



LUND UNIVERSITY

The Dominant Pole Design Toolbox - The Matlab Code

Persson, Per

1992

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Persson, P. (1992). *The Dominant Pole Design Toolbox - The Matlab Code*. (Technical Reports TFRT-7498). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

1

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

ISSN 0280-5316
ISRN LUTFD2/TFRT--7498--SE

The Dominant Pole Design Toolbox – the Matlab Code

Per Persson

Department of Automatic Control
Lund Institute of Technology
December 1992

Department of Automatic Control Lund Institute of Technology P.O. Box 118 S-221 00 Lund Sweden		Document name	
		Date of issue December 1992	
		Document Number ISRN LUTFD2/TFRT--7498--SE	
Author(s) Per Persson		Supervisor	
		Sponsoring organisation	
Title and subtitle The Dominant Pole Design Toolbox – the Matlab Code			
Abstract This report contains the listing of the Matlab functions presented in the report Persson, P.: <i>"The Dominant Pole Design Toolbox,"</i> TFRT-7497.			
Key words			
Classification system and/or index terms (if any)			
Supplementary bibliographical information			
ISSN and key title 0280-5316			ISBN
Language English	Number of pages 39	Recipient's notes	
Security classification			

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

1. Introduction

This report contains the source code listing of the .m files which are the Dominant Pole Design Toolbox. The files can be run on Matlab Version 4.0 on a Sparcstation ELC. The version listed here are the version of December 1992.

2. Source Code Listing

amarg

```
function [am, vx] = amarg(cstr, pstr, ws, tol)

%AMARG Computes the amplitude margin of a system.
%
% [am, vx] = AMARG(cstr, pstr, ws, tol)
% Input arguments:
% cstr - the controller expressed as a string
% pstr - the process expressed as a string
% ws - the frequency interval where the amplitude margin is found
% tol - the tolerance of the solution (default: deftol)
% Output arguments:
% am - the amplitude margin
% vx - the frequency for the amplitude margin

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 15:58:10 1992

if nargin==2,
    ws = logspace(-2, 2, 200)'; tol = deftol;
elseif nargin==3,
    tol = deftol;
end;
vx = psolveol(cstr, pstr, -pi, ws, tol);
am = 1/abs(evals(cstr, i*vx)*evals(pstr, i*vx));
```

asolvecl

```
function vx = asolvecl(cstr, pstr, y, ws, tol)

%ASOLVECL Find the solution of  $|gc(i\omega)*gp(i\omega)/(1+gc(i\omega)*gp(i\omega))| = y$ 
%
% vx = ASOLVECL(cstr, pstr, y, ws, tol)
% Input arguments:
% cstr - the controller expressed as a string
% pstr - the process expressed as a string
% y - the value of the amplitude
% ws - the frequency interval where the amplitude margin is found
% tol - the tolerance of the solution (default: deftol)
% Output arguments:
% vx - the solution frequency

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 15:58:10 1992

if nargin==4, tol = deftol; end;
tmp = ['abs(closeit(evals('' cstr '', i*x).*evals('' pstr '', i*x)))'];
vx = solve(tmp, y, ws, 1, tol);
```

asolveol

```
function vx = asolveol(cstr, pstr, y, ws, tol)

%ASOLVEOL Find the solution of  $|gc(i\omega)*gp(i\omega)| = y$ 
%
% vx = ASOLVEOL(cstr, pstr, y, ws, tol)
% Input arguments:
% cstr - the controller expressed as a string
% pstr - the process expressed as a string
% ws - the frequency interval where the amplitude margin is found
% tol - the tolerance of the solution (default: deftol)
```

```

%           Output arguments:
%           wx - the solution frequency

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 15:58:09 1992

if nargin==4, tol = deftol; end;
tmp = ['abs(evals('' cstr '', i*x)).*evals('' pstr '', i*x))'];
wx = solve(tmp, y, ws, 1, tol);



---


betades


---



function b = betades(pstr, con, mp, tol)

%BETADES Design of the set point weighting factor, beta. beta is chosen
%         such that Mp assumes a specified value.
%
%         b = BETADES(pstr, con, mp, tol)
%         pstr - the process expressed as a string
%         con - the controller, in the standard form
%         mp - desired Mp value (default: 1.001)
%         tol - the tolerance of the solution (default: deftol());
%         Output arguments:
%         b - beta, the set point weighting factor

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Nov 12 16:34:34 1992

if nargin==3,
    tol = deftol;
elseif nargin==2,
    mp = 1.001;
    tol = deftol;
end;

mp1 = mpbeta(pstr, getk(con), getti(con), gettd(con), 0);
if mp1 > mp + eps,
    b = 0;
else
    st = ['mpbeta(' qstring(pstr) ', ' numtostr(getk(con)) ', ' ,...
          numtostr(getti(con)) ', ' numtostr(gettd(con)) ', x)'];
    b = solve(st, mp, linspace(0, 5, 20), 1, tol);
end;

```

closeit

```

function res = closeit(f)

%CLOSEIT Given a frequency response f, CLOSEIT(f) computes the frequency
%         response of the closed system. f may be the frequency response
%         generated from FRC, or a one column matrix of complex numbers.

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Sep 26 11:12:52 1992

```

```

tmp = [];
if cols(f)>1, tmp = f(:, 1); f(:, 1) = []; end;

res = f./(1 + f);

if ~isempty(tmp), res = [tmp res]; end;

```

cols

```

function res = cols(matrix)

%COLS Returns the number of columns of a matrix.

```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Sep 29 08:04:51 1992
```

```
[x, res] = size(matrix);
```

con2str

```
function str = con2str(k, ti, td, n)
```

```
%CON2STR Converts the controller data structure into a text string.
```

```
%
%      str = CON2STR(k)
%      str = CON2STR(k, ti)
%      str = CON2STR(k, ti, td)
%      str = CON2STR(k, ti, td, n)
%      str = CON2STR(con)
%      str = CON2STR(con, n)
%      Input arguments:
%      k, ti, td - the PID parameters
%      n          - the filter factor
%      con        - the controller data structure from the design routines
%      Output arguments:
%      str        - the controller expressed as a string
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 18 10:48:16 1992
```

```
for ix = 1:rows(k),
    if nargin==1,
        if cols(k)>1,
            %      k = k(rows(k), :);
            tdp = gettd(k(ix, :)); tip = getti(k(ix, :)); kip = getki(k(ix, :));
            kdp = getkd(k(ix, :));
            kp = getk(k(ix, :)); np = Inf;
        else
            kp = k(ix); tip = Inf; tdp = 0; np = Inf;
        end;
    elseif nargin==2,
        if cols(k)>1,
            %      k = k(rows(k), :);
            np = ti;
            tdp = gettd(k(ix, :)); tip = getti(k(ix, :)); kip = getki(k(ix, :));
            kdp = getkd(k(ix, :));
            kp = getk(k(ix, :));
        else
            kp = k(ix); tip = ti(ix); tdp = 0; np = Inf;
        end;
    elseif nargin==3,
        kp = k(ix); tip = ti(ix); tdp = td(ix);
        np = Inf;
    elseif nargin==4,
        kp = k(ix); tip = ti(ix); tdp = td(ix);
        np = n(max(ix, rows(n)));
    end;

    kpart = sprintf('%.10f)*(1 ', kp);
    tipart = ''; if tip~=Inf, tipart = sprintf('+(s*(%.10f)).^(-1)', tip); end;
    tdp = '';
    if tdp~=0, tdp = sprintf('+(%.10f)*s', tdp); end;
    if ~isinf(np),
        tdp = [tdpart sprintf('./(1 + s*(%.10f)/(%.10f))', tdp, np)];
    end;
    if kp==0 & tip==0, % This is an I controller!
        kpart = '';
        tipart = sprintf('%.10f)*s.^(-1)', kip);
        tdp = '';
    end;
```

```

if ix==1
    str = [kpart, tipart, tdpert ' '];
else
    str = str2mat(str, [kpart, tipart, tdpert ' ']);
end;
end;
end;

```

convert

```

function str = convert(pstr)

%CONVERT Convert a string describing a process to a string that accept
%         vector arguments when evaluated.
%
%         str = CONVERT(pstr)

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 6 11:12:02 1992

tmp = pstr;
p2 = tmp(1); tmp(1) = [];

while tmp,
    p1 = p2;
    p2 = tmp(1);
    tmp(1) = [];
    if p1=='.' & (p2=='*' | p2=='^' | p2=='/' | p2=='\' | p2==''),
        str = [str p1 p2];
        p2 = tmp(1); tmp(1) = [];
    elseif (p1=='*' | p1=='^' | p1=='/' | p1=='\' | p1==''),
        str = [str ' ' p1];
    else
        str = [str p1];
    end;
end;
str = [str p2];

```

defprint

```

function dp = defprint()

%DEFPRT Get the default printing status to the optimizers and
%         the equation solvers. Nice to watch while they are working.
%         The default printing status is set to 1, but can be overridden
%         by the global variable GPRINT (if it exists).

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Wed Oct 21 17:12:00 1992

global GPRINT
if exist('GPRINT'),
    dp = GPRINT;
else
    dp = 1;
end;

```

deftol

```

function dt = deftol()

%DEFTOL Get the default tolerance for all routines in the DPD toolbox.
%         The default tolerance is set to 1e-4, but can be overridden by
%         the global variable GTOL (if it exists).

