The bodeplot package*

Rushikesh Kamalapurkar
rlkamalapurkar@gmail.com

December 2, 2021

Contents

*This document corresponds to bodeplot v1.0.6, dated November 18, 2021.

1 Introduction

Generate Bode, Nyquist, and Nichols plots for transfer functions in the canonical (TF) form

$$G(s) = e^{-Ts} \frac{b_m s^m + \cdots + b_1 s + b_0}{a_n s^n + \cdots + a_1 s + a_0} \tag{1}$$

and the zero-pole-gain (ZPK) form

$$G(s) = Ke^{-Ts} \frac{(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)}. \tag{2}$$

In the equations above, $b_m, \cdots, b_0$ and $a_n, \cdots, a_0$ are real coefficients, $T \geq 0$ is the loop delay, $z_1, \cdots, z_m$ and $p_1, \cdots, p_n$ are complex zeros and poles of the transfer function, respectively, and $K \in \Re$ is the loop gain. For transfer functions in the ZPK format in (2) with zero delay, this package also supports linear and asymptotic approximation of Bode plots. **Limitation:** in TF form, the phase angles are always between 0 and 360°, As such, the Bode phase plots and the Nyquist and Nichols plots will have phase wrapping discontinuities. I do not know how this can be rectified, pull requests are welcome!

2 Usage

2.1 Bode plots

`\BodeZPK` `\BodeZPK` [⟨*obj1/typ1/*{⟨*opt1*⟩}*,obj2/typ2/*{⟨*opt2*⟩}*,...*⟩]
{⟨*z/*{⟨*zeros*⟩}*,p/*{⟨*poles*⟩}*,k/*{⟨*gain*⟩}*,d/*{⟨*delay*⟩}⟩}
{⟨*min-freq*⟩}{⟨*max-freq*⟩}

Plots the Bode plot of a transfer function given in ZPK format using the `groupplot` environment. The three mandatory arguments include: (1) a list of tuples, comprised of the zeros, the poles, the gain, and the transport delay of the transfer function, (2) the lower end of the frequency range for the $x-$ axis, and (3) the higher end of the frequency range for the $x-$axis. The zeros and the poles are complex numbers, entered as a comma-separated list of comma-separated lists, of the form `{{real part 1,imaginary part 1},{real part 2,imaginary part 2},...}`. If the imaginary part is not provided, it is assumed to be zero.

The optional argument is comprised of a comma separated list of tuples, either `obj/typ/{opt}`, or `obj/{opt}`, or just `{opt}`. Each tuple passes options to different `pgfplots` macros that generate the group, the axes, and the plots according to:

- Tuples of the form `obj/typ/{opt}`:
  - `plot/typ/{opt}`: modify plot properties by adding options `{opt}` to the `\addplot` macro for the magnitude plot if `typ` is `mag` and the phase plot if `typ` is `ph`.

- `axes/typ/{opt}`: modify axis properties by adding options `{opt}` to the `\nextgroupplot` macro for the magnitude plot if `typ` is `mag` and the phase plot if `typ` is `ph`.
- `commands/typ/{opt}`: add any valid TikZ commands (including the the parametric function generator macros in this package, such as `\addBodeZPKPlots`, `\addBodeTFPlot`, and `\addBodeComponentPlot`) to the magnitude axes plot if `typ` is `mag` and the phase plot if `typ` is `ph`. The commands passed to `opt` need to be valid TikZ commands, separated by semicolons as usual. For example, a TikZ command is used in the description of the `\BodeTF` macro below to mark the gain crossover frequency on the Bode Magnitude plot.

- Tuples of the form `obj/{opt}`:
  - `plot/{opt}`: adds options `{opt}` to `\addplot` macros for both the magnitude and the phase plots.
  - `axes/{opt}`: adds options `{opt}` to `\nextgroupplot` macros for both the magnitude and the phase plots.
  - `group/{opt}`: adds options `{opt}` to the `groupplot` environment.
  - `tikz/{opt}`: adds options `{opt}` to the `tikzpicture` environment.
  - `approx/linear`: plots linear approximation.
  - `approx/asymptotic`: plots asymptotic approximation.
- Tuples of the form `{opts}` add all of the supplied options to `\addplot` macros for both the magnitude and the phase plots.

The options `{opt}` can be any `key=value` options that are supported by the `pgfplots` macros they are added to. *Linear or asymptotic approximation of transfer functions that include a transport delay is not supported.*

For example, given a transfer function

$$G(s) = 10\frac{s(s+0.1+0.5\mathrm{i})(s+0.1-0.5\mathrm{i})}{(s+0.5+10\mathrm{i})(s+0.5-10\mathrm{i})}, \tag{3}$$

its Bode plot over the frequency range $[0.01, 100]$ can be generated using

```
\BodeZPK [blue,thick]
  {z/{0,{-0.1,-0.5},{-0.1,0.5}},p/{{-0.5,-10},{-0.5,10}},k/10}
  {0.01}{100}
```

which generates the plot in Figure 1. If a delay is not specified, it is assumed to be zero. If a gain is not specified, it is assumed to be 1. By default, each of the axes, excluding ticks and labels, are 5cm wide and 2.5cm high. The width and the height, along with other properties of the plots, the axes, and the group can be customized using native `pgf` keys as shown in the example below.

As demonstrated in this example, if a single comma-separated list of options is passed, it applies to both the magnitude and the phase plots. Without any optional arguments, we gets a thick black Bode plot.

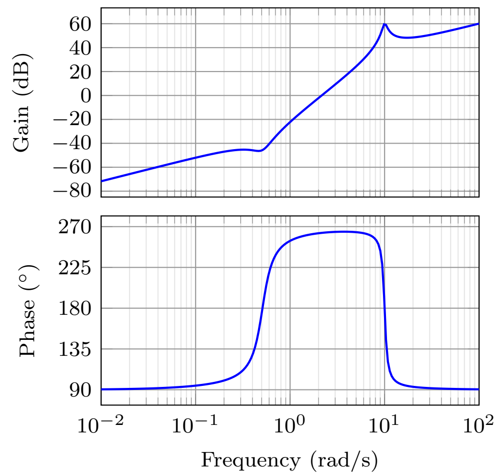

Figure 1: Output of the default `\BodeZPK` macro.

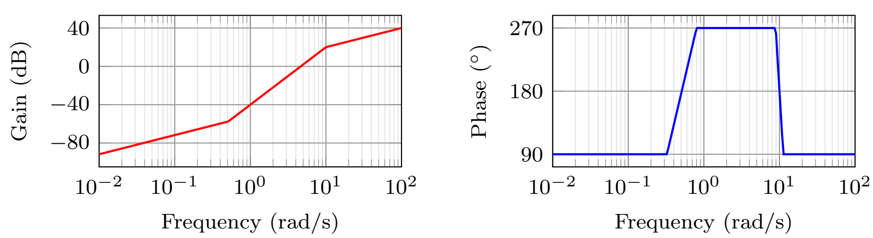

Figure 2: Customization of the default `\BodeZPK` macro.

A linear approximation of the Bode plot with customization of the plots, the axes, and the group can be generated using

```
\BodeZPK[plot/mag/{red,thick},plot/ph/{blue,thick},
  axes/mag/{ytick distance=40,xmajorticks=true,
  xlabel={Frequency (rad/s)}},axes/ph/{ytick distance=90},
  group/{group style={group size=2 by 1,horizontal sep=2cm,
  width=4cm,height=2cm}},approx/linear]
  {z/{0,{-0.1,-0.5},{-0.1,0.5}},p/{{-0.5,-10},{-0.5,10}},k/10}
  {0.01}{100}
```

which generates the plot in Figure 2.

`\BodeTF` `\BodeTF` [⟨*obj1/typ1/*{⟨*opt1*⟩}*,obj2/typ2/*{⟨*opt2*⟩}*,...*⟩]
{⟨*num/*{⟨*coeffs*⟩}*,den/*{⟨*coeffs*⟩}*,d/*{⟨*delay*⟩}⟩}
{⟨*min-freq*⟩}{⟨*max-freq*⟩}

Plots the Bode plot of a transfer function given in TF format. The three mandatory arguments include: (1) a list of tuples comprised of the coefficients in the

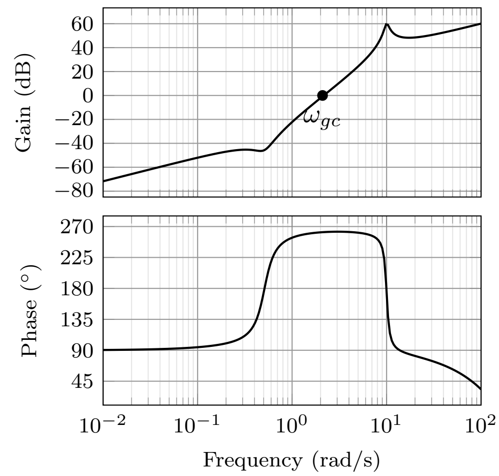

Figure 3: Output of the `\BodeTF` macro with an optional TikZ command used to mark the gain crossover frequency.

numerator and the denominator of the transfer function and the transport delay, (2) the lower end of the frequency range for the $x-$ axis, and (3) the higher end of the frequency range for the $x-$axis. The coefficients are entered as a comma-separated list, in order from the highest degree of s to the lowest, with zeros for missing degrees. The optional arguments are the same as `\BodeZPK`, except that linear/asymptotic approximation is not supported, so `approx/...` is ignored.

For example, given the same transfer function as (3) in TF form and with a small transport delay,

$$G(s) = e^{-0.01s}\frac{s(10s^2+2s+2.6)}{(s^2+s+100.25)}, \tag{4}$$

its Bode plot over the frequency range $[0.01, 100]$ can be generated using

```
\BodeTF[commands/mag/{\node at (axis cs: 2.1,0)
  [circle,fill,inner sep=0.05cm,label=below:{$\omega_{gc}$}]{};}]
  {num/{10,2,2.6,0},den/{1,0.2,100},d/0.01}
  {0.01}{100}
```

which generates the plot in Figure 3. Note the 0 added to the numerator coefficients to account for the fact that the numerator does not have a constant term in it. Note the semicolon after the TikZ command passed to the `\commands` option.

