controltheory

Adam Vogt

June 3, 2014

List of Figures


```
library(controltheory)
```

This package helps to treat linear systems in R. Specifically root locus and a time domain simulation (using deSolve) are implemented.

Polynomials are represented as vectors, as done in `base::polyroot` and the `polynom` package. A shorthand for `polynom::polynomial` is defined:

```
p( 1, 2, 3 )

## 1 + 2*x + 3*x^2
```

1 Root Locus

The root locus of the transfer function:

$$G_1 = N/D = \frac{(5+s)(1+s)}{s(3+s)(2+2s+s^2)} \tag{1}$$

shows where the poles of the closed-loop system, $kG_1/(1+kG_1)$, move as k increases from 0. Specifically, the poles are defined by the roots of the characteristic polynomial which will be called Δ ($\Delta = kN + D$). Points of interest, such as break-points, asymptotes and intervals in which k leads to a stable system are calculated following `http://lpsa.swarthmore.edu/Root_Locus/RootLocusReviewRules.html`

```r
eq1 <- rational( p(5, 1) *p(1, 1), p() * p(3, 1) * p(2, 2, 1))
print(eq1)
```

```
## 5 + 6*x + x^2
## -------------------------
## 6*x + 8*x^2 + 5*x^3 + x^4
```

```r
r1 <- rlocus(eq1, k.expand.f = 2)
lapply(r1, head, 3)
```

```
## $poles
##   k.idx pole k.int.idx Im          Re     k    k.int stability
## 1     1    1         1  0 -0.0008334 0.001 [0,14.5]    stable
## 2     1    2         1  1 -0.9994500 0.001 [0,14.5]    stable
## 3     1    3         1 -1 -0.9994500 0.001 [0,14.5]    stable
##
## $asymptotes
##     f pole  Re     Im
## 1 0.0    2 0.5 0.0000
## 2 0.0    3 0.5 0.0000
## 3 0.1    2 0.5 0.7879
##
## $points
##   type Re         Im
## 1    z -1 -5.457e-15
## 2    z -5  5.457e-15
## 3    p  0  0.000e+00
##
## $ks.stable
## $ks.stable$pp
##         ppInts sign
## 1 (-Inf,-2.34]   -1
## 2 (-2.34,14.5]    1
## 3  (14.5, Inf]   -1
##
## $ks.stable$cuts
## [1]   -Inf -2.343 14.510    Inf
```

```
library(ggplot2)
plot(ggplot( r1$poles, aes(Re, Im)) +
    geom_path(aes(group=interaction(pole, stability),
              col=pole, linetype=stability)) +
    geom_path(data=r1$asymptotes, linetype=3,
              aes(group=pole, col=pole)) +
    geom_point(data=subset(r1$points, type != 'centroid'),
              aes(shape=type))
    )
```

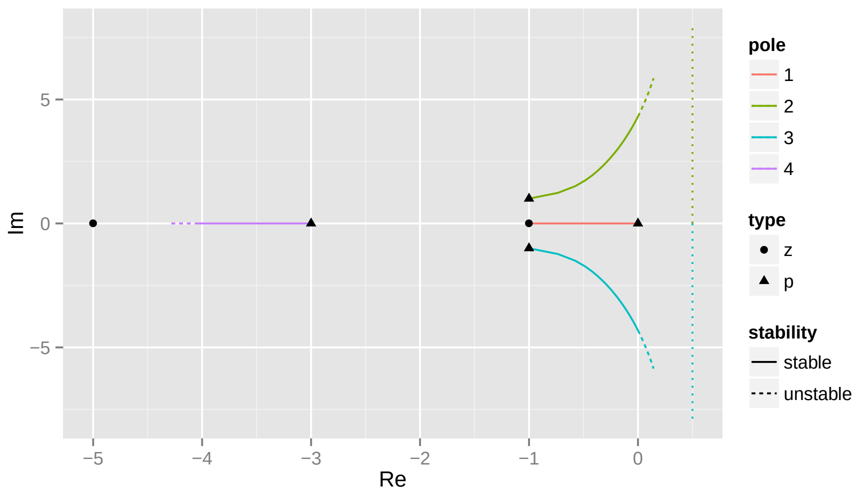


Figure 1: Root locus for equation (1)

2 Time domain

`ilt` is provided as an adapter for `deSolve::radau`. It numerically calculates an inverse laplace transform, at least if the provided transfer function is a ratio of polynomials.