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%

```

%LastEditDate : Wed Oct 21 17:10:27 1992

```
global GTOL
if exist('GTOL'),
    dt = GTOL;
else
    dt = 1e-4;
end;
```

dpamhlp4

```
function res = dpamhlp4(pstr, contype, w, zs, p1, tol)
```

%DPAMHLP4 A help routine used in DPTABLE4.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden

%
%LastEditDate : Tue Nov 17 16:50:18 1992

```
zs = zs(:);
res = [];
ws = logspace(log10(w)-1, log10(w)+1, 100);
for ix=zs',
    [a, x] = amarg(con2str(dptable1(pstr, contype, w, ix, p1)), pstr, ws, tol);
    res = [res; a];
end;
```

dpi

```
function con = dpi(pstr, w0)
```

%DPI Computes the dominant pole design for an I controller
% with poles in -w0. w0 can be a vector

%
% con = DPI(pstr, w0)
% Input arguments:
% pstr - the process expressed as a string
% w0 - the locations of the poles
% Output arguments:
% con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden

%
%LastEditDate : Sat Nov 7 10:30:41 1992

```
ki = w0./evals(pstr, -w0);
os = ones(size(w0));
con = [w0 NaN*os NaN*os 0*os ki 0*os 0*os 0*os 2*os];
```

dpmshlp2

```
function res = dpmshlp2(pstr, contype, w, zs, p1, tol)
```

%DPMSHLP2 A help routine used in DPTABLE2.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden

%
%LastEditDate : Fri Dec 4 11:13:22 1992

```
zs = zs(:);
res = [];
ws = logspace(log10(w)-1, log10(w)+1, 2000);
for ix=zs',
    [m, x] = mscl(con2str(dptable1(pstr, contype, w, ix, p1)), pstr, ws, tol);
    res = [res; m];
end;
```

dpp

```
function con = dpp(pstr, w0)
```

```
%DPP Dominant pole design of a P controller with poles in -w0.
%
%   con = DPP(pstr, w0)
%   Input arguments:
%   pstr - the process expressed as a string
%   w0   - the locations of the poles
%   Output arguments:
%   con  - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 10:30:41 1992

k = -evals(pstr, -w0).^(-1);
os = ones(size(w0));
con = [w0 NaN*os NaN*os k 0*os Inf*os 0*os 0*os 1*os];
```

dppd

```
function con = dppd(pstr, w0, z0)

%DPPD Dominant pole design of a PD controller with poles in
%   w0(-z0 +- i*sqrt(1 - z0*z0)). w0 can be a vector.
%
%   con = DPPD(pstr, w0, z0)
%   Input arguments:
%   pstr - the process expressed as a string
%   w0   - the distance from the origin to the poles
%   z0   - the relative damping of the poles
%   Output arguments:
%   con  - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 27 09:08:58 1992
```

```
if z0 < 1,
    sqz = sqrt(1 - z0*z0);
    p1 = w0*(-z0 + i*sqz);
    tmp = evals(pstr, p1);
    a = real(tmp);
    b = imag(tmp);
    n = (a.^2+b.^2)*sqz;
    k = (-sqz*a+b*z0)./n;
    kd = b./w0./n;
elseif z0 > 1,
    p1 = w0*(-z0 + sqrt(z0*z0 - 1));
    p2 = w0*(-z0 - sqrt(z0*z0 - 1));
    a = -1./evals(pstr, p1);
    b = -1./evals(pstr, p2);
    kd = (b - a)./(p1 - p2);
    k = a - kd.*p1;
end;
os = ones(size(w0));
con = [w0 z0*os NaN*os k 0*os Inf*os kd kd./k 5*os];
```

dppl

```
function con = dppl(pstr, w0, z0)

%DPPI Dominant pole design of a PI controller with poles in
%   w0(-z0 +- i*sqrt(1 - z0*z0)). w0 can be a vector.
%
%   con = DPPI(pstr, w0, z0)
%   Input arguments:
%   pstr - the process expressed as a string
%   w0   - the distance from the origin to the poles
%   z0   - the relative damping of the poles
%   Output arguments:
%   con  - the controller data structure
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 27 09:03:45 1992
```

```
if z0 < 1,
    zp = sqrt(1 - z0*z0);
    p1 = w0*(-z0 + i*zp);
    tmp = evals(pstr, p1);
    a = real(tmp);
    b = imag(tmp);
    n = zp*(a.^2+b.^2);
    k = -(b*z0+a*zp)./n;
    ki = -b.*w0./n;
elseif z0 > 1,
    p1 = w0*(-z0 + sqrt(z0*z0 - 1));
    p2 = w0*(-z0 - sqrt(z0*z0 - 1));
    a = -p1./evals(pstr, p1);
    b = -p2./evals(pstr, p2);
    k = (a - b)./(p1 - p2);
    ki = a - k.*p1;
end;
os = ones(size(w0));
con = [w0 z0*os NaN*os k ki k./ki 0*os 0*os 4*os];
```

dppid

```
function con = dppid(pstr, w0, z0, alpha0)
```

```
%DPPID Dominant pole design of a PID controller with poles in
% w0(-z0 +- i*sqrt(1 - z0*z0)) and -w0*alpha0. w0 can be a vector.
%
% con = DPPID(pstr, w0, z0, alpha0)
% Input arguments:
% pstr - the process expressed as a string
% w0 - the distance from the origin to the poles
% z0 - the relative damping of the poles
% alpha0 - w0*alpha0 = the distance from the origin to the third pole
% Output arguments:
% con - the controller data structure
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 27 17:32:04 1992
```

```
if z0 < 1,
    sqz = sqrt(1 - z0^2);
    p1 = w0*(-z0 + i*sqz);
    p2 = w0*(-z0 - i*sqz);
    p3 = -w0*alpha0;
    tmp = evals(pstr, p1);
    a = real(tmp);
    b = imag(tmp);
    c = evals(pstr, p3);

    a2b2 = a.^2 + b.^2;
    ac = a.*c;

    d = sqz*c.*a2b2*(1 - 2*alpha0*z0 + alpha0^2);

    k = -(sqz*(-2*alpha0*z0*a2b2 + (1 + alpha0^2)*ac) + ...
        z0*b.*c*(alpha0^2 - 1))./d;
    ki = -(alpha0*w0.*((alpha0 - z0)*b.*c + sqz*(ac - a2b2)))./d;
    kd = -((alpha0*z0 - 1)*b.*c + alpha0*sqz*(ac - a2b2))./w0./d;
elseif z0 > 1,
    p1 = w0*(-z0 + sqrt(z0*z0 - 1));
    p2 = w0*(-z0 - sqrt(z0*z0 - 1));
    p3 = -alpha0*w0;
    a = -p1./evals(pstr, p1);
    b = -p2./evals(pstr, p2);
    c = -p3./evals(pstr, p3);
```

```

tmp = (p1.*p2 - p1.*p3 + p3.^2 - p2.*p3).*(p1 - p2);
kd = (p2.*a - c.*p2 + p1.*c + p3.*b - p1.*b - a.*p3)./tmp;
k = -(a.*p2.^2 - c.*p2.^2 + c.*p1.^2 - b.*p1.^2 + b.*p3.^2 - a.*p3.^2)./tmp;
ki = (a.*p3.*p2.^2 - p1.*c.*p2.^2 + c.*p2.*p1.^2 - a.*p2.*p3.^2 - ...
      p3.*b.*p1.^2 + b.*p1.*p3.^2)./tmp;
end;
os = ones(size(w0));
con = [w0 z0*os alpha0*os k ki k./ki kd kd./k 5*os];

```

dppid2

```
function con = dppid2(pstr, w0, z0, kd0)
```

```
%DPPID2 Dominant pole design of a PID controller with poles in
% w0(-z0 +- i*sqrt(1 - z0*z0)) and a given kd.
```

```
%
% con = DPPID2(pstr, w0, z0, kd0)
% Input arguments:
% pstr - the process expressed as a string
% w0 - the distance from the origin to the poles
% z0 - the relative damping of the poles
% kd0 - a specified kd of the controller
% Output arguments:
% con - the controller data structure
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
```

```
%
%LastEditDate : Fri Nov 27 10:00:10 1992
```

```

if z0 < 1,
    zq0 = sqrt(1 - z0^2);
    p1 = w0*(-z0 + i*zq0);
    tmp = evals(pstr, p1);
    a = real(tmp);
    b = imag(tmp);
    n = zq0*(a.^2+b.^2);
    k = 2*kd0*w0*z0 -(b*z0+a*zq0)./n;
    ki = kd0*w0.^2 - b.*w0./n;
elseif z0 > 1,
    p1 = w0*(-z0 + sqrt(z0*z0 - 1));
    p2 = w0*(-z0 - sqrt(z0*z0 - 1));
    a = -p1./evals(pstr, p1) - p1.^2*kd;
    b = -p2./evals(pstr, p2) - p2.^2*kd;
    k = (a - b)./(p1 - p2);
    ki = a - k.*p1;
end;

```

```

os = ones(size(w0));
kd = kd0*os;
con = [w0 z0*os NaN*os k ki k./ki kd kd./k 6*os];

```

dppmhl3

```
function res = dppmhl3(pstr, contype, w, zs, p1, tol)
```

```
%DPPMHLP3 A help routine used in DPTABLE3.
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
```

```
%
%LastEditDate : Tue Nov 17 16:49:59 1992
```

```

zs = zs(:);
res = [];
ws = logspace(log10(w)-1, log10(w)+1, 100);
for ix=zs',
    [p, x] = pmarg(con2str(dptable1(pstr, contype, w, ix, p1)), pstr, ws, tol);
    res = [res; p];
end;

```

dptable1

```
function con = dptable1(pstr, contype, wOs, zO, p1)

%DPTABLE1 Computes the controller parameters from specified poles or
%          other parameters. This is an interface routine to all the
%          fundamental routines.
%
%          con = DPTABLE1(pstr, contype, wOs, zO, p1)
%          Input arguments:
%          pstr - the process transfer function as a string
%          contype - 'p', 'i', 'pi', 'pd', 'pid', 'pid1', 'pid2'
%          wOs - the distance of the poles from the origin
%          zO - the relative damping of the poles
%          p1 - alpha0 or kd0 depending on which type of PID controller
%              is specified.
%          Output arguments:
%          con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 10:37:21 1992

wOs = wOs(:);
if strcmp(contype, 'p'), % P 1
    con = dpp(pstr, wOs);
elseif strcmp(contype, 'i'), % PI 2
    con = dpi(pstr, wOs);
elseif strcmp(contype, 'pd'), % PD 3
    con = dppd(pstr, wOs, zO);
elseif strcmp(contype, 'pi'), % PI 4
    con = dppi(pstr, wOs, zO);
elseif strcmp(contype, 'pid') | strcmp(contype, 'pid1'), % PID 5
    alpha = p1;
    con = dppid(pstr, wOs, zO, alpha);
elseif strcmp(contype, 'pid2'), % PID 6
    kd0 = p1;
    con = dppid2(pstr, wOs, zO, kd0);
else
    con = [];
end;
```

dptable2

```
function con = dptable2(pstr, contype, wOs, ms, p1, tol)

%DPTABLE2 Computes the controller parameters from specified poles or
%          other parameters. This routine makes use of DPTABLE1 and
%          solve an equation of Ms with respect to zO.
%
%          con = DPTABLE2(pstr, contype, wOs, ms, p1, tol)
%          Input arguments:
%          pstr - the process transfer function as a string
%          contype - 'p', 'i', 'pi', 'pd', 'pid', 'pid1', 'pid2'
%          wOs - the distance of the poles to the origin
%          ms - the maximum of the sensitivity function of the
%              control system
%          p1 - alpha0 or kd0 depending on which type of PID
%              controller is specified.
%          Output arguments:
%          con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 18 10:48:16 1992

if nargin==5,
    tol = deftol;
```

```

elseif nargin==4,
    tol = deftol;
    p1 = -100;
end;
wOs = wOs(:);
con = [];
for ix=wOs',
    st = ['dpmshlp2(' qstring(pstr) ', ' qstring(contype) ', ' numtostr(ix) ...
        ', x, ' numtostr(p1) ', ' numtostr(tol) ')'];
    tmp = solve(st, ms, linspace(0.01, 2, 20), 1, tol);
    if ~isempty(tmp),
        con = [con; dptable1(pstr, contype, ix, tmp, p1)];
    end;
end;
end;

```

dptable3

