Transfer Function Analysis by Manipulation of Poles and Zeros

Initialization Code (optional)

Manipulate

```
Manipulate[

 Module[{graphicsOptions, frameThickness = 0.001},

  If[plotType == "root Locus", useClosedLoop = False];

  graphicsOptions = {ImageSize → {240, 240}, ImagePadding → 0,
    Axes → True,
    PlotRange → {{-ext - 3 del, ext + 3 del}, {-ext - 3 del, ext + 3 del}},
    GridLines → {Range[-ext - 3 del, ext + 3 del, del], Range[-ext - 3 del, ext + 3 del, del]},
    GridLinesStyle → Directive[{LightGray, Thickness[frameThickness]}],
    AspectRatio → 1,
    Background → White,
    TicksStyle → Small,
    Ticks → {{-1, -0.5, 0, 0.5, 1}, {-1, -0.5, 0, 0.5, 1}},
    AxesStyle → Directive[{Blue, Thickness[frameThickness]}]};

  If[showGrid,
   baseGraphics = {{LightGray, Rectangle[{-ext - 3 del, -ext - 3 del}, {ext + 3 del, ext + 3 del}]},
     {White, Rectangle[{-ext, -ext}, {ext, ext}]},
     {Gray, PointSize[0.001],
      Point[#] & /@ Flatten[Outer[List, Range[-ext , ext , del], Range[-ext , ext , del]], 1]},
     {Black, Circle[{0, 0}, 1]}}
   ,
   baseGraphics = {{LightGray, Rectangle[{-ext - 3 del, -ext - 3 del}, {ext + 3 del, ext + 3 del}]},
     {White, Rectangle[{-ext, -ext}, {ext, ext}]},
     {Black, Circle[{0, 0}, 1]}}
  ];

  mousePos = If[showToolTip, MousePosition["Graphics"], 0];

  Row[{
    Grid[{
      {Dynamic[formatedTransferFunctions], SpanFromLeft},
      {
       Switch[pointType,
        POLE,
        LocatorPane[Dynamic[poles], Graphics[
          {
           Sequence@baseGraphics,

           Dynamic@If[showToolTip, formatRadialLine[MousePosition["Graphics"], ts],
             Text["", {0, 0}], Enabled -> showToolTip == True],

           Dynamic[Refresh[format[Chop@poles, Chop@zeros, ext],
             TrackedSymbols → {poles}], Enabled -> showToolTip == False]
```

```
          }, graphicsOptions],
         {{-ext - 2 del, 0}, {ext + 2 del, ext + 2 del}, {gridSpacing, gridSpacing}},
         Appearance → None, LocatorAutoCreate → {All}, AutoAction → False, Enabled -> showToolTip == False
        ],
        _,
        LocatorPane[Dynamic[zeros], Graphics[
          {
           Sequence@baseGraphics,

           Dynamic@If[showToolTip, formatRadialLine[MousePosition["Graphics"], ts],
             Text["", {0, 0}], Enabled -> showToolTip == True],

           Dynamic[Refresh[format[Chop@poles, Chop@zeros, ext],
             TrackedSymbols → {zeros}], Enabled -> showToolTip == False]

          }, graphicsOptions],
         {{-ext - 2 del, 0}, {ext + 2 del, ext + 2 del}, {gridSpacing, gridSpacing}},
         Appearance → None, LocatorAutoCreate → {All}, AutoAction → False, Enabled -> showToolTip == False
        ]
       ],

      Dynamic[processResult]
     }
    }, Spacings → {0, 0}, Frame → All,
    Alignment → Center, FrameStyle -> Directive[Thickness[frameThickness], Gray]],

   (*format returned is {order,{x,y}} *)
   Dynamic[Refresh[internalPoles = findInternalPoints[Chop@poles, ext]; "", TrackedSymbols → {poles}]],

   Dynamic[Refresh[internalZeros = findInternalPoints[Chop@zeros, ext]; "", TrackedSymbols → {zeros}]],

   Dynamic[Refresh[hz = makeHz[internalPoles, internalZeros, z, useClosedLoop];
     "", TrackedSymbols → {poles, zeros, useClosedLoop}]],

   Dynamic[Refresh[{formatedTransferFunctions,
       tf, sys, statusOfConversion, statusMessage, polesInSPlane , zerosInSPlane} =
      formatTransferFunctions[{z, s, y, t}, internalPoles, internalZeros, hz, conversionType, ts];
     "", TrackedSymbols → {poles, zeros, conversionType, ts, useClosedLoop}]],

   Dynamic[
    Refresh[processResult = First@process[z, ts, plotType, simulationTime, rootLocusK, bodeMagScales,
        bodePhaseScales, mousePos, showToolTip, responseJoined, showPlotsGrid, showStabilityMargins,
        tf, sys, internalPoles, internalZeros, hz, addContResponse, statusOfConversion,
        statusMessage, polesInSPlane , zerosInSPlane, conversionType, forcingFrequency];
     "", TrackedSymbols -> {mousePos, poles, zeros, ts, plotType, simulationTime, rootLocusK,
       bodeMagScales, bodePhaseScales, responseJoined, showPlotsGrid, showStabilityMargins,
       addContResponse, conversionType, forcingFrequency, useClosedLoop}
    ], ContinuousAction → True
   ]

  }](*Row*)
], (*Manipulate expression completed*)

(*control variables layout. Top panel*)
Item[
 Grid[{{

    Grid[{{
       Grid[{

         {Control[{{pointType, POLE, ""},
            {POLE → Style["pole", 11], ZERO → Style["zero", 11]}, ControlType → RadioButtonBar,
            ImageSize → Tiny, ImageMargins → 0, Appearance → "Vertical", Background → LightGray}]}
```

```
    }, Spacings → {0, .2}],

   Grid[{
     {Text@Style["plot type", 12]},
     {Control[{ {plotType, "step", ""},
        (# -> Text@Style[#, 12]) & /@ {"Bode", "root Locus", "Nyquist", "Nichols", "H(z) margins",
          "H(s) margins", "S-plane map", "impulse", "step", "ramp", "cos(2πfn)", "e^-n cos(2πfn)"},
        ControlType → PopupMenu, ImageSize → All}]
     }
    }, Spacings → {0, .2}]

  }}, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {-.5, .4}]
,
Grid[{
  {Text@Row[{Style[" k", Italic, 12]}],
   Control[{{rootLocusK, 3, ""}, 1, 20, 1, ImageSize → Tiny, Enabled -> plotType == "root Locus"}],
   Text@Row[{" ", Style[Dynamic[rootLocusK], 10]}]
  },
  {
   Text@Row[{Style[" f", Italic, 12]}],
   Control[{{forcingFrequency, 1, ""}, 0, 2, 0.05,
     ImageSize → Tiny, Enabled -> plotType == "cos(2πfn)" || plotType == "e^-n cos(2πfn)"}],
   Text@Row[{" ", Style[Dynamic[padIt1[forcingFrequency, {2, 2}]], 10]}]
  }
 }, Frame → True, Alignment → Left, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {0, 0.3}
],

Grid[{
  {Text@Style["Bode units (mag)", 12],
   Text@Style["Bode units (phase)", 12]
  },
  {Control[{ {bodeMagScales, {"Linear", "Absolute"}, ""}, (# -> Style[#, 10]) & /@
      {{"Log10", "dB"}, {"Log10", "Absolute"}, {"Linear", "dB"}, {"Linear", "Absolute"}},
     ControlType → PopupMenu, ImageSize → All, Enabled -> plotType == "Bode"}],
   Control[{ {bodePhaseScales, {"Linear", "Degree"}, ""}, (# -> Style[#, 10]) & /@
      {{"Log10", "Degree"}, {"Log10", "Radian"}, {"Linear", "Degree"}, {"Linear", "Radian"}},
     ControlType → PopupMenu, ImageSize → All, Enabled -> plotType == "Bode"}]
  }
 }, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {0, 0.3}],

Grid[{
  {Text@Style["grid lines", 10],