2.1 Sin

As an example, here is $\mathcal{L}^{-1} 1/(1+s^2) = \sin(t)$

```
library(controltheory)
sint <- ilt( rational( 1, c(1, 0, 1)), times = seq(0, pi, by=0.3) )
unclass(sint)

##        time      dy.1       y
```

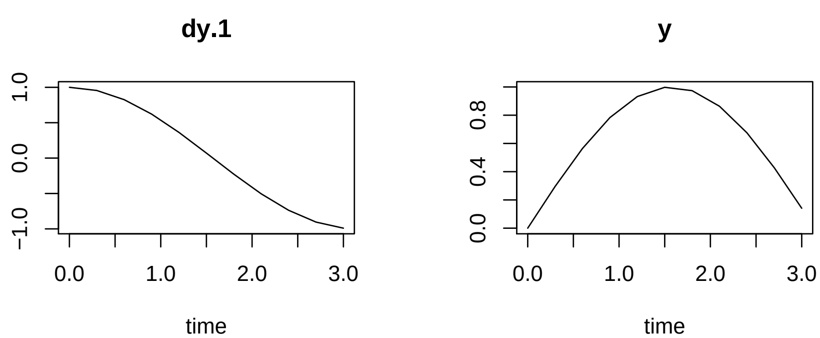
```
##  [1,]  0.0  1.00000 0.0000
##  [2,]  0.3  0.95534 0.2955
##  [3,]  0.6  0.82534 0.5646
##  [4,]  0.9  0.62161 0.7833
##  [5,]  1.2  0.36236 0.9320
##  [6,]  1.5  0.07074 0.9975
##  [7,]  1.8 -0.22720 0.9738
##  [8,]  2.1 -0.50485 0.8632
##  [9,]  2.4 -0.73739 0.6755
## [10,]  2.7 -0.90407 0.4274
## [11,]  3.0 -0.98999 0.1411
## attr(,"istate")
##  [1]  1  7 33  1 NA NA NA NA NA  2 NA NA  0 NA NA NA NA NA NA NA NA
## attr(,"rstate")
## [1] 0.2277 3.0000 0.0010 0.0000     NA
## attr(,"valroot")
##           [,1]
## [1,] 3.772e-13
## [2,] 1.000e+00
## attr(,"indroot")
## [1] 1
## attr(,"troot")
## [1] 1.571
## attr(,"nroot")
## [1] 1
## attr(,"lengthvar")
## [1] 2
## attr(,"type")
## [1] "radau5"
## attr(,"tf")
## 1
## -------
## 1 + x^2
## attr(,"roots")
## attr(,"roots")$dy.1
## attr(,"roots")$dy.1$exclude
## function (t, y, ...)
## t < tMin | (y - y.inf)/(y.inf - y.init) < 0.1
## <environment: 0x3474470>
##
## attr(,"roots")$dy.1$final
## [1] 0
##
##
## attr(,"roots")$y
```

```
## attr(,"roots")$y$exclude
## function (t, y, ...)
## t < tMin
## <environment: 0x4b4f0f8>
##
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")$y
## attr(,"roots")$y$exclude
## function (t, ...)
## t < tMin
## <environment: 0x4b4f0f8>
##
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")$y
## attr(,"roots")$y$exclude
## function (t, y, ...)
## t < tMin
## <environment: 0x4b50350>
##
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")$y
## attr(,"roots")$y$exclude
## function (t, ...)
## t < tMin
## <environment: 0x4b50350>
##
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")$y
## attr(,"roots")$y$exclude
## function (t, y, ...)
## t < tMin
## <environment: 0x4b4fe80>
##
```

```
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")$y
## attr(,"roots")$y$exclude
## function (t, ...)
## t < tMin
## <environment: 0x4b4fe80>
##
## attr(,"roots")$y$final
## [1] 0
##
##
## attr(,"roots")attr(,"settledRtol")
## [1] 0.01 0.02 0.05
## attr(,"roots")attr(,"tMin")
## [1] 0.1
```