```
function con = dptable3(pstr, contype, wOs, pm, p1, tol)
```

```

%DPTABLE3 Computes the controller parameters from specified poles or
% other parameters. This routine makes use of DPTABLE1 and
% solve an equation of Pm with respect to x0.
%
% con = DPTABLE3(pstr, contype, wOs, pm, p1, tol)
% Input arguments:
% pstr - the process transfer function as a string
% contype - 'p', 'i', 'pi', 'pd', 'pid', 'pid1', 'pid2'
% wOs - the distance of the poles to the origin
% pm - the phase margin
% p1 - alpha0 or kd0 depending on which type of PID
% controller is specified.
% Output arguments:
% con - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 18 10:48:16 1992

```

```

if nargin==5,
    tol = deftol;
elseif nargin==4,
    tol = deftol;
    p1 = -100;
end;
wOs = wOs(:);
con = [];
for ix=wOs',
    st = ['dppmshlp3(' qstring(pstr) ', ' qstring(contype) ', ' numtostr(ix) ...
        ', x, ' numtostr(p1) ', ' numtostr(tol) ')'];
    tmp = solve(st, pm, linspace(0.01, 0.99, 20), 1, tol);
    if ~isempty(tmp),
        con = [con; dptable1(pstr, contype, ix, tmp, p1)];
    end;
end;
end;

```

dptable4

```
function con = dptable4(pstr, contype, wOs, am, p1, tol)
```

```

%DPTABLE4 Computes the controller parameters from specified poles or
% other parameters. This routine makes use of DPTABLE1 and
% solve an equation of Am with respect to x0.
%
% con = DPTABLE4(pstr, contype, wOs, am, p1, tol)
% Input arguments:
% pstr - the process transfer function as a string
% contype - 'p', 'i', 'pi', 'pd', 'pid', 'pid1', 'pid2'
% wOs - the distance of the poles to the origin
% am - the phase margin
% p1 - alpha0 or kd0 depending on which type of PID
% controller is specified.

```

```
%      Output arguments:
%      con      - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 18 10:48:16 1992

if nargin==5,
    tol = deftol;
elseif nargin==4,
    tol = deftol;
    p1 = -100;
end;
wOs = wOs(:);
con = [];
for ix=wOs',
    st = ['dpamhlp4(' qstring(pstr) ', ' qstring(contype) ', ' numtostr(ix) ...
        ', x, ' numtostr(p1) ', ' numtostr(tol) ')'];
    tmp = solve(st, am, linspace(0.01, 0.99, 20), 1, tol);
    if ~isempty(tmp),
        con = [con; dptable1(pstr, contype, ix, tmp, p1)];
    end;
end;
```

evals

```
function r = evals(str, s)

%EVALS  EVALS(str, s) evaluates the string str where the variable "s"
%      in the string is bound to the argument s.
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Nov 10 13:17:07 1992
```

```
r = eval(str);
```

evalx

```
function r = evalx(str, x)

%EVALX  EVALX(str, x) evaluates the string str where the variable "x"
%      in the string is bound to the argument x.
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 6 14:30:44 1992
```

```
r = eval(str);
```

geta0

```
function a0 = geta0(con)

%GETA0  Access function to get alpha0 (variable or vector) from the controller
%      data structure.
%
%      a0 = GETA0(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:36 1992
```

```
a0 = con(:,3);
```

getk

```
function k = getk(con)
```

```
%GETK Access function to get k (variable or vector) from the controller
% data structure.
%
% k = GETK(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:35 1992
```

```
k = con(:, 4);
```

getkd

```
function kd = getkd(con)
```

```
%GETKD Access function to get kd (variable or vector) from the controller
% data structure.
%
% kd = GETKD(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:35 1992
```

```
kd = con(:, 7);
```

getki

```
function ki = getki(con)
```

```
%GETKI Access function to get ki (variable or vector) from the controller
% data structure.
%
% ki = GETKI(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:35 1992
```

```
ki = con(:, 5);
```

gettd

```
function td = gettd(con)
```

```
%GETTD Access function to get td (variable or vector) from the controller
% data structure.
%
% td = GETTD(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:35 1992
```

```
td = con(:, 8);
```

getti

```
function ti = getti(con)
```

```
%GETTI Access function to get ti (variable or vector) from the controller
% data structure.
%
% ti = GETTI(con)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:35 1992
```

```
ti = con(:, 6);
```

getw0

```
function w0 = getw0(con)

%GETWO Access function to get w0 (variable or vector) from the controller
%      data structure.
%
%      w0 = GETWO(con)

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:44:34 1992

w0 = con(:, 1);
```

getz0

```
function z0 = getz0(con)

%GETZO Access function to get z0 (variable or vector) from the controller
%      data structure.
%
%      z0 = GETZO(con)

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 9 16:49:00 1992

z0 = con(:, 2);
```

ides

```
function con = pides(pstr, w0s, tol)

%IDES Design of a I controller with dominant poles with w0 chosen to
%      maximize ki.
%
%      con = IDES(pstr, w0s, tol)
%      Input arguments:
%      pstr - the process expressed as a string
%      w0s - array of w0 values, the interval must contain w0max
%      tol - the tolerance of the solution (default: deftol)
%      Output arguments:
%      con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 20 14:16:49 1992

if nargin==2, tol = deftol; end;
w0s = w0s(:);

tmp = ['getki((dptable1(' qstring(pstr) ', 'i'', x)))'];
wx = opt(tmp, w0s, tol);
if ~isempty(wx),
    con = dptable1(pstr, 'i', wx);
else
    error('No controller found with ides.');
```

kdguess

```
function kd = kdguess(pstr, w0s, z0)

%KDGUESS Makes an estimation of maximal allowable kd in a pid controller.
%      See. P. Persson: Towards Autonomous PID control, page 104.
%      The guess may fail.
%
%      kd = KDGUESS(pstr, w0s, z0)
%      Input arguments:
%      pstr - the process expressed as a string
```

```
%      w0s - an array of w0 values
%      z0 - the relative damping of the poles
%      Output arguments:
%      kd - the maximum kd
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 10:46:36 1992
```

```
w0s = w0s(:);
r = dptable1(pstr, 'pi', w0s, z0);
ki = getki(r);
kip = diff(ki)./diff(w0s);
wsp = w0s(1:(length(w0s)-1));
tmpx = -kip./(2*wsp); m = locmax(tmpx);
kd = tmpx(m(1));
```

locmax

```
function res = locmax(array)
```

```
%LOCMAX Finds the local maximum of a vector.
%
%      res = locmax(array)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Oct 2 11:13:51 1992
```

```
res = [];
if (array(1) > array(2)), res = [1]; end;
for ix = [2:1:(length(array) - 1)],
    if ((array(ix) > array(ix - 1)) & (array(ix) > array(ix + 1))),
        res = [res ix];
    end;
end;
if array(length(array)) > array(length(array)-1), res = [res length(array)]; end;
if rows(array) == 1, res = res'; end;
```

makep

```
function str = makep(sysnr, p1, p2, p3, p4)
```

```
%MAKEP A routine for generating transfer function strings for
%      a number of standard systems.
```

```
%
%      str = MAKEP(sysnr, p1, p2, p3)
%      str = MAKEP(arr)
%
%      arr - may be an array containing [sysnr p1 ... p4]
%
%      sysnr = 1 => p1*exp(-s p2)/(s p3 + 1)
%      sysnr = 2 => 1/(s+1)^p1
%      sysnr = 3 => (1 - s p1)/(s + 1)^3
%      sysnr = 4 => exp(-s p1)/(s p2 + 1)(s p3 + 1)
%      sysnr = 5 => exp(-s p1) p2^2/(s^2 + 2 s p2 p3 + p2^2)
%      sysnr = 6 => exp(-s p1) p2^3 p4/(s + p2 p4)(s + 2 s p2 p3 + p2^2)
%      sysnr = 7 => 1/(s + 1)(s p1 + 1)(s p1^2 + 1)
%      sysnr = 8 => 1/(s + 1)(s p1 + 1)(s p1^2 + 1)(s p1^3 + 1)
%      sysnr = 9 => (1 - s p1/2)/(1 + s p1/2)(1 + s p2)
%      sysnr = 10 => exp(-s p1)/(1 + s p2)(1 + s p3)(1 + s p4)
%      sysnr = 11 => exp(-s p1)/s(s p2 + 1)
%      sysnr = 12 => exp(-s p1)/s(s p2 + 1)(s p3 + 1)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Dec 19 11:35:45 1992
```

```
l = length(sysnr);
```

```

if l > 1,
    p1 = sysnr(2);
    if l > 2, p2 = sysnr(3); end;
    if l > 3, p3 = sysnr(4); end;
    if l > 4, p4 = sysnr(5); end;
    sysnr = sysnr(1);
end;

if sysnr == 1
    str = ['(' numtostr(p1) '*exp(-' numtostr(p2) '*s)/(1+s*( ...
        numtostr(p3) '))')'];
elseif sysnr == 2
    str = sprintf('((1 + s).^(-%.10g))', p1);
elseif sysnr == 3
    str = sprintf('((1 - %.10g*s)/((1 + s).^3))', p1);
elseif sysnr == 4
    str = ['(exp(-s*' numtostr(p1) ')/(1+( ' numtostr(p2) ') *s)/(1+( ' ...
        numtostr(p3) ') *s))' ];
elseif sysnr == 5
    str = ['(exp(-s*' numtostr(p1) ') * ' numtostr(p2) '^2/(s.^2 + 2*s*' ...
        numtostr(p2) '*' numtostr(p3) '+' numtostr(p3) '^2))' ];
elseif sysnr == 6
    if p1==0, st1 = []; else st1 = ['exp(-s*' numtostr(p1) ') *']; end;
    str = ['(' st1 numtostr(p4*p2^3) ' ./ ' ...
        '(s + ' numtostr(p2*p4) ') ./ (s.^2 + ' ...
        numtostr(2*p2*p3) '*s + ' numtostr(p2^2) '))' ];
elseif sysnr == 7
    str = sprintf('(((s+1).*(s*%.10g+1).*(s*%.10g^2+1)).^(-1))', p1, p1);
elseif sysnr == 8
    str = sprintf('(((s+1).*(s*%.10g+1).*(s*%.10g^2+1).*(s*%.10g^3+1)).^(-1))', ...
        p1, p1, p1);
elseif sysnr == 9
    str = ['((1-' numtostr(p1) '*s/2)/(1+' numtostr(p1) ...
        '*s/2)/(1+' numtostr(p2) '*s))'];
elseif sysnr == 10
    str = ['(exp(-' numtostr(p1) '*s)/(1 + ' numtostr(p2) ...
        '*s)/(1 + ' numtostr(p3) '*s)/(1 + ' numtostr(p4) '*s))'];
elseif sysnr == 11
    str = ['(exp(-' numtostr(p1) '*s)/s/(1 + ' numtostr(p2) '*s))'];
elseif sysnr == 12
    str = ['(exp(-' numtostr(p1) '*s)/s/(1 + ' numtostr(p2) ...
        '*s)/(1 + ' numtostr(p3) '*s))'];
end;

```

mpbeta