   Control[{{showPlotsGrid, True, ""}, {True, False}, ImageSize → Tiny, ImagePadding → 0,
     Enabled -> plotType == "Bode" || plotType == "Nyquist" || plotType == "Nichols" ||
       plotType == "impulse" || plotType == "step" || plotType == "ramp" ||
       plotType == "cos(2πfn)" || plotType == "e^-n cos(2πfn)"}]
  },
  {Text@Style["margins", 10], Control[
    {{showStabilityMargins, False, ""}, {True, False}, ImageSize → Tiny, ImagePadding → 0, Enabled ->
      plotType == "Bode" || plotType == "Nyquist" || plotType == "Nichols" || plotType == "root Locus"}]
  },
  {Text@Style["closed loop", 10], Control[{{useClosedLoop, False, ""},
     {True, False}, ImageSize → Tiny, ImagePadding → 0, Enabled -> plotType ≠ "root Locus"}]
  }
 }, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {.5, 0}, Alignment → Left]

}}, Frame → None, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {.1, 0}, Alignment → Center
], ControlPlacement → Top, ItemSize → All, Alignment → Left],

(*Bottom panel*)
Item[
```

```
Grid[{
  {
   Grid[{
     {Text@Style["background", 11],
      Control[{{showGrid, True, ""}, {True, False}, ControlType → Checkbox, ImageSize → Tiny}]},
     {Text@Style["tooltip", 11], Control[{{showToolTip, False, ""},
        {True, False}, ControlType → Checkbox, ImageSize → Tiny}]}
    }, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray],
    Spacings → {0.3, 0.6}],

   Grid[{
     {Text@Style[" grid spacing", 10]},
     {Control[{ {gridSpacing, 0.1, ""}, {0.1 → Style["0.1", 10], 0 → Style["0", 10]},
        ControlType → PopupMenu, ImageSize → All}]
     }
    }, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {0.2, 0.3}],

   Grid[{
     {Text@Style["time", 11],
      Control[{{simulationTime, 25, ""}, 2, 50, 1, ImageSize → Tiny}],
      Text@Style[Dynamic[simulationTime], 10]
     },
     {Text@Style["T_s", 11],
      Control[{{ts, 1, ""}, 1, 4, 0.1, ImageSize → Tiny, ImagePadding → 0}],
      Text@Style[Dynamic[padIt1[ts, {2, 2}]], 10]
     }
    }
    , Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {.2, .3}],

   Grid[{
     {Text[Style["response shape", 11]]},
     {Control[{{responseJoined, True, Text@Style["joined plot ", 10]}, {True, False},
        ImageSize → Tiny, ImagePadding → 0, Enabled -> plotType == "impulse" || plotType == "step" ||
          plotType == "ramp" || plotType == "cos(2πfn)" || plotType == "e^-n cos(2πfn)"}]
     },
     {Control[{{addContResponse, True, Text@Style["superimpose", 10]}, {True, False},
        ImageSize → Tiny, ImagePadding → 0, Enabled -> plotType == "impulse" || plotType == "step" ||
          plotType == "ramp" || plotType == "cos(2πfn)" || plotType == "e^-n cos(2πfn)"}]
     }
    }, Frame → True,
    FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {.5, .1}, Alignment → Left],

   Grid[{
     {Text@Style["conversion method", 12]},
     {Control[{ {conversionType, "BilinearTransform", ""}, (#[[2]] -> Text@Style[#[[2]], 11]) & /@
          {"FirstOrderHold" -> "FirstOrderHold", "ZeroOrderHold" -> "ZeroOrderHold", "ZeroPoleMapping" ->
            "ZeroPoleMapping", "BilinearTransform" -> "BilinearTransform", "BackwardRectangularRule" ->
            "BackwardRectangularRule", "ForwardRectangularRule" -> "ForwardRectangularRule"},
        ControlType → PopupMenu, ImageSize → All}]
     }
    }, Frame → True, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {0, 0.3}],

   Button[Text@Style["init", 10],
    {showGrid = True; showToolTip = False; gridSpacing = 0.1; ts = 1; showPlotsGrid = True; simulationTime = 25;
     rootLocusK = 2; bodePhaseScales = {"Linear", "Degree"}; bodeMagScales = {"Linear", "Absolute"};
     plotType = "impulse"; poles = {{0.7, .1}, {1.3, .1}, {1.3, .3}, {1.3, .5}};
     zeros = {{0, 0}, {-1.3, .1}, {-1.3, .3}, {-1.3, .5}}}, ImageSize → {30, 50}]

  }}, Frame → None, FrameStyle -> Directive[Thickness[.001], Gray], Spacings → {.1, 0}, Alignment → Center
 ], ControlPlacement → Bottom],

{{POLE, 2}, ControlType → None},
```

```
{{ZERO, 3}, ControlType → None},
{{poles, {{0.2, 0.8}, {1.3, .1}, {1.3, .3}, {1.3, .5}}}, ControlType → None},
{{zeros, {{0.0, 0.0}, {-1.3, .1}, {-1.3, .3}, {-1.3, .5}}}, ControlType → None},
{{ext, 1.2}, ControlType → None},
{{del, 0.1}, ControlType → None}, (*grid size*)

{{statusOfConversion, True}, ControlType → None},
{{statusMessage, ""}, ControlType → None},
{{polesInSPlane, {}}, ControlType → None},
{{zerosInSPlane , {}}, ControlType → None},
{{internalPoles, {{1, {0.2, 0.8}}}}, ControlType → None},
{{internalZeros, {{1, {0.0, 0.0}}}}, ControlType → None},
{{hz, z / (z^2 - 0.4 z + 0.68)}, ControlType → None},
{{mousePos, 0}, ControlType → None},
{{processResult, {}}, ControlType → None},

(*This below is initialized here for proper use with AutoRunSequence*)
{{formatedTransferFunctions, Grid[{
    {Graphics[
      Text@Row[{ Style["H(z)", Italic, 12], Style[" = "],
          Style[" z / (z^2 - 0.4 z + 0.68) ", 14], Style[" ROC |", 9], Style["z", Italic, 12],
          Style["| > ", 12], 0.825, Style[" " <> "stable", 12]}], ImageSize → {527, 28}
     ]
    },
    {
     Graphics[Text@Row[{Style["H(s)", Italic, 12], Style[" = "], Style[" (1 - 0.25 s^2) / (0.52 s^2 + 0.32 s + 1.28) ", 14]}]],
      ImageSize → {527, 30}]
    },
    {
     Graphics[Text@Row[{Style["0.52 y''(t)+0.32 y(t)+1.28 y(t)", 10]}], ImageSize → {527, 15}]
    }
   }, Spacings → 0, Frame → None]}, ControlType → None},

TrackedSymbols :→
 {pointType, showGrid, showToolTip, gridSpacing, responseJoined, responseGrid, showPlotsGrid,
  showStabilityMargins, addContResponse, plotType, mousePos, useClosedLoop, forcingFrequency},

ImageMargins → 0,
Alignment → Center,
FrameMargins → 0,
ContinuousAction → True,
AutoAction → False,
SynchronousUpdating → True,
ContentSize → Automatic,
AutorunSequencing → {{13, 1}},

Initialization :→
 (
  $MinPrecision = $MachinePrecision;
  $MaxPrecision = $MachinePrecision;

  (*----------------------------------------------------------*)
  makeHz[internalPoles_List,
    internalZeros_List,
```

```
  z_,
  useClosedLoop_ /; Element[useClosedLoop, Booleans]] :=
 Module[{num, den, numCoeffList, denCoeffList, plant},
  den = Replace[internalPoles, {o_, {x_, y_}} :> (z - (x + I y)) ^o, {1}];
  num = Replace[internalZeros, {o_, {x_, y_}} :> (z - (x + I y)) ^o, {1}];

  numCoeffList = Chop@CoefficientList[Expand@Apply[Times, num], z];
  denCoeffList = Chop@CoefficientList[Expand@Apply[Times, den], z];