`BodePlot` `\begin{BodePlot}[`⟨*obj1*/{⟨*opt1*⟩},*obj2*/{⟨*opt2*⟩},...⟩`]`
`{`⟨*min-frequency*⟩`}{`⟨*max-frequency*⟩`}`
`\addBode...`
`\end{BodePlot}`

The `BodePlot` environment works in conjunction with the parametric function generator macros `\addBodeZPKPlots`, `\addBodeTFPlot`, and `\addBodeComponentPlot`.

The optional argument is comprised of a comma separated list of tuples, either `obj/{opt}` or just `{opt}`. Each tuple passes options to different `pgfplots` macros that generate the axes and the plots according to:

- Tuples of the form `obj/{opt}`:
  - `tikz/{opt}`: modify picture properties by adding options `{opt}` to the `tikzpicture` environment.
  - `axes/{opt}`: modify axis properties by adding options `{opt}` to the `semilogaxis` environment.
  - `commands/{opt}`: add any valid TikZ commands inside `semilogaxis` environment. The commands passed to `opt` need to be valid TikZ commands, separated by semicolons as usual.
- Tuples of the form `{opt}` are passed directly to the `semilogaxis` environment.

The frequency limits are translated to the x-axis limits and the domain of the `semilogaxis` environment. Example usage in the description of `\addBodeZPKPlots`, `\addBodeTFPlot`, and `\addBodeComponentPlot`.

`\addBodeZPKPlots` `\addBodeZPKPlots [`⟨*approx1/*`{`⟨*opt1*⟩`}`*,approx2/*`{`⟨*opt2*⟩`}`*,...*⟩`]`
`{`⟨*plot-type*⟩`}`
`{`⟨*z/*`{`⟨*zeros*⟩`}`*,p/*`{`⟨*poles*⟩`}`*,k/*`{`⟨*gain*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Generates the appropriate parametric functions and supplies them to multiple `\addplot` macros, one for each `approx/{opt}` pair in the optional argument. If no optional argument is supplied, then a single `\addplot` command corresponding to a thick true Bode plot is generated. If an optional argument is supplied, it needs to be one of `true/{opt}`, `linear/{opt}`, or `asymptotic/{opt}`. This macro can be used inside any `semilogaxis` environment as long as a domain for the x-axis is supplied through either the `approx/{opt}` interface or directly in the optional argument of the `semilogaxis` environment. Use with the `BodePlot` environment supplied with this package is recommended. The second mandatory argument, `plot-type` is either `magnitude` or `phase`. If it is not equal to `phase`, it is assumed to be `magnitude`. The last mandatory argument is the same as `\BodeZPK`.

For example, given the transfer function in (3), its linear, asymptotic, and true Bode plots can be superimposed using

```
\begin{BodePlot}[ ylabel={Gain (dB)}, ytick distance=40,
  height=2cm, width=4cm] {0.01} {100}
  \addBodeZPKPlots[
    true/{black,thick},
    linear/{red,dashed,thick},
    asymptotic/{blue,dotted,thick}]
    {magnitude}
    {z/{0,{-0.1,-0.5},{-0.1,0.5}},p/{{-0.1,-10},{-0.1,10}},k/10}
\end{BodePlot}

\begin{BodePlot}[ylabel={Phase ($^{\circ}$)},
```

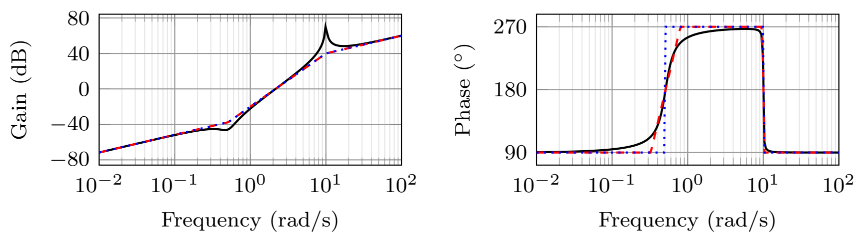

Figure 4: Superimposed approximate and true Bode plots using the `BodePlot` environment and the `\addBodeZPKPlots` macro.

```
  height=2cm, width=4cm, ytick distance=90] {0.01} {100}
  \addBodeZPKPlots[
    true/{black,thick},
    linear/{red,dashed,thick},
    asymptotic/{blue,dotted,thick}]
    {phase}
    {z/{0,{-0.1,-0.5},{-0.1,0.5}},p/{{-0.1,-10},{-0.1,10}},k/10}
\end{BodePlot}
```

which generates the plot in Figure 4.

`\addBodeTFPlot` `\addBodeTFPlot[`⟨*plot-options*⟩`]`
`{`⟨*plot-type*⟩`}`
`{`⟨*num/*`{`⟨*coeffs*⟩`}`*,den/*`{`⟨*coeffs*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Generates a single parametric function for either Bode magnitude or phase plot of a transfer function in TF form. The generated parametric function is passed to the `\addplot` macro. This macro can be used inside any `semilogaxis` environment as long as a domain for the x-axis is supplied through either the `plot-options` interface or directly in the optional argument of the container `semilogaxis` environment. Use with the `BodePlot` environment supplied with this package is recommended. The second mandatory argument, `plot-type` is either magnitude or `phase`. If it is not equal to `phase`, it is assumed to be `magnitude`. The last mandatory argument is the same as `\BodeTF`.

`\addBodeComponentPlot` `\addBodeComponentPlot[`⟨*plot-options*⟩`]{`⟨*plot-command*⟩`}`

Generates a single parametric function corresponding to the mandatory argument `plot-command` and passes it to the `\addplot` macro. The plot command can be any parametric function that uses `t` as the independent variable. The parametric function must be `gnuplot` compatible (or `pgfplots` compatible if the package is loaded using the `pgf` option). The intended use of this macro is to plot the parametric functions generated using the basic component macros described in Section 2.1.1 below.

2.1.1 Basic components up to first order

`\TypeFeatureApprox` `\TypeFeatureApprox{`⟨*real-part*⟩`}{`⟨*imaginary-part*⟩`}`

This entry describes 20 different macros of the form `\TypeFeatureApprox` that take the real part and the imaginary part of a complex number as arguments. The `Type` in the macro name should be replaced by either `Mag` or `Ph` to generate a parametric function corresponding to the magnitude or the phase plot, respectively. The `Feature` in the macro name should be replaced by one of `K`, `Pole`, `Zero`, or `Del`, to generate the Bode plot of a gain, a complex pole, a complex zero, or a transport delay, respectively. If the `Feature` is set to either `K` or `Del`, the `imaginary-part` mandatory argument is ignored. The `Approx` in the macro name should either be removed, or it should be replaced by `Lin` or `Asymp` to generate the true Bode plot, the linear approximation, or the asymptotic approximation, respectively. If the `Feature` is set to `Del`, then `Approx` has to be removed. For example,

- `\MagK{k}{0}` or `\MagK{k}{400}` generates a parametric function for the true Bode magnitude of $G(s) = k$
- `\PhPoleLin{a}{b}` generates a parametric function for the linear approximation of the Bode phase of $G(s) = \frac{1}{s-a-ib}$.
- `\PhDel{T}{200}` or `\PhDel{T}{0}` generates a parametric function for the Bode phase of $G(s) = e^{-Ts}$.

All 20 of the macros defined by combinations of `Type`, `Feature`, and `Approx`, and any `gnuplot` (or `pgfplot` if the `pgf` class option is loaded) compatible function of the 20 macros can be used as `plot-command` in the `addBodeComponentPlot` macro. This is sufficient to generate the Bode plot of any rational transfer function with delay. For example, the Bode phase plot in Figure 4 can also be generated using:

```
\begin{BodePlot}[ylabel={Phase (degree)},ytick distance=90]{0.01}{100}
  \addBodeComponentPlot[black,thick]{\PhZero{0}{0} + \PhZero{-0.1}{-0.5} +
    \PhZero{-0.1}{0.5} + \PhPole{-0.5}{-10} + \PhPole{-0.5}{10} +
    \PhK{10}{0}}
  \addBodeComponentPlot[red,dashed,thick] {\PhZeroLin{0}{0} +
    \PhZeroLin{-0.1}{-0.5} + \PhZeroLin{-0.1}{0.5} +
    \PhPoleLin{-0.5}{-10} + \PhPoleLin{-0.5}{10} + \PhKLin{10}{20}}
  \addBodeComponentPlot[blue,dotted,thick] {\PhZeroAsymp{0}{0} +
    \PhZeroAsymp{-0.1}{-0.5} + \PhZeroAsymp{-0.1}{0.5} +
    \PhPoleAsymp{-0.5}{-10} + \PhPoleAsymp{-0.5}{10} + \PhKAsymp{10}{40}}
\end{BodePlot}
```

which gives us the plot in Figure 5.

2.1.2 Basic components of the second order

`\TypeSOFeatureApprox`

`\TypeSOFeatureApprox{`$\langle a1\rangle$`}{`$\langle a0\rangle$`}`

This entry describes 12 different macros of the form `\TypeSOFeatureApprox` that take the coefficients a_1 and a_0 of a general second order system as inputs. The `Feature` in the macro name should be replaced by either `Poles` or `Zeros` to generate the Bode plot of $G(s) = \frac{1}{s^2+a_1s+a_0}$ or $G(s) = s^2+a_1s+a_0$, respectively. The

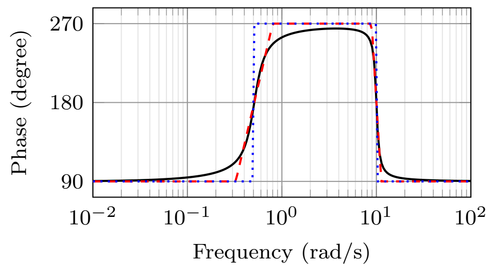

Figure 5: Superimposed approximate and true Bode Phase plot using the `BodePlot` environment, the `\addBodeComponentPlot` macro, and several macros of the `\TypeFeatureApprox` form.