```

function mpr = mpbeta(pstr, k, ti, td, betas, tol)

%MPBETA Compute the M_p value given process, controller, and beta.
%      A help routine for BETADESIGN.
%
%      mpr = MPBETA(pstr, k, ti, td, betas, tol)

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Mon Oct 12 15:00:03 1992

if nargin==5, tol = deftol; end;

betas=betas(:)';
mpr = [];
for ix = betas,
    tmp1 = sprintf('(%f)*(%f) + (s*(%f)).^(-1))', k, ix, ti);
    gcl2 = [ tmp1 '*' pstr './(1 + ' con2str(k, ti, td) '*' pstr ')'];

    s2 = ['abs(evals(' ' ' gcl2 ' ' ' , i*x))'];
    [wr, mp] = opt(s2, logspace(-2, 2, 100), tol);
    if isempty(wr), wr = 0; mp = 1; end;

    mpr = [mpr; mp];
end;

```

mscl

```
function [ms, ws] = mscl(cstr, pstr, wx, tol)

%MSCL Computation of the maximum of the sensitivity function, and the
% frequency at which it occurs.
%
% ms = max |1/(1 + gc(iw)gp(iw))|
% w
%
% [ms, ws] = MSCL(cstr, pstr, ws, tol)
% Input arguments:
% cstr - the controller expressed as a string
% pstr - the process expressed as a string
% wx - the frequency interval where the maximum sensitivity is found
% tol - the tolerance of the solution (default: deftol)
% Output arguments:
% ms - the maximum of the sensitivity function
% ws - the frequency of the maximum

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 4 13:45:22 1992

if nargin==3, tol = deftol; end;

for ix=1:rows(cstr),
    s1 = ['(' cstr(ix, :) ')'.*( ' pstr ')'];
    s2 = ['abs(evals(' ' ' '(1 + ' s1 ')'.^(-1)' ' ' ', i*x))'];
    [w, m] = optg(s2, wx, tol);

% If we fail to find an optimum the it is very likely that something is wrong.

    if isempty(w),
        fprintf('mscl : cannot find a maximal Ms. Guessing Ms = 1.\n');
        % error('Cannot find a maximal Ms. ');
        m = 1; w = inf;
    end;
    ms = [ms; m];
    ws = [ws; w];
end
```

mshelp

```
function res = mshelp(pstr, w0s, z0, kds, tol)

%MSHELP Computes the Ms values with a PID controller for a range of kds
% This is a help routine used in PIDDESM2.
%
% res = MSHELP(pstr, w0s, z0, kds, tol)
% Input arguments:
% pstr - process expressed as a string
% w0s - array of w0 values, the interval must contain w0max
% z0 - the relative damping
% kds - an array of kd values
% tol - the tolerance of the solution (default: deftol)
% Output arguments:
% res - an array of corresponding Ms values

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 16:00:36 1992

if nargin==4, tol = deftol; end;
rpar = [];
for ix = 1:length(kds),
    r = piddekd(pstr, w0s, z0, kds(ix));
    if ~isempty(r),
        [b, a] = mscl(con2str(r), pstr, w0s, tol);
        rpar = [rpar; r b];
    end;
end
```

```

    end;
end;
if ~isempty(rpar), res = rpar(:, cols(rpar)); end;

```

numtostr

```

function str = numtostr(num, n)

%NUMTOSTR Number to string conversion.
%
%      str = NUMTOSTR(num, n)
%      Input arguments:
%      num - the number to convert
%      n   - number of decimals (default 10)
%      Output arguments:
%      str - num as a string

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Nov 10 11:23:52 1992

if nargin == 1, n = 10; end;
str = sprintf(['%.', sprintf('%g', n) 'g'], num);

```

opt

```

function [xsol, fx] = opt(str, xOs, tol)

%OPT Find the first maximum of an expression of 'x' defined in str,
%      where x is in the range xOs.
%
%      [xsol, fx] = OPT(str, xOs, tol)
%      Input arguments:
%      str - a function expressed as a string
%      xOs - the interval of x-guesses
%      tol - the tolerance of the solution (default: deftol)
%      Output arguments:
%      xsol - the x coordinate of the solution
%      fx   - the maximum value

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:17 1993

xOs = xOs(:);
if nargin==2, tol = deftol; end;
fi = (3 - sqrt(5))/2;
tmp = [];

f1 = evalx(str, xOs(1)); f2 = evalx(str, xOs(2)); f3 = evalx(str, xOs(3));
if (f2 > f1) & (f2 > f3), tmp = 2; end;
ix = 4;
while (ix < rows(xOs)) & isempty(tmp),
    f1 = f2;
    f2 = f3;
    f3 = evalx(str, xOs(ix));
    if (f2 > f1) & (f2 > f3), tmp = ix - 1; end;
    ix = ix + 1;
end;

%r = evalx(str, xOs);
%tmp = locmax1(r);

if isempty(tmp),
    fprintf('opt: no maximum in [%f, %f]\n', xOs(1), xOs(length(xOs)));
    xsol = []; fx = []; return;
end;

a = xOs(tmp(1) - 1);
c = xOs(tmp(1) + 1);
b = a + (c - a)*fi;

```

```

    fprintf('      %f %f %f \n', a, b, c);
    fprintf('      %f %f %f \n', evalx(str, a), evalx(str, b), evalx(str, c));

    while abs(c - a) > tol*abs((c + a)),
        xx = b + fi*(c - b);
        fxx = evalx(str, xx);
        fb = evalx(str, b);
    % fprintf('      %f \n', fx-fb);
    % fprintf('      %f %f %f \n', a, b, c);
    % fprintf('      %f %f %f \n', evalx(str, a), evalx(str, b), evalx(str, c));
    % fprintf('-----\n');
    if fxx - fb >= 0,
        a = b; b = xx; % fprintf('type 1\n');
    elseif fxx - fb == 0,
        a = b; c = xx; b = a + (c - a)*fi; % fprintf('type 2\n');
    else
        c = xx; b = a + fi*(c - a); % fprintf('type 3\n');
    end;
end;
xsol = b; fx = fb;

----- optg -----
function [xsol, fx] = optg(str, xOs, tol)

%OPTG Find the global maximum of an expression of 'x' defined in str,
%      where x is in the range xOs.
%
%      [xsol, fx] = OPTG(str, xOs, tol)
%      Input arguments:
%      str - a function expressed as a string
%      xOs - the interval of x-guesses
%      tol - the tolerance of the solution (default: deftol)
%      Output arguments:
%      xsol - the x coordinate of the solution
%      fx - the maximum value

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:17 1993

xOs = xOs(:);
if nargin==2, tol = deftol; end;
fi = (3 - sqrt(5))/2;
tmp = [];

%f1 = evalx(str, xOs(1)); f2 = evalx(str, xOs(2)); f3 = evalx(str, xOs(3));
%if (f2 > f1) & (f2 > f3), tmp = 2; end;
%ix = 4;
%while (ix < rows(xOs)) & isempty(tmp),
% f1 = f2;
% f2 = f3;
% f3 = evalx(str, xOs(ix));
% if (f2 > f1) & (f2 > f3), tmp = ix - 1; end;
% ix = ix + 1;
%end;

r = evalx(str, xOs);
[y, tmp] = max(r);

if tmp==1 | tmp==length(r),
    fprintf('opt: no maximum available in [%f, %f]\n', xOs(1), xOs(length(xOs)));
    xsol = []; fx = []; return;
%error(sprintf('opt: no local maximum in [%f, %f]\n', xOs(1), xOs(length(xOs))));
end;
%if isempty(tmp),
% fprintf('opt: no maximum available in [%f, %f]\n', xOs(1), xOs(length(xOs)));
% xsol = []; fx = []; return;
%end;

```

```

a = x0s(tmp(1) - 1);
c = x0s(tmp(1) + 1);
b = a + (c - a)*fi;

fprintf('      %f %f %f \n', a, b, c);
fprintf('      %f %f %f \n', evalx(str, a), evalx(str, b), evalx(str, c));

while abs(c - a) > tol*abs((c + a)),
    xx = b + fi*(c - b);
    fxx = evalx(str, xx);
    fb = evalx(str, b);
    % fprintf('      %f \n', fx-fb);
    % fprintf('      %f %f %f \n', a, b, c);
    % fprintf('      %f %f %f \n', evalx(str, a), evalx(str, b), evalx(str, c));
    % fprintf('-----\n');
    if fxx - fb >= 0,
        a = b; b = xx; % fprintf('type 1\n');
    elseif fxx - fb == 0,
        a = b; c = xx; b = a + (c - a)*fi; % fprintf('type 2\n');
    else
        c = xx; b = a + fi*(c - a); % fprintf('type 3\n');
    end;
end;
xsol = b; fx = fb;

```

par2con

```

function con = par2con(k, ti, td)

%PAR2CON Converts the PID controller parameters to the standard controller
%          data structure.
%
%          con = PAR2CON(k)
%          con = PAR2CON(k, ti)
%          con = PAR2CON(k, ti, td)

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Dec 15 11:45:46 1992

ns = NaN*ones(size(k));
os = zeros(size(k));

if nargin==1,
    ti = Inf; td = 0;
elseif nargin==2
    td = os;
end;

con = [ns ns ns k k./ti ti k.*td td os];

```

pddes

```

function [k, td] = pddes(pstr, w0s, z0, tol)

%PDDES Design of a PD controller with dominant poles with a specified z0
%          and w0 chosen to maximize k.
%
%          [k, td] = PDDES(pstr, w0s, z0, tol) or
%          con = PDDES(pstr, w0s, z0, tol)
%          Input arguments:
%          pstr - process expressed as a string
%          w0s - array of w0 values, the interval must contain wOmax
%          z0 - the relative damping of the dominant poles
%          tol - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, td - PD parameters
%          con - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%

```

