```
function BSSDRE14
%Three-D, multi-tooth, boring stability analysis with speed dependency from both specifiec energy and rotational/TV effects
%Started back up at MTU on June 13, 2002
clear
%-----------------------------------------------------------------------
%Start - Input Paramaters
%Analysis
    de_min = 0.1;           %Minimum Equilibrium Depth of Cut Level in Solution Grid (mm): 0 for min valid
    de_max = 5.1;           %Maximum Equilibrium Depth of Cut Level in Solution Grid (mm)
    de_convpcnt = 5.;       %Percent Change for Convergence when Seeking Tang. Limit for Higher Lobe Numbers (%): recom = 1
    de_intervals = 20;      %Equilibrium Depth of Cut Intervals in Solution Grid (-): recommended min = 10
    epsc_intervals = 100;   %Chatter-Phase Intervals Between 0 and 2pi (-): recommended = 100
    nucpower = 2.;          %Power for Incresing Resolution Near epsc = 0, pi and 2pi (-): min = 1, recom = 2
    lobe_min = 3;           %Minimum (Highest-Speed) of Desired Lobes in Stability Diagram (-): min = 0
    lobe_max = 5;           %Maximum (Lowest-Speed) of Desired Lobes in Stability Diagram (-): min = lobe_min + 1
    lobe_tan = 3;           %Lobe Number at and above which only Minimum (Tangential) Borderline is Found (-): min = 0
    multifreqterms = 1;     %Number of Fourier Series Expansion Terms to Use in Addition to Mean Term (-): min = 0, max = 2 (for cont., single-mat'l borin↙
g)
    useavgspeed = 1;        %A Flag to Use the Averge Cutting Speed (in Spec. Energy) from Zero Lobes to Avoid Extrapolation (-)
    dyn_order = 2;

%Tool
    pr = 15.;               %Lead Angle (deg)
    re = 1.;                %Corner Radius (mm)
    Nt = 1;                 %Number of Teeth (-)
    Dt = 75.;               %Tool/Work Diameter (mm)

%Machine settings
    toolrotates = 1;        %Tool Rotation (-): rotating tool = 1, rotating workpiece = 0
    svsign = +1.;           %Sign of Spindle Velocity (-): cw = +1., ccw = -1.
    ftsign = +1.;           %Direction of Relative Feed during Cutting (-): T and W together = +1., T and W apart = -1.
    ft = 0.1;               %Feed per Tooth (mm): > 0.

%Structural dynamics
    wnx = 120.;             %x-Direction Natural Frequency (Hz)
    wny = 40.;              %y-Direction Natural Frequency (Hz)
    wnz = 170.;             %z-Direction Natural Frequency (Hz)
    kx = 2.5e6;             %x-Direction Stiffness (N/m)
    ky = 1.8e6;             %y-Direction Stiffness (N/m)
    kz = 1.8e6;             %z-Direction Stiffness (N/m)
    zetax = 0.05;           %x-Direction Damping Ratio (-)
    zetay = 0.05;           %Y-Direction Damping Ratio (-)
    zetaz = 0.05;           %z-Direction Damping Ratio (-)

%Specific energy models
%Specific Cutting Energy (N/m^2):  uC = bC0*(V^bCv)*(h^bCh), V (m/s) and h (m)
%Specific Thrust Energy (N/m^2):   uT = bT0*(V^bTv)*(h^bTh), V (m/s) and h (m)
%Specific Lateral Energy (N/m^2):  uL = bL0*(V^bLv)*(h^bLh), V (m/s) and h (m)
%Note:  The numerical values are supposed to be calculated for
%       each tooling/workpiece material combination.
%       Note that ul is typically not used for such tooling (only for
%       end milling and drilling, e.g.), but is included here for generality.
%-----------------------------------
%6061-T6 Aluminum; 1/32 corner radius
%Cutting
    bC0 = 5.5644e+006; %?????????
    bCv = -0.13192; %?????????
    bCh = -0.5000; %?????????
```

```
%Thrust
    bT0 = 1.6669e+006;
    bTv = -0.31448;
    bTh = -0.76719;
%Lateral
    bL0 = 0.;
    bLv = 0.;
    bLh = 0.;
%-----------------------------------
%6061-T6 Aluminum; 1/16 corner radius
%Cutting
    bC0 = 5.5644e+006; %?????????
    bCv = -0.13192; %?????????
    bCh = -0.5000; %?????????
