Using the Julia Control Toolbox

KTH Royal Institute of Technology, Stockholm
School of Electrical Engineering, Department of Automatic Control

written by

Linus Härenstam-Nielsen

linusnie@kth.se

August 31, 2015

1 Introduction

This document will help you get started with using the toolbox ControlKTH in Julia and provides a brief introductions to the functions needed in a basic course in automatic control.

1.1 Getting Started

First, make sure to download and install Julia version 0.4 or higher from `http://julialang.org/downloads/`. At the time of writing you will need to scroll down to "Nightly builds". Once there, click the appropriate link for your system.

For the plotting tools used in ControlKTH to work you will also need to have Python and Matplotlib installed. Anaconda is a python distribution which includes both and can be downloaded from `http://continuum.io/downloads`.

If you haven't used Julia before make sure to familiarize yourself with the command line (called the REPL) and basic functionality. You can visit `http://docs.julialang.org/` for the official documentation and `https://www.youtube.com/watch?v=vWkgEddb4-A` for an introductory lecture.

1.2 Installation

Make sure that you have the package Requires installed by running `Pkg.add("Requires")`. Once that is done head to the EL1000 home page and download the ControlKTH .zip file.

Unzip the files in your Julia package directory (find it by calling `Pkg.dir()`). You can then load the ControlKTH package with `using ControlKTH`. The first time you do this several other packages that are needed will also be installed automatically and it may take some time. After the installation you are ready to start using the toolbox.

2 Features

This section will introduce the key features needed for a basic course in control theory.

2.1 LTI Systems

The toolbox stores LTI systems as objects. Systems can be manipulated using `+`, `-`, `*` and `/` as you would expect. Similarly, you can use `series`, `parallel` and `feedback` to create system connections.

There are three equivalent ways to represent an LTI-system, each of which correspond to a julia type (noted in parentheses): transfer function (`TransferFunction`), state space (`StateSpace`)and zero-pole-gain (`ZPK`). `TranferFunction`, `StateSpace` and `ZPK` are all subtypes of the abstract type `LTISystem`.

2.1.1 Transfer Function

An object of type `TranferFunction` stores the numerator and denominator polynomials of a transfer function. To create a `TranferFunction` object use `tf(num,den)`, where `num` and `den` are vectors containing the polynomial coefficients.

You can also construct a laplace variable using `s = tf("s")` which in turn can be used to construct transfer functions. For example: `s = tf("s")` followed by `H = (s+5)/(s^2+2s+2)` is equivalent to `H = tf([1,5],[1,2,2])`, both resulting in

```
TransferFunction:
    s + 5.0
----------------
s^2 + 2.0s + 2.0

Continuous-time transfer function model
```

2.1.2 State Space

An object of type `StateSpace` stores the state space matrices of an LTI system. To create a `StateSpace` object use `ss(A,B,C,D)`, where `A`, `B`, `C` and `D` are the state space matrices.

2.1.3 Zero-Pole-Gain

An object of type `ZPK` stores the zeros, poles and gain of a system. To create a `ZPK` object use `zpk(z,p,k)` where z and p are vectors containing the poles and zeroes of the system and k is the system gain. For example: `zpk([-5,-2],[1+im,1-im],3)` results in

```
ZPK:
    (s + 5.0)(s + 2.0)
3.0 ------------------
    (s^2 - 2.0s + 2.0)

Continuous-time zero-pole-gain model
```

2.2 Functions

Here is a summary of useful functions. For a more detailed explanations you can always get the documentation of a function by typing a "`?`" followed by the function name in the REPL (for example: `? bodeplot`).

You can also get a list of what combinations of arguments are accepted by a function using `methods` (for example `methods(bodeplot)`).

- `tf`, `ss` and `zpk` - initializes and LTI system. Also converts between system types.
- `zpkdata` - computes the zeros, poles and gain of a system.
- `bodeplot` - plots the bode-diagram of a system.
- `phasemargin` and `gainmargin` - calculates the phase and gain margins of a system. Use `marginplot` to plot the margins on a bode-diagram.
- `nyquistplot` - plots the nyquist diagram of a system.
- `rlocusplot` - plots the root locus of a system.
- `stepplot` and `impulseplot` - plots the step and impulse responce of a system.
- `stepinfo` - calculates rise time, setting time, overshoot, undershoot, peak value, peak time and final value of a system. There are also separate functions for calculating single properties (for example `risetime` calculates the rise time of a system).
- `minreal` - computes the minimal realization of a system (cancelling unnecessary pole/zero pairs).
- `ctrb` and `obsv` - computes the controllability and observability matrices of a system
- `acker` - calculates a feedback vector for pole placement using Ackermann's formula.

For a list of all avaliable functions run `names(ControlKTH)`.

2.3 Plotting

ControlKTH uses PyPlot to produce plots and diagrams. You can visit `https://github.com/stevengj/PyPlot.jl` for more information on how it works.

In ControlKTH all functions that produce plots end with `plot`. Plotting functions often have optional arguments so make sure to check the documentation.

3 Feedback and bug reports

If you have any questions/feedback or want to report a bug please send an email to `linusnie@kth.se`. As always if reporting a bug, try to provide a minimal working example (the smallest amount of code that reproduces the issue). Remember to include which version of Julia you are using as provided by the `versioninfo()` command.