```
%LastEditDate : Fri Nov 13 15:58:09 1992
```

```
if nargin==3, tol = deftol; end;
w0s = w0s(:);

tmp = ['getk((dptable1('' pstr, '', 'pd', x, ' numtostr(z0) ')))'];
wx = opt(tmp, w0s, tol);
if isempty(wx),
    con = dptable1(pstr, 'pd', wx, z0);
    if nargin==1,
        k = con;
    else
        k = getk(con); td = gettd(con);
    end;
else
    error('No controller found with pddes.');
```

pddesam

```
function [k, td] = pddesam(pstr, w0s, am, zguess, tol)
```

```
%PDDESAM Design of a PD controller with w0 chosen to maximize k and
%          z0 chosen to get a specified amplitude margin of the closed system.
%
%          [k, td] = PDDESAM(pstr, w0s, am, zguess, tol)
%          con = PDDESAM(pstr, w0s, am, zguess, tol)
%          Input arguments:
%          pstr - process expressed as a string
%          w0s - array of w0 values, the interval must contain w0max
%          am - the specified phase margin, in degrees
%          zguess - a guess of z0 interval (default: linspace(0.01, 2, 10))
%          tol - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, td - PD parameters
%          con - the controller data structure
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Dec 19 11:39:26 1992
```

```
lsp = linspace(0.01, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;
```

```
w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;
```

```
delete('amfunpd.m');
fn = fopen('amfunpd.m', 'w');
fprintf(fn, 'function am = amfunpd(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs'', \n');
fprintf(fn, [' con = pddes(' qstring(pstr) ', [' numtostr(w1,10) ...
    ':' numtostr(dw,10) ':' numtostr(w2,10) ']]', ix, ' ...
    numtostr(tol) ']);\n']);
fprintf(fn, [' [am, wx] = amarg(' qstring(pstr) ...
    ', con2str(con), linspace(0.2*getw0(con), 5*getw0(con), 20), ' ...
    numtostr(tol) ']);\n']);
fprintf(fn, ' tmp = [tmp; am];\n');
fprintf(fn, 'end;\nam = tmp;');
fclose(fn);
clear('amfunpd');

zx = solve('amfunpd(x)', am, zguess, 1, tol);
con = pddes(pstr, w0s, zx, tol);
```

```

if nargout==1,
    k = con;
else
    k = getk(con);
    td = gettd(con);
end;

```

pddesms

```
function [k, td] = pddesms(pstr, w0s, ms, zguess, tol)
```

```

%PDDESMS Design of a PD controller with z0 chosen to give the controller
%          specified Ms-value, and w0 chosen to maximize k.
%
%          [k, td] = PDDESMS(pstr, w0s, ms, zguess, tol)
%          con = PDDESMS(pstr, w0s, ms, zguess, tol)
%          Input arguments:
%          pstr - process expressed as a string
%          w0s - array of w0 values, the interval must contain w0max
%          ms - the specified ms value
%          zguess - a guess of z0 interval (default: linspace(0.01, 2, 10))
%          tol - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, td - PD parameters
%          con - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Dec 1 11:20:44 1992

```

```

lsp = linspace(0.01, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; end;
end;

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('msfunpd.m')
fn = fopen('msfunpd.m', 'w');
fprintf(fn, 'function ms = msfunpd(zs)\n');
fprintf(fn, 'zs=zs(:);tmp = []; \nfor ix=zs', '\n');
fprintf(fn, [' r = pddes(' qstring(pstr) ', linspace(' numtostr(w1,10) ', ' numtostr(w2,10)
', ' numtostr(lw,1) ')', ix, ' numtostr(tol) '); \n']]);
fprintf(fn, 'if ~isempty(r), w0 = getw0(r);\n');
fprintf(fn, [' [ms, wmp] = mscl(con2str(r), ' qstring(pstr) ', linspace(0.1*w0, 10*w0,
200), ' numtostr(tol) '); \n']]);
fprintf(fn, ' tmp = [tmp; ms];\n');
fprintf(fn, 'else error(''Failure in msfunpd.''); end; end;\nms = tmp; return;');
fclose(fn);
clear('msfunpd');

zx = solve('msfunpd(x)', ms, zguess, 1, tol);
if ~isempty(zx),
    con = pddes(pstr, w0s, zx);
    if nargout==1,
        k = con;
    else
        k = getk(con);
        td = gettd(con);
    end;
else
    error('No controller found with pddesms.')
end;

```

pddespm

```
function [k, td] = pddespm(pstr, w0s, pm, zguess, tol)

%PDDSPM Design of a PD controller with w0 chosen to maximize k and
%        z0 chosen to get a specified phase margin of the closed system.
%
%        [k, td] = PDDSPM(pstr, w0s, pm, zguess, tol)
%        con = PDDSPM(pstr, w0s, pm, zguess, tol)
%        Input arguments:
%        pstr - process expressed as a string
%        w0s - array of w0 values, the interval must contain w0max
%        pm - the specified phase margin, in degrees
%        zguess - a guess of z0 interval (default: linspace(0.01, 2, 10))
%        tol - the tolerance of the solution (default: deftol)
%        Output arguments:
%        k, td - PD parameters
%        con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 11 16:40:01 1992

lsp = linspace(0.01, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('pmfunpd.m');
fn = fopen('pmfunpd.m', 'w');
fprintf(fn, 'function pm = pmfunpd(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs', '\n');
fprintf(fn, [' con = pddes(' qstring(pstr) ', [' numtostr(w1,10) ':' numtostr(dw,10) ':'
numtostr(w2,10) ']', ix, ' numtostr(tol) '); \n']);
fprintf(fn, [' [pm, wx] = pmarg(' qstring(pstr) ', con2str(con), linspace(0.2*getw0(con),
5*getw0(con), 20), ' numtostr(tol) '); \n']);
fprintf(fn, ' tmp = [tmp; pm]; \n');
fprintf(fn, 'end; \nnpn = tmp;');
fclose(fn);
clear('pmfunpd');

zx = solve('pmfunpd(x)', pm, zguess, 1, tol);
con = pddes(pstr, w0s, zx, tol);

if nargout==1,
    k = con;
else
    k = getk(con);
    td = gettd(con);
end;
```

phase

```
function phi = phase(g)

%PHASE Computes the phase of a complex vector
%
%        phi = phase(g)
%
%        g is a complex-valued column vector and phi is returned as its
%        phase (in radians), with an effort made to keep it continuous
%        over the pi-borders.
```

```
%L. Ljung 10-2-86/ Modified by Per Persson September 14, 1992
%Copyright (c) 1986-90 by the MathWorks, Inc.
%All Rights Reserved.
%
%LastEditDate : Sat Nov 7 10:57:21 1992
```

```
g = g(:);
phi = atan2(imag(g), real(g));
n = length(phi);
df = phi(1:n-1) - phi(2:n);
tmp = find(abs(df)>3.5);
for ix=tmp',
    if ix~=0,
        phi = phi + 2*pi*sign(df(ix))*[zeros([1,ix])'; ones([1,n-ix])'];
    end;
end;
```

piddes

```
function [k, ti, td] = piddes(pstr, wOs, zO, alphaO, tol)
```

```
%PIDDES Design of a PID controller with dominant poles with a specified zO
%          alphaO, and wO chosen to maximize ki.
%
%          [k, ti, td] = PIDDES(pstr, wOs, zO, alphaO, tol) or
%          con = PIDDES(pstr, wOs, zO, alphaO, tol)
%          Input arguments:
%          pstr      - the process expressed as a string
%          wOs       - array of wO values, the interval must contain wOmax
%          zO        - relative damping of the dominant poles
%          alphaO    - the relative distance of the third pole
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
%          con       - the controller data structure
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 15:58:09 1992
```

```
if nargin==4, tol = deftol; end;
wOs = wOs(:);
```

```
tmp=['getki(dptable1(' qstring(pstr) ', 'pid' ,x , ' numtostr(zO) ', ' numtostr(alphaO)
'))'];
wx = opt(tmp, wOs, tol);
con = dptable1(pstr, 'pid', wx, zO, alphaO);
if nargin==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
    td = gettd(con);
end;
```

piddes2

```
function [k, ti, td] = piddes2(pstr, wOs, ms, alphaO, tol)
```

```
%PIDDES2 Design of a PID controller with a specified Ms
%          and wO chosen to maximize ki.
%
%          [k, ti, td] = PIDDES2(pstr, wOs, ms, alphaO, tol)
%          con = PIDDES2(pstr, wOs, ms, alphaO, tol)
%          Input arguments:
%          pstr      - the process expressed as a string
%          wOs       - array of wO values, the interval must contain wOmax
%          ms        - the Ms value of the controller
%          alphaO    - the relative location of the third dominant pole
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
```

```

%          con          - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:23:13 1993

if nargin==4, tol = deftol; end;
wOs = wOs(:);

tmp = ['getki((dptable2(' qstring(pstr) ', 'pid', x, ' numtostr(ms) ...
', ' numtostr(alpha0) ', ' numtostr(tol) ')))'];
wx = opt(tmp, wOs, tol);
if ~isempty(wx),
    con = dptable2(pstr, 'pid', wx, ms, alpha0, tol);
    if nargout==1,
        k = con;
    else
        k = getk(con);
        ti = getti(con);
        td = gettd(con);
    end;
else
    error('No controller found with piddes2.');
```

```

end;

                                piddesam

function [k, ti, td] = pidesam(pstr, wOs, am, alpha0, zguess, tol)

%PIDDESAM Design of a PID controller with w0 chosen to maximize ki and
%          z0 chosen to get a specified amplitude margin of the closed system.
%
%          [k, ti, td] = PIDDESAM(pstr, wOs, am, alpha0, zguess, tol)
%          con = PIDDESAM(pstr, wOs, am, alpha0, zguess, tol)
%          Input arguments:
%          pstr      - process expressed as a string
%          wOs       - array of w0 values, the interval must contain wOmax
%          am        - the specified amplitude margin
%          alpha0    - the relative location of the third pole
%          zguess    - a guess of z0 interval
%                    (default: linspace(0.005, 2, 10))
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
%          con       - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Dec 19 14:23:48 1992

lsp = linspace(0.1, 2, 10)';
if nargin==4,
    zguess = lsp; tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==6,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;

wOs = wOs(:);
lw = length(wOs); w1 = wOs(1); w2 = wOs(lw); dw = (w2 - w1)/lw;

delete('amfunpid.m');
fn = fopen('amfunpid.m', 'w');
fprintf(fn, 'function am = amfunpid(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs'', \n');
fprintf(fn, [' con = piddes(' qstring(pstr) ', [' numtostr(w1,10) ':' numtostr(dw,10) ':'
numtostr(w2,10) ']]', ix, ' numtostr(alpha0) ', ' numtostr(tol) ');\n']);
fprintf(fn, [' [am, wx] = amarg(' qstring(pstr) ', con2str(con), linspace(0.2*getw0(con),
```

