also clear that the chosen 2nd and 3rd order models are not well adapted to this system which is of higher order. An idea is to use only a second order approximant which will make the rise time difference smaller.

2.9 Conclusions

All control designs have been iterative until a presentable result was reached. Our example is no exception, and above the first few steps have been shown. We have shown some common pitfalls:

- templates computed by methods that do not give the true template edges;
- too few and/or unsuitable frequencies for which templates and Horowitz bounds are computed;
- time domain specifications that do not reflect the nature of the closed loop system and therefore give rise to too loose frequency domain specifications, while in other cases one may get too tight frequency domain specifications that may result in over design, i.e. excessive bandwidth of the complementary sensitivity function, or a failure to find a controller;
- too great an effort to satisfy the Horowitz bound constraints strictly, by "clever" loop shaping, which may result in violations of the specification for frequencies between the bound frequencies, and may also make it necessary to spend a great effort to loop shape the prefilter so that the servo specification is strictly satisfied.

2.10 Summary of the Qsyn command sequence

```
% edit the plant description file ex2 1a.m, Sec. 2.1
plnt ex2 1a

% plot selected plant frequency function cases in a Bode diagram
cases('ex2_1a',[2 5; 1 3; .3 .3; 8 4],[],1);

% compute the plant templates and place them in the template file
% ex2 1a.tpl, Section 2.2
ctpl('ex2 1a')

% display the templates
showtpl('ex2_1a');

% define servo specifications, and translate them to the frequency
% domain, with the result saved in the specification file ex2 1a.spc,
% Section 2.3. Plot the specifications
rsrs('ex2 1a',[],[1.2 0.2],10,1.5,[],logspace(-1,2),2.85,3.1);
showspc('ex2 1a','rsrs','time'); showspc('ex2 1a','rsrs','freq');

% study the plant amplitude function and the servo specifications,
% Figure 2.11
fdesign('ex2_1a.tpl'),
hold on, showspc('ex2 1a','rsrs','freq',[],gcf);

% define sensitivity specifications, store them in ex2 1a.spc, plot
add2spc('ex2 1a','odsrs',logspace(-1,2),6); % create and insert
showspc('ex2 1a','odsrs','freq');           % show


% compute tolerance bounds from the servo specification, place them
in the bounds file ex2_1a.bnd, Section 2.4. Show the bounds
cbnd('ex2 1a','rsrs');
hngrid, hold, showbnd('ex2_1a',gcf,[],'rsrs')
```

```
% compute sensitivity bounds, and place them in ex2 1a.bnd. Plot
cbnd('ex2 1a','odsrs');
hngrid([],[],1),  hold on, showbnd('ex2 1a',gcf,[],'odsrs');

% plot dominant bounds
clg,hold off, hngrid, axis([-360 0 -50 50]), mgrid(12,10), hold on
showbnd('ex2 1a',gcf,[0.2 0.5 1 2],'rsrs',[],[5 10 20 50],'odsrs')

% edit the feedback compensator file g1.m, by copying fbcomp.m.
% Section 2.5

% show the nominal open loop with the dominant bounds
clg,hold off, hngrid, axis([-360 0 -50 50]), mgrid(12,10), hold on
showbnd('ex2_1a',gcf,[.2 .5 1 2],'rsrs','roll', ...
  [5 10 20 50],'odsrs','roll' );
h1=cdesign('ex2 1a.m','g1',[], [],[],[],logspace(-2,3));

% edit g1.m interatively , until the nominal open loop satisfies the
% bounds, plot
h2=cdesign('ex2_1a.m','g1',[],[],[],[],logspace(-2,3));

% display open loop nominal with templates
tplfop('oex2 1a','*',[],'ex2 1a',1,'g1');%create open loop tpl file
showtpl('oex2 1a',[],[],'-');,hold on, hngrid([],[],1)

% close the loop. Place the complementary sensitivity function
templates in the file cex2 1a.m.  Section 2.6
tplfop('cex2_1a','iosrs',[],'ex2_1a',1,'g1');

% compare the complementary sensitivity function with the servo spec
fdesign('cex2 1a.tpl');
hold on, showspc('ex2 1a','rsrs','freq',[],gcf);

% edit the prefilter file f1.m, by copying prefil.m.
% Section 2.7

% compute the closed loop, and compare with servo specifications
fdesign('cex2_1a.tpl','f1');
hold on, showspc('ex2 1a','rsrs','freq',[],gcf);

% edit f1.m interatively, until the nominal open loop satisfies the
% servo specifications, plot
fdesign('cex2_1a.tpl','f1'),
hold on, showspc('ex2_1a','rsrs','freq',[],gcf);

% simulate the closed loop in the frequency and time domains, as
% described in Section 2.8

% if necessary, redesign
```