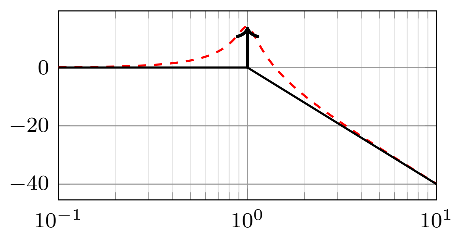

Figure 6: Resonant peak in asymptotic Bode plot using `\MagSOPolesPeak`.

`Type` in the macro name should be replaced by either `Mag` or `Ph` to generate a parametric function corresponding to the magnitude or the phase plot, respectively. The `Approx` in the macro name should either be removed, or it should be replaced by `Lin` or `Asymp` to generate the true Bode plot, the linear approximation, or the asymptotic approximation, respectively.

`\MagSOFeaturePeak`

`\MagSOFeaturePeak[`⟨*draw-options*⟩`]{`⟨*a1*⟩`}{`⟨*a0*⟩`}`

This entry describes 2 different macros of the form `\MagSOFeaturePeak` that take the the coefficients a_1 and a_0 of a general second order system as inputs, and draw a resonant peak using the `\draw` TikZ macro. The `Feature` in the macro name should be replaced by either `Poles` or `Zeros` to generate a peak for poles and a valley for zeros, respectively. For example, the command

```
\begin{BodePlot}[xlabel={}]{0.1}{10}
  \addBodeComponentPlot[red,dashed,thick]{\MagSOPoles{0.2}{1}}
  \addBodeComponentPlot[black,thick]{\MagSOPolesLin{0.2}{1}}
  \MagSOPolesPeak[thick]{0.2}{1}
\end{BodePlot}
```

generates the plot in Figure 6.

`\TypeCSFeatureApprox`

`\TypeCSFeatureApprox{`⟨*zeta*⟩`}{`⟨*omega-n*⟩`}`

This entry describes 12 different macros of the form `\TypeCSFeatureApprox` that take the damping ratio, ζ, and the natural frequency, ω_n of a canonical second order system as inputs. The `Type` in the macro name should be replaced by either `Mag` or `Ph` to generate a parametric function corresponding to the magnitude or the phase plot, respectively. The `Feature` in the macro name should be replaced by either `Poles` or `Zeros` to generate the Bode plot of $G(s) = \frac{1}{s^2+2\zeta\omega_n s+\omega_n^2}$ or $G(s) = s^2+2\zeta\omega_n s+\omega_n^2$, respectively. The `Approx` in the macro name should either be removed, or it should be replaced by `Lin` or `Asymp` to generate the true Bode plot, the linear approximation, or the asymptotic approximation, respectively.

`\MagCSFeaturePeak` `\MagCSFeaturePeak[`⟨*draw-options*⟩`]{`⟨*zeta*⟩`}{`⟨*omega-n*⟩`}`

This entry describes 2 different macros of the form `\MagCSFeaturePeak` that take the damping ratio, ζ, and the natural frequency, ω_n of a canonical second order system as inputs, and draw a resonant peak using the `\draw` TikZ macro. The `Feature` in the macro name should be replaced by either `Poles` or `Zeros` to generate a peak for poles and a valley for zeros, respectively.

`\MagCCFeaturePeak` `\MagCCFeaturePeak[`⟨*draw-options*⟩`]{`⟨*real-part*⟩`}{`⟨*imaginary-part*⟩`}`

This entry describes 2 different macros of the form `\MagCCFeaturePeak` that take the real and imaginary parts of a pair of complex conjugate poles or zeros as inputs, and draw a resonant peak using the `\draw` TikZ macro. The `Feature` in the macro name should be replaced by either `Poles` or `Zeros` to generate a peak for poles and a valley for zeros, respectively.

2.2 Nyquist plots

`\NyquistZPK` `\NyquistZPK [`⟨*plot/{*⟨*opt*⟩*},axes/{*⟨*opt*⟩*}*⟩`]`
`{`⟨*z/{*⟨*zeros*⟩*},p/{*⟨*poles*⟩*},k/{*⟨*gain*⟩*},d/{*⟨*delay*⟩*}*⟩`}`
`{`⟨*min-freq*⟩`}{`⟨*max-freq*⟩`}`

Plots the Nyquist plot of a transfer function given in ZPK format with a thick red + marking the critical point (-1,0). The mandatory arguments are the same as `\BodeZPK`. Since there is only one plot in a Nyquist diagram, the `\typ` specifier in the optional argument tuples is not needed. As such, the supported optional argument tuples are `plot/{opt}`, which passes `{opt}` to `\addplot`, `axes/{opt}`, which passes `{\opt}` to the `axis` environment, and `tikz/{opt}`, which passes `{\opt}` to the `tikzpicture` environment. Asymptotic/linear approximations are not supported in Nyquist plots. If just `{opt}` is provided as the optional argument, it is interpreted as `plot/{opt}`. Arrows to indicate the direction of increasing ω can be added by adding `\usetikzlibrary{decorations.markings}` and `\usetikzlibrary{arrows.meta}` to the preamble and then passing a tuple of the form

```
plot/{postaction=decorate,decoration={markings,
  mark=between positions 0.1 and 0.9 step 5em with
  {\arrow{Stealth [length=2mm, blue]}}}}
```

Caution: with a high number of samples, adding arrows in this way may cause the error message `! Dimension too big`.

For example, the command

```
\NyquistZPK[plot/{red,thick,samples=2000},axes/{blue,thick}]
```

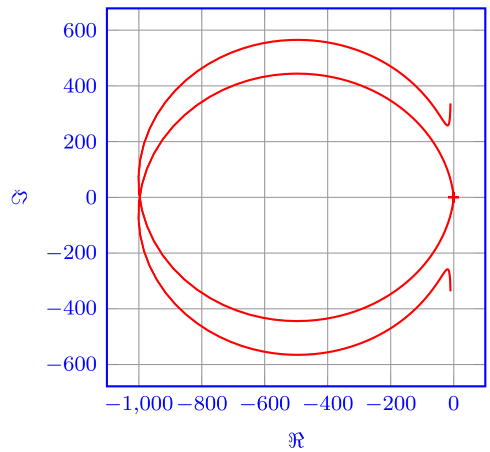

Figure 7: Output of the `\NyquistZPK` macro.

```
  {z/{0,{-0.1,-0.5},{-0.1,0.5}},p/{{-0.5,-10},{-0.5,10}},k/10}
  {-30}{30}
```

generates the Nyquist plot in Figure 7.

`\NyquistTF` `\NyquistTF [`⟨*plot/*`{`⟨*opt*⟩`}`*,axes/*`{`⟨*opt*⟩`}`⟩`]`
`{`⟨*num/*`{`⟨*coeffs*⟩`}`*,den/*`{`⟨*coeffs*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`
`{`⟨*min-freq*⟩`}{`⟨*max-freq*⟩`}`

Nyquist plot of a transfer function given in TF format. Same mandatory arguments as `\BodeTF` and same optional arguments as `\NyquistZPK`. For example, the command

```
\NyquistTF[plot/{green,thick,samples=500,postaction=decorate,
  decoration={markings,
  mark=between positions 0.1 and 0.9 step 5em
  with{\arrow{Stealth[length=2mm, blue]}}}}]
  {num/{10,2,2.6,0},den/{1,1,100.25}}
  {-30}{30}
```

generates the Nyquist plot in Figure 8.

`NyquistPlot` `\begin{NyquistPlot}[`⟨*obj1/*`{`⟨*opt1*⟩`}`*,obj2/*`{`⟨*opt2*⟩`}`*,...*⟩`]`
`{`⟨*min-frequency*⟩`}{`⟨*max-frequency*⟩`}`
`\addNyquist...`
`\end{NyquistPlot}`

The `NyquistPlot` environment works in conjunction with the parametric function generator macros `\addNyquistZPKPlot` and `\addNyquistTFPlot`. The optional argument is comprised of a comma separated list of tuples, either `obj/{opt}` or just `{opt}`. Each tuple passes options to different `pgfplots` macros that generate the axes and the plots according to:

- Tuples of the form `obj/{opt}`:

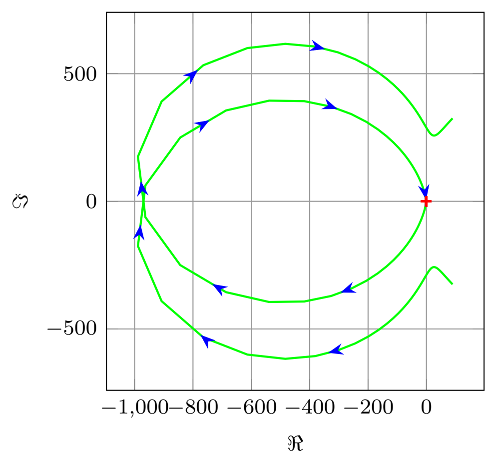

Figure 8: Output of the `\NyquistTF` macro with direction arrows. Increasing the number of samples can cause `decorations.markings` to throw errors.

- `tikz/{opt}`: modify picture properties by adding options `{opt}` to the `tikzpicture` environment.
- `axes/{opt}`: modify axis properties by adding options `{opt}` to the `axis` environment.
- `commands/{opt}`: add any valid TikZ commands inside `axis` environment. The commands passed to `opt` need to be valid TikZ commands, separated by semicolons as usual.