%Thrust
    bT0 = 1.2123e+006;
    bTv = -0.17378;
    bTh = -0.68870;
%Lateral
    bL0 = 0.;
    bLv = 0.;
    bLh = 0.;
    bL0 = 0.25*1.2123e+006;
    bLv = 0.8*-0.17378;
    bLh = 0.8*-0.68870;
%-----------------------------------

%End - Input Paramers
%-------------------------------------------------------------------
%-------------------------------------------------------------------
%------------ No Need for Standard User to Look Past Here ------------
%-------------------------------------------------------------------
%-------------------------------------------------------------------
    fprintf('Inputs Complete\n');

    tic;

%Start - Client-to-Std Unit Conversions
%Analysis
    de_min = de_min*0.001;  %Convert from mm to m
    de_max = de_max*0.001;  %Convert from mm to m
    de_convpcnt = de_convpcnt *0.01;%Convert from percent to fraction

%Tool
    pr = pr*pi/180.;        %Convert from deg to rad
    re = re*0.001;          %Convert from mm to m
    Dt = Dt*0.001;          %Convert from mm to m

%Machine settings
    ft = ft*0.001;          %Convert from mm to m

%Structural dynamics
    wnx = wnx*2*pi;         %Convert from Hz to rad/sec
    wny = wny*2*pi;         %Convert from Hz to rad/sec
    wnz = wnz*2*pi;         %Convert from Hz to rad/sec

%Specific energy models
    %None
```

```
%End - Client-to-Std Unit Conversions
    fprintf('Unit Coversions Complete\n');

%Start - Initializations
%Misc
    nuc_intervals = epsc_intervals;
    subsystemsize = 6; %For this case of second-order dynamics in all three directions.
    systemsize = subsystemsize*(1 + 2*multifreqterms);

%Machine-tool system architecture constants
    if (toolrotates == 0)
        coschi = -1.;
    else %Else - if (toolrotates = 0)
        coschi = +1.;
    end %End - if (toolrotates = 0)

%Repetitively used trig functions
    sinpr = sin(pr);
    cospr = cos(pr);
    tanpr = tan(pr);

%Other repetitively used terms
    wnx2 = wnx*wnx;
    wny2 = wny*wny;
    wnz2 = wnz*wnz;
    twozetawnx = 2.*zetax*wnx;
    twozetawny = 2.*zetay*wny;
    twozetawnz = 2.*zetaz*wnz;
    mx = kx/wnx2; %Mass for x-node
    my = ky/wny2; %Mass for y-node
    mz = kz/wnz2; %Mass for z-node
    ft_2 = 0.5*ft;
    wn_min = min(wnx,wny);
    wn_min = min(wn_min,wnz);

%Compute chip area model coefficients
    de_trans = re*(1. - sinpr); %Transition depth of cut - where corner blends with lead edge
    cft = sqrt(re*re - 0.25*ft*ft); %Feed-per-tooth constant in chip area linearization
    cpr = de_trans/cospr; %Lead-angle constant in chip area linearization

%Force model constants with respect to state variables (all but h-term)
    wn_mean = (wnx + wny + wnz)/3.; %Mean natural frequency (rad/s)
    candidate_spindlespeed = wn_mean/(2.*pi); %Spinlde speed near stability peaks (rps)
    cuttingspeed_init = pi*Dt*candidate_spindlespeed; %Cutting speed near stability peaks (m/s)

%Set de_min to cusp height if requested with an entry of zero in inputs
    if (de_min < re - cft)
        de_min = 1.001*(re - cft);
    end %End - if (de_min == 0.)

%Depth grid increment and levels
    de_increment = (de_max - de_min)/de_intervals;
    de_numlevels = de_intervals + 1; %Number of levels is one more than intervals

%Chatter phase (nu-c) increment and levels
    if (nuc_intervals < 3)
        nuc_intervals = 3;
```

```
    end %End - if (nuc_intervals < 3)
    nuc_numlevels = nuc_intervals + 1; %Number of levels is one more than intervals
%Want number of levels to be one less than an even multiple of 4.
    while (mod(nuc_numlevels,4) ~= 0)
        nuc_numlevels = nuc_numlevels + 1;
    end %End - while (mod(nuc_numlevels,4) ~= 0)
    nuc_numlevels = nuc_numlevels - 1;

    for (k1 = 1 : nuc_numlevels)
        %Set values based on 4-piece piecewise root law for increased resolution near 0, 0.5, and 1.
        dblTemp = 4.*k1/(nuc_numlevels + 1);
        if (dblTemp <= 1.)
            nuc_levels(k1) = 0.00 + 0.25*(dblTemp - 0.)^nucpower;
        elseif (dblTemp <= 2.)
            nuc_levels(k1) = 0.50 - 0.25*(2. - dblTemp)^nucpower;
        elseif (dblTemp <= 3.)
            nuc_levels(k1) = 0.50 + 0.25*(dblTemp - 2.)^nucpower;
        else %Else - if (dblTemp <= 1.)