```

20*getw0(con), 20), ' numtostr(tol) '); \n']);
fprintf(fn, ' tmp = [tmp; am]; \n');
fprintf(fn, 'end; \nam = tmp;');
fclose(fn);
clear('amfunpid');

```

```

zx = solve('amfunpid(x)', am, zguess, 1, tol);
con = pidde(pstr, w0s, zx, alpha0, tol);

```

```

if nargout==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
    td = gettd(con);
end;

```

piddeskd

```
function [k, ti, td] = piddeskd(pstr, w0s, z0, kd0, tol)
```

```

%PIDDESKD Design of a PID controller with specified z0 and kd, and w0
%          chosen to maximize ki.
%
%          [k, ti, td] = PIDDESKD(pstr, w0s, z0, kd0)
%          con = PIDDESKD(pstr, w0s, z0, kd0)
%          Input arguments:
%          pstr      - process expressed as a string
%          w0s       - array of w0 values, the interval must contain w0max
%          z0        - specified value of z0
%          kd0       - a specified value of kd
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
%          con       - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 15:58:09 1992

```

```
if nargin==4, tol = deftol; end;
```

```

w0s = w0s(:);
tmp = ['getki(dptable1(' qstring(pstr) ', 'pid2' ',x , numtostr(z0) ', ' numtostr(kd0)
'))'];
wx = opt(tmp, w0s, tol); if isempty(wx), return; end;
con = dptable1(pstr, 'pid2', wx, z0, kd0);
if nargout==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
    td = gettd(con);
end;

```

piddesm2

```
function cons = piddesm2(pstr, w0s, ms1, ms2, kdx, zguess, tol)
```

```

%PIDDESM2 Design of a PID controller with z0 and kd chosen to give the
%          controller specified Ms-value, and w0 chosen to maximize ki.
%          The parameter alpha0 is not used in this routine alpha0 is
%          not used. z0 is chosen by designing a PI controller with Ms=
%          Ms = ms1. kd is then increased until Ms for the system
%          controlled by the PID controller is Ms = ms2.
%          This method assumes that the process can be controlled
%          by a PI controller.
%
%          cons = PIDDESM2(pstr, w0s, ms1, ms2, kdx, zguess, tol)
%          Input arguments:
%          pstr      - process expressed as a string
%          w0s       - array of w0 values the interval must contain w0max

```

```

%      ms1      - the specified ms value for the PI controller
%      ms2      - the specified ms value for the PID controller
%      kdx      - a maximal kd (should be reasonably guessed)
%                  (default: guessed by kdguess)
%      zguess   - a guess of z0 interval
%                  (default: linspace(0.01, 0.99, 40))
%      tol      - the tolerance of the solution (default: deftol)
%      Output arguments:
%      cons     - both the PI and the PID controller are returned.
%                  The PI controller is the first row in cons, and the
%                  PID controller is the second row.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 10:57:24 1993

if nargin<7, tol = deftol; end;

if nargin==6,
    rpi = pidesms(pstr, wOs, ms1, zguess, tol);
else
    rpi = pidesms(pstr, wOs, ms1, [], tol);
end;

z0 = getz0(rpi);

if nargin==4, kdx = kdguess(pstr, wOs, z0); end;
if nargin>4,
    if isempty(kdx), kdx = kdguess(pstr, wOs, z0); end;
end;

kds = 3*kdx*linspace(0, 3, 50)';

w1 = wOs(1); wn = wOs(length(wOs)); dw = (wn-w1)/(length(wOs)-1);

st = ['mshelp(' qstring(pstr) ', [' numtostr(w1) ':' numtostr(dw) ':' numtostr(wn) '], '
numtostr(z0) ', x, ' numtostr(tol) ')'];

kdsol = solve(st, ms2, kds, 1, tol);
rpid = piddeksd(pstr, wOs, z0, kdsol, tol);
cons = [rpi; rpid];



---



piddesms



---


function [k, ti, td] = piddesms(pstr, wOs, ms, alpha0, zguess, tol)

%PIDDESMS Design of a PID controller with z0 chosen to give the controller
%          specified Ms-value, and w0 chosen to maximize ki. The parameter
%          alpha0 is chosen separately.
%
%          [k, ti, td] = PIDDESMS(pstr, wOs, ms, alpha0, zguess, tol)
%          con = PIDDESMS(pstr, wOs, ms, alpha0, zguess, tol)
%          Input arguments:
%          pstr      - process expressed as a string
%          wOs       - array of w0 values the interval must contain wOmax
%          ms        - the specified ms value
%          alpha0    - the relative location of the third dominant pole
%          zguess    - a guess of z0 interval
%                      (default: linspace(0.01, 2, 10))
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
%          con       - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 09:51:00 1993

lsp = linspace(0.01, 2, 10)';
if nargin==4,

```

```

    zguess = lsp;
    tol = deftol;
elseif nargin==5,
    tol = deftol;
elseif nargin==6,
    if isempty(zguess), zguess = lsp; end;
end;

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('msfunpid.m');
fn = fopen('msfunpid.m', 'w');
fprintf(fn, 'function ms = msfunpid(zs)\n');
fprintf(fn, 'zs=zs(:);tmp = []; \nfor ix=zs'', \n');
fprintf(fn, [' r = pidde(' qstring(pstr) ', linspace(' numtostr(w1,10) ', ' numtostr(w2,10)
', ' numtostr(lw) ')', ix, ' numtostr(alpha0) ', ' numtostr(tol) '); \n']);
fprintf(fn, 'if ~isempty(r), w0 = getw0(r);\n');
fprintf(fn, [' [ms, wmp] = mscl(con2str(r), ' qstring(pstr) ', linspace(0.1*w0, 10*w0,
200), ' numtostr(tol) '); \n']);
fprintf(fn, ' tmp = [tmp; ms];\n');
fprintf(fn, 'else error(''Failure in msfunpid.''); end; end;\nms = tmp; return;');
fclose(fn);
clear('msfunpid');

zx = solve('msfunpid(x)', ms, zguess, 1, tol);
if ~isempty(zx),
    con = pidde(pstr, w0s, zx, alpha0);
    if nargin==1,
        k = con;
    else
        k = getk(con);
        ti = getti(con);
        td = gettd(con);
    end;
else
    error('No controller found with piddeSMS.')
end;

```

piddespm

```

function [k, ti, td] = piddespm(pstr, w0s, pm, alpha0, zguess, tol)

%PIDDESPM Design of a PID controller with w0 chosen to maximize ki and
%          z0 chosen to get a specified phase margin of the closed system.
%
%          [k, ti, td] = PIDDESPM(pstr, w0s, pm, alpha0, zguess, tol)
%          con = PIDDESPM(pstr, w0s, pm, alpha0, zguess, tol)
%          Input arguments:
%          pstr      - process expressed as a string
%          w0s       - array of w0 values, the interval must contain w0max
%          pm        - the specified phase margin, in degrees
%          alpha0    - the relative location of the third pole
%          zguess    - a guess of z0 interval
%                   (default: linspace(0.01, 2, 10))
%          tol       - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti, td - PID parameters
%          con       - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:19 1993

lsp = linspace(0.01, 2, 10)';
if nargin==4,
    zguess = lsp; tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==6,

```

```

    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('pmfunpid.m');
fn = fopen('pmfunpid.m', 'w');
fprintf(fn, 'function pm = pmfunpid(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs', '\n');
fprintf(fn, [' con = pides(' qstring(pstr) ', [' numtostr(w1,10) ':' numtostr(dw,10) ':'
numtostr(w2,10) ']', ix, ' numtostr(alpha0) ', ' numtostr(tol) '); \n']);
fprintf(fn, [' [pm, wx] = pmarg(' qstring(pstr) ', con2str(con), linspace(0.2*getw0(con),
5*getw0(con), 20), ' numtostr(tol) '); \n']);
fprintf(fn, ' tmp = [tmp; pm]; \n');
fprintf(fn, 'end; \n\npm = tmp;');
fclose(fn);
clear('pmfunpid');

zx = solve('pmfunpid(x)', pm, zguess, 1, tol);
con = pides(pstr, w0s, zx, alpha0, tol);

if nargout==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
    td = gettd(con);
end;

```

pides

```

function [k, ti] = pides(pstr, w0s, z0, tol)

%PIDES Design of a PI controller with dominant poles with a specified z0
% and w0 chosen to maximize ki.
%
% [k, ti] = PIDES(pstr, w0s, z0, tol) or
% con = PIDES(pstr, w0s, z0, tol)
% Input arguments:
% pstr - the process expressed as a string
% w0s - array of w0 values, the interval must contain w0max
% z0 - relative damping of the dominant poles
% tol - the tolerance of the solution (default: deftol)
% Output arguments:
% k, ti - PI parameters
% con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Dec 1 11:19:27 1992

if nargin==3, tol = deftol; end;
w0s = w0s(:);

tmp = ['getki((dptable1(' qstring(pstr) ', 'pi', x, ' numtostr(z0) ')))'];
wx = opt(tmp, w0s, tol);
if ~isempty(wx),
    con = dptable1(pstr, 'pi', wx, z0);
    if nargout==1,
        k = con;
    else
        k = getk(con);
        ti = getti(con);
    end;
else
    error('No controller found with pides. ');
end;

```

pides2

```

function [k, ti] = pides2(pstr, w0s, ms, tol)

```

```

%PIDES2 Design of a PI controller with with a specified Ms
%       and w0 chosen to maximize ki.
%
%       [k, ti] = PIDES2(pstr, w0s, ms, tol) or
%       con = PIDES2(pstr, w0s, ms, tol)
%       Input arguments:
%       pstr - the process expressed as a string
%       w0s - array of w0 values, the interval must contain wOmax
%       ms - the Ms value of the controller
%       tol - the tolerance of the solution (default: deftol)
%       Output arguments:
%       k, ti - PI parameters
%       con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 10:46:50 1993

if nargin==3, tol = deftol; end;
w0s = w0s(:);

tmp = ['getki((dptable2(' qstring(pstr) ', 'pi', x, ' numtostr(ms) ...
      ', NaN, ' numtostr(tol) ')))'];
wx = opt(tmp, w0s, tol);
if ~isempty(wx),
    con = dptable2(pstr, 'pi', wx, ms, NaN, tol);
    if nargin==1,
        k = con;
    else
        k = getk(con);
        ti = getti(con);
    end;
else
    error('No controller found with pides2.');
```

pidesam