• Tuples of the form `{opt}` are passed directly to the `axis` environment.

The frequency limits are translated to the x-axis limits and the domain of the `axis` environment.

`\addNyquistZPKPlot` `\addNyquistZPKPlot[`⟨*plot-options*⟩`]`
`{`⟨*z/*`{`⟨*zeros*⟩`}`*,p/*`{`⟨*poles*⟩`}`*,k/*`{`⟨*gain*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Generates a twp parametric functions for the magnitude and the phase a transfer function in ZPK form. The generated magnitude and phase parametric functions are converted to real and imaginary part parametric functions and passed to the `\addplot` macro. This macro can be used inside any `axis` environment as long as a domain for the x-axis is supplied through either the `plot-options` interface or directly in the optional argument of the container `axis` environment. Use with the `NyquistPlot` environment supplied with this package is recommended. The mandatory argument is the same as `\BodeZPK`.

`\addNyquistTFPlot` `\addNyquistTFPlot[`⟨*plot-options*⟩`]`
`{`⟨*num/*`{`⟨*coeffs*⟩`}`*,den/*`{`⟨*coeffs*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Similar to `\addNyquistZPKPlot`, with a transfer function input in the TF form.

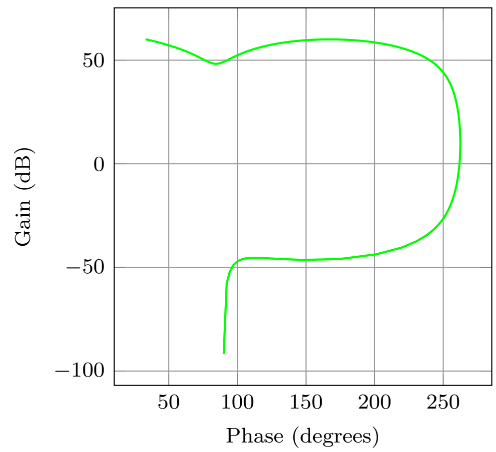

Figure 9: Output of the `\NyquistZPK` macro.

2.3 Nichols charts

`\NicholsZPK` `\NicholsZPK [`⟨*plot/{*⟨*opt*⟩*},axes/{*⟨*opt*⟩*}*⟩`]`
`{`⟨*z/{*⟨*zeros*⟩*},p/{*⟨*poles*⟩*},k/{*⟨*gain*⟩*},d/{*⟨*delay*⟩*}*⟩`}`
`{`⟨*min-freq*⟩`}{`⟨*max-freq*⟩`}`

Nichols chart of a transfer function given in ZPK format. Same arguments as `\NyquistZPK`.

`\NicholsTF` `\NicholsTF [`⟨*plot/{*⟨*opt*⟩*},axes/{*⟨*opt*⟩*}*⟩`]`
`{`⟨*num/{*⟨*coeffs*⟩*},den/{*⟨*coeffs*⟩*},d/{*⟨*delay*⟩*}*⟩`}`
`{`⟨*min-freq*⟩`}{`⟨*max-freq*⟩`}`

Nichols chart of a transfer function given in TF format. Same arguments as `\NyquistTF`. For example, the command

```
\NicholsTF[plot/{green,thick,samples=2000}]
  {num/{10,2,2.6,0},den/{1,1,100.25},d/0.01}
  {0.001}{100}
```

generates the Nichols chart in Figure 9.

`NicholsChart` `\begin{NicholsChart}[`⟨*obj1/{*⟨*opt1*⟩*},obj2/{*⟨*opt2*⟩*},...*⟩`]`
`{`⟨*min-frequency*⟩`}{`⟨*max-frequency*⟩`}`
`\addNichols...`
`\end{NicholsChart}`

The `NicholsChart` environment works in conjunction with the parametric function generator macros `\addNicholsZPKChart` and `\addNicholsTFChart`. The optional argument is comprised of a comma separated list of tuples, either `obj/{opt}` or just `{opt}`. Each tuple passes options to different `pgfplots` macros that generate the axes and the plots according to:

- Tuples of the form `obj/{opt}`:

- `tikz/{opt}`: modify picture properties by adding options `{opt}` to the `tikzpicture` environment.
- `axes/{opt}`: modify axis properties by adding options `{opt}` to the `axis` environment.
- `commands/{opt}`: add any valid TikZ commands inside `axis` environment. The commands passed to `opt` need to be valid TikZ commands, separated by semicolons as usual.

• Tuples of the form `{opt}` are passed directly to the `axis` environment.

The frequency limits are translated to the x-axis limits and the domain of the `axis` environment.

`\addNicholsZPKChart`

`\addNicholsZPKChart[`⟨*plot-options*⟩`]`
`{`⟨*z/*`{`⟨*zeros*⟩`}`*,p/*`{`⟨*poles*⟩`}`*,k/*`{`⟨*gain*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Generates a twp parametric functions for the magnitude and the phase a transfer function in ZPK form. The generated magnitude and phase parametric functions are passed to the `\addplot` macro. This macro can be used inside any `axis` environment as long as a domain for the x-axis is supplied through either the `plot-options` interface or directly in the optional argument of the container `axis` environment. Use with the `NicholsChart` environment supplied with this package is recommended. The mandatory argument is the same as `\BodeZPK`.

`\addNicholsTFChart`

`\addNicholsTFChart[`⟨*plot-options*⟩`]`
`{`⟨*num/*`{`⟨*coeffs*⟩`}`*,den/*`{`⟨*coeffs*⟩`}`*,d/*`{`⟨*delay*⟩`}`⟩`}`

Similar to `\addNicholsZPKChart`, with a transfer function input in the TF form.

3 Implementation

3.1 Initialization

`\pdfstrcmp`

The package makes extensive use of the `\pdfstrcmp` macro to parse options. Since that macro is not available in `lualatex`, this code is needed.

```
\RequirePackage{ifluatex}%
\ifluatex
  \let\pdfstrcmp\pdf@strcmp
\fi
```

`\n@mod`
`\n@pow`
`gnuplot@id`
`gnuplot@prefix`
`gnuplot@degrees`

This code is needed to support both `pgfplots` and `gnuplot` simultaneously. New macros are defined for the `pow` and `mod` functions to address differences between the two math engines. We start by processing the `pgf` class option.

```
\newif\if@pgfarg\@pgfargfalse
\DeclareOption{pgf}{%
  \@pgfargtrue
}
\ProcessOptions\relax
```

Then, we define two new macros to unify `pgfplots` and `gnuplot`.

```
\if@pgfarg
```

```
  \newcommand{\n@pow}[2]{(#1)^(#2)}%
  \newcommand{\n@mod}[2]{mod((#1),(#2))}%
\else
  \newcommand{\n@pow}[2]{(#1)**(#2)}%
  \newcommand{\n@mod}[2]{(#1)-(floor((#1)/(#2))*(#2))}%
```

Then, we create a counter so that a new data table is generated and for each new plot. If the plot macros have not changed, the tables, once generated, can be reused by `gnuplot`, which reduces compilation time.

```
  \newcounter{gnuplot@id}%
  \setcounter{gnuplot@id}{0}%
  \tikzset{%
    gnuplot@prefix/.style={%
      id=\arabic{gnuplot@id},
      prefix=gnuplot/\jobname
    }%
  }
```

Then, we add `set angles degrees` to all `gnuplot` macros to avoid having to convert from degrees to radians everywhere.

```
  \pgfplotsset{%
    gnuplot@degrees/.code={%
      \ifnum\value{gnuplot@id}=1
        \xdef\pgfplots@gnuplot@format{\pgfplots@gnuplot@format set angles degrees;}%
      \fi
    }%
  }
```

If the operating system is not Windows, we create the `gnuplot` folder if it does not already exist.

```
  \ifwindows\else
    \immediate\write18{mkdir -p gnuplot}%
  \fi
\fi
```

`bode@style` Default axis properties for all plot macros are collected in this `pgf` style.

```
\pgfplotsset{%
  bode@style/.style = {%
    label style={font=\footnotesize},
    tick label style={font=\footnotesize},
    grid=both,
    major grid style={color=gray!80},
    minor grid style={color=gray!20},
    x label style={at={(ticklabel cs:0.5)},anchor=near ticklabel},
    y label style={at={(ticklabel cs:0.5)},anchor=near ticklabel},
    scale only axis,
    samples=200,
    width=5cm,
  }%
}
```

3.2 Parametric function generators for poles, zeros, gains, and delays.

`\MagK` `\MagKAsymp` `\MagKLin` `\PhK` `\PhKAsymp` `\PhKLin`

True, linear, and asymptotic magnitude and phase parametric functions for a pure gain $G(s) = k + 0\mathrm{i}$. The macros take two arguments corresponding to real and imaginary part of the gain to facilitate code reuse between delays, gains, poles, and zeros, but only real gains are supported. The second argument, if supplied, is ignored.

```
\newcommand*{\MagK}[2]{(20*log10(abs(#1)))}
\newcommand*{\MagKAsymp}{\MagK}
\newcommand*{\MagKLin}{\MagK}
\newcommand*{\PhK}[2]{(#1<0?-180:0)}
\newcommand*{\PhKAsymp}{\PhK}
\newcommand*{\PhKLin}{\PhK}
```

`\PhKAsymp` `\PhKLin`

True magnitude and phase parametric functions for a pure delay $G(s) = e^{-Ts}$. The macros take two arguments corresponding to real and imaginary part of the gain to facilitate code reuse between delays, gains, poles, and zeros, but only real gains are supported. The second argument, if supplied, is ignored.

```
\newcommand*{\MagDel}[2]{0}
\newcommand*{\PhDel}[2]{-#1*180*t/pi}
```

`\MagPole` `\MagPoleAsymp` `\MagPoleLin` `\PhPole` `\PhPoleAsymp` `\PhPoleLin`

These macros are the building blocks for most of the plotting functions provided by this package. We start with Parametric function for the true magnitude of a complex pole.

```
\newcommand*{\MagPole}[2]
  {(-20*log10(sqrt(\n@pow{#1}{2} + \n@pow{t - (#2)}{2})))}
```