  plant = Expand@FromDigits[Reverse[numCoeffList], z] / Expand@FromDigits[Reverse[denCoeffList], z];

  If[useClosedLoop, Simplify[plant / Expand[1 + plant]], plant]
 ];

(*-----------------------------------------------------------*)
(*input format {order,{x,y}}*)
polesAndZerosAsList[p_List, z_List] := {
  Flatten[Replace[p, {n_, {x_, y_}} :> Table[{x, y}, {n}], {1}], 1],
  Flatten[Replace[z, {n_, {x_, y_}} :> Table[{x, y}, {n}], {1}], 1]
 };

(*-----------------------------------------------------------*)
getContinuouseTimeTFfromHz[{z_, s_, t_, y_},
  hz_,
  conversionType_String,
  ts_?(NumberQ[#] && Positive[#] &),
  poles_List,
  zeros_List] := Module[{tf, sys, statusOfConversion = True, statusMessage = "", polesInSPlane = {},
   zerosInSPlane = {}, poly, num, den, lap, out, odeOutput, numberOfPoles, numberOfZeros},

  tf = TransferFunctionModel[1. * hz, z, SamplingPeriod → ts];
  numberOfPoles = Length[poles[[All, 2]]];
  numberOfZeros = Length[zeros[[All, 2]]];

  If[ Length[Cases[poles, {xx_, yy_} /; (Abs[yy] < $MachineEpsilon && xx ≤ 0)]] > 0 ||
    numberOfZeros > numberOfPoles,
   (
    If[conversionType == "ZeroOrderHold",
     (
      statusOfConversion = False;
      statusMessage = "Warning: Negative pure real pole using ZeroOrderHold detected"
     )
    ]
   ),
   (
    If[
     Length[Cases[poles, {xx_, yy_} /; (Abs[yy] < $MachineEpsilon && (Abs[xx - 1] < $MachineEpsilon))]] > 0,
     If[conversionType == "ZeroOrderHold" || conversionType == "FirstOrderHold",
      (
       statusOfConversion = False;
       statusMessage = "Warning: Pole at 1 using ZeroOrderHold or FirstOrderHold detected"
      )
     ]
    ]
   )
  ];

  If[statusOfConversion == False,
```

```
    -
   (
    odeOutput = "";
    out = Style[statusMessage, Red, 10]
   ),
   (
    sys = ToContinuousTimeModel[tf, s, Method ->conversionType];
    lap = Chop@Expand@Denominator@First@sys[s][[1]];

    If[lap === 1,
     (
      polesInSPlane = {0, 0};
      zerosInSPlane = {0, 0}
     ),
     (
      polesInSPlane = s /. NSolve[lap == 0, s];
      If[polesInSPlane === s, polesInSPlane = {0, 0}];
      zerosInSPlane = s /. NSolve[Chop@Expand@Numerator@First@sys[s][[1]] == 0, s];
      If[zerosInSPlane === s, zerosInSPlane = {0, 0}]
     )
    ];

    odeOutput = InverseLaplaceTransform[lap, s, t];
    odeOutput = ReplaceAll[odeOutput, {a_. DiracDelta[t] :> round[Chop@a, 2] y[t],
        b_. Derivative[n_][DiracDelta][t] :> round[Chop@b, 2] Derivative[n][y][t]}];

    poly = First@sys;
    num = First@CoefficientRules[First@poly[[1]], s];
    num = Table[num[[i, 1]] → round[Chop@num[[i, 2]], 2], {i, 1, Length[num]}];

    den = poly[[2]];
    den = CoefficientRules[First@Flatten[{den}], s];
    den = Table[den[[i, 1]] → round[Chop@den[[i, 2]], 2], {i, 1, Length[den]}];
    out = FromCoefficientRules[num, s] / FromCoefficientRules[den, s]
   )
  ];

  {tf, sys, statusOfConversion, statusMessage, polesInSPlane , zerosInSPlane, out, odeOutput}
 ];

(*-----------------------------------------------------------*)
formatTransferFunctions[{z_, s_, y_, t_},
  internalPoles_List,
  internalZeros_List,
  hz_,
  conversionType_String,
  ts_?(NumberQ[#] && Positive[#] &)
 ] := Module[{largestPole, stable, poles, zeros, tf, sys, out, odeOutput,
   fontSize = 12, statusOfConversion, statusMessage, zerosInSPlane, polesInSPlane},

  {poles, zeros} = polesAndZerosAsList[internalPoles, internalZeros];

  {tf, sys, statusOfConversion, statusMessage, polesInSPlane , zerosInSPlane, out, odeOutput} =
   getContinuouseTimeTFfromHz[{z, s, t, y}, hz, conversionType, ts, poles, zeros];

  largestPole = Max[Norm[#] & /@ poles];

  Which[largestPole < 1, stable = "stable",
   Abs[largestPole - 1] ≤ $MachineEpsilon, stable = "marginal",
   True, stable = "unstable"
  ];

  {
```

```
   Grid[{
     {Graphics[
       Text@Row[{ Style["H(z)", Italic, fontSize], Style[" = "], Style[NumberForm[hz, 3], 16],
           Style[" ROC |", fontSize], Style["z", Italic, fontSize], Style["| > ", fontSize],
           NumberForm[largestPole, 3], Style[" " <> stable, fontSize]}], ImageSize → {527, 28}
      ]
     },
     {
      Graphics[
       Text@Row[{Style["H(s)", Italic, fontSize], Style[" = "], Style[out, 16]}], ImageSize → {527, 30}]
     },

     {
      Graphics[Text@Row[{Style[odeOutput, 10]}], ImageSize → {527, 15}]
     }

    }, Spacings → 0, Frame → None]
   , tf, sys, statusOfConversion, statusMessage, polesInSPlane , zerosInSPlane
  }
 ];

(*-----------------------------------------------------------*)
formatRadialLine[pt_, ts_?(NumberQ[#] && Positive[#] &)] :=
 Module[{norm, line, text, point, angle, displayedAngle, offset, at},

  norm = Norm[pt];

  If[pt === None ,
   (
    line = Line[{{0, 0}, {0, 0}}];
    text = Text["", {0, 0}];
    point = {PointSize[0], Point[{0, 0}]}
   )
   ,
   (
    angle = ArcTan[pt[[1]], pt[[2]]];
    displayedAngle = angle;

    If[angle ≤ 0.0, displayedAngle = 2 Pi + angle];
    Which[displayedAngle ≥ 3 / 2 Pi || displayedAngle ≤ Pi / 2, offset = {1.3, 0},
     displayedAngle > Pi / 2 || displayedAngle < 3 / 2 Pi, offset = {-1.3, 0}
    ];

    If[norm > 0.8 && norm < 1.2,
     (
      at = {Cos[angle], Sin[angle]};
      line = {Thick, Dashed, Red, Arrow[{{0, 0}, at}]};
      point = {PointSize[0.03], Blue, Point[at]};
      text = Text[Grid[{
           {Row[{Style[NumberForm[(1 / (2 Pi ts)) * displayedAngle, 3], 12, Red], Style[" hz", 10]}]}
          }, Spacings → {0.1, 0.1}, Alignment → Left, Frame → None], at, offset]
     ),
     (
      line = Line[{{0, 0}, {0, 0}}];
      text = Text["", {0, 0}];
      point = {PointSize[0], Point[{0, 0}]}
     )
    ]
   )
  ];

  {line, text, point}
 ];
```

```
(*-----------------------------------------------------------*)
round2[digit_, delta_?(IntegerQ[#] && Positive[#] &)] := Module[{f, d, dd, r, res},

  If[Abs[digit] ≤ $MachineEpsilon,
   (
    res = digit
   ),
   (
    f = FractionalPart[digit];
    d = RealDigits[f, 10, 10];
    r = Flatten[Prepend[d[[1]], Table[0, {Abs[d[[2]] ]}]]];

    dd = If[Length[r] ≤ delta, FromDigits[r] / 10^Length[r],
      Round[ FromDigits[r[[1 ;; delta + 1]]] / 10^(delta + 1), 1 / 10^delta]
     ];
    res = N[IntegerPart[digit] + Sign[digit] * dd]
   )
  ];

  If[res == 1.0, res = 1];
  If[res == -1.0, res = -1];

  res
 ];

(*-----------------------------------------------------------*)
round[digit_, delta_?(IntegerQ[#] && Positive[#] &)] := Module[{},
