# Root Locus for Systems with Delays

## ECE 461/661 Controls Systems

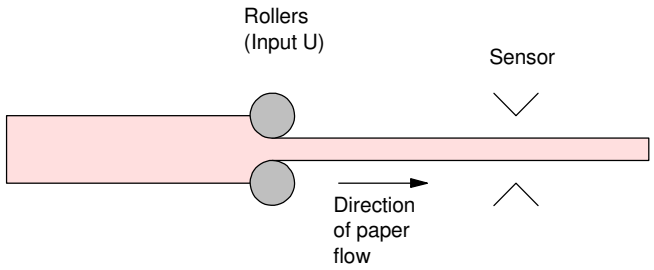
## Jake Glower - Lecture #27

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

# Source of Delays

Delays can result from many situations:

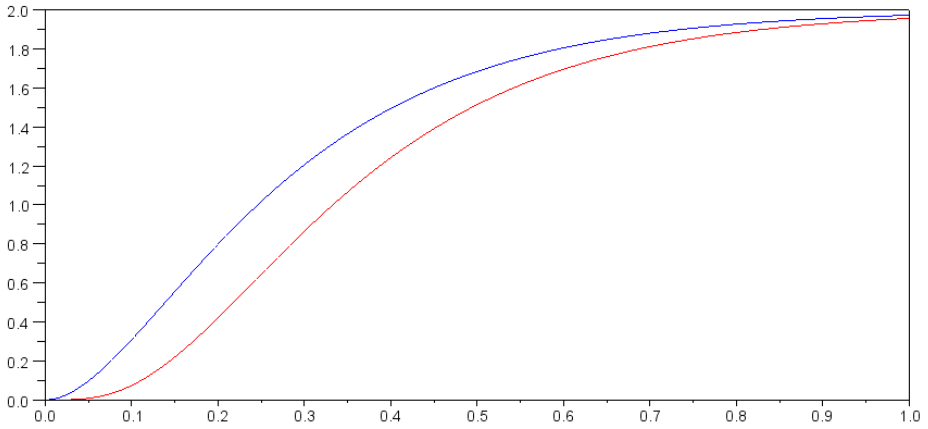
- Placing a sensor far from the input
  *Example: paper mill*
- Ignored dynamics
  *fast poles do have some affect on a system*
  *A delay is a way to model the poles you're ignoring*