Parametric function for linear approximation of the magnitude of a complex pole.

```
\newcommand*{\MagPoleLin}[2]{(t < sqrt(\n@pow{#1}{2} + \n@pow{#2}{2}) ?
  -20*log10(sqrt(\n@pow{#1}{2} + \n@pow{#2}{2})) :
  -20*log10(t)
  )}
```

Parametric function for asymptotic approximation of the magnitude of a complex pole, same as linear approximation.

```
\newcommand*{\MagPoleAsymp}{\MagPoleLin}
```

Parametric function for the true phase of a complex pole.

```
\newcommand*{\PhPole}[2]{(#1 > 0 ? (#2 > 0 ?
  (\n@mod{-atan2((t - (#2)),-(#1))+360}{360}) :
  (-atan2((t - (#2)),-(#1)))) :
  (-atan2((t - (#2)),-(#1))))}
```

Parametric function for linear approximation of the phase of a complex pole.

```
\newcommand*{\PhPoleLin}[2]{%
  (abs(#1)+abs(#2) == 0 ? -90 :
  (t < (sqrt(\n@pow{#1}{2} + \n@pow{#2}{2}) /
    (\n@pow{10}{sqrt(\n@pow{#1}{2}/(\n@pow{#1}{2} + \n@pow{#2}{2}))})) ?
  (-atan2(-(#2),-(#1))) :
```

```
  (t >= (sqrt(\n@pow{#1}{2} + \n@pow{#2}{2}) *
    (\n@pow{10}{sqrt(\n@pow{#1}{2}/(\n@pow{#1}{2} + \n@pow{#2}{2}))})) ?
  (#2>0?(#1>0?270:-90):-90) :
  (-atan2(-(#2),-(#1)) + (log10(t/(sqrt(\n@pow{#1}{2} + \n@pow{#2}{2}) /
    (\n@pow{10}{sqrt(\n@pow{#1}{2}/(\n@pow{#1}{2} +
    \n@pow{#2}{2}))}))))*((#2>0?(#1>0?270:-90):-90) + atan2(-(#2),-(#1)))/
    (log10(\n@pow{10}{sqrt((4*\n@pow{#1}{2})/
    (\n@pow{#1}{2} + \n@pow{#2}{2}))})))))))}
```

Parametric function for asymptotic approximation of the phase of a complex pole.

```
\newcommand*{\PhPoleAsymp}[2]{(t < (sqrt(\n@pow{#1}{2} + \n@pow{#2}{2})) ?
  (-atan2(-(#2),-(#1))) :
  (#2>0?(#1>0?270:-90):-90))}
```

\MagZero
\MagZeroAsymp
\MagZeroLin
\PhZero
\PhZeroAsymp
\PhZeroLin

Plots of zeros are defined to be negative of plots of poles. The `0-` is necessary due to a bug in `gnuplot` (fixed in version 5.4, patchlevel 3).

```
\newcommand*{\MagZero}{0-\MagPole}
\newcommand*{\MagZeroLin}{0-\MagPoleLin}
\newcommand*{\MagZeroAsymp}{0-\MagPoleAsymp}
\newcommand*{\PhZero}{0-\PhPole}
\newcommand*{\PhZeroLin}{0-\PhPoleLin}
\newcommand*{\PhZeroAsymp}{0-\PhPoleAsymp}
```

3.3 Second order systems.

Although second order systems can be dealt with using the macros defined so far, the following dedicated macros for second order systems involve less computation.

\MagCSPoles
\MagCSPolesAsymp
\MagCSPolesLin
\PhCSPoles
\PhCSPolesAsymp
\PhCSPolesLin
\MagCSZeros
\MagCSZerosAsymp
\MagCSZerosLin
\PhCSZeros
\PhCSZerosAsymp
\PhCSZerosLin

Consider the canonical second order transfer function $G(s) = \frac{1}{s^2+2\zeta w_n s+w_n^2}$. We start with true, linear, and asymptotic magnitude plots for this transfer function.

```
\newcommand*{\MagCSPoles}[2]{(-20*log10(sqrt(\n@pow{\n@pow{#2}{2}
    - \n@pow{t}{2}}{2} + \n@pow{2*#1*#2*t}{2})))}
\newcommand*{\MagCSPolesLin}[2]{(t < #2 ? -40*log10(#2) : - 40*log10(t))}
\newcommand*{\MagCSPolesAsymp}{\MagCSPolesLin}
```

Then, we have true, linear, and asymptotic phase plots for the canonical second order transfer function.

```
\newcommand*{\PhCSPoles}[2]{(-atan2((2*(#1)*(#2)*t),(\n@pow{#2}{2}
  - \n@pow{t}{2})))}
\newcommand*{\PhCSPolesLin}[2]{(t < (#2 / (\n@pow{10}{abs(#1)})) ?
  0 :
  (t >= (#2 * (\n@pow{10}{abs(#1)})) ?
  (#1>0 ? -180 : 180) :
  (#1>0 ? (-180*(log10(t*(\n@pow{10}{#1})/#2))/(2*#1)) :
    (180*(log10(t*(\n@pow{10}{abs(#1)})/#2))/(2*abs(#1))))))}
\newcommand*{\PhCSPolesAsymp}[2]{(#1>0?(t<#2?0:-180):(t<#2?0:180))}
```

Plots of the inverse function $G(s) = s^2 + 2\zeta\omega_n s + \omega_n^2$ are defined to be negative of plots of poles. The `0-` is necessary due to a bug in `gnuplot` (fixed in version 5.4, patchlevel 3).

```
\newcommand*{\MagCSZeros}{0-\MagCSPoles}
\newcommand*{\MagCSZerosLin}{0-\MagCSPolesLin}
\newcommand*{\MagCSZerosAsymp}{0-\MagCSPolesAsymp}
\newcommand*{\PhCSZeros}{0-\PhCSPoles}
\newcommand*{\PhCSZerosLin}{0-\PhCSPolesLin}
\newcommand*{\PhCSZerosAsymp}{0-\PhCSPolesAsymp}
```

`\MagCSPolesPeak` `\MagCSZerosPeak`

These macros are used to add a resonant peak to linear and asymptotic plots of canonical second order poles and zeros. Since the plots are parametric, a separate `\draw` command is needed to add a vertical arrow.

```
\newcommand*{\MagCSPolesPeak}[3][]{%
  \draw[#1,->] (axis cs:{#3},{-40*log10(#3)}) --
  (axis cs:{#3},{-40*log10(#3)-20*log10(2*abs(#2))})
}
\newcommand*{\MagCSZerosPeak}[3][]{%
  \draw[#1,->] (axis cs:{#3},{40*log10(#3)}) --
  (axis cs:{#3},{40*log10(#3)+20*log10(2*abs(#2))})
}
```

`\MagSOPoles` `\MagSOPolesAsymp` `\MagSOPolesLin` `\PhSOPoles` `\PhSOPolesAsymp` `\PhSOPolesLin` `\MagSOZeros` `\MagSOZerosAsymp` `\MagSOZerosLin` `\PhSOZeros` `\PhSOZerosAsymp` `\PhSOZerosLin`

Consider a general second order transfer function $G(s) = \frac{1}{s^2+as+b}$. We start with true, linear, and asymptotic magnitude plots for this transfer function.

```
\newcommand*{\MagSOPoles}[2]{%
  (-20*log10(sqrt(\n@pow{#2 - \n@pow{t}{2}}{2} + \n@pow{#1*t}{2})))}
\newcommand*{\MagSOPolesLin}[2]{%
  (t < sqrt(abs(#2)) ? -20*log10(abs(#2)) : - 40*log10(t))}
\newcommand*{\MagSOPolesAsymp}{\MagSOPolesLin}
```

Then, we have true, linear, and asymptotic phase plots for the general second order transfer function.

```
\newcommand*{\PhSOPoles}[2]{(-atan2((#1)*t,((#2) - \n@pow{t}{2})))}
\newcommand*{\PhSOPolesLin}[2]{(#2>0 ?
  \PhCSPolesLin{(#1/(2*sqrt(#2)))}{(sqrt(#2))} :
  (#1>0 ? -180 : 180))}
\newcommand*{\PhSOPolesAsymp}[2]{(#2>0 ?
  \PhCSPolesAsymp{(#1/(2*sqrt(#2)))}{(sqrt(#2))} :
  (#1>0 ? -180 : 180))}
```

Plots of the inverse function $G(s) = s^2 + as + b$ are defined to be negative of plots of poles. The `0-` is necessary due to a bug in `gnuplot` (fixed in version 5.4, patchlevel 3).

```
\newcommand*{\MagSOZeros}{0-\MagSOPoles}
\newcommand*{\MagSOZerosLin}{0-\MagSOPolesLin}
\newcommand*{\MagSOZerosAsymp}{0-\MagSOPolesAsymp}
\newcommand*{\PhSOZeros}{0-\PhSOPoles}
\newcommand*{\PhSOZerosLin}{0-\PhSOPolesLin}
\newcommand*{\PhSOZerosAsymp}{0-\PhSOPolesAsymp}
```

`\MagSOPolesPeak` `\MagSOZerosPeak`

These macros are used to add a resonant peak to linear and asymptotic plots of general second order poles and zeros. Since the plots are parametric, a separate `\draw` command is needed to add a vertical arrow.

```
\newcommand*{\MagSOPolesPeak}[3][]{%
  \draw[#1,->] (axis cs:{sqrt(abs(#3))},{-20*log10(abs(#3))}) --
  (axis cs:{sqrt(abs(#3))},{-20*log10(abs(#3)) -
    20*log10(abs(#2/sqrt(abs(#3))))});
}
\newcommand*{\MagSOZerosPeak}[3][]{%
  \draw[#1,->] (axis cs:{sqrt(abs(#3))},{20*log10(abs(#3))}) --
  (axis cs:{sqrt(abs(#3))},{20*log10(abs(#3)) +
    20*log10(abs(#2/sqrt(abs(#3))))});
}
```

