University of Ljubljana

J. Stefan Institute, Ljubljana, Slovenia

IJS Delovno poročilo
Report DP-7054

AMPLITUDE AND PHASE MARGIN DETECTION WITH ON-LINE PID CONTROLLER

Damir Vrančić
Youbin Peng*

August, 1994

*Department of Control Engineering
Free University of Brussels
Belgium

Table of Contents

1. Introduction

The goal of this report was to find such controller parameters that amplitude margin (A_m) and phase margin (ϕ_m) of the controlled system will be as desired so that, if process have transfer function $G_P(s)$, we want to find such controller $G_C(s)$ to reach desired points A and B on the Nyquist curve in Fig. 1.

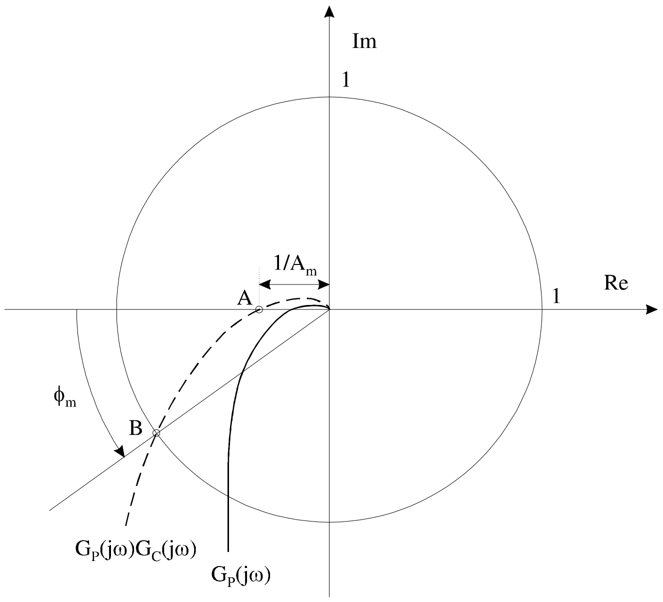

Fig. 1. Nyquist plot of the process ($G_P(s)$) and process with a controller ($G_P(s)G_C(s)$)

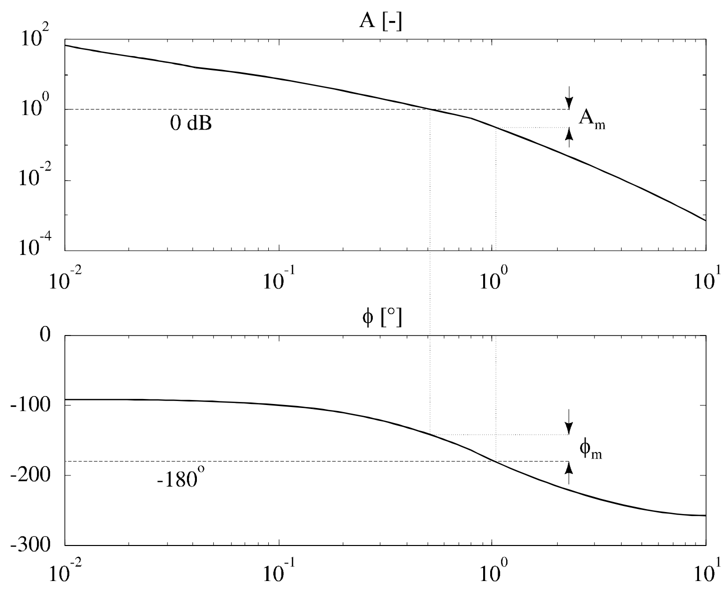

Fig. 2. Bode plot of the process with a controller

Full line (see Fig. 1) represents process' Nyquist curve and dashed line represents correction of the curve made by controller. Fig. 2 shows representation of the amplitude and phase margin in Bode plot.

Points A and B (see fig. 1) can be detected by many ways. One of the most practical way to do that is to use the relay feedback method [1]. In this method, relay is connected in closed-loop with a process as shown in Fig. 3.

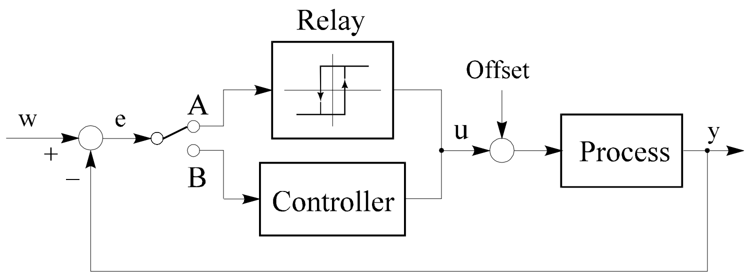

Fig. 3. Relay feedback method

Characteristics of the used relay is as follows:

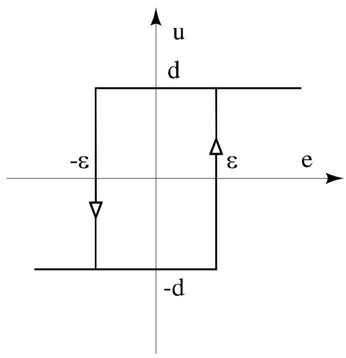

Fig. 4. Relay characteristics

where ε represents hysteresis of the relay and d is an output value. Describing function of such relay is represented in Fig. 5.

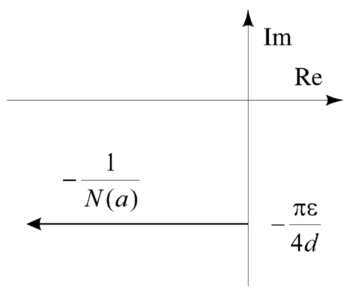

Fig. 5. Describing function of the relay

where -1/N(a) is

$$-\frac{1}{N(a)} = -\frac{\pi}{4d}\sqrt{a^2 - \varepsilon^2} - i\frac{\pi\varepsilon}{4d} \tag{1}$$

The system in Fig. 3 will oscillate at the point where $G_P(j\omega)$ intersects the relay describing function. To detect point A (see Fig. 1), relay without hysteresis (ε=0) should be used. In that case, imaginary part in (1) will be 0 and $G_P(j\omega)$ and relay describing function will intersect on real axis as shown in Fig. 6.

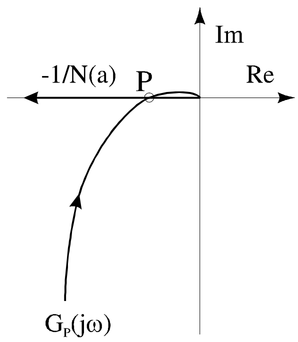

Fig. 6. Detection of the amplitude margin

P is an intersection point and it represents the point at which system described in Fig. 3 oscillates. With detected point P we can calculate amplitude margin as shown in Fig. 1.

To detect phase margin, we have to change hysteresis of used relay such that describing function will intersect desired phase margin (see Fig. 7).