            nuc_levels(k1) = 1.00 - 0.25*(4. - dblTemp)^nucpower;
        end %End - if (dblTemp <= 1.)
    end %End - for (nuc_level = 1 : nuc_numlevels)
    for (lobe_index = 1 : lobe_max - lobe_min + 1)
        lobe_num = lobe_min + lobe_index - 1;
        for (series = 1 : systemsize)
            solnuc_level(lobe_index,series) = 0;
            nuc_solution(lobe_index,1,series) = 0.;
            wc_solution(lobe_index,1,series) = 0.;
            d_solution(lobe_index,1,series) = 0.;
            spindlespeed_solution(lobe_index,1,series) = 0.;
        end %End - for (series = 1 : systemsize)
    end %End - for (lobe_index = 1 : lobe_max - lobe_min + 1)

%Structural response matrix, for state-space representation.
%It is assumed that each direction is second-order with m, c, k, and no inherent coupling.
%Refer to Li's paper, Eq. 3, where time variation has been averaged out here ("Bar":cos and sin -> 0)
%of [C] and [K] terms; this removes structural (rotation-induced) coupling beween x and y directions.
    As0 = [
            [0.,      1.,          0.,      0.,          0.,      0.]
            [-wnx2, -twozetawnx, 0.,      0.,          0.,      0.]
            [0.,      0.,          0.,      1.,          0.,      0.]
            [0.,      0.,          -wny2, -twozetawny, 0.,      0.]
            [0.,      0.,          0.,      0.,          0.,      1.]
            [0.,      0.,          0.,      0.,          -wnz2, -twozetawnz]
          ];

%Transformation from 6x1 state vector to 3x1 displacement vector.
    Td0 = [
            [1., 0., 0., 0., 0., 0.]
            [0., 0., 1., 0., 0., 0.]
            [0., 0., 0., 0., 1., 0.]
          ];

%Transformation from 3x1 force vector to 6x1 force vector to be compatible with 6x6 * 6x1
%structural matrix times state vector.
    Tf0 = [
            [0.,     0.,     0.]
            [1./mx, 0.,     0.]
            [0.,     0.,     0.]
```

```
            [0.,    1./my, 0.]
            [0.,    0.,    0.]
            [0.,    0.,    1./mz]
          ];


%Constant coefficients in the process gain matrices (for boring) that multiply by Forier
%series expansion coefficnets.
    Pxxcos2 =    svsign*ftsign;
    Pxxsincos = -coschi*ftsign;
    Pxysincos =  Pxxcos2;
    Pxysin2 =    Pxxsincos;
    Pxzcos =     coschi*svsign;
    Pxzsin =     -1.;

    Pyxcos2 =   -Pxxsincos;
    Pyxsincos =  Pxxcos2;
    Pyysincos =  Pyxcos2;
    Pyysin2 =    Pyxsincos;
    Pyzcos =    -Pxzsin;
    Pyzsin =     Pxzcos;

    Pzxcos =     coschi*svsign;
    Pzysin =     Pzxcos;
    Pzz =       -svsign*ftsign;

%End - Initializations
    fprintf('Initializations Complete\n')

%Cycle through each lobe set.
    for (lobe_index = 1 : lobe_max - lobe_min + 1)
        lobe_num = lobe_min + lobe_index - 1;
        if (mod(lobe_index,floor((lobe_max - lobe_min + 1)/10)) == 0)
            fprintf('   %d%%\n',ceil(lobe_index/(lobe_max - lobe_min + 1)*100));
        end %End - if (mod(lobe_index,floor((lobe_max - lobe_min + 1)/10)) == 0)

        if (lobe_num >= lobe_tan) %Initialize for tangential limit convergence seeking.
            de_unstable = 0.;
            de_stable = 0.;%%%% HERE, not sure about resetting de_stable, etc to zeor, seems a faster way could exist.
            de = 0.;
        end %End - if (lobe_num >= lobe_tan)
        de_converged = 0;
        unstable_nuc_level = 0;
        unstable_series = 0;

%Loop over depth of cut grid
%   for (de_level = 1 : de_numlevels)
        de_level = 0;
        while (de_level < de_numlevels) %Use while so counter can be changed in loop (Matlab issue).
%             if (mod(de_level,floor(de_numlevels/10)) == 0)
%                 fprintf('   %d%%\n',ceil(de_level/de_numlevels*100));
%             end %End - if (mod(de_level,floor(de_numlevels/10)) == 0)
            if (lobe_num >= lobe_tan)
                if (de_unstable > de_stable) %A solution was found; split between old solution and new soution.
                    de = 0.5*abs(de_stable + de_unstable);
                else %Else - if (de_unstable > de_stable)
                    de = de_stable + de_min;
                end %End - if (de_unstable > de_stable)
```