Example

$$G(s) = \left( \frac{60{,}000}{(s+5)(s+10)(s+20)(s+30)} \right)$$

$$G(s) \approx \left( \frac{100}{(s+5)(s+10)} \right)$$

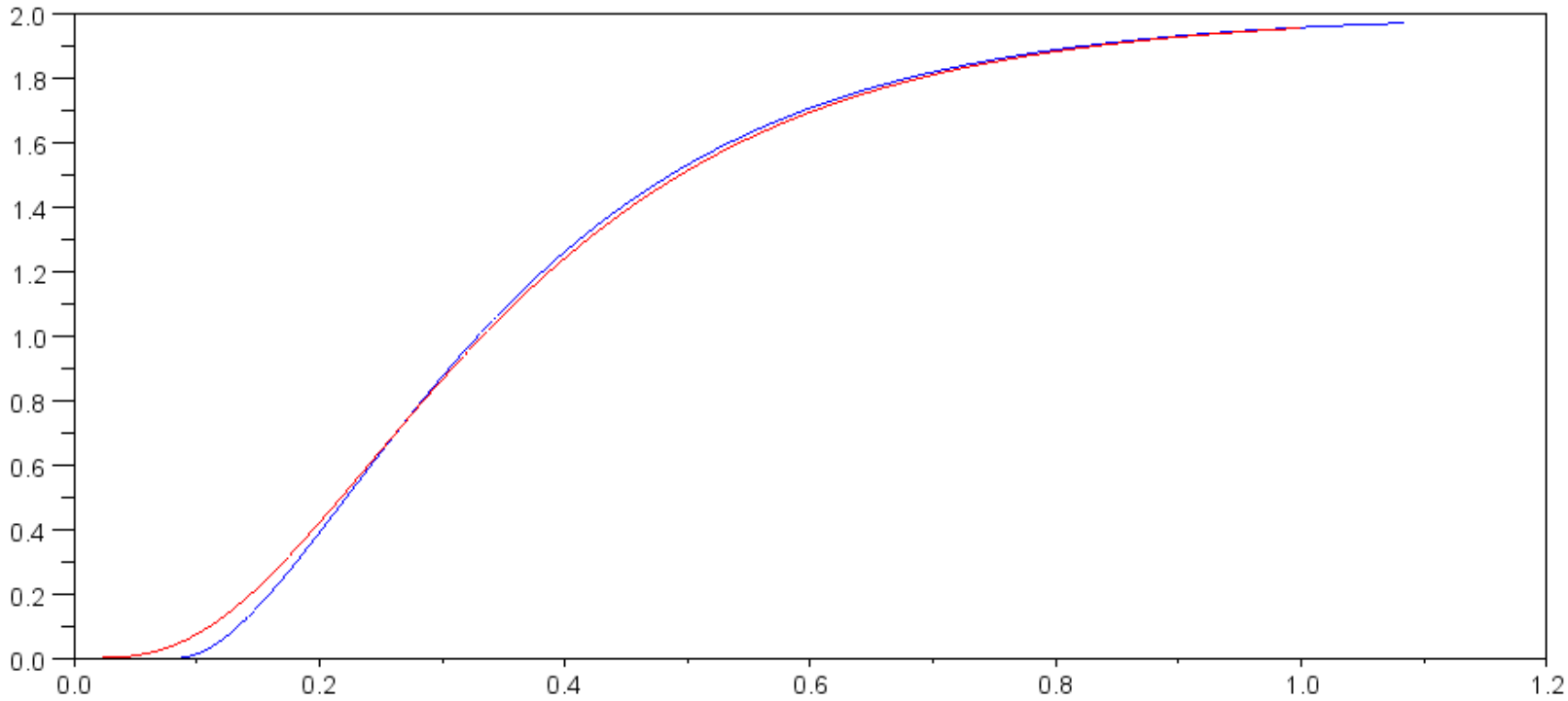


## Approximating the delay

$$G(s) = \left(\frac{60{,}000}{(s+5)(s+10)(s+20)(s+30)}\right) \approx \left(\frac{100}{(s+5)(s+10)}\right) \cdot e^{-sT}$$

## Find T to match the phase shift at s = j1

$$(e^{-sT})_{s=j1} = 1\angle -4.7716^0 \quad \Rightarrow \quad T = 0.0833 \text{ seconds}$$



## Pade Approximation:

To use root locus techniques, we need the poles and zeros:

$$Delay(T) = e^{-sT} \approx k\frac{z(s)}{p(s)}$$

Rewrite as

$$e^{-sT} = \left(\frac{e^{-\frac{sT}{2}}}{e^{\frac{sT}{2}}}\right)$$

Expand as a Taylor's series

$$e^{-sT} = \left(\frac{1-\left(\frac{T}{2}\right)s+\left(\frac{T^2}{8}\right)s^2-\left(\frac{T^3}{48}\right)s^3+\left(\frac{T^4}{384}\right)s^4+\ldots}{1+\left(\frac{T}{2}\right)s+\left(\frac{T^2}{8}\right)s^2+\left(\frac{T^3}{48}\right)s^3+\left(\frac{T^4}{384}\right)s^4+\ldots}\right)$$

The more terms you add, the better the approximation.