The result comes from `deSolve::radau`. Derivatives of the output y that were needed to calculate the response are included with names `dy.1`, `dy.2` for the first and second derivatives. Additionally zeroes of the first derivative are included in the attributes `valroot` and `troot`.

```
plot(sint)
```



2.2 A more complicated transfer function

For the next example, the closed-loop step response for the system given by equation (1) is found.

First define the open-loop transfer function

```r
library(controltheory)
(gOL <- rational( p(5,1)*p(1,1),
                  p(0, 1) * p(3, 1) * p(2, 2, 1)))

## 5 + 6*x + x^2
## -------------------------
## 6*x + 8*x^2 + 5*x^3 + x^4

gCL <- function(k) k*gOL / (1 + k*gOL)
gCL(1)

## 30*x + 76*x^2 + 79*x^3 + 43*x^4 + 11*x^5 + x^6
## -----------------------------------------------------------------------
## 30*x + 112*x^2 + 175*x^3 + 167*x^4 + 103*x^5 + 42*x^6 + 10*x^7 + x^8
```

The above transfer function could be simplified by dividing through by x. The other common factors could be found by manually finding the the poles and zeroes:

```r
polyroot(gCL(1)@den)

## [1]  0.0000+0.000i -0.6157+0.000i -1.0000+1.000i -1.0000-1.000i
## [5] -0.5874-1.478i -0.5874+1.478i -3.0000-0.000i -3.2096+0.000i

polyroot(gCL(1)@num)

## [1]  0+0i -1-0i -1+1i -1-1i -3+0i -5-0i
```

and then making a new transfer function without the factors 0 $-1+i$ and $-1-i$ from the numerator and denominator.

Alternatively, this process is automated with `rationalize`:

```r
rationalize(gCL(1))

## 5 + 6*x + x^2
## ---------------------------------------------
## 5 + 12*x + 9*x^2 + 4.99999999999999*x^3 + x^4

round( rationalize(gCL(1)), digits=10)

## 5 + 6*x + x^2
## ------------------------------
## 5 + 12*x + 9*x^2 + 5*x^3 + x^4
```

```r
library(doMC)
registerDoMC(2)
```

```r
mkStepResponses <- function(ks, ...) mlply( data.frame(k = ks),
  function(k)
      ilt(rationalize( gCL(k) / p() ),
          ...),
     .parallel=T)
```

```r
library(ggplot2)
plot( ggplot( ldply(mkStepResponses(c(1,5,10),
                                    times=seq(0, 20, length.out=400)),
                     identity),
            aes(time, y, col=factor(k)))
    + geom_path())
```