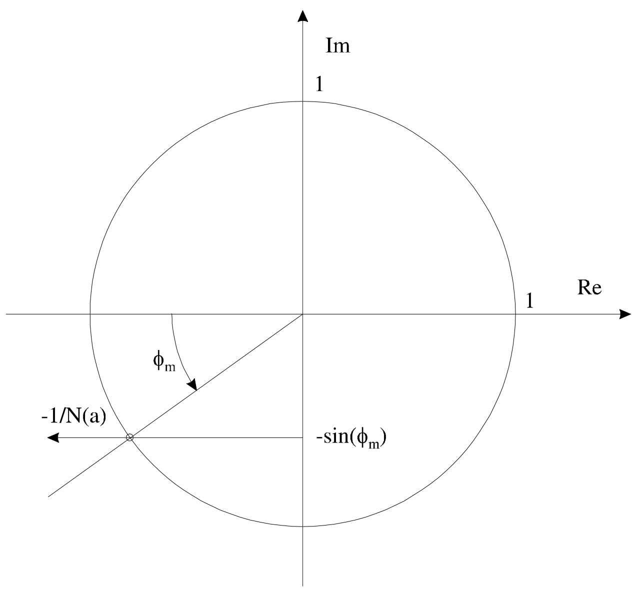

Fig. 7. Detection of the phase margin

From (1) and Fig. 7, we can calculate ε:

$$\varepsilon = \frac{4d}{\pi}\sin\phi_m \tag{2}$$

If process phase margin is bigger than desired, $G_P(j\omega)$ and describing function will intersect at point D_1 and if process phase margin is smaller than desired, intersection point would be D_2 (see Fig. 8).

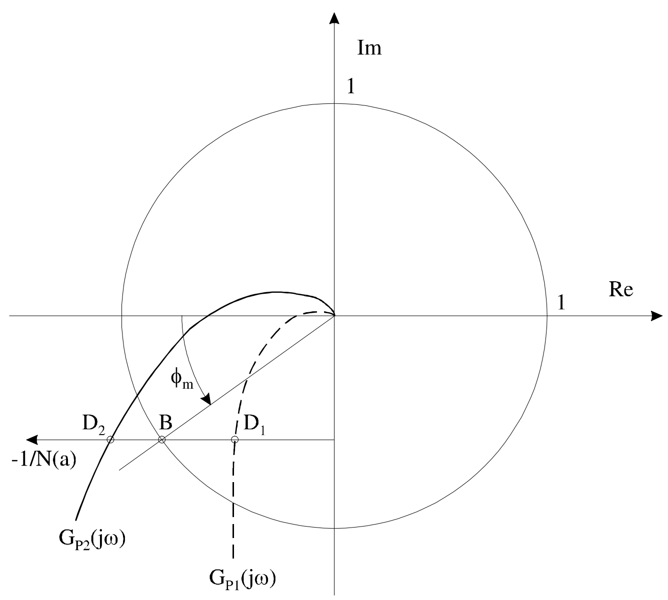

Fig. 8. __ System with too small phase margin, -- system with too big phase margin

From intersection point position we can calculate controller parameters to achieve desired amplitude or phase margin. Then we can change the position of the switch in Fig. 3 from A to B and push the controller into closed-loop configuration. If our goal is to satisfy amplitude *and* phase margin as well, we have to detect more points on the Nyquist curve, usually by changing relay hysteresis or adding some additional function blocks in line with relay [2, 3].

The idea, presented here, is to use modified relay feedback method as shown in Fig. 9.

By this method, controller is always connected in line with the process. It solves some problems related with classical relay feedback method:

- We doesn't have to add offset signal at the process input to achieve desired set-point (see Fig. 3)
- In classical method, if the process is low-order, we have to use some additional blocks (like integrator [3]) in line with tested process to achieve oscillation, while in new scheme the controller already does it.

- In classical method we can't take into account some controller specialities like input filter (analog or/and digital), delayed output, sampling time, ... or if we can do so, the computation would be too complex. Usually fine tuning is required afterwards. In presented method, the controller is all the time in line with a process, so when switching from A to B (see Fig. 9), no fine tuning is required.

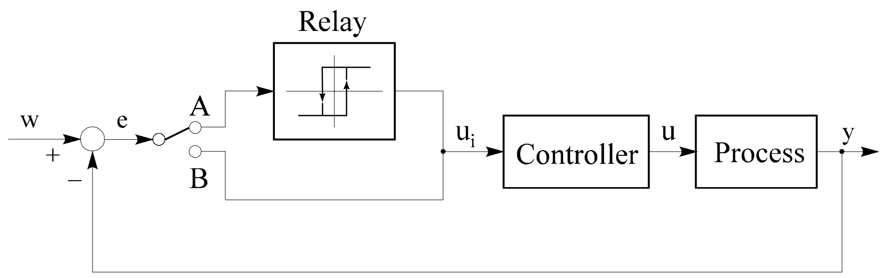

Fig. 9. Modified scheme of the relay tuning method

2. Tuning methods

Here some tuning methods to achieve amplitude and phase margin will be shown. To simplify presentation, we will use PI controller. PID controller can be used as well with some modifications [1, 2, 3]. Actual amplitude and phase margin will be detected as described in previous section (Fig. 6 and 7). Supported simulations are simplified in the way we calculated intersection points from Nyquist curve of $G_P(j\omega)G_C(j\omega)$ obtained in program package MATLAB. Next section (3.) describes how to use tuning methods with a relay.

Basic idea of tuning PI controller is to satisfy both, amplitude and phase margin (A_m and ϕ_m respectively) by iteratively changing K_a and T_i. Three different methods of tuning are presented.

2.1 Direct method

Fig. 10 shows typical situation during tuning procedure. Points C and D are actually detected and our goal is to move them toward points A and B respectively.

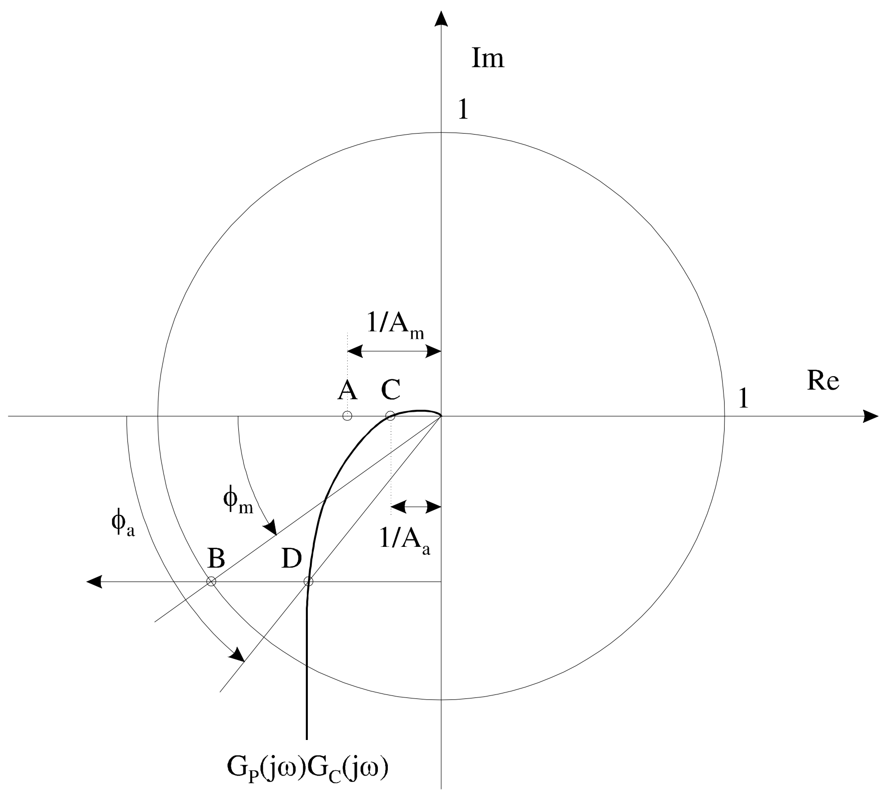

Fig. 10. Tuning procedure: moving point C toward A and point D toward B

To move point C toward A, we can change controller proportional gain K_P:

$$K_P = K_P \frac{A_a}{A_m} \ , \tag{3}$$

where A_a is measured amplitude margin and A_m is a desired one. K_P on the right side is the previous (old) one.