Example: Find 'k' for 20% overshoot

$$Y = \left( \left( \frac{100}{s(s+5)(s+10)} \right) \cdot e^{-0.5s} \right) U$$

Solution #1: Use a Pade approximation with 2 terms: (Matlab function)

```
[num, den] = pade(0.5, 2);
Delay = tf(num, den);
```

$$e^{-0.5s} \approx \left( \frac{s^2 - 12s + 48}{s^2 + 12s + 48} \right) = \left( \frac{(s-6+j3.464)(s-6-j3.464)}{(s+6+j3.464)(s+6-j3.464)} \right)$$

In Matlab:

```
G = zpk([],[0,-5,-10],100);
[num,den] = pade(0.5, 2);
Delay = tf(num,den);
k = logspace(-2,2,1000)';
R = rlocus(G*Delay,k);
```

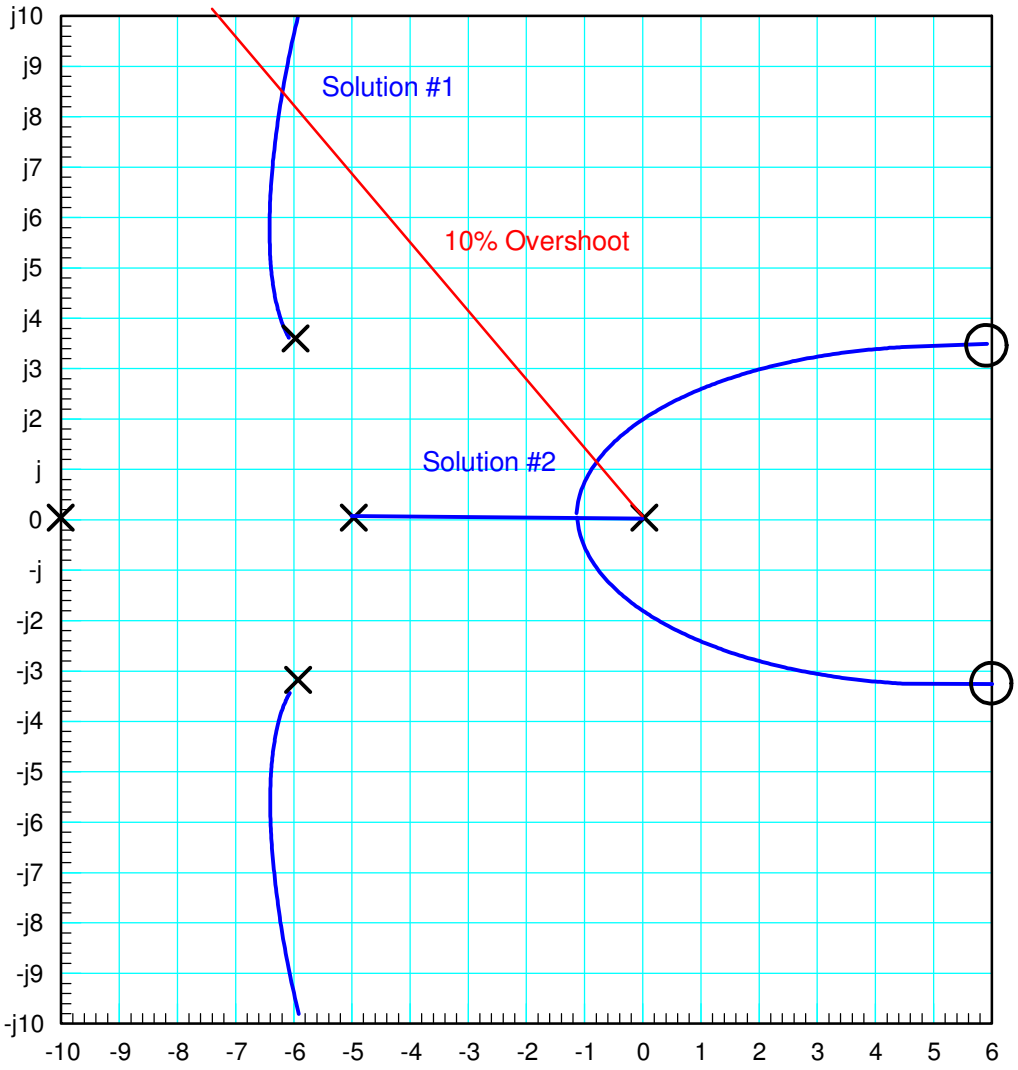
Two solutions:

s = -0.6646 + j1.3293

- k = 0.4484
- dominant pole

s = -4.5751 + j9.1502

- k = 4.3299
- stray solution



# Checking in Matlab