  Chop@round2[N@Re[digit], delta] + Chop@round2[N@Im[digit], delta] I
 ];

(*-----------------------------------------------------------*)
padIt1[v_?(NumberQ[#] &), f_List] :=
 AccountingForm[Chop[v] , f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True
 ];

(*-----------------------------------------------------------*)
padIt2[v_?(NumberQ[#] &), f_List] :=
 AccountingForm[Chop[v] , f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True
 ];

(*-----------------------------------------------------------*)
findInternalPoints[points_List, ext_?(NumberQ[#] && Positive[#] &)] := Module[{pointsFound},

  (*find points inside region*)
  pointsFound = Cases[points, {x_, y_} /; (Abs[x] - ext) ≤ $MachineEpsilon];

  (*generate order, convert {x,y} to {order,{x,y}} *)
  pointsFound =
   Union[Replace[pointsFound, {x_, y_} :> {Length[Cases[pointsFound, {x, y}]], {x, y}}, {1}]];

  (*add complex conjugate points*)
  Replace[pointsFound, {o_, {x_, y_}} /; y > 0.0 ↦ Sequence[{o, {x, y}}, {o, {x, -y}}], {1}]

 ];

(*-----------------------------------*)
findExternalPoints[points_List, ext_?(NumberQ[#] && Positive[#] &)] := Module[{pointsFound},

  pointsFound = Cases[points, {x_, y_} /; Abs[x] > ext];

  (*add complex conjugate points*)
  Union[Replace[pointsFound, {x_, y_} :> {Length[Cases[pointsFound, {x, y}]], {x, y}}, {1}]
```

```
  ]
 ];

(*-----------------------------------------------------------*)
format[poles_List, zeros_List, ext_?(NumberQ[#] && Positive[#] &)] := Module[{p, z},

  padIt1[x_] := NumberForm[N[x], 3];

  p = Join[findInternalPoints[poles, ext], findExternalPoints[poles, ext]];
  z = Join[findInternalPoints[zeros, ext], findExternalPoints[zeros, ext]];

  (* each entry in the p and z is now in the form {order,{x,y}} *)
  {
   Replace[p, {
     {o_, {x_, y_}} /; o > 1 :>
      Sequence[{Tooltip[Text[Style["×", Red, 16], {x, y}], {padIt1[x], padIt1[y]}]},
       {Text[Style[o, Blue, 10], {x, y}, {-2, 0}]}],
     {o_, {x_, y_}} :> Tooltip[Text[Style["×", Red, 16], {x, y}], {padIt1[x], padIt1[y]}]}, {1}]
   ,
   Replace[z, {
     {o_, {x_, y_}} /; o > 1 :>
      Sequence[{Tooltip[Text[Style["○", Black, 16], {x, y}], {padIt1[x], padIt1[y]}]},
       {Text[Style[o, Blue, 10], {x, y}, {-2, 0}]}],
     {o_, {x_, y_}} :> Tooltip[Text[Style["○", Black, 16], {x, y}], {padIt1[x], padIt1[y]}]}, {1}]

  }
 ];

(*-----------------------------------------------------------*)
phaseBodePlot[phasePlot_, {xScale_, yScale_}, bodePlotOptions_, bodePhaseTicks_, opt___] :=
 Module[{x, y, yToUse, xToUse, yPlotRange, data, p},

  data = Cases[Normal@phasePlot, Line[pts_] → pts, Infinity];

  x = data[[1, All, 1]];
  y = data[[1, All, 2]];

  xToUse = x;
  If[yScale == "Degree",
   (
    yToUse = Mod[y, 360, -180];
    yPlotRange = {-180, 180};
    If[xScale == "Linear",
     (
      p = ListPlot[Transpose[{xToUse, yToUse}], Evaluate@bodePlotOptions,
        Ticks -> Evaluate@bodePhaseTicks, Evaluate@opt, PlotRange → {Automatic, {-200, 200}}]
     ),
     (
      p = ListPlot[Transpose[{xToUse, yToUse}],
        GridLinesStyle → Black,
        (*Evaluate@First@AbsoluteOptions[phasePlot,GridLines],*)
        PlotRange → {Automatic, yPlotRange},
        Evaluate@bodePlotOptions,
        Evaluate@opt,
        Ticks → {{{-3, 0.001}, {-2, 0.01}, {-1, 0.1}, {0, 1}, {10, 1}}, Automatic},
        AxesOrigin → {Min[xToUse], 0},
        PlotRange → {Automatic, {-185, 185}}
       ]
     )]
   ),
   (
    yToUse = Mod[y, 2 Pi, -Pi];
```

```
    yPlotRange = {-Pi, Pi};
    If[xScale == "Linear",
     (
      p = ListPlot[Transpose[{xToUse, yToUse}], Evaluate@bodePlotOptions,
         Ticks -> Evaluate@bodePhaseTicks, Evaluate@opt, PlotRange → {Automatic, {-1.2 Pi, 1.2 Pi}}]
     ),
     (
      p = ListPlot[Transpose[{xToUse, yToUse}],
         GridLinesStyle → Black(*,
         Evaluate@First@AbsoluteOptions[phasePlot,GridLines]*),
         PlotRange → {Automatic, yPlotRange}, Evaluate@bodePlotOptions, Evaluate@opt,
         Ticks → {{{-3, 0.001}, {-2, 0.01}, {-1, 0.1}, {0, 1}, {10, 1}}, Automatic},
         AxesOrigin → {Min[xToUse], 0}, PlotRange → {Automatic, {- Pi, Pi}}]
     )
    ]
   )
  ];

  p
 ];

(*-----------------------------------------------------------*)
makeBodeDiagrams[mousePtIn_,
  z_,
  ts_?(NumberQ[#] && Positive[#] &),
  bodeMagScales_List,
  bodePhaseScales_List,
  showToolTip_ /; (Element[showToolTip, Booleans]),
  hzIn_,
  tf_,
  showPlotsGrid_ /; (Element[showPlotsGrid, Booleans]),
  showStabilityMargins_ /; (Element[showStabilityMargins, Booleans])] :=
 Module[{bodePhase, bodePlotOptions, yLabelBodeMag, xLabelBodeMag, yLabelBodePhase,
   xLabelBodePhase, xForBodeMag, yForBodeMag, xForBodePhase, yForBodePhase, bodePlot,
   w, pt, norm, freq = 0, bodeMagTicks, bodePhaseTicks, scaledFreq, ticks, hz = hzIn,
   bodeMagTicksX, bodeMagTicksY, xLabel, mousePt = mousePtIn, labelFontSize = 9},

  If[Not[showToolTip], mousePt = None];

  If[Not[mousePt === None],
   (
    freq = ArcTan[mousePt[[1]], mousePt[[2]]];
    If[freq ≤ 0.0, freq = 2 Pi + freq];
    norm = Norm[mousePt];
    pt = mousePt
   )
   ,
   (
    norm = Infinity
   )
  ];

  If[norm < 0.9 || norm > 1.1, pt = None]; (*do not tooltip if far from circle*)
  hz = ReplaceAll[hz, z → Exp[I*w]]; (*needed just for the tracking point *)
  scaledFreq = freq/ts;
  xLabel = Grid[{{Style["ω", labelFontSize]}, {Style["rad/sec", labelFontSize]}}, Spacings → {0, -0.2}];
  xLabelBodeMag = xLabel;

  (*find tracking point coordinates for Magnitude plot*)
  If[Order[bodeMagScales[[1]], "Linear"] == 0,
```

```
  (
   xForBodeMag = scaledFreq;
   bodeMagTicksX = {0, Pi/4, Pi/2, 3/4 Pi, Pi, 5 Pi/4, 3 Pi/2, 7/4 Pi, 2 Pi};
   bodeMagTicksY = Automatic;
   bodeMagTicks = {bodeMagTicksX, bodeMagTicksY}
  ),
  (
   xForBodeMag = Log[10, scaledFreq];
   bodeMagTicksX = Automatic;
   bodeMagTicksY = Automatic;
   bodeMagTicks = {bodeMagTicksX, bodeMagTicksY}
  )
 ];

If[Order[bodeMagScales[[2]], "Absolute"] == 0,
 (
  yLabelBodeMag = Text@Style["|H(ω)|", labelFontSize];
  yForBodeMag = Abs[hz /. w → freq]
 ),
 (
  yLabelBodeMag = Text@Row[{Style["|H(ω)| dB", labelFontSize]}];
  yForBodeMag = 20 * Log[10, Abs[hz /. w → freq]]
 )
];

(*find tracking point coordinates for Phase plot*)
xLabelBodePhase = xLabel;
If[Order[bodePhaseScales[[1]], "Linear"] == 0,
 (
  xForBodePhase = scaledFreq;
  bodePhaseTicks = {{0, Pi/4, Pi/2, 3/4 Pi, Pi, 5 Pi/4, 3 Pi/2, 7/4 Pi, 2 Pi}, Automatic}