Now, Nyquist curve changes the shape:

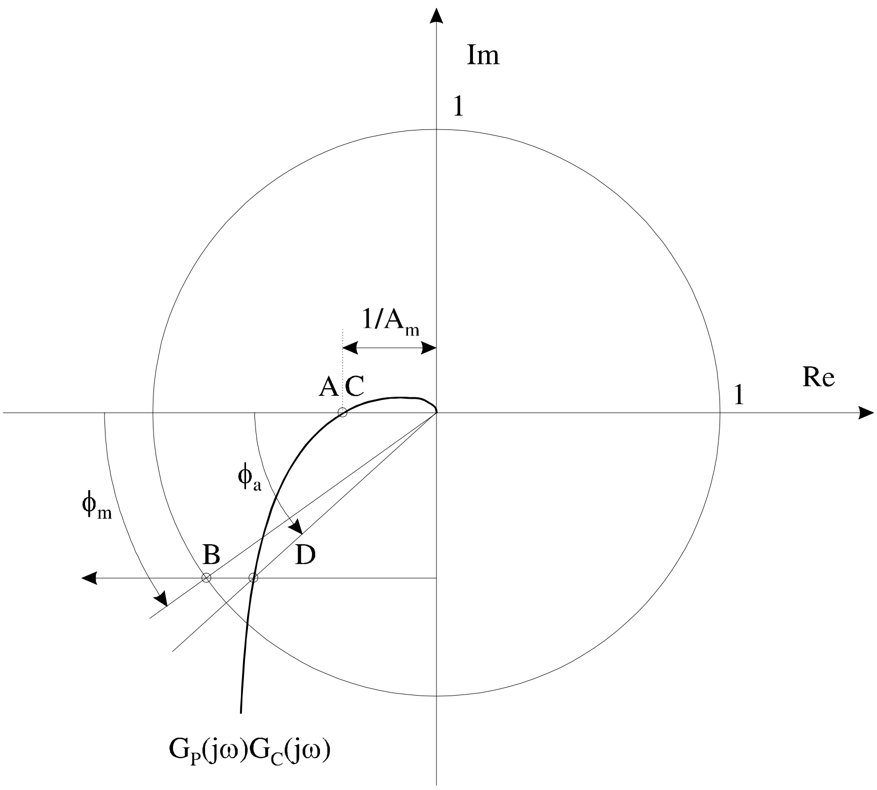

Fig. 11. Nyquist curve after changing K_P

Point C moved to point A. But point D can not move directly to point B. With T_i of the controller we can rotate point D from ϕ_a to ϕ_m. But if we would like to move rotated point D directly to B, we would have to change K_P and point C will move out from A. So, in that method we will only rotate point D to angle ϕ_m with integral time constant T_i of controller.

Controller transfer function is:

$$\left|G_C(j\omega)\right| = K_P \sqrt{1 + \frac{1}{\omega^2 T_i^2}} \tag{4}$$

$$\phi_C(j\omega) = \arctan\left(-\frac{1}{\omega T_i}\right) , \tag{5}$$

where $\phi_C(\omega)$ represents controller's phase shift. To rotate point D from angle ϕ_a to ϕ_m, T_i have to be changed:

$$T_i = \frac{1}{\omega_D \tan\left[\arctan\left(\frac{1}{\omega_D T_i}\right) + \phi_a - \phi_m\right]} , \tag{6}$$

where ω_D represents ultimate frequency at point D and T_i on the right side of equation represents old value of T_i. In the same time change of T_i will cause change of absolute gain at point C (4). Intersection point will move out from the point A and procedure have to be repeated.

The result of tuning with presented method is shown in Fig. 12. X axis represents number of iterations, where one iteration means calculating a new pair of K_P and T_i. Dashed line represents the phase error in degrees. It can be seen that procedure converges, but slowly and practically can't be used successfully.

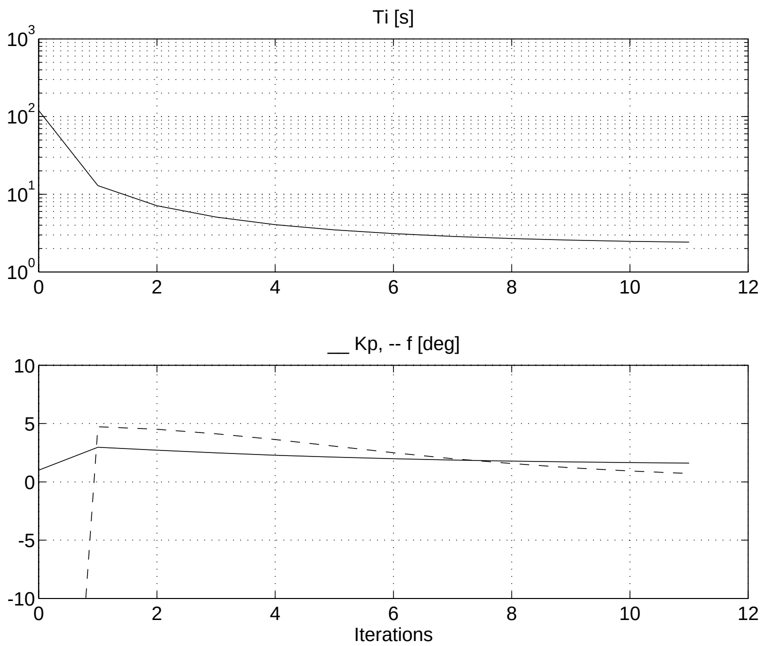

Fig. 12. Tuning procedure: proportional gain (K_P), integration constant (T_i) and difference between actual and desired phase margin (f) for direct method

In all presented simulations we used the same process:

$$G_P = \frac{1}{(1+s)^2(1+2s)} \tag{7}$$

and desired amplitude and phase margin:

$$\begin{aligned} A_m &= 3 \\ \phi_m &= 36° \end{aligned} \tag{8}$$

Proportional constant (K_P) at the beginning is 1 and integral constant (T_i) is 120s.

All used programs (in MATLAB) are printed in appendix.

2.2 Successive change of Ti

The method is relatively simple. At first we detect point C on Nyquist curve (see Fig. 10) and with K_P we can move it to point A (3). Then we detect point D. If $\phi_a > \phi_m$ and there exist no previous T_i such that $\phi_a < \phi_m$, then divide T_i by 2:

$$T_i = \frac{T_i}{2} \tag{9}$$

If $\phi_a < \phi_m$ and there exist no previous T_i such that $\phi_a > \phi_m$, then multiply T_i by 2:

$$T_i = 2T_i \tag{10}$$

In other case calculate T_i as:

$$T_i = \sqrt{T_{i1} T_{i2}} \tag{11}$$

where T_{i1} means the last T_i which caused $\phi_a < \phi_m$ and T_{i2} is the last T_i when $\phi_a > \phi_m$.

The result of presented method is shown in Fig. 13. We can see, this time algorithm converges faster than previous one, but still slow. It also needs some time to find appropriate range of T_i (from ≈ 10^2 s to about 10^0 s).