3.4 Commands for Bode plots

3.4.1 User macros

`\BodeZPK` This macro takes lists of complex poles and zeros of the form `{re,im}`, and values of gain and delay as inputs and constructs parametric functions for the Bode magnitude and phase plots. This is done by adding together the parametric functions generated by the macros for individual zeros, poles, gain, and delay, described above. The parametric functions are then plotted in a `tikzpicture` environment using the `\addplot` macro. Unless the package is loaded with the option `pgf`, the parametric functions are evaluated using `gnuplot`.

```
\newcommand{\BodeZPK}[4][approx/true]{%
```

Most of the work is done by the `\parse@opt` and the `\build@ZPK@plot` macros, described in the 'Internal macros' section. The former is used to parse the optional arguments and the latter to extract poles, zeros, gain, and delay from the first mandatory argument and to generate macros `\func@mag` and `\func@ph` that hold the magnitude and phase parametric functions.

```
  \parse@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@ZPK@plot{\func@mag}{\func@ph}{\opt@approx}{#2}%
```

The `\noexpand` macros below are needed to so that only the macro `\opt@group` is expanded.

```
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{groupplot}[%
      bode@style,
      xmin={#3},
      xmax={#4},
      domain=#3:#4,
      height=2.5cm,
      xmode=log,
      group style = {group size = 1 by 2,vertical sep=0.25cm},
      \opt@group
    ]%
  }%
  \temp@cmd
```

To ensure frequency tick marks on magnitude and the phase plots are always aligned, we use the groupplot library. The \expandafter chain below is used to expand macros in the plot and group optional arguments.

```
      \if@pgfarg
        \expandafter\nextgroupplot\expandafter[ytick distance=20,
          ylabel={Gain (dB)},xmajorticks=false,\optmag@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optmag@plot]}%
        \temp@cmd {\func@mag};
        \optmag@commands
        \expandafter\nextgroupplot\expandafter[ytick distance=45,
          ylabel={Phase ($^{\circ}$)},xlabel={Frequency (rad/s)},\optph@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optph@plot]}%
        \temp@cmd {\func@ph};
        \optph@commands
      \else
```

In gnuplot mode, we increment the gnuplot@id counter before every plot to make sure that new and reusable .gnuplot and .table files are generated for every plot.

```
      \stepcounter{gnuplot@id}
      \expandafter\nextgroupplot\expandafter[ytick distance=20,
        ylabel={Gain (dB)},xmajorticks=false,\optmag@axes]
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optmag@plot]}%
      \temp@cmd gnuplot[gnuplot@degrees,gnuplot@prefix] {\func@mag};
      \optmag@commands
      \stepcounter{gnuplot@id}
      \expandafter\nextgroupplot\expandafter[ytick distance=45,
        ylabel={Phase ($^{\circ}$)},xlabel={Frequency (rad/s)},\optph@axes]
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optph@plot]}%
      \temp@cmd gnuplot[gnuplot@degrees,gnuplot@prefix] {\func@ph};
      \optph@commands
    \fi
    \end{groupplot}
  \end{tikzpicture}
}
```

\BodeTF Implementation of this macro is very similar to the \BodeZPK macro above. The only difference is the lack of linear and asymptotic plots and slightly different parsing of the mandatory arguments.

```
\newcommand{\BodeTF}[4][]{%
  \parse@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@TF@plot{\func@mag}{\func@ph}{#2}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{groupplot}[%
      bode@style,
      xmin={#3},
      xmax={#4},
      domain=#3:#4,
```

```
      height=2.5cm,
      xmode=log,
      group style = {group size = 1 by 2,vertical sep=0.25cm},
      \opt@group
    ]%
  }%
  \temp@cmd
      \if@pgfarg
        \expandafter\nextgroupplot\expandafter[ytick distance=20,
          ylabel={Gain (dB)},xmajorticks=false,\optmag@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optmag@plot]}%
        \temp@cmd {\func@mag};
        \optmag@commands
        \expandafter\nextgroupplot\expandafter[ytick distance=45,
          ylabel={Phase ($^{\circ}$)},xlabel={Frequency (rad/s)},\optph@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optph@plot]}%
        \temp@cmd {\func@ph};
        \optph@commands
      \else
        \stepcounter{gnuplot@id}%
        \expandafter\nextgroupplot\expandafter[ytick distance=20,
          ylabel={Gain (dB)},xmajorticks=false,\optmag@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optmag@plot]}%
        \temp@cmd gnuplot[gnuplot@degrees,gnuplot@prefix] {\func@mag};
        \optmag@commands
        \stepcounter{gnuplot@id}%
        \expandafter\nextgroupplot\expandafter[ytick distance=45,
          ylabel={Phase ($^{\circ}$)},xlabel={Frequency (rad/s)},\optph@axes]
        \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\optph@plot]}%
        \temp@cmd gnuplot[gnuplot@degrees,gnuplot@prefix] {\func@ph};
        \optph@commands
      \fi
    \end{groupplot}
  \end{tikzpicture}
}
```

`\addBodeZPKPlots` This macro is designed to issues multiple `\addplot` macros for the same set of poles, zeros, gain, and delay. All of the work is done by the `\build@ZPK@plot` macro.

```
\newcommand{\addBodeZPKPlots}[3][true/{}]{%
  \foreach \approx/\opt in {#1} {%
    \gdef\plot@macro{}%
    \gdef\temp@macro{}%
    \ifnum\pdfstrcmp{#2}{phase}=0
      \build@ZPK@plot{\temp@macro}{\plot@macro}{\approx}{#3}%
    \else
      \build@ZPK@plot{\plot@macro}{\temp@macro}{\approx}{#3}%
    \fi
    \if@pgfarg
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\opt]}%
```

```
      \temp@cmd {\plot@macro};
    \else
      \stepcounter{gnuplot@id}%
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\opt]}
      \temp@cmd gnuplot[gnuplot@degrees,gnuplot@prefix] {\plot@macro};
    \fi
  }%
}
```

`\addBodeTFPlot` This macro is designed to issues a single `\addplot` macros for the set of coefficients and delay. All of the work is done by the `\build@TF@plot` macro.

```
\newcommand{\addBodeTFPlot}[3][thick]{%
  \gdef\plot@macro{}%
  \gdef\temp@macro{}%
  \ifnum\pdfstrcmp{#2}{phase}=0
    \build@TF@plot{\temp@macro}{\plot@macro}{#3}%
  \else
    \build@TF@plot{\plot@macro}{\temp@macro}{#3}%
  \fi
  \if@pgfarg
    \addplot[variable=t,#1]{\plot@macro};
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[gnuplot@degrees, gnuplot@prefix] {\plot@macro};
  \fi
}
```

`\addBodeComponentPlot` This macro is designed to issue a single `\addplot` macro capable of plotting linear combinations of the basic components described in Section 2.1.1. The only work to do here is to handle the `pgf` package option.

```
\newcommand{\addBodeComponentPlot}[2][thick]{%
  \if@pgfarg
    \addplot[variable=t,#1]{#2};
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[gnuplot@degrees,gnuplot@prefix] {#2};
  \fi
}
```

`BodePlot` An environment to host macros that pass parametric functions to `\addplot` macros. Uses the defaults specified in `bode@style` to create a shortcut that includes the `tikzpicture` and `semilogaxis` environments.

```
\newenvironment{BodePlot}[3][]{%
  \parse@env@opt{#1}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]
    \noexpand\begin{semilogxaxis}[%
      bode@style,
      xmin={#2},
      xmax={#3},
```

```
      domain=#2:#3,
      height=2.5cm,
      xlabel={Frequency (rad/s)},
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
}{
    \end{semilogxaxis}
  \end{tikzpicture}
}
```

3.4.2 Internal macros

`\add@feature` This is an internal macro to add a basic component (pole, zero, gain, or delay), described using one of the macros in Section 2.1.1 (input `#2`), to a parametric function stored in a global macro (input `#1`). The basic component value (input `#3`) is a complex number of the form `{re,im}`. If the imaginary part is missing, it is assumed to be zero. Implementation made possible by this StackExchange answer.

```
\newcommand*{\add@feature}[3]{%
  \ifcat$\detokenize\expandafter{#1}$%
    \xdef#1{\unexpanded\expandafter{#1 0+#2}}%
  \else
    \xdef#1{\unexpanded\expandafter{#1+#2}}%
  \fi
  \foreach \y [count=\n] in #3 {%
    \xdef#1{\unexpanded\expandafter{#1}{\y}}%
    \xdef\Last@LoopValue{\n}%
  }%
  \ifnum\Last@LoopValue=1%
    \xdef#1{\unexpanded\expandafter{#1}{0}}%
  \fi
}
```