 ),
 (
  xForBodePhase = Log[10, scaledFreq];
  bodePhaseTicks = {Automatic, Automatic}
 )
];

If[Order[bodePhaseScales[[2]], "Degree"] == 0,
 (
  yLabelBodePhase = Text@Style["phase (deg)", labelFontSize];
  yForBodePhase = Arg[hz /. w → freq] * 180 / Pi
 ),
 (
  yLabelBodePhase = Style["phase (rad)", labelFontSize];
  yForBodePhase = Arg[hz /. w → freq]
 )
];

ticks = Ticks → {bodeMagTicks, bodePhaseTicks};

bodePlotOptions =
 {If[showPlotsGrid, {GridLines → Automatic, GridLinesStyle → Directive[{LightGray, Thickness[0.001]}]},
   GridLines → None], PlotStyle → Red, AspectRatio → 0.25, PlotRangeClipping → False};
```

```
bodePlot = BodePlot[tf, {0.001, 2 * Pi - 0.0001},
  Evaluate@If[showStabilityMargins,
    {StabilityMargins → True, StabilityMarginsStyle → {Blue, Blue}}, StabilityMargins → False],
  Frame → False,
  ScalingFunctions → {bodeMagScales, bodePhaseScales},
  Evaluate@bodePlotOptions,
  ImageSize → {{280, 121}, {280, 121}}, ImagePadding → {{{25, 30}, {25, 25}}, {{25, 30}, {25, 25}}},
  TicksStyle → {Directive[8], Directive[8]},
  PlotLayout → "List",
  PlotLabel → Evaluate@If[pt === None || Not[showToolTip ], {"", ""},

    {Row[{Style["|H(", 10], Style[padIt1[scaledFreq * 180 / Pi, {5, 2}], 10],
       Style["°)| = ", 10], Style[padIt1[yForBodeMag, {5, 2}], 10]}],
     Row[{Style["Arg(H(", 10], Style[padIt1[scaledFreq * 180 / Pi, {5, 2}], 10],
       Style["°) = ", 10], Style[padIt2[yForBodePhase, {5, 2}], 10]}]
    }
   ],
  Evaluate@ticks,
  PlotRange → {{Full, Full}, {Full, Full}},
  AxesLabel → {{xLabelBodeMag, yLabelBodeMag}, {xLabelBodePhase, yLabelBodePhase}}];

bodePhase = phaseBodePlot[bodePlot[[2]],
  bodePhaseScales,
  bodePlotOptions,
  bodePhaseTicks,
  {ImageSize → {280, 121},
   AspectRatio → 0.33,
   ImagePadding → {{25, 30}, {5, 25}},
   TicksStyle → Directive[8],
   Frame → False,
   Joined -> True,
   Axes → True,
   AxesLabel → {xLabelBodePhase, yLabelBodePhase},
   PlotLabel → Evaluate@If[pt === None || Not[showToolTip ], "",
      Row[{Style["Arg(H(", 10], Style[padIt1[scaledFreq * 180 / Pi, {5, 2}], 10],
        Style["°) = ", 10], Style[padIt2[yForBodePhase, {5, 2}], 10]}]]
  }];

If[pt === None || Not[showToolTip ],
 Grid[{
   {bodePlot[[1]]},
   {bodePhase}
  }, Spacings → {0, 0}, Frame → All,
  FrameStyle -> Directive[Thickness[.001], LightGray], Alignment → Center],
 Grid[{
   {
    Show[
     bodePlot[[1]],
     Graphics[{PointSize[0.03], Blue, Point[{xForBodeMag, yForBodeMag}]}],
     PlotRange → All
    ]
   },
   {
    Show[
     bodePhase,
     Graphics[{PointSize[0.03], Blue, Point[{xForBodePhase, yForBodePhase}]}],
     PlotRange → All
    ]

   }, Spacings → {0, 0}, Frame → All,
  FrameStyle -> Directive[Thickness[.001], LightGray], Alignment → Center]
```

```
  ]
 ];

(*-----------------------------------------------------------*)
formatSPlane[p_, z_, ts_?(NumberQ[#] && Positive[#] &)] := Module[{},
  {
   Replace[p, {x_, y_} :> Tooltip[{PointSize[.2], White, Opacity → 0, Point[{x, y}]},
       Grid[{
         {Row[{Style["(", 14], round[Chop@x, 2],
            Style[" , ", 10], round[Chop@y, 2], Style[")", 14]}], SpanFromLeft},
         {Style["ω = ", 10], round[Chop@Norm[{x, y}] * ts / (2 Pi), 2], Style["hz", Italic, 10]},
         {Style["ζ = ", 10], -round[Chop@Cos[If[Abs[x] < $MachineEpsilon, 0, ArcTan[x, y]]], 2]}
        }, Spacings → {0, 0}, Alignment → Left]], {1}]
   ,
   Replace[z, {x_, y_} :> Tooltip[{PointSize[.2], White, Opacity → 0, Point[{x, y}]},
       Grid[{
         {Row[{Style["(", 14], round[Chop@x, 2], Style[" , ", 10], round[Chop@y, 2], Style[")", 14]}]}
        }, Spacings → {0, 0}, Alignment → Left]], {1}]
  }
 ];

(*-----------------------------------------------------------*)
gainPhaseMargins[tf_TransferFunctionModel, title_] :=
 Module[{phasecross, gaincross, g, roundDelta = 3, len, emptySpace = Text@Style[" ", 12]},
  g = GainPhaseMargins[tf];
  phasecross = g[[1]];
  gaincross = g[[2]];

  phasecross = Replace[phasecross, {{None, y_} :> {Text@Style["None", 12], Text@Style[y, 12]}, {x_, y_} :>
      {Text@Style[round2[x / (2 * Pi), roundDelta], 12], Text@Style[round2[y, roundDelta], 12]}}, {1}];

  PrependTo[phasecross, {Text@Style["phase crossover (hz)", 12], Text@Style["gain margins", 12]}];
  len = Length[phasecross];
  If[len < 5, Table[AppendTo[phasecross, {emptySpace, emptySpace}], {i, 1, 5 - len}]];



  gaincross =
   Replace[gaincross, {{None, y_} :> {Text@Style["None", 12], Text@Style[y, 12]}, {x_, y_} :> {Text@Style[
         round2[x / (2 * Pi), roundDelta], 12], Text@Style[round2[y * 180 / Pi, roundDelta], 12]}}, {1}];

  PrependTo[gaincross,
   {Text@Style["gain crossover (hz)", 12], Text@Style["phase margins (deg)", 12]}];
  len = Length[gaincross];
  If[len < 5, Table[AppendTo[gaincross, {emptySpace, emptySpace}], {i, 1, 5 - len}]];

  Grid[{
    {title},
    {Grid[phasecross, Alignment → Left,
      Spacings → {.4, .2}, Frame → All, FrameStyle -> Directive[Thickness[.001], Gray]]},
    {Grid[gaincross, Alignment → Left, Spacings → {.4, .2}, Frame → All,
      FrameStyle -> Directive[Thickness[.001], Gray]]}
   }, Frame → None, Alignment → Left, Spacings → {0, 1}]

 ];

(*-----------------------------------------------------------*)
```

```
process[z_,
  ts_?(NumberQ[#] && Positive[#] &),
  plotType_String,
  simulationTime_?(NumberQ[#] && Positive[#] &),
  rootLocusK_?(IntegerQ[#] &),
  bodeMagScales_List,
  bodePhaseScales_List,
  mousePt_,
  showToolTip_ /; (Element[showToolTip, Booleans]),
  responseJoined_ /; (Element[responseJoined, Booleans]),
  showPlotsGrid_ /; (Element[showPlotsGrid, Booleans]),
  showStabilityMargins_ /; (Element[showStabilityMargins, Booleans]),
  tf_,
  sys_,
  ppoles_List,
  zzeros_List,
  hz_,
  addContResponse_ /; (Element[addContResponse, Booleans]),
  statusOfConversion_,
  statusMessage_String,
  polesInSPlane_List,
  zerosInSPlane_List,
  conversionType_String,
  forcingFrequency_?(NumberQ[#] &)] :=
 Module[{input, res, largestPole, numberOfZeros, numberOfPoles, n, imageSize = {280, 240}, t, contResponse,
   poles, zeros, labels, okToDisplayContPlot = True, discretePlotOptions, analogPlotOptions},

  {poles, zeros} = polesAndZerosAsList[ppoles, zzeros];

  (*do not display analog repsonse for Bilinear when poles<-1*)
  If[Length[Cases[poles, {x_, y_} /; x ≤ -1]] > 0,
   If[conversionType == "BilinearTransform" || conversionType == "BackwardRectangularRule",