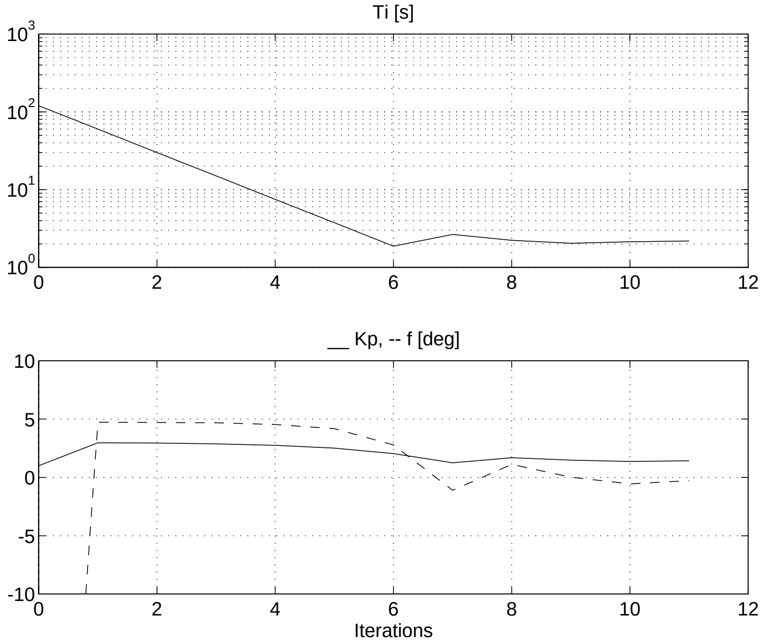

Fig. 13. Tuning procedure: proportional gain (K_P), integration constant (T_i) and difference between actual and desired phase margin (f) for successive change method

That leads us to modify the algorithm. The first step is to find point C and move it toward point A (see Fig. 10 and eq. 3), then (with new K_P) we find point D and calculate new T_i using equation 6 in section 2.1. If new (calculated) T_i is more than two times bigger or more than two times smaller than old one, new T_i would be as calculated (6). Otherwise procedure would be the same as detected by equations 9 and 10.

To speed up the optimisation method when error of phase margin changes the sign, (11) have to be changed:

$$T_i = T_{i1}\left(\frac{T_{i2}}{T_{i1}}\right)^{\frac{\phi_1-\phi_m}{\phi_1-\phi_2}} \tag{12}$$

where T_{i1} represents last T_i for which $\phi_a>\phi_m$, T_{i2} represents last T_i for which $\phi_a<\phi_m$ and ϕ_1 and ϕ_2 represents ϕ_a obtained when using T_{i1} and T_{i2} respectively.

Results using such improved method is shown in Fig. 14. Improved method converges faster.

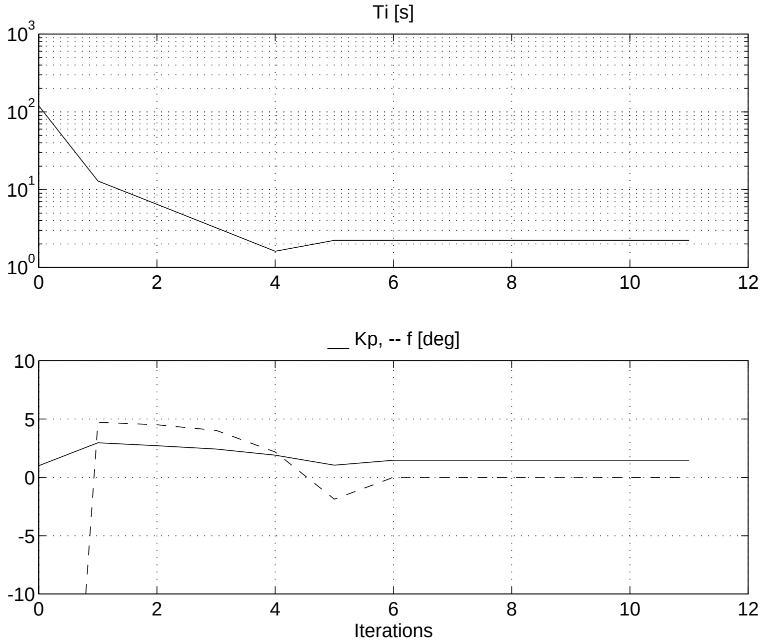

Fig. 14. Tuning procedure: proportional gain (K_P), integration constant (T_i) and difference between actual and desired phase margin (f) for improved successive change method

2.3 Correlation compensation method

The method is based on correlation between amplitude and phase margin. Change of amplitude margin when changing phase margin can be measured and vice versa. Then we can predict T_i for which both, amplitude and phase margin, will be fulfilled.

At first we determine K_P as shown in section 2.1 (see Fig. 10 and eq. 3). We could have the situation as shown with solid line in Fig. 15.

If we rotate point D for the angle $\Delta\phi$, Nyquist curve intersects describing function at point D_1 instead of B (dashed line). Actually, the curve rotates the angle $\Delta\phi_1$:

$$\Delta\phi_1 = k_1 \Delta\phi \tag{13}$$

where k_1 is a gain factor between $\Delta\phi$ and $\Delta\phi_1$:

$$k_1 = \frac{\Delta\phi_1}{\Delta\phi} \tag{14}$$

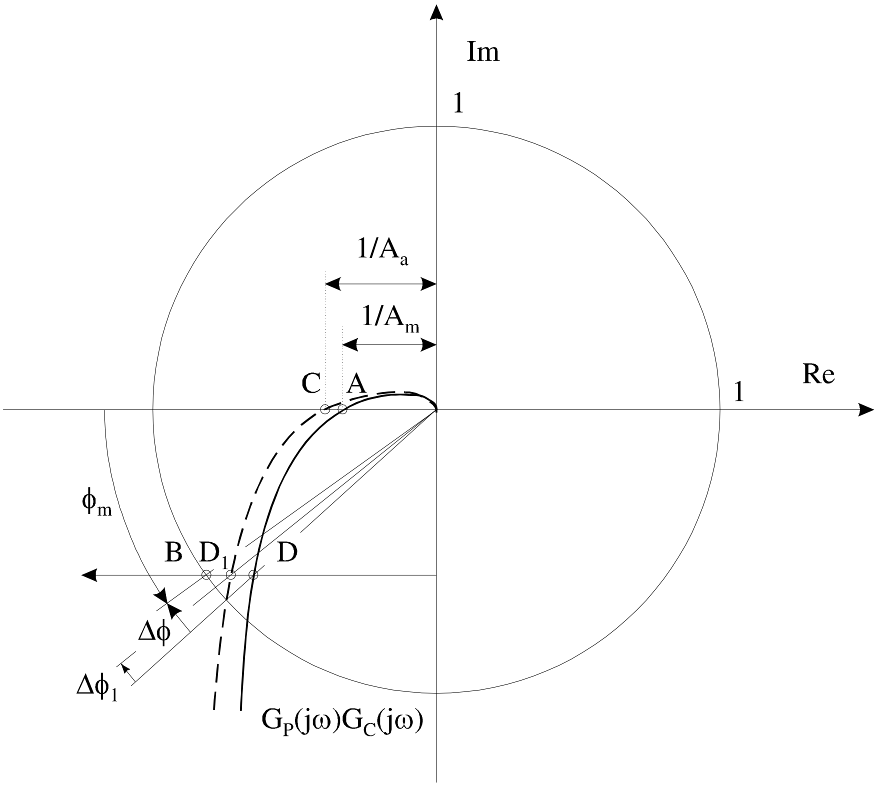

Fig. 15. Change of T_i changes desired amplitude margin

In Fig. 15 we can also see that change of T_i also causes change of amplitude margin. It changes from A_m to A_1:

$$A_1 = k_2 A_m \Delta\phi_1 \tag{15}$$

where k_2 represents correlation factor from phase to amplitude margin:

$$k_2 = \frac{A}{A_m \Delta\phi_1} \tag{16}$$

To correct amplitude margin, we have to calculate K_P again (see Fig. 10 and eq. 3). This correction changes phase margin (see Fig. 16).