```

function [k, ti] = pidesam(pstr, w0s, am, zguess, tol)

%PIDESAM Design of a PI controller with w0 chosen to maximize ki and
%       z0 chosen to get a specified amplitude margin of the closed system.
%
%       [k, ti] = PIDESAM(pstr, w0s, am, zguess, tol)
%       con = PIDESAM(pstr, w0s, am, zguess, tol)
%       Input arguments:
%       pstr - process expressed as as a string
%       w0s - array of w0 values, the interval must contain wOmax
%       am - the specified amplitude margin, in degrees
%       zguess - a guess of z0 interval (default: linspace(0.1, 2, 10))
%       tol - the tolerance of the solution (default: deftol)
%       Output arguments:
%       k, ti - PI parameters
%       con - the controller data structure

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Dec 19 11:38:06 1992

lsp = linspace(0.1, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;
```

```

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('amfunpi.m');
fn = fopen('amfunpi.m', 'w');
fprintf(fn, 'function am = amfunpi(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs'', \n');
fprintf(fn, [' con = pides(' qstring(pstr) ', [' numtostr(w1,10) ...
            ': ' numtostr(dw,10) ': ' numtostr(w2,10) ']'', ix, ' ...
            numtostr(tol) ']);\n']);
fprintf(fn, [' [am, wx] = amarg(' qstring(pstr) ...
            ', con2str(con), linspace(0.2*getw0(con), 5*getw0(con), 20), ' ...
            numtostr(tol) ']);\n']);
fprintf(fn, ' tmp = [tmp; am];\n');
fprintf(fn, 'end;\nam = tmp;');
fclose(fn);
clear('amfunpi');

zx = solve('amfunpi(x)', am, zguess, 1, tol);
con = pides(pstr, w0s, zx, tol);

if nargout==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
end;

```

pidesms

```
function [k, ti] = pidesms(pstr, w0s, ms, zguess, tol)
```

```

%PIDESMS Design of a PI controller with z0 chosen to give the controller
%         specified Ms-value, and w0 chosen to maximize ki.
%
%         [k, ti] = PIDESMS(pstr, w0s, ms, zguess, tol)
%         con = PIDESMS(pstr, w0s, ms, zguess, tol)
%         Input arguments:
%         pstr - process expressed as a string
%         w0s - array of w0 values, the interval must contain w0max
%         ms - the specified ms value
%         zguess - a guess of z0 interval (default: linspace(0.01, 2, 10))
%                 if zguess = [] then the default will be assumed
%         tol - the tolerance of the solution (default: deftol)
%         Output arguments:
%         k, ti - PI parameters
%         con - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Dec 1 11:17:59 1992

```

```

lsp = linspace(0.01, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; end;
end;

```

```

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;
delete('msfunpi.m');
fn = fopen('msfunpi.m', 'w');
fprintf(fn, 'function ms = msfunpi(zs)\n');
fprintf(fn, 'zs=zs(:);tmp = []; \nfor ix=zs'', \n');
fprintf(fn, [' r = pides(' qstring(pstr) ', linspace(' numtostr(w1,10) ', ' numtostr(w2,10)
            ', ' numtostr(lw,1) ']'', ix, ' numtostr(tol) ']); \n']);
fprintf(fn, 'if isempty(r), w0 = getw0(r);\n');

```

```

fprintf(fn, ['      [ms, wmp] = mscl(con2str(x), ' qstring(pstr) ', linspace(0.1*w0, 10*w0,
200), ' numtostr(tol) '); \n']);
fprintf(fn, '      tmp = [tmp; ms]; \n');
fprintf(fn, 'else error(''Failure in msfunpi.''); end; end; \nms = tmp; return;');
fclose(fn);
clear('msfunpi');

zx = solve('msfunpi(x)', ms, zguess, 1, tol);
if isempty(zx),
    con = pides(pstr, w0s, zx);
    if nargin==1,
        k = con;
    else
        k = getk(con);
        ti = getti(con);
    end;
else
    error('No controller found with pidesms.')
end;

```

pidespm

```
function [k, ti] = pidespm(pstr, w0s, pm, zguess, tol)
```

```

%PIDESPM Design of a PI controller with w0 chosen to maximize ki and
%          z0 chosen to get a specified phase margin of the closed system.
%
%          [k, ti] = PIDESPM(pstr, w0s, pm, zguess, tol)
%          con = PIDESPM(pstr, w0s, pm, zguess, tol)
%          Input arguments:
%          pstr - process expressed as a string
%          w0s - array of w0 values, the interval must contain w0max
%          pm - the specified phase margin, in degrees
%          zguess - a guess of z0 interval (default: linspace(0.01, 2, 10))
%          tol - the tolerance of the solution (default: deftol)
%          Output arguments:
%          k, ti - PI parameters
%          con - the controller data structure

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 18 10:52:19 1992

```

```

lsp = linspace(0.01, 2, 10)';
if nargin==3,
    zguess = lsp; tol = deftol;
elseif nargin==4,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
    tol = deftol;
elseif nargin==5,
    if isempty(zguess), zguess = lsp; else zguess = zguess(:); end;
end;

w0s = w0s(:);
lw = length(w0s); w1 = w0s(1); w2 = w0s(lw); dw = (w2 - w1)/lw;

delete('pmfunpi.m');
fn = fopen('pmfunpi.m', 'w');
fprintf(fn, 'function pm = pmfunpi(zs)\n');
fprintf(fn, 'tmp = []; \nfor ix=zs', \n');
fprintf(fn, [' con = pides(' qstring(pstr) ', [' numtostr(w1,10) ': ' ...
numtostr(dw,10) ': ' numtostr(w2,10) ']', ix, ' ...
numtostr(tol) '); \n']);
fprintf(fn, [' [pm, wx] = pmarg(' qstring(pstr) ', con2str(con), linspace(0.2*getw0(con),
5*getw0(con), 20), ' numtostr(tol) '); \n']);
fprintf(fn, ' tmp = [tmp; pm]; \n');
fprintf(fn, 'end; \nend; \n');
fclose(fn);
clear('pmfunpi');

zx = solve('pmfunpi(x)', pm, zguess, 1, tol);

```

```
con = pides(pstr, wOs, zx, tol);
```

```
if nargin==1,
    k = con;
else
    k = getk(con);
    ti = getti(con);
end;
```

plotc

```
function plotc(z, rad, phi1, phi2, lt, ddeg)
```

```
%PLOTc Plots a circle in an existing plot, leaving the plot state
% unchanged.
```

```
%
%
%      res = PLOTc(z, rad, phi1, phi2, lt, ddeg)
%      Input arguments:
%      z      - a complex number of the center
%      rad    - the radius of the circle
%      phi1, phi2 - angles of the segment
%      lt     - line type (default: '-')
%      ddeg   - plotting discretization (default: 1 deg)
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 16:03:24 1992
```

```
holdstate = get(gcf, 'NextPlot');
if nargin==5,
    ddeg = 1;
elseif nargin==4,
    ddeg = 1; lt = '-';
end;
x = real(z); y = imag(z);
if phi2 < phi1, tmp = phi1; phi1 = phi2; phi2 = tmp; end;
tmp = [phi1:(ddeg*pi/180):phi2]';
res = rad*[cos(tmp) sin(tmp)];
hold on;
plot(res(:,1) + x, res(:,2) + y, lt);
if holdstate == 0, hold off; end;
```

pmarg

```
function [pm, wx] = pmarg(cstr, pstr, ws, tol)
```

```
%PMARG Computes the phase margin of a system.
% The phase margin is expressed in DEGREES, NOT RADIANS.
%
%      [pm, wx] = PMARG(cstr, pstr, ws, tol)
%      Input arguments:
%      cstr - the controller expressed as a string
%      pstr - the process expressed as a string
%      ws   - the frequency interval where the amplitude margin is found
%      tol  - the tolerance of the solution (default: deftol)
%      Output arguments:
%      pm   - the phase margin
%      wx   - the frequency of the phase margin
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 16:52:34 1992
```

```
if nargin==2,
    ws = logspace(-2, 2, 200)'; tol = deftol;
elseif nargin==3,
    tol = deftol;
end;
wx = asolveol(cstr, pstr, 1, ws, tol);
```

```
tmp = evals(cstr, i*wx)*evals(pstr, i*wx);
pm = 180*atan(abs(imag(tmp)/real(tmp)))/pi;
```

psolvecl

```
function wx = psolvecl(cstr, pstr, y, ws, tol)

%PSOLVECL Find the solution of
%      arg(gc(i*w)*gp(i*w)/(1 + gc(i*w)*gp(i*w))) = y
%
%      wx = PSOLVECL(cstr, pstr, y, ws, tol)
%      Input arguments:
%      cstr - the controller expressed as a string
%      pstr - the process expressed as a string
%      y    - the value of the phase (in radians)
%      ws   - the frequency interval where the phase is found
%      tol  - the tolerance of the solution (default: deftol)
%      Output arguments:
%      wx   - the solution frequency

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 16:03:56 1992

if nargin==4, tol = deftol; end;
tmp = ['phase(closeit(evals('' rstr '', i*x).*evals('' pstr '', i*x)))'];
wx = solveb(tmp, y, ws, 1, tol);
```

psolveol

```
function wx = psolveol(cstr, pstr, y, ws, tol)

%PSOLVEOL Finds the solution of arg(Gc(i*w)*Gp(i*w)) = y
%
%      wx = PSOLVEOL(cstr, pstr, y, ws, tol)
%      Input arguments:
%      cstr - the controller expressed as a string
%      pstr - the process expressed as a string
%      y    - the value of the phase (in radians)
%      ws   - the frequency interval where the phase is found
%      tol  - the tolerance of the solution (default: deftol)
%      Output arguments:
%      wx   - the solution frequency

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Nov 13 16:03:56 1992

if nargin==4, tol = deftol; end;
tmp = ['phase(evals('' cstr '', i*x).*evals('' pstr '', i*x))'];
wx = solveb(tmp, y, ws, 1, tol);
```

qstring

```
function str = qstring(str)

%QSTRING Returns a s string WITH quotes.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 11:18:35 1992

tmp = [];
for ix = str,
    if ix==39, tmp = [tmp 39]; end;
    tmp = [tmp ix];
end;

str = [''' tmp '''];
```

ROWS

```
function res = rows(matrix)

%ROWS Returns the number of rows of a matrix.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Sep 26 10:57:04 1992

[res, x] = size(matrix);
```

sfrcol

```
function fr = sfrcol(cstr, pstr, w1, w2, n)

%SFRCOL Computes the frequency frponse of a continuous time
% transfer function.
%
%
% fr = SFRCOL(cstr, pstr, lgw1, lgw2, n)
% fr = SFRCOL(cstr, pstr, wvec)
% Input arguments:
% cstr = the controller expressed as a string
% pstr = the process expressed as a string
%
% The value of  $G(s) = cstr \cdot pstr$  is calculated either for
% the frequencies in wvec [rad/s] or for n logarithmically spaced
% frequency points [rad/s] between  $10^{w1}$  and  $10^{w2}$ . The argument
% n is optional with default value 50.
%
% The output fr takes the form [w G(iw)], and can be plotted with
% BOPL or NYPL from FRBOX.