`\build@ZPK@plot` This is an internal macro to build parametric Bode magnitude and phase plots by concatenating basic component (pole, zero, gain, or delay) macros (Section 2.1.1) to global magnitude and phase macros (inputs `#1` and `#2`). The `\add@feature` macro is used to do the concatenation. The basic component macros are inferred from a `feature/{values}` list, where `feature` is one of `z`,`p`,`k`, and `d`, for zeros, poles, gain, and delay, respectively, and `{values}` is a comma separated list of comma separated lists (complex numbers of the form `{re,im}`). If the imaginary part is missing, it is assumed to be zero.

```
\newcommand{\build@ZPK@plot}[4]{%
  \foreach \feature/\values in {#4} {%
    \ifnum\pdfstrcmp{\feature}{z}=0
      \foreach \z in \values {%
        \ifnum\pdfstrcmp{#3}{linear}=0
```

```
        \add@feature{#2}{\PhZeroLin}{\z}%
        \add@feature{#1}{\MagZeroLin}{\z}%
      \else
        \ifnum\pdfstrcmp{#3}{asymptotic}=0
          \add@feature{#2}{\PhZeroAsymp}{\z}%
          \add@feature{#1}{\MagZeroAsymp}{\z}%
        \else
          \add@feature{#2}{\PhZero}{\z}%
          \add@feature{#1}{\MagZero}{\z}%
        \fi
      \fi
    }%
  \fi
  \ifnum\pdfstrcmp{\feature}{p}=0
    \foreach \p in \values {%
      \ifnum\pdfstrcmp{#3}{linear}=0
        \add@feature{#2}{\PhPoleLin}{\p}%
        \add@feature{#1}{\MagPoleLin}{\p}%
      \else
        \ifnum\pdfstrcmp{#3}{asymptotic}=0
          \add@feature{#2}{\PhPoleAsymp}{\p}%
          \add@feature{#1}{\MagPoleAsymp}{\p}%
        \else
          \add@feature{#2}{\PhPole}{\p}%
          \add@feature{#1}{\MagPole}{\p}%
        \fi
      \fi
    }%
  \fi
  \ifnum\pdfstrcmp{\feature}{k}=0
    \ifnum\pdfstrcmp{#3}{linear}=0
      \add@feature{#2}{\PhKLin}{\values}%
      \add@feature{#1}{\MagKLin}{\values}%
    \else
      \ifnum\pdfstrcmp{#3}{asymptotic}=0
        \add@feature{#2}{\PhKAsymp}{\values}%
        \add@feature{#1}{\MagKAsymp}{\values}%
      \else
        \add@feature{#2}{\PhK}{\values}%
        \add@feature{#1}{\MagK}{\values}%
      \fi
    \fi
  \fi
  \ifnum\pdfstrcmp{\feature}{d}=0
    \ifnum\pdfstrcmp{#3}{linear}=0
      \PackageError {bodeplot} {Linear approximation for pure delays is not
      supported.} {Plot the true Bode plot using ‘true’ instead of ‘linear’.}
    \else
      \ifnum\pdfstrcmp{#3}{asymptotic}=0
        \PackageError {bodeplot} {Asymptotic approximation for pure delays is not
```

```
          supported.} {Plot the true Bode plot using ‘true’ instead of ‘asymptotic’.}
        \else
          \ifdim\values pt < 0pt
            \PackageError {bodeplot} {Delay needs to be a positive number.}
          \fi
          \add@feature{#2}{\PhDel}{\values}%
          \add@feature{#1}{\MagDel}{\values}%
        \fi
      \fi
    \fi
  }%
}
```

`\build@TF@plot` This is an internal macro to build parametric Bode magnitude and phase functions by computing the magnitude and the phase given numerator and denominator coefficients and delay (input #3). The functions are assigned to user-supplied global magnitude and phase macros (inputs #1 and #2).

```
\newcommand{\build@TF@plot}[3]{%
  \gdef\num@real{0}%
  \gdef\num@im{0}%
  \gdef\den@real{0}%
  \gdef\den@im{0}%
  \gdef\loop@delay{0}%
  \foreach \feature/\values in {#3} {%
    \ifnum\pdfstrcmp{\feature}{num}=0
      \foreach \numcoeff [count=\numpow] in \values {%
        \xdef\num@degree{\numpow}%
      }%
      \foreach \numcoeff [count=\numpow] in \values {%
        \pgfmathtruncatemacro{\currentdegree}{\num@degree-\numpow}%
        \ifnum\currentdegree = 0
          \xdef\num@real{\num@real+\numcoeff}%
        \else
          \ifodd\currentdegree
            \xdef\num@im{\num@im+(\numcoeff*(\n@pow{-1}{(\currentdegree-1)/2})*%
              (\n@pow{t}{\currentdegree}))}%
          \else
            \xdef\num@real{\num@real+(\numcoeff*(\n@pow{-1}{(\currentdegree)/2})*%
              (\n@pow{t}{\currentdegree}))}%
          \fi
        \fi
      }%
    \fi
    \ifnum\pdfstrcmp{\feature}{den}=0
      \foreach \dencoeff [count=\denpow] in \values {%
        \xdef\den@degree{\denpow}%
      }%
      \foreach \dencoeff [count=\denpow] in \values {%
        \pgfmathtruncatemacro{\currentdegree}{\den@degree-\denpow}%
        \ifnum\currentdegree = 0
```

```
          \xdef\den@real{\den@real+\dencoeff}%
        \else
          \ifodd\currentdegree
            \xdef\den@im{\den@im+(\dencoeff*(\n@pow{-1}{(\currentdegree-1)/2})*%
              (\n@pow{t}{\currentdegree}))}%
          \else
            \xdef\den@real{\den@real+(\dencoeff*(\n@pow{-1}{(\currentdegree)/2})*%
              (\n@pow{t}{\currentdegree}))}%
          \fi
        \fi
      }%
    \fi
    \ifnum\pdfstrcmp{\feature}{d}=0
      \xdef\loop@delay{\values}%
    \fi
  }%
  \xdef#2{(\n@mod{atan2((\num@im),(\num@real))-atan2((\den@im),%
    (\den@real))+360}{360}-\loop@delay*180*t/pi)}%
  \xdef#1{(20*log10(sqrt((\n@pow{\num@real}{2})+(\n@pow{\num@im}{2})))-%
    20*log10(sqrt((\n@pow{\den@real}{2})+(\n@pow{\den@im}{2}))))}%
}
```

`\parse@opt` Parses options supplied to the main Bode macros. A `for` loop over tuples of the form `\obj/\typ/\opt` with a long list of nested if-else statements does the job. If the input `\obj` is `plot`, `axes`, `group`, `approx`, or `tikz` the corresponding `\opt` are passed, unexpanded, to the `\addplot` macro, the `\nextgroupplot` macro, the `groupplot` environment, the `\build@ZPK@plot` macro, and the `tikzpicture` environment, respectively. If `\obj` is `commands`, the corresponding `\opt` are stored, unexpanded, in the macros `\optph@commands` and `\optmag@commands`, to be executed in appropriate `axis` environments.

```
\newcommand{\parse@opt}[1]{%
  \gdef\optmag@axes{}%
  \gdef\optph@axes{}%
  \gdef\optph@plot{}%
  \gdef\optmag@plot{}%
  \gdef\opt@group{}%
  \gdef\opt@approx{}%
  \gdef\optph@commands{}%
  \gdef\optmag@commands{}%
  \gdef\opt@tikz{}%
  \foreach \obj/\typ/\opt in {#1} {%
    \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{plot}=0
      \ifnum\pdfstrcmp{\unexpanded\expandafter{\typ}}{mag}=0
        \xdef\optmag@plot{\unexpanded\expandafter{\opt}}%
      \else
        \ifnum\pdfstrcmp{\unexpanded\expandafter{\typ}}{ph}=0
          \xdef\optph@plot{\unexpanded\expandafter{\opt}}%
        \else
          \xdef\optmag@plot{\unexpanded\expandafter{\opt}}%
```

```
          \xdef\optph@plot{\unexpanded\expandafter{\opt}}%
        \fi
      \fi
    \else
      \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{axes}=0
        \ifnum\pdfstrcmp{\unexpanded\expandafter{\typ}}{mag}=0
          \xdef\optmag@axes{\unexpanded\expandafter{\opt}}%
        \else
          \ifnum\pdfstrcmp{\unexpanded\expandafter{\typ}}{ph}=0
            \xdef\optph@axes{\unexpanded\expandafter{\opt}}%
          \else
            \xdef\optmag@axes{\unexpanded\expandafter{\opt}}%
            \xdef\optph@axes{\unexpanded\expandafter{\opt}}%
          \fi
        \fi
      \else
        \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{group}=0
          \xdef\opt@group{\unexpanded\expandafter{\opt}}%
        \else
          \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{approx}=0
            \xdef\opt@approx{\unexpanded\expandafter{\opt}}%
          \else
            \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{commands}=0
              \ifnum\pdfstrcmp{\unexpanded\expandafter{\typ}}{ph}=0
                \xdef\optph@commands{\unexpanded\expandafter{\opt}}%
              \else
                \xdef\optmag@commands{\unexpanded\expandafter{\opt}}%
              \fi
            \else
              \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{tikz}=0
                \xdef\opt@tikz{\unexpanded\expandafter{\opt}}%
              \else
                \xdef\optmag@plot{\unexpanded\expandafter{\optmag@plot},
                  \unexpanded\expandafter{\obj}}%
                \xdef\optph@plot{\unexpanded\expandafter{\optph@plot},
                  \unexpanded\expandafter{\obj}}%
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
  }%
}
```

`\parse@env@opt` Parses options supplied to the Bode, Nyquist, and Nichols environments. A `for` loop over tuples of the form `\obj/\opt`, processed using nested if-else statements does the job. The input `\obj` should either be `axes` or `tikz`, and the corresponding `\opt` are passed, unexpanded, to the `axis` environment and the `tikzpicture` environment, respectively.

```
\newcommand{\parse@env@opt}[1]{%
  \gdef\opt@axes{}%
  \gdef\opt@tikz{}%
  \foreach \obj/\opt in {#1} {%
    \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{axes}=0
      \xdef\opt@axes{\unexpanded\expandafter{\opt}}%
    \else
      \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{tikz}=0
        \xdef\opt@tikz{\unexpanded\expandafter{\opt}}%
      \else
        \xdef\opt@axes{\unexpanded\expandafter{\opt@axes},
          \unexpanded\expandafter{\obj}}%
      \fi
    \fi
  }%
}
```