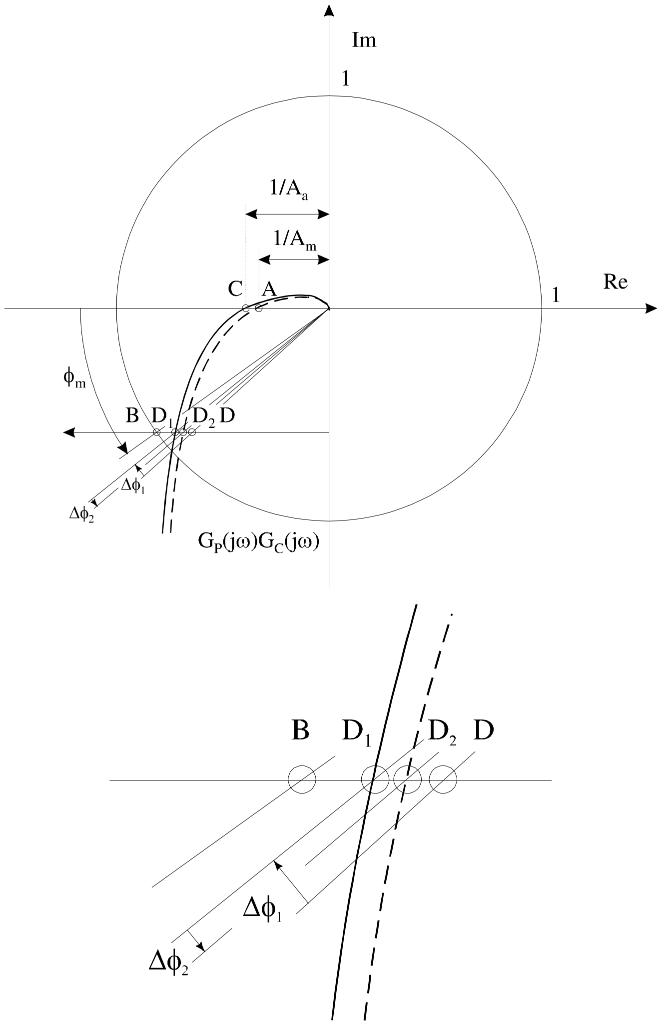

Fig. 16. Correction of amplitude margin changes phase margin (lower Figure is magnified part of upper Nyquist diagram)

Intersection point with describing function changes from D_1 to D_2. Phase margin increases for the angle $\Delta\phi_2$:

$$\Delta\phi_2 = k_3 \Delta\phi_1 \left(1 - \frac{A}{A_m}\right) \tag{17}$$

where k_3 is a correlation factor from amplitude to phase margin:

$$k_3 = \frac{\Delta\phi_1}{\Delta\phi_2}\left(1 - \frac{A}{A_m}\right)^{-1} \tag{18}$$

If we want to correct the phase margin exactly from D to B, angles $\Delta\phi_1$ and $\Delta\phi_2$ have to be such that

$$\Delta\phi_1 - \Delta\phi_2 = \phi_{DOB} \tag{19}$$

where ϕ_{DOB} represents angle DOB (in Fig. 15 marked as $\Delta\phi$). From (13) to (18) we can calculate such $\Delta\phi$ which will satisfy (19):

$$\Delta\phi = \frac{k_3 - 1 \pm \sqrt{(k_3 - 1)^2 + 4k_2k_3\phi_{DOB}}}{2k_1k_2k_3} \tag{20}$$

From (6), if we substitute ϕ_a-ϕ_m = $\Delta\phi$, we can calculate and change T_i and start again new iteration with determining K_P.

Results of described algorithm are shown in Fig. 17. We can see the method is the fastest one. Drawback of such method is, when we are close to desired amplitude and phase margin, $\Delta\phi$ and related $\Delta\phi_1$ and $\Delta\phi_2$ become small and factors k_1, k_2 and k_3 (14, 16 and 18) become inaccurate. Then we should stop this method and continue with e.g. successive change of T_i method (chapter 2.2).

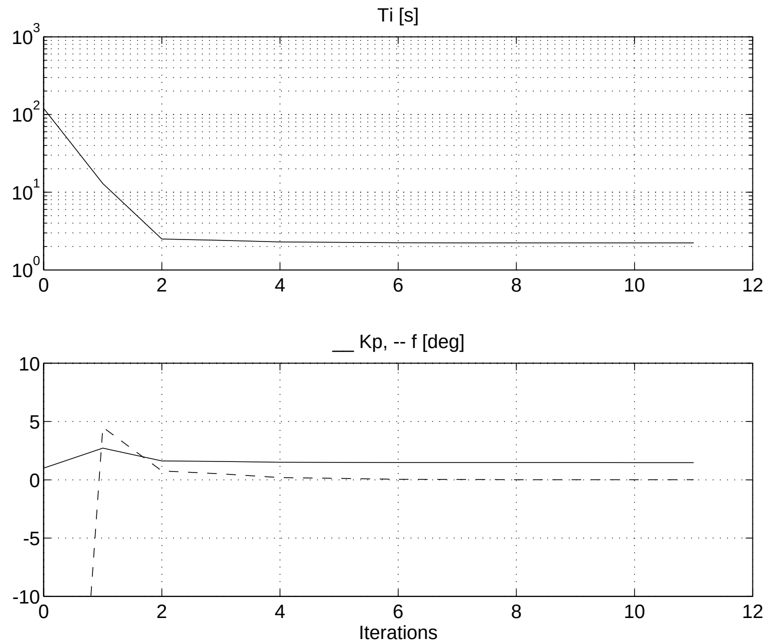

Fig. 17. *Tuning procedure: proportional gain (K_P), integration constant (T_i) and difference between actual and desired phase margin (f) for the correlation compensation method*

3. Detection of Nyquist curve by relay excitation

In previous chapter, some methods of tuning PI controller according to detected Nyquist points were discussed. Here, a procedure how to detect points on Nyquist curve by relay method (chapter 1) is presented.

3.1 Amplitude and phase computation

Limit cycle (oscillation) appears at the point where transfer function $G_P(j\omega)G_C(j\omega)$ crosses describing function of relay (-1/N(a)). From relay and process output (see Fig. 9) we can calculate amplitude and phase of our system ($G_P(j\omega)G_C(j\omega)$). Fig. 18 shows typical time response.

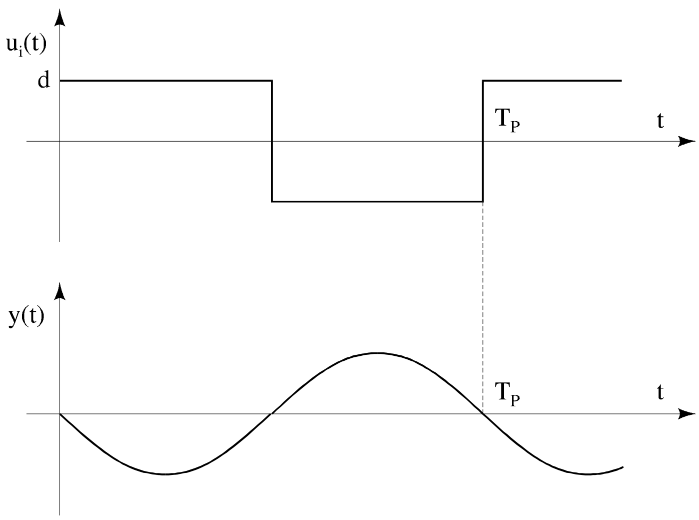

Fig. 18. Time response of relay output (u_i) and process output (y)