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:18 1993
```

```
fr = [];
if nargin==5,
    w = logspace(w1, w2, n)';
elseif nargin==4,
    w = logspace(w1, w2)';
elseif nargin==2,
    w = logspace(-2, 2, 200)';
else
    w = w1(:);
end

fr = [w evals(cstr, i*w).evals(pstr, i*w)];
```

sfun

```
function res = sfun(sysnr, s, p1, p2, p3, p4)

if sysnr == 1
    res = p1*exp(-p1*s)./(p3*s+1);
elseif sysnr == 2
    res = 1./(s+1).^p1;
elseif sysnr == 3
    res = (1 - p1*s)./(s + 1).^3;
elseif sysnr == 4
    res = exp(-p1*s)./(p2*s + 1)./(p3*s + 1);
elseif sysnr == 5
    res = exp(-p1*s)*p2^2./(s.^2 + 2*p2*p3*s + p2^2);
elseif sysnr == 6
    res = exp(-p1*s)*p2^3*p4./(s + p2*p4)./(s^2 + 2*p2*p3*s + p2^2);
elseif sysnr == 7
    res = 1./(s + 1)./(s*p1 + 1)./(s*p1^2 + 1);
elseif sysnr == 8
    res = 1./(s + 1)./(s*p1 + 1)./(s*p1^2 + 1)./(s*p1^3 + 1);
elseif sysnr == 9
```

```

    res = (1 - s*p1/2)./(1 + s*p1/2)./(1 + p2*s);
elseif sysnr == 10
    res = exp(-p1*s)./(1 + s*p2)./(1 + s*p3)./(1 + s*p4);
elseif sysnr == 11
    res = exp(-s*p1)./s./(s + 1);
elseif sysnr == 12
    res = exp(-s*p1)./s./(p2*s + 1)./(p3*s + 1);
end;

```

solve

```

function xx = solve(str, y, x0, fol, tol)

%SOLVE Solves the equation f(x) = y.
%       If there is no solution, the value [] is returned, and no error is
%       signalled.
%
%       xx = SOLVE(str, y, x0, fol, tol)
%       Input arguments:
%       str - the function f(x) expressed as a string
%       y   - value of f to solve for
%       x0  - the range searched by the routine
%       fol - first (1) or last solution (2)
%       tol - tolerance (default: deftol)
%       Output arguments:
%       xx  - the x coordinate of the solution

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:18 1993

if nargin==4, tol = deftol; end;
w = x0+eps;
if fol==2, w = reverse(w); fol = 1; end; %hack

rt = evalx(str, w(1)) - y; r = rt;
if defprint, fprintf('solve: %.4f %.4f\n', w(1), rt + y); end;
rt = evalx(str, w(2)) - y; r = [r; rt];
if defprint, fprintf('solve: %.4f %.4f\n', w(2), rt + y); end;
ix = 2;
while r(ix-1)*r(ix)>0,
    ix = ix + 1; if ix>length(w), break; end;
    rt = evalx(str, w(ix)) - y; r = [r; rt];
    if defprint, fprintf('solve: %.4f %.4f\n', w(ix), rt + y); end;
end
if ix>length(w),
    xx = [];
    if defprint,
        fprintf('solve: No solution in [%.4f %.4f]\n', w(1), w(length(w)));
    end;
    return;
end;

tmp1 = find(r > 0);
tmp2 = find(r < 0);

if fol==2,
    c1 = tmp1(length(tmp1)); c2 = length(r);
    if c1==c2, tmp = tmp2; tmp2 = tmp1; tmp1 = tmp; end;
    ix1 = tmp1(length(tmp1)); ix2 = ix1 + 1;
else
    c1 = tmp2(1); c2 = 1;
    if c1==c2, tmp = tmp2; tmp2 = tmp1; tmp1 = tmp; end;
    ix1 = tmp2(1); ix2 = ix1 - 1;
end;

a = w(ix1);
b = w(ix2);
c = (a + b)/2;

if a > b, tmp = a; a = b; b = tmp; end; % a, c, b must be in increasing order

```

```

fa = evalx(str, a) - y; fb = evalx(str, b) - y;
while abs(b-a) > tol*abs(a+b),
    c = (a + b)/2;
    fc = evalx(str, c) - y;
    if (fa>0 & fb>0 & fc>0)|(fa<0 & fb<0 & fc<0), error('error in solve'); end;
    if fa*fc > 0, a = c; fa = fc; else b = c; fb = fc; end;
end;
xx = c;

```

solveb

```
function xx = solveb(str, y, x0, fol, tol)
```

```

%SOLVEB Solves the equation f(x) = y
%       Note that this is a special version of solve used for PHASESOLVE*
%       to get the phase computation right. This routine is slightly more
%       inefficient than SOLVE.
%
%       xx = SOLVEB(str, y, x0, fol, tol)
%       Input arguments:
%       str - the function f(x) expressed as a string
%       y   - value of f to solve for
%       x0  - the range searched by the routine
%       fol - first (1) or last solution (2)
%       tol - tolerance (default deftol;)
%       Output arguments:
%       xx  - the x coordinate of the solution

```

```

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Thu Jan 7 13:38:18 1993

```

```

if nargin==4, tol = deftol; end;
w = x0(:);
r = evalx(str, w) - y;
tmp1 = find(r > 0);
tmp2 = find(r < 0);

if fol==2,
    c1 = tmp1(length(tmp1)); c2 = length(r);
    if c1==c2, tmp = tmp2; tmp2 = tmp1; tmp1 = tmp; end;
    ix1=tmp1(length(tmp1)); ix2 = ix1 + 1;
else
    c1 = tmp2(1); c2 = 1;
    if c1==c2, tmp = tmp2; tmp2 = tmp1; tmp1 = tmp; end;
    ix1 = tmp2(1); ix2 = ix1 - 1;
end;

a = w(ix1);
b = w(ix2);
c = (a + b)/2;
if a > b, tmp = a; a = b; b = tmp; end; % a, c, b must be in increasing order

while abs(b-a) > tol*abs(a+b),
    c = (a + b)/2;
    tmp = evalx(str, [linspace(x0(1),0.9*a,10)';a;c;b]) - y;
    % trick to get phase right
    % must be computed simultaneously!
    fa = tmp(11); fc = tmp(12);
    if fa*fc > 0, a = c; else b = c; end;
end;
xx = c;

```

tf2str

```
function str = tf2str(num, den)
```

```

%TF2STR Converts a transfer function to a text string representation.
%
%       str = TF2STR(num, den)

```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Tue Nov 10 11:18:27 1992
```

```
if length(num)==1,
    str = [numtostr(num)];
else
    str = 'polyval([';
    for ix = num, str = [str numtostr(ix) ' ']; end; str(length(str)) = [];
    str = [str '], s)'];
end;
if length(den)==1,
    str = [str '/' numtostr(den)];
else
    str = [str './polyval([';
    for ix = den, str = [str numtostr(ix) ' ']; end; str(length(str)) = [];
    str = [str '], s)'];
end;
```

zn1pi

```
function [k, ti] = zn1pi(pstr, wOs, tol)
```

```
%ZN1PI Design of a PI controller with Ziegler-Nichols oscillating method.
%
% [k, ti] = ZN1PI(pstr, wOs, tol)
% [k, ti] = ZN1PI(pstr, wOs)
% [k, ti] = ZN1PI(pstr)
% Input arguments:
% pstr - the process expressed as a string
% wOs - the frequency interval where the phase is -pi
%       (default: [0.1:0.5:50])
% tol - the tolerance of the solution (default: deftol())
% Output arguments:
% k, ti - PI parameters
```

```
%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 11 16:37:03 1992
```

```
if nargin==2,
    tol = deftol;
elseif nargin==1,
    wOs = linspace(0.1, 50, 100);
    tol = deftol;
end;

w = psolveol(pstr, '1', -pi, wOs, tol);

t0 = 2*pi/w;
kc = abs(1/evals(pstr, i*w));

k = 0.45*kc;
ti = t0/1.2;
```

zn1pid

```
function [k, ti, td] = zn1pid(pstr, wOs, tol)
```

```
%ZN1PID Design of a PID controller with Ziegler-Nichols oscillating method.
%
% [k, ti, td] = ZN1PID(pstr, wOs, tol)
% [k, ti, td] = ZN1PID(pstr, wOs)
% [k, ti, td] = ZN1PID(pstr)
% Input arguments:
% pstr - the process expressed as a string
% wOs - the frequency interval where the phase is -pi
%       (default: [0.1:0.5:50])
% tol - the tolerance of the solution (default: deftol())
% Output arguments:
```

```

%      k, ti, td - PID parameters

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Fri Dec 11 16:37:03 1992

if nargin==2,
    tol = deftol;
elseif nargin==1,
    w0s = linspace(0.1, 50, 100);
    tol = deftol;
end;

w = psolveol(pstr, '1', -pi, w0s, tol);

t0 = 2*pi/w;
kc = abs(1/evals(pstr, i*w));

k = 0.6*kc;
ti = t0/2;
td = t0/8;



---


                        zn2pi


---


function [k, ti] = zn2pi(kp, l, t)

%ZN2PI Design of a PI controller with Ziegler-Nichols step response method.
%
%      [k, ti] = ZN2PI(kp, l, t)
%      Input arguments:
%      kp      - process gain
%      l       - apparent time delay
%      t       - apparent time constant
%      Output arguments:
%      k, ti   - PI parameters

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 11:18:33 1992

my = 1/t;
k = (0.9/my)/kp;
ti = 1*3;



---


                        zn2pid


---


function [k, ti, td] = zn2pid(kp, l, t)

%ZN2PID Design of a PID controller with Ziegler-Nichols step response method.
%
%      [k, ti, td] = ZN2PID(kp, l, t)
%      Input arguments:
%      kp      - process gain
%      l       - apparent time delay
%      t       - apparent time constant
%      Output arguments:
%      k, ti, td - PI parameters

%Copyright (c) 1992 by Per Persson, Department of Automatic Control,
%Lund Institute of Technology, Lund, Sweden
%
%LastEditDate : Sat Nov 7 11:18:33 1992

my = 1/t;
k = (1.2/my)/kp;
ti = 1*2;
td = 1*0.5;

```