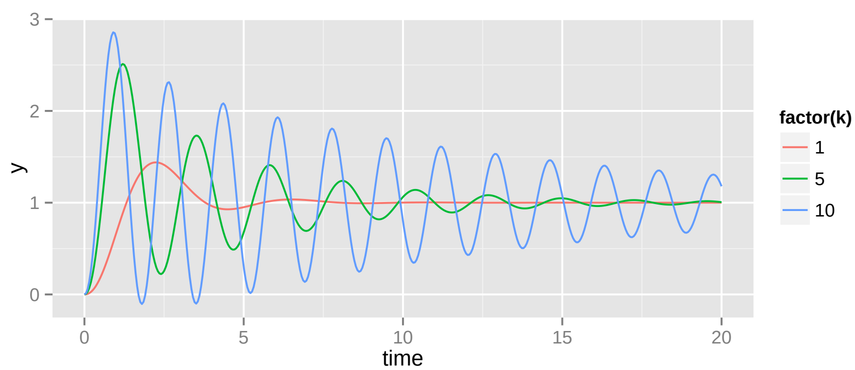


Figure 2: Step responses for the system in equation 1 with different proportional gains k

```r
responsesK <- mkStepResponses(seq(1, 8, length.out=20),
                     times = seq(0, 100, length.out=100))
```

```r
responsesK.stationary <- ldply(responsesK,
      function(x) {
        ix <- attr(x, 'indroot') == 1
        data.frame(t=attr(x, 'troot')[ix],
                   n=seq_along(attr(x, 'troot')[ix]),
                   y=attr(x, 'valroot')[ ncol(x)-1,ix])
      })
responsesK.settling <- within(ldply(responsesK, settlingTimes), {
                              yU <- round(yU, digits=3)
                              yL <- round(yL, digits=3) })
```

```
library(directlabels)
plot( ggplot(subset(responsesK.stationary, n<13), aes(k, y))
     + geom_line(aes(col=factor(n)))
     + geom_dl(aes(col=factor(n),
                   label=n),
               method='last.bumpup')
     + scale_color_discrete(guide='none'))
```

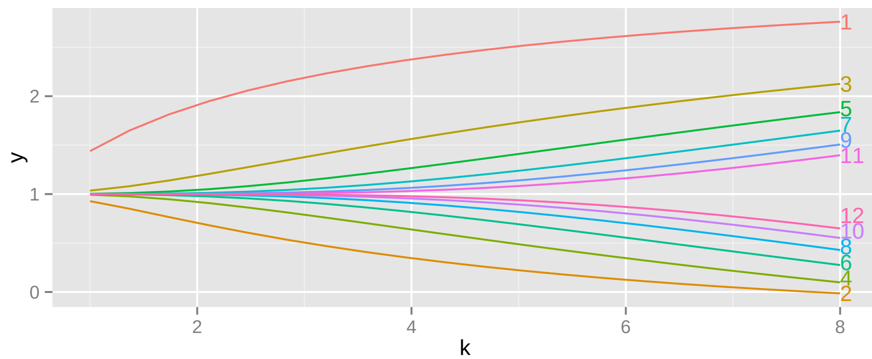


Figure 3: The peaks in figure 2 increase in amplitude as k increases.

```
plot( ggplot(responsesK.settling,
             aes(k , t, col=factor(yU)))
     + geom_line()
     + ylab('settling time'))
```

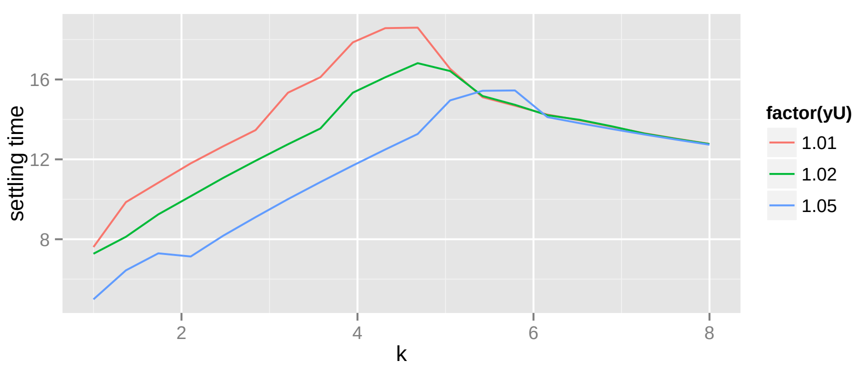


Figure 4: Settling times