System input is therefore square wave signal which consists of main harmonic component at frequency $\omega_0 = 2\pi/T_P$ and other higher harmonic components:

$$u_i(t) = \frac{4d}{\pi}\sum_{n=0}^{\infty}\frac{1}{2n+1}\sin\big((2n+1)\omega_0 t\big) = \frac{4d}{\pi}\left[\sin\omega_0 t + \frac{1}{3}\sin 3\omega_0 t + ...\right] \tag{21}$$

So, process output (y) contains response on all harmonic components of the input signal. To detect the first (main) harmonic (n=0), we have to use “filter” which is in fact Fourier transformation of y at the main frequency (ω_0):

$$A_{re} = \frac{2}{T_P}\int_0^{T_P} y(t)\sin(\omega_0 t)dt \tag{22a}$$

$$A_{im} = \frac{2}{T_P}\int_0^{T_P} y(t)\cos(\omega_0 t)dt \tag{22b}$$

$$A = \sqrt{{A_{re}}^2 + {A_{im}}^2} \tag{22c}$$

$$\phi = \arctan\frac{A_{im}}{A_{re}} \tag{22d}$$

where A_{re} and A_{im} are real and imaginary component of amplitude respectively, A is an amplitude of the first harmonic and ϕ is a phase shift of the first harmonic signal. To compute a gain of $G_P(j\omega_0)G_C(j\omega_0)$, we have to divide A (22c) with the amplitude of the input signal. Amplitude and phase became:

$$A_a = \frac{4d}{\pi A} \tag{23a}$$

$$\phi_a = \phi \tag{23b}$$

where in (22d) we used two quadrant function atan. If we use function atan in all 4 quadrants (e.g. function atan2 in MATLAB), we have to change (23b) into:

$$\phi_a = \pi + \phi \tag{24}$$

Algorithm for detection amplitude and phase is digital, so we changed equations 22a and 22b into next form:

$$A_{re} = \frac{T_S}{T_P}\sum_{k=0}^{n-1}\left[y(k)\sin(kT_S\omega_0) + y(k+1)\sin((k+1)T_S\omega_0)\right] \tag{25a}$$

$$A_{im} = \frac{T_S}{T_P}\sum_{k=0}^{n-1}\left[y(k)\cos(kT_S\omega_0) + y(k+1)\cos((k+1)T_S\omega_0)\right] \tag{25b}$$

T_S is sampling time, y(k) means k-th sample of y and y(n) represents the last sample ($y(T_P)$).

Actual phase margin can be calculated directly from detected amplitude margin if relay hysteresis is set as in (2):

$$\phi_a = \arcsin(A_a \sin\phi_m) \tag{26}$$

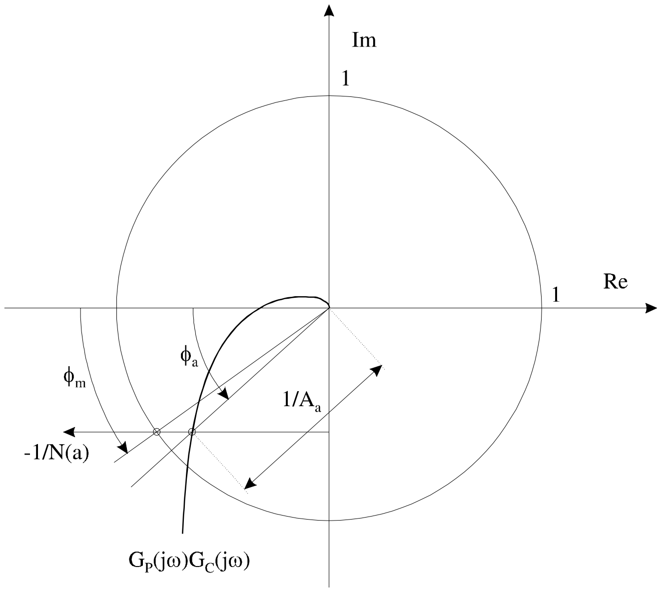

Fig. 19. Calculating phase margin from amplitude margin

To improve accuracy of the calculated phase margin, we can calculate it as a mean value of (22d) and (26):

$$\phi_a = \frac{\arctan\left(\frac{A_{im}}{A_{re}}\right) + \arcsin\left(A_a \sin\phi_m\right)}{2} \tag{27}$$

Fig. 20 shows tuning procedure when using relay excitation instead of Nyquist curve. We used improved successive change method (chapter 2.2). We can see, comparing with Fig. 14, the relay excitation method converges slower. The reason lays in errors when detecting characteristic Nyquist points.

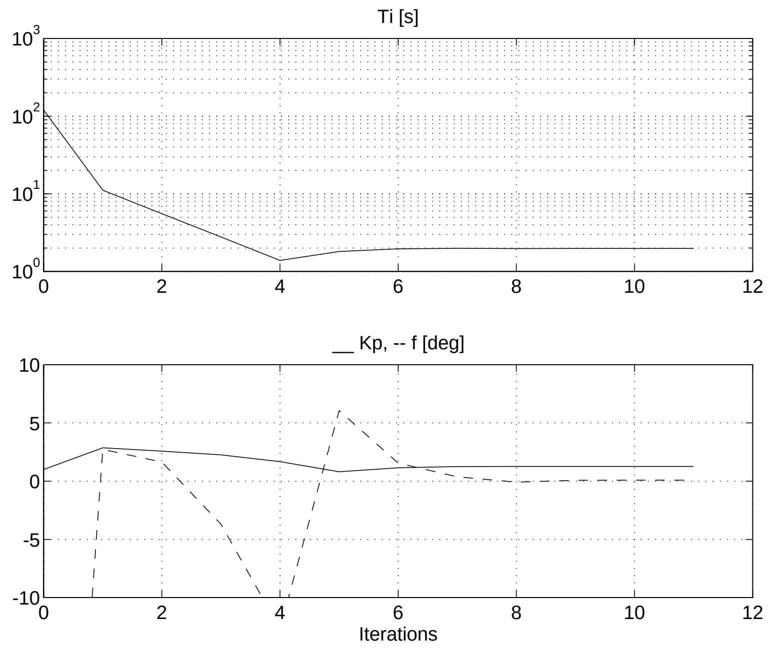

Fig. 20. Tuning procedure: proportional gain (K_P), integration constant (T_i) and difference between actual and desired phase margin (f) for relay excitation and improved successive change method

3.2 Errors

Some error can appear when calculating A_a and ϕ_a (23a and 27) because of time discretisation. If detected period from t=0 to $t=T_P$ consists of n equidistant sampling intervals, then phase margin can not be detected more accurate than:

$$\Delta\phi = \frac{\pm 360°}{n} \tag{28}$$

Period of the oscillation (T_P) can also be inaccurate in a range:

$$\Delta T_P = \pm 2\frac{T_P}{n} \tag{29}$$

what leads to inaccuracy of amplitude margin:

$$\Delta A_a = \frac{A_a}{n} \tag{30}$$

when n is relatively big.

Noise in the system can also have strong influence on result, specially when hysteresis of used relay is small or equal to zero. In that case we could add a filter at the relay input and leave some hysteresis. Then, from the first (main) harmonic and higher harmonics, we can find an approximation of the position of the point C (see fig. 10).

4. References

[1] K. J. Åström and T. Hägglund: "Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins", Automatica, Vol. 20, No. 5, pp. 645-651, 1984.

[2] W. K. Ho, C. C. Hang and L. S. Cao: "Tuning of PID Controllers Based on Gain and Phase Margin Specifications", 12th world congress IFAC, Sydney, Vol. 5, pp 267-270, 1993.