3.5 Nyquist plots

3.5.1 User macros

`\NyquistZPK` Converts magnitude and phase parametric functions built using `\build@ZPK@plot` into real part and imaginary part parametric functions. A plot of these is the Nyquist plot. The parametric functions are then plotted in a `tikzpicture` environment using the `\addplot` macro. Unless the package is loaded with the option `pgf`, the parametric functions are evaluated using `gnuplot`. A large number of samples is typically needed to get a smooth plot because frequencies near 0 result in plot points that are very close to each other. Linear frequency sampling is unnecessarily fine near zero and very coarse for large ω. Logarithmic sampling makes it worse, perhaps inverse logarithmic sampling will help, pull requests to fix that are welcome!

```
\newcommand{\NyquistZPK}[4][]{%
  \parse@N@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@ZPK@plot{\func@mag}{\func@ph}{}{#2}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      domain=#3:#4,
      height=5cm,
      xlabel={$\Re$},
      ylabel={$\Im$},
      samples=500,
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
```

```
      \addplot [only marks,mark=+,thick,red] (-1 , 0);
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\unexpanded\expandafter{\opt@plot}]}%
      \if@pgfarg
        \temp@cmd ( {\n@pow{10}{((\func@mag)/20)}*cos(\func@ph)},
          {\n@pow{10}{((\func@mag)/20)}*sin(\func@ph)} );
        \opt@commands
      \else
        \stepcounter{gnuplot@id}%
        \temp@cmd gnuplot[parametric,gnuplot@degrees,gnuplot@prefix] {%
          \n@pow{10}{((\func@mag)/20)}*cos(\func@ph),
          \n@pow{10}{((\func@mag)/20)}*sin(\func@ph)};
        \opt@commands
      \fi
    \end{axis}
  \end{tikzpicture}
}
```

`\NyquistTF` Implementation of this macro is very similar to the `\NyquistZPK` macro above. The only difference is a slightly different parsing of the mandatory arguments via `\build@TF@plot`.

```
\newcommand{\NyquistTF}[4][]{%
  \parse@N@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@TF@plot{\func@mag}{\func@ph}{#2}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      domain=#3:#4,
      height=5cm,
      xlabel={$\Re$},
      ylabel={$\Im$},
      samples=500,
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
      \addplot [only marks,mark=+,thick,red] (-1 , 0);
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\unexpanded\expandafter{\opt@plot}]}%
      \if@pgfarg
        \temp@cmd ( {\n@pow{10}{((\func@mag)/20)}*cos(\func@ph)},
          {\n@pow{10}{((\func@mag)/20)}*sin(\func@ph)} );
        \opt@commands
      \else
        \stepcounter{gnuplot@id}%
        \temp@cmd gnuplot[parametric,gnuplot@degrees,gnuplot@prefix]{%
          \n@pow{10}{((\func@mag)/20)}*cos(\func@ph),
          \n@pow{10}{((\func@mag)/20)}*sin(\func@ph)};
        \opt@commands
      \fi
```

```
    \end{axis}
  \end{tikzpicture}
}
```

\addNyquistZPKPlot

Adds Nyquist plot of a transfer function in ZPK form. This macro is designed to pass two parametric function to an `\addplot` macro. The parametric functions for phase (`\func@ph`) and magnitude (`\func@mag`) are built using the `\build@ZPK@plot` macro, converted to real and imaginary parts and passed to `\addplot` commands.

```
\newcommand{\addNyquistZPKPlot}[2][]{%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@ZPK@plot{\func@mag}{\func@ph}{}{#2}%
  \if@pgfarg
    \addplot[variable=t,#1] ( {\n@pow{10}{((\func@mag)/20)}*cos(\func@ph)},
      {\n@pow{10}{((\func@mag)/20)}*sin(\func@ph)} );
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[parametric,gnuplot@degrees,gnuplot@prefix]{%
      \n@pow{10}{((\func@mag)/20)}*cos(\func@ph),
      \n@pow{10}{((\func@mag)/20)}*sin(\func@ph)};
  \fi
}
```

\addNyquistTFPlot

Adds Nyquist plot of a transfer function in TF form. This macro is designed to pass two parametric function to an `\addplot` macro. The parametric functions for phase (`\func@ph`) and magnitude (`\func@mag`) are built using the `\build@TF@plot` macro, converted to real and imaginary parts and passed to `\addplot` commands.

```
\newcommand{\addNyquistTFPlot}[2][]{%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@TF@plot{\func@mag}{\func@ph}{#2}%
  \if@pgfarg
    \addplot[variable=t,#1] ( {\n@pow{10}{((\func@mag)/20)}*cos(\func@ph)},
      {\n@pow{10}{((\func@mag)/20)}*sin(\func@ph)} );
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[parametric,gnuplot@degrees,gnuplot@prefix]{%
      \n@pow{10}{((\func@mag)/20)}*cos(\func@ph),
      \n@pow{10}{((\func@mag)/20)}*sin(\func@ph)};
  \fi
}
```

NyquistPlot

An environment to host `\addNyquist...` macros that pass parametric functions to `\addplot`. Uses the defaults specified in `bode@style` to create a shortcut that includes the `tikzpicture` and `axis` environments.

```
\newenvironment{NyquistPlot}[3][]{%
  \parse@env@opt{#1}%
```

```
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      height=5cm,
      domain=#2:#3,
      xlabel={$\Re$},
      ylabel={$\Im$},
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
    \addplot [only marks,mark=+,thick,red] (-1 , 0);
}{%
    \end{axis}
  \end{tikzpicture}
}
```

3.5.2 Internal commands

`\parse@N@opt` Parses options supplied to the main Nyquist and Nichols macros. A `for` loop over tuples of the form `\obj/\opt`, processed using nested if-else statements does the job. If the input `\obj` is `plot`, `axes`, or `tikz` then the corresponding `\opt` are passed, unexpanded, to the `\addplot` macro, the `axis` environment, and the `tikzpicture` environment, respectively.

```
\newcommand{\parse@N@opt}[1]{%
  \gdef\opt@axes{}%
  \gdef\opt@plot{}%
  \gdef\opt@commands{}%
  \gdef\opt@tikz{}
  \foreach \obj/\opt in {#1} {%
    \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{axes}=0
      \xdef\opt@axes{\unexpanded\expandafter{\opt}}%
    \else
      \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{plot}=0
        \xdef\opt@plot{\unexpanded\expandafter{\opt}}%
      \else
        \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{commands}=0
          \xdef\opt@commands{\unexpanded\expandafter{\opt}}%
        \else
          \ifnum\pdfstrcmp{\unexpanded\expandafter{\obj}}{tikz}=0
            \xdef\opt@tikz{\unexpanded\expandafter{\opt}}%
          \else
            \xdef\opt@plot{\unexpanded\expandafter{\opt@plot},
              \unexpanded\expandafter{\obj}}%
          \fi
        \fi
      \fi
    \fi
  }%
```

```
}
```

3.6 Nichols charts

\NicholsZPK
\NicholsTF
NicholsChart
\addNicholsZPKChart
\addNicholsTFChart

These macros and the NicholsChart environment generate Nichols charts, and they are implemented similar to their Nyquist counterparts.

```
\newcommand{\NicholsZPK}[4][]{%
  \parse@N@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@ZPK@plot{\func@mag}{\func@ph}{}{#2}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      domain=#3:#4,
      height=5cm,
      xlabel={Phase (degrees)},
      ylabel={Gain (dB)},
      samples=500,
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\opt@plot]}%
      \if@pgfarg
        \temp@cmd ( {\func@ph} , {\func@mag} );
        \opt@commands
      \else
        \stepcounter{gnuplot@id}%
        \temp@cmd gnuplot[parametric, gnuplot@degrees, gnuplot@prefix]
          { \func@ph , \func@mag };
        \opt@commands
      \fi
    \end{axis}
  \end{tikzpicture}
}
\newcommand{\NicholsTF}[4][]{%
  \parse@N@opt{#1}%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@TF@plot{\func@mag}{\func@ph}{#2}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      domain=#3:#4,
      height=5cm,
      xlabel={Phase (degrees)},
      ylabel={Gain (dB)},
      samples=500,
```

```
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
      \edef\temp@cmd{\noexpand\addplot[variable=t,thick,\opt@plot]}%
      \if@pgfarg
        \temp@cmd ( {\func@ph} , {\func@mag} );
        \opt@commands
      \else
        \stepcounter{gnuplot@id}%
        \temp@cmd gnuplot[parametric, gnuplot@degrees, gnuplot@prefix]
          { \func@ph , \func@mag };
        \opt@commands
      \fi
    \end{axis}
  \end{tikzpicture}
}
\newenvironment{NicholsChart}[3][]{%
  \parse@env@opt{#1}%
  \edef\temp@cmd{\noexpand\begin{tikzpicture}[\unexpanded\expandafter{\opt@tikz}]%
    \noexpand\begin{axis}[%
      bode@style,
      domain=#2:#3,
      height=5cm,
      xlabel={Phase (degrees)},
      ylabel={Gain (dB)},
      \unexpanded\expandafter{\opt@axes}
    ]%
  }%
  \temp@cmd
}{
    \end{axis}
  \end{tikzpicture}
}
\newcommand{\addNicholsZPKChart}[2][]{%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@ZPK@plot{\func@mag}{\func@ph}{}{#2}%
  \if@pgfarg
    \addplot[variable=t,#1] ( {\func@ph} , {\func@mag} );
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[parametric,gnuplot@degrees,gnuplot@prefix]
      {\func@ph , \func@mag};
  \fi
}
\newcommand{\addNicholsTFChart}[2][]{%
  \gdef\func@mag{}%
  \gdef\func@ph{}%
  \build@TF@plot{\func@mag}{\func@ph}{#2}%
```

```
  \if@pgfarg
    \addplot[variable=t,#1] ( {\func@ph} , {\func@mag} );
  \else
    \stepcounter{gnuplot@id}%
    \addplot[variable=t,#1] gnuplot[gnuplot@degrees,gnuplot@prefix]
      {\func@ph , \func@mag};
  \fi
}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History