    okToDisplayContPlot = False
   ]
  ];

  If[Length[Cases[poles, {x_, y_} /; x ≤ 0]] > 0 && conversionType == "BackwardRectangularRule",
   okToDisplayContPlot = False
  ];

  largestPole = Max[Norm[#] & /@ poles];
  numberOfPoles = Length[poles];
  numberOfZeros = Length[zeros];

  If[numberOfZeros > numberOfPoles,
   Framed[
    Text@Grid[{
       {Style["Warning:", Red, 12]}
       , {Row[{Style["number of zeros (", 12], numberOfZeros, Style[")", 12]}]}
       , {Row[{Style["is larger than number of poles (", 12], numberOfPoles, ")."}]}
       , {Style["Not a physical system.", 12]}
      }, Spacings → {0, 0}, Alignment → Left]
   ],
   {
    discretePlotOptions = {Evaluate@If[responseJoined, Sequence[{Joined → True, InterpolationOrder → 0}],
        {Filling → Axis, FillingStyle → Red, PlotMarkers → Graphics[{PointSize[0.05], Point[{0, 0}]}]}],
      Evaluate@If[showPlotsGrid, {GridLines → Automatic, GridLinesStyle →
          Directive[{LightGray, Thickness[0.001]}]}, GridLines → None],
      PlotRange → All,
      ImageSize → imageSize,
```

```
  ImagePadding → {{40, 10}, {35, 10}},
  AspectRatio → 0.8,
  AxesOrigin → {0, 0},
  FrameLabel → {{None, None}, {Text[Style["time (sec)", 10]], None}},
  Frame → True,
  PlotStyle → Red,
  FrameTicksStyle → Directive[Black, 8],
  Axes → None,
  DataRange → {0, simulationTime}};

analogPlotOptions = {Joined → True, InterpolationOrder → 0,
  PlotRange → All,
  ImageSize → imageSize,
  ImagePadding → {{40, 10}, {35, 10}},
  AspectRatio → 0.9,
  AxesOrigin → {0, 0},
  PlotStyle → {Dashed, Black},
  FrameTicksStyle → Directive[Black, 8],
  DataRange → {0, simulationTime},
  Axes → None};

Which[plotType == "Bode",
 (
  makeBodeDiagrams[mousePt, z, ts, bodeMagScales,
   bodePhaseScales, showToolTip, hz, tf, showPlotsGrid, showStabilityMargins]
 ),

 plotType == "root Locus",
 (
  RootLocusPlot[TransferFunctionModel[n hz, z, SamplingPeriod → ts],
   {n, 0, rootLocusK}, AspectRatio → 0.7, PlotRange → {{-2, 2}, {-2, 2}},
   PoleZeroMarkers → {Text[Style["×", Red, 16]], Automatic, Text[Style["○", Black, 16]]},
   ImageSize → imageSize, ImagePadding → {{30, 10}, {25, 30}},
   PerformanceGoal → "Speed", MaxRecursion → 1, Method → "NDSolve", PlotPoints → 7]
 ),

 plotType == "Nyquist",
 (
  Quiet[NyquistPlot[tf, {-Pi, Pi}, ImagePadding → {{30, 30}, {25, 30}},
    Exclusions → True, PlotRange → Automatic, ImageSize → imageSize, AxesLabel → {"Re", "Im"},
    Evaluate@If[showPlotsGrid, NyquistGridLines → Automatic, NyquistGridLines → None],
    PlotStyle → Red, PerformanceGoal → "Speed", MaxRecursion → 1, Method → "NDSolve",
    PlotPoints → Automatic, Evaluate@If[showStabilityMargins, {StabilityMargins → True,
       StabilityMarginsStyle → Blue}, StabilityMargins → False], AspectRatio → 1]]
 ),

 plotType == "Nichols",
 (
  NicholsPlot[tf, {-Pi, Pi},
   Evaluate@If[showPlotsGrid, NicholsGridLines → Automatic, NicholsGridLines → None],
   ImagePadding → {{30, 40}, {25, 30}},
   ScalingFunctions → {"Degree", "Absolute"},
   AxesLabel → {Text@Column[{Style["phase", 10], Style["(deg)", 10]}, Alignment → Center],
     Text[Style["magnitude", 10]]}, ImageSize → imageSize,
   PlotStyle → Red,
   AspectRatio → 0.8,
   PerformanceGoal → "Speed", MaxRecursion → 1, Method → "NDSolve",
   Evaluate@If[showStabilityMargins,
     {StabilityMargins → True, StabilityMarginsStyle → Blue}, StabilityMargins → False]]
 ),

 plotType == "impulse",
 (
  res = First@Simplify@OutputResponse[tf, DiscreteDelta[n], {n, 0, Ceiling[simulationTime / ts]}];
```

```
 If[okToDisplayContPlot,
  (
   If[addContResponse && statusOfConversion,
    (
     contResponse =
      Chop@First@Simplify@OutputResponse[sys, DiracDelta[t], {t, 0, simulationTime}]
    )
    ,
    contResponse = {}
   ]
  ),
  contResponse = {}
 ];

 Show[
  ListPlot[res, Sequence@discretePlotOptions],
  Plot[ts * contResponse, {t, 0, simulationTime }, PlotRange → All],
  PlotRange → All
 ]

),

plotType == "step",
(
 input = Table[{n, UnitStep[n]}, {n, 0, Ceiling[simulationTime / ts]}];
 res = First@Simplify@OutputResponse[tf, UnitStep[n], {n, 0, simulationTime}];

 If[okToDisplayContPlot,
  (
   If[addContResponse && statusOfConversion,
    contResponse = Chop@First@OutputResponse[sys, UnitStep[t], {t, 0, simulationTime}]
    ,
    contResponse = {}
   ]
  ),
  contResponse = {}
 ];

 Show[

  ListPlot[res, Sequence@discretePlotOptions],
  ListPlot[input[[All, 2]], Sequence@analogPlotOptions],
  Plot[contResponse, {t, 0.001, simulationTime}, PlotRange → All],
  PlotRange → All
 ]
),

plotType == "ramp",
(
 input = Table[{n, n}, {n, 0, Ceiling[simulationTime / ts]}];
 res = Simplify@OutputResponse[tf, input[[All, 2]]][[1]];

 If[okToDisplayContPlot,
  (
   If[addContResponse && statusOfConversion,
    (
     contResponse = First@Simplify@OutputResponse[sys, t UnitStep[t], {t, simulationTime}]
    ),
    (
     contResponse = {}
    )
   ]
  ),
```

```
  contResponse = {}
 ];

 Show[
  ListPlot[res, Sequence@discretePlotOptions],
  ListPlot[input[[All, 2]], Sequence@analogPlotOptions],
  Plot[contResponse, {t, 0.001, simulationTime}, PlotRange → All],
  PlotRange → All
 ]
),

plotType == "cos(2πfn)",
(
 input = Table[{n, Cos[2 Pi forcingFrequency n]}, {n, 0, Ceiling[simulationTime / ts]}];
 res = Simplify@OutputResponse[tf, input[[All, 2]]][[1]];

 If[okToDisplayContPlot,
  (
   If[addContResponse && statusOfConversion,
    (
     contResponse =
      First@Simplify@OutputResponse[sys, Cos[2 Pi forcingFrequency t], {t, 0, simulationTime}]
    ),
    (
     contResponse = {}
    )
   ]
  ),
  contResponse = {}
 ];

 Show[
  ListPlot[res, Sequence@discretePlotOptions],
  ListPlot[input[[All, 2]], Sequence@analogPlotOptions],
  Plot[contResponse, {t, 0, simulationTime}, PlotRange → All],
  PlotRange → All
 ]

),

plotType == "e^-ncos(2πfn)",
(
 input = Table[{n, Exp[-n] Cos[2 Pi forcingFrequency n]}, {n, 0, Ceiling[simulationTime / ts]}];
 res = Simplify@OutputResponse[tf, input[[All, 2]]][[1]];

 If[okToDisplayContPlot,
  (
   If[addContResponse && statusOfConversion,
    (
     contResponse = First@Simplify@
         OutputResponse[sys, Exp[-t] Cos[2 Pi forcingFrequency t], {t, 0, simulationTime}]
    ),
    (
     contResponse = {}
    )
   ]
  ),
  contResponse = {}
 ];

 Show[
  ListPlot[res, Sequence@discretePlotOptions],
  ListPlot[input[[All, 2]], Sequence@analogPlotOptions],
  Plot[contResponse, {t, 0, simulationTime}, PlotRange → All],
```

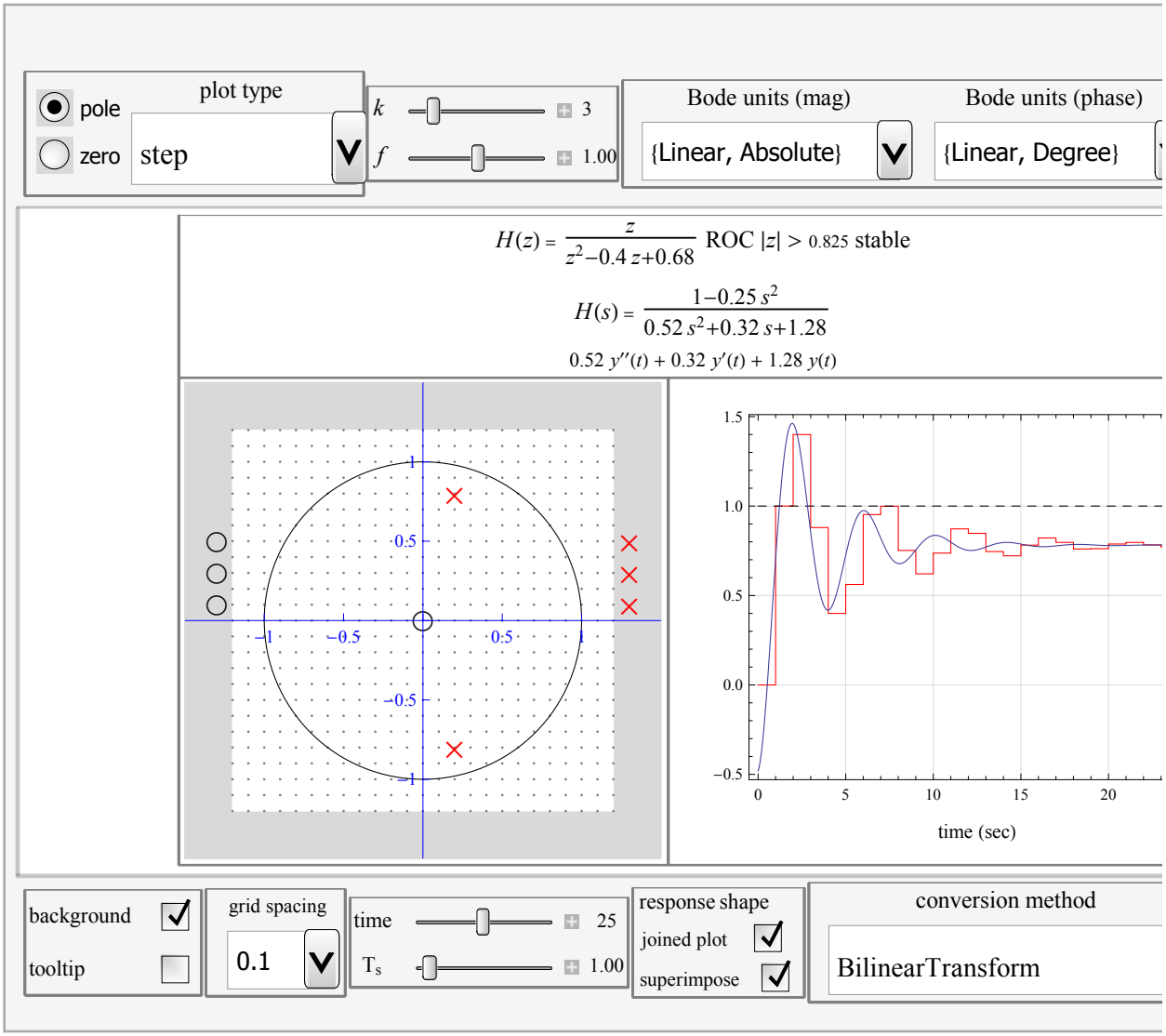
```
    PlotRange → All
   ]
  ),

  plotType == "H(z) margins",
  (
   gainPhaseMargins[tf, Text@Style["discrete system gain phase margins", 12]]
  ),
  plotType == "H(s) margins",
  (
   gainPhaseMargins[sys, Text@Style["continuous system gain phase margins", 12]]
  ),

  plotType == "S-plane map",
  (
   If[statusOfConversion,
    (
     sPoles = Replace[polesInSPlane, x_ :> {Re[x], Im[x]}, {1}];
     sZeros = Replace[zerosInSPlane, x_ :> {Re[x], Im[x]}, {1}];
     labels = formatSPlane[sPoles, sZeros, ts];

     Show[
      ListPlot[{sPoles, sZeros},
       AxesOrigin → {0, 0},
       PlotMarkers → {Style["×", 18, Bold, Red], Style["○", 16, Bold, Red]},
       PlotRange → {Automatic, Automatic},
       ImageSize → imageSize,
       ImagePadding → {{45, 45}, {25, 25}},
       Frame → False,
       Axes → True,
       AxesStyle → Thick,
       Ticks → None,
       PlotRangeClipping → False,
       GridLines → Automatic,
       GridLinesStyle → {{Thickness[0.005], LightGray}, {Thickness[0.005], LightGray}},
       AspectRatio → 1,
       AxesLabel → {Text@Row[{Style["Re(", 10], Style["s", Italic, 10], Style[")", 10]}], Text@
          Row[{Style["Im(", 10], Style["s", Italic, 10], Style[")", 10]}]},
       ImageMargins → 0, Epilog → {labels}],
      PlotRange → All
     ]
    ),
    (
     Graphics[Text[Style[statusMessage, Red, 10]], ImageSize → imageSize]
    )
   ]

  )
 ]
}
]
];
)
]
```



Caption

This demonstration shows visually how the locations of poles and zeros of the system transfer function affect the system properties. Using the mouse you can drag a pole or a zeros of a discrete system transfer function $H(z)$ to a different location and observe the effect on the system. $H(z) = \frac{Y(z)}{X(z)}$ represents the transfer function of a discrete time system where $Y(z)$ is the Z transform of the output signal and $X(z)$ is the Z transform of the input signal. Writing $H(z)$ as a ratio of two polynomials in z, the poles of $H(z)$ are the roots of the denominator polynomial and the zeros are the roots of the numerator polynomial.
This demonstration uses the new control system functions added to *Mathematica* 8.0 in the implementation.

The continuous time transfer function approximation $H(s)$ is generated using all supported $H(z)$ to $H(s)$ mapping methods in *Mathematica* 8. The mapping of the z plane to the s plane is also generated. A total of 12 different plots are available to examine as you move the poles and zeros to different locations.

Thumbnail

Snapshots

Details (optional)

There are 3 parts to the main display: The left part represents the domain of $H(z)$ and shows the unit circle where the poles and zeros can be dragged using the mouse. The right side of the display is used to show different plots such as Bode and system response to different inputs. The top part of the display shows the generated analytical expression for $H(z)$ and $H(s)$ and the differential equation that represents the inverse Laplace transform of the denominator of the transfer function $H(s)$.

In this demonstration it is assumed that the system is causal and right sided which implies that the system is stable when all the poles are inside the unit circle and hence the ROC (region of convergence) extends from the largest pole to infinity. Therefore, the system will be stable when the ROC contains the unit circle.

The following is a description of how to use this demonstration: Starting at the left side of the display, we see the unit circle surrounded by a small gray area. Poles and zeros that are inside the gray area do not contribute at all to the system. A pole is marked with $\times$ and a zero is marked with a $\circ$. To add a pole or a zero to the system, simply drag the pole or the zero from the gray area to the white area using the mouse. To remove a pole or a zero from the system, drag it back to the gray area. All operations are done using the mouse only. The keyboard is not used.

To drag a pole, the radio button selection at the top left of the display must be set to pole. Similarly, to drag a zero, the selection should be set to zero. Dragging is active only in the upper half of the diagram. Complex conjugate poles and zeros in the lower half are automatically created, removed and dragged with the upper half poles and zeros.

To keep the display small, the active area where poles and zeros can be located extend from -1.2 to $+1.2$ in both the x and y directions. This is the white area surrounded by the gray area. The gray area is used only to keep poles and zeros that are not being used at the moment.

The following is a description of the controls in the top of the demonstration from left to right: The radio buttons labeled 'pole' and 'zero' are used to select the type of the root to move (a pole or zero). For example, If you try to drag a pole when the selection is set to 'zero' it will have no effect.

Next is a popup menu labeled 'plot type' which is used to select which plot to show in the right side of the display. The slider labeled 'f' represents the forcing frequency in units of hz for the input signal which is used when selecting the last 2 choices in the 'plot type' popup menu. The slider labeled 'k' is the root locus gain parameter used in the *Mathematica* RootLocusPlot command.

The next 2 popup menus are used to select the units for the Bode plot. The *Mathematica* BodePlot command contains more description of these units. The checkbox labeled 'gridline' is used to turn on grid lines for the plots generated in the right side of the display. The checkbox labeled 'margins' is used to add the option StabilityMargins to BodePlot, NyquistPlot and NicholsPlot. The checkbox labeled 'closed loop' is used to generate the closed loop system and use that instead of the open loop system. By using this option the system $\frac{H(z)}{1+H(z)}$ will be used instead of $H(z)$ in all the analysis except for the RootLocusPlot where the open loop transfer function $H(z)$ is used.

The following is a description of the controls located in the bottom side of the demonstration from left to right: The checkbox labeled 'background' is used to show the fine grid dots shown in the domain of the $H(z)$. The checkbox labeled 'tooltip' is used to turn on the tooltip support which allows you to view the system frequency as you move the mouse around the perimeter of the unit circle and shows the corresponding location on the Bode plots (if it is selected at the same time). Therefore to see this in action, start by selecting the Bode plot option, then turn on the tooltip checkbox, then move the mouse near the edge of the unit circle.

When the tooltip is active, dragging of poles and zeros is disabled automatically. To drag poles and zeros again, make sure the tooltip is checked off. The popup menu labeled 'grid spacing' is used to select the grid spacing for locating poles and zeros. By default, poles and zeros will be located only at intervals of 0.1 distance from each others. This is the default grid granularity. This allows better control on the location of poles and zeros. By setting this option to 0, you can position poles and zeros any where and not only at the grid points, however this could make it harder to position more than one pole or zero at the exact same location depending on the mouse sensitivity.

When the same physical location contain more than one pole or one zero, the order is shown in smaller text next to the location. Putting a zero on top of a pole do not cause a pole zero cancellation in this demonstration.

The slider labeled 'time' is used to specify the time duration for the response of the system in seconds. The slider labeled 'T_s' is the sampling period in seconds used in the conversion of the discrete system to continuous system. The check box labeled 'joined plot' is used to change the display of the response of the discrete system. The check box labeled 'superimpose' is used to show the continuous response of the system with the discrete response on the same plot. The popup menu labeled 'conversion method' is used to select the method to convert $H(z)$ to $H(s)$. *Mathematica* documentation contains more information on these conversion methods. Finally the button labeled 'init' is used to initialize the demonstration to the original settings.

Additional information: In some cases (such as with marginal stability) the magnitude bode plot might not be generated. The plots of Nyquist, Nichlos and Root locus are all plots of the discrete system.

In the response plots, the dashed black line represents the input to the system (step, ramp, sinusoidal, or the exponential decaying sinusoidal). The red line is the discrete system response, and the blue line is the response of the continuous time approximation of the discrete system.

Nichlos plot is plotted using degrees for phase units and absolute for the magnitude.

The S-plane map plot shows the poles and zeros locations in the Laplace domain corresponding to the poles and zeros in the z domain. Moving the mouse close to a pole in the s-domain displays the damping ratio ξ and the natural frequency ω corresponding to this pole. This demonstration supports a maximum of 8 poles and 8 zeros.

references

[1] Alan V. Oppenheim and Ronald W. Schafer, Discrete-time signal processing, Prenctice Hall, NJ. 1999.

[2] Alan V. Oppenheim and Ronald W. Schafer, Digital Signal processing, Prenctice Hall, NJ. 1975.

Control Suggestions (optional)

- ☐ Resize Images
- ☐ Rotate and Zoom in 3D
- ☐ Drag Locators
- ☐ Create and Delete Locators
- ☐ Slider Zoom
- ☐ Gamepad Controls
- ☐ Automatic Animation
- ☐ Bookmark Animation

Search Terms (optional)

BodePlot
NyquistPlot
NicholsPlot
TransferFunctionModel
StabilityMargins
StateSpaceModel
ToContinuousTimeModel
ZTransform
LaplaceTransform
InverseLaplaceTransform
RootLocusPlot
OutputResponse
LocatorPane
Transfer function
poles and zeros
InverseLaplaceTransform
gainPhaseMargins

Related Links (optional)

Mathematica control systems

Authoring Information

Contributed by: Nasser M. Abbasi