[3] A. Leva: "PID autotuning algorithm based on relay feedback", IEE Proceedings, Part D, Vol. 140, No. 5, September 1993.

5. Appendix

Program in MATLAB for tuning with *direct method* (section 2.1)

```
imag = sqrt(-1);
w = logspace (-2,1,200)+0.001;
fi=0:0.05*pi:2*pi;
re = cos(fi);
im = sin (fi);
points1 = re+imag*im;
fm1 = pi+fm;
rez = [];
Kptmp = Kp;
Titmp = Ti;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));


for l=1:12,

  rez = [rez; Kp Ti (f02-fm)*180/pi];

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  i = 1;
  while (im(i+1) < 0)
    i = i+1;
  end
  rex2 = re(i)+(re(i+1)-re(i))*(im(i))/(im(i)-im(i+1));
  w01 = w(i)+(w(i+1)-w(i))*(im(i))/(im(i)-im(i+1));
  A01 = -rex2;

  Kp = Kp/(Am*A01);
```

```
  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));
  w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  A02 = sqrt(rex1^2 + (sin(fm))^2);

  df = f02-fm;
  Ti = (w02*tan(atan(1/(w02*Ti))+df))^(-1)

end

Kp = Kptmp;
Ti = Titmp;

subplot(211);
semilogy([0:11],rez(:,2))
grid
title('Ti [s]')
subplot(212)
plot([0:11],rez(:,1),[0:11],rez(:,3),'--')
grid
axis([0,12,-10,10])
title('__ Kp, -- f [deg]')
xlabel('Iterations')
```

Program in MATLAB for tuning with *successive change method* (section 2.2)

```
imag = sqrt(-1);
w = logspace (-2,1,200)+0.001;
fi=0:0.05*pi:2*pi;
re = cos(fi);
im = sin (fi);
points1 = re+imag*im;
fm1 = pi+fm;
d = 0.1;
rez = [];
more = 0;
less = 0;
Kptmp = Kp;
Titmp = Ti;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));

for l=1:12,

  rez = [rez; Kp Ti (f02-fm)*180/pi];

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;

  i = 1;
  while (im(i+1) < 0)
    i = i+1;
  end
  rex2 = re(i)+(re(i+1)-re(i))*(im(i))/(im(i)-im(i+1));
  w01 = w(i)+(w(i+1)-w(i))*(im(i))/(im(i)-im(i+1));
  A01 = -rex2;

  Kp = Kp/(Am*A01);

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);
```

```
  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));
  w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  A02 = sqrt(rex1^2 + (sin(fm))^2);

  if (f02 < fm)
    less = Ti;
    if (more == 0)
      Ti = 2*Ti;
    else
      Ti = sqrt (Ti*more);
    end

  else
    more = Ti;
    if (less == 0)
      Ti = Ti/2;
    else
      Ti = sqrt (Ti*less);
    end
  end

end

Kp = Kptmp;
Ti = Titmp;

subplot(211);
semilogy([0:11],rez(:,2))
grid
title('Ti [s]')
subplot(212)
plot([0:11],rez(:,1),[0:11],rez(:,3),'--')
grid
axis([0,12,-10,10])
title('__ Kp, -- f [deg]')
xlabel('Iterations')
```

Program in MATLAB for tuning with *improved successive change method* (section 2.2)

```
imag = sqrt(-1);
w = logspace (-2,1,200)+0.001;
fi=0:0.05*pi:2*pi;
re = cos(fi);
im = sin (fi);
points1 = re+imag*im;
fm1 = pi+fm;
d = 0.1;
rez = [];
more = 0;
less = 0;
moref = 0;
lessf = 0;
Kptmp = Kp;
Titmp = Ti;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));

for l=1:12,

  rez = [rez; Kp Ti (f02-fm)*180/pi];

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;

  i = 1;
  while (im(i+1) < 0)
    i = i+1;
  end
  rex2 = re(i)+(re(i+1)-re(i))*(im(i))/(im(i)-im(i+1));
  w01 = w(i)+(w(i+1)-w(i))*(im(i))/(im(i)-im(i+1));
  A01 = -rex2;

  Kp = Kp/(Am*A01);

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
```

```
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));
  w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  A02 = sqrt(rex1^2 + (sin(fm))^2);


  Tix = 1/(w02*tan(atan(1/(w02*Ti))+f02-fm));

   if ((Tix/Ti > 2) | (Tix/Ti < 0.5))
      Ti = Tix;

  else

    if (f02 < fm)
      less = Ti;
      lessf = f02;
      if (more == 0)
        Ti = 2*Ti;
      else
        a = Ti/more;
        Ti = more*a^((moref-fm)/(moref-f02));
      end

    else
      more = Ti;
      moref = f02;
      if (less == 0)
        Ti = Ti/2;
      else
        a = less/Ti;
        Ti = Ti*a^((f02-fm)/(f02-lessf));
      end
    end
  end
 end

Kp = Kptmp;
Ti = Titmp;

subplot(211);
semilogy([0:11],rez(:,2))
grid
title('Ti [s]')
subplot(212)
plot([0:11],rez(:,1),[0:11],rez(:,3),'--')
grid
axis([0,12,-10,10])
title('__ Kp, -- f [deg]')
xlabel('Iterations')
```

Program in MATLAB for tuning with *correlation compensation method* (section 2.3)

```
imag = sqrt(-1);
w = logspace (-2,1,200)+0.001;
fi=0:0.05*pi:2*pi;
re = cos(fi);
im = sin (fi);
points1 = re+imag*im;
fm1 = pi+fm;
d = 0.1;
rez = [];
Kptmp = Kp;
Titmp = Ti;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));

  rez = [rez; Kp Ti (f02-fm)*180/pi];

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  i = 1;
  while (im(i+1) < 0)
    i = i+1;
  end
  rex2 = re(i)+(re(i+1)-re(i))*(im(i))/(im(i)-im(i+1));
  w01 = w(i)+(w(i+1)-w(i))*(im(i))/(im(i)-im(i+1));
  A01 = -rex2;

  Kp01 = Kp/(Am*A01);
  Kp = Kp01;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
```

```
 plot([cos(fm1)+imag*sin(fm1) 0]);
 plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
 axis([-1.5 1.5 -1.5 1.5]);
 grid
 hold off;
 pause (1);

 w=w';
 i = 1;
 while (im(i+1) < -sin(fm))
   i = i+1;
 end
 rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
 f02 = atan(sin(fm)/(-rex1));
 w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
 A02 = sqrt(rex1^2 + (sin(fm))^2);


 fx = f02-fm;
 df = fx;


 Ti = (w02*tan(atan(1/(w02*Ti))+(f02-fm)))^(-1);


for l=1:11,


 [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
 points = re+imag*im;
 plot(points);
 hold on;
 plot(points1);
 plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
 plot([cos(fm1)+imag*sin(fm1) 0]);
 plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
 axis([-1.5 1.5 -1.5 1.5]);
 grid
 hold off;
 pause (1);

 w=w';
 i = 1;
 while (im(i+1) < -sin(fm))
   i = i+1;
 end
 rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
 f02 = atan(sin(fm)/(-rex1));
 w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
 A02 = sqrt(rex1^2 + (sin(fm))^2);

 fx = f02-fm;
 df1 = df-fx;
 k1 = df1/df;

 [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
 points = re+imag*im;
 plot(points);
 hold on;
 plot(points1);
 plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
 plot([cos(fm1)+imag*sin(fm1) 0]);
 plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
 axis([-1.5 1.5 -1.5 1.5]);
 grid
 hold off;
 pause (1);

 i = 1;
```

```
while (im(i+1) < 0)
  i = i+1;
end
rex2 = re(i)+(re(i+1)-re(i))*(im(i))/(im(i)-im(i+1));
w01 = w(i)+(w(i+1)-w(i))*(im(i))/(im(i)-im(i+1));
A01 = -rex2;


k2 = A01/(Am*df1);
Kp01 = Kp/(Am*A01);
Kp = Kp01;


[re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
points = re+imag*im;
plot(points);
hold on;
plot(points1);
plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
plot([cos(fm1)+imag*sin(fm1) 0]);
plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
axis([-1.5 1.5 -1.5 1.5]);
grid
hold off;
pause (1);

w=w';
i = 1;
while (im(i+1) < -sin(fm))
  i = i+1;
end
rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
f02 = atan(sin(fm)/(-rex1));
w02 = w(i)+(w(i+1)-w(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
A02 = sqrt(rex1^2 + (sin(fm))^2);

rez = [rez; Kp Ti (f02-fm)*180/pi];

fx = f02-fm;
df2 = df-df1-fx;
k3 = df2/df1*(1-A02/Am)^(-1);

df01 = 1/(2*k1*k2*k3)*(1+k3+sqrt((1+k3)^2-4*k2*k3*fx))
df02 = 1/(2*k1*k2*k3)*(1+k3-sqrt((1+k3)^2-4*k2*k3*fx))


if (fx > 0)
  if (df02 < df01)
    df = df01;
  else
    df = df02;
  end
else
  if (df02 < df01)
    df = df02;
  else
    df = df01;
  end
end


if (abs(df01) == abs(df02))
  break;
else
  Ti = (w02*tan(atan(1/(w02*Ti))+df))^(-1)
end
if (Ti < 0)
  Ti = 0.1;
end
```

```
 end

Kp = Kptmp;
Ti = Titmp;

subplot(211);
semilogy([0:11],rez(:,2))
grid
title('Ti [s]')
subplot(212)
plot([0:11],rez(:,1),[0:11],rez(:,3),'--')
grid
axis([0,12,-10,10])
title('__ Kp, -- f [deg]')
xlabel('Iterations')
```

Program in MATLAB for tuning with *relay excitation* (section 3.1)

```
imag = sqrt(-1);
w = logspace (-2,1,100)+0.001;
fi=0:0.05*pi:2*pi;
re = cos(fi);
im = sin (fi);
points1 = re+imag*im;
fm1 = pi+fm;
d = 0.1;
rez = [];
more = 0;
less = 0;
Kptmp = Kp;
Titmp = Ti;

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));


for i = 1:12,
  rez = [rez; Kp Ti (f02-fm)*180/pi];

  eps = 0;
  [t,x,y] = gear ('rele',60,[],[1e-3,1e-5,0.01,0,3,0]);
  a = size(yout,1);
  j = a;

  while ((yout(j,3) < 0) | (yout(j-1,3) > 0))
    j = j-1;
  end
  k = j;

  t2 = yout(k,1);

  j = k-1;
  while ((yout(j,3) < 0) | (yout(j-1,3) > 0))
    j = j-1;
  end
  l = j;

  t1 = yout(l,1);
  tp = t2-t1;
  w0 = 2*pi/tp;

  t = yout(l:k,1)-t1;
  y = yout(l:k,2);

  rea = 0;
```

```
ima = 0;

for j = 1:k-l,
  rea = rea + 0.5*(y(j)*sin(w0*t(j))+y(j+1)*sin(w0*t(j+1)))*(t(j+1)-t(j));
  ima = ima + 0.5*(y(j)*cos(w0*t(j))+y(j+1)*cos(w0*t(j+1)))*(t(j+1)-t(j));
end

rea=rea*2/tp;
ima=ima*2/tp;
tp1vect(i) = tp;
A0 = sqrt(rea^2+ima^2);
A01vect(i) = A0;
Kp = Kp*4*d/(Am*A0*pi);
Kpvect(i) = Kp;

[re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
points = re+imag*im;
plot(points);
hold on;
plot(points1);
plot([-1/Am+imag -1/Am-imag]);
plot([cos(fm1)+imag*sin(fm1) 0]);
axis([-1 1 -1 1]);
grid
hold off;
pause (1);


eps = 4*d*sin(fm)/pi;

[t,x,y] = gear ('rele',60,[],[1e-3,1e-5,0.01,0,3,0]);
a = size(yout,1);

j = a;
while ((yout(j,3) < 0) | (yout(j-1,3) > 0))
  j = j-1;
end
k = j;

t2 = yout(k,1);

j = k-1;
while ((yout(j,3) < 0) | (yout(j-1,3) > 0))
  j = j-1;
end
l = j;

t1 = yout(l,1);

tp = t2-t1;
tp2vect(i) = tp;

w0 = 2*pi/tp;

t = yout(l:k,1)-t1;
y = yout(l:k,2);

rea = 0;
ima = 0;

for j = 1:k-l,
  rea = rea + 0.5*(y(j)*sin(w0*t(j))+y(j+1)*sin(w0*t(j+1)))*(t(j+1)-t(j));
  ima = ima + 0.5*(y(j)*cos(w0*t(j))+y(j+1)*cos(w0*t(j+1)))*(t(j+1)-t(j));
end

rea=rea*2/tp;
ima=ima*2/tp;
tp1vect(i) = tp;
A0 = sqrt(rea^2+ima^2);
```

```
  f2 = atan(ima/rea);
  f21 = asin (sin (fm)*4*d/A0/pi);
  f2 = (f2+f21)/2;


  f2vect(i) = f2;
  f2nn = pi/2 - asin (sin (fm)/(tan (f2-pi)));
  w2 = 2*pi/tp;

  Tix = 1/(w2*tan(atan(1/(w2*Ti))+f2-fm));

   if ((Tix/Ti > 2) | (Tix/Ti < 0.5))
      Ti = Tix;

  else

    if (f2 < fm)
      less = Ti;
      lessf = f2;
      if (more == 0)
        Ti = 2*Ti;
      else
        a1 = Ti/more;
        Ti = more*a1^((moref-fm)/(moref-f2));
      end

    else
      more = Ti;
      moref = f2;
      if (less == 0)
        Ti = Ti/2;
      else
        a1 = less/Ti;
        Ti = Ti*a1^((f2-fm)/(f2-lessf));
      end
    end
  end

  [re,im] = nyquist([Kp*Ti Kp],[2*Ti 5*Ti 4*Ti Ti 0],w);
  points = re+imag*im;
  plot(points);
  hold on;
  plot(points1);
  plot([-1/Am+1.5*imag -1/Am-1.5*imag]);
  plot([cos(fm1)+imag*sin(fm1) 0]);
  plot([-1.5-imag*sin(fm) 1.5-imag*sin(fm)]);
  axis([-1.5 1.5 -1.5 1.5]);
  grid
  hold off;
  pause (1);

  w=w';
  i = 1;
  while (im(i+1) < -sin(fm))
    i = i+1;
  end
  rex1 = re(i)+(re(i+1)-re(i))*(sin(fm)+im(i))/(im(i)-im(i+1));
  f02 = atan(sin(fm)/(-rex1));



end

Kp = Kptmp;
Ti = Titmp;

subplot(211);
semilogy([0:11],rez(:,2))
grid
```

```
title('Ti [s]')
subplot(212)
plot([0:11],rez(:,1),[0:11],rez(:,3),'--')
grid
axis([0,12,-10,10])
title('__ Kp, -- f [deg]')
xlabel('Iterations')
```

where RELE.M represents next scheme in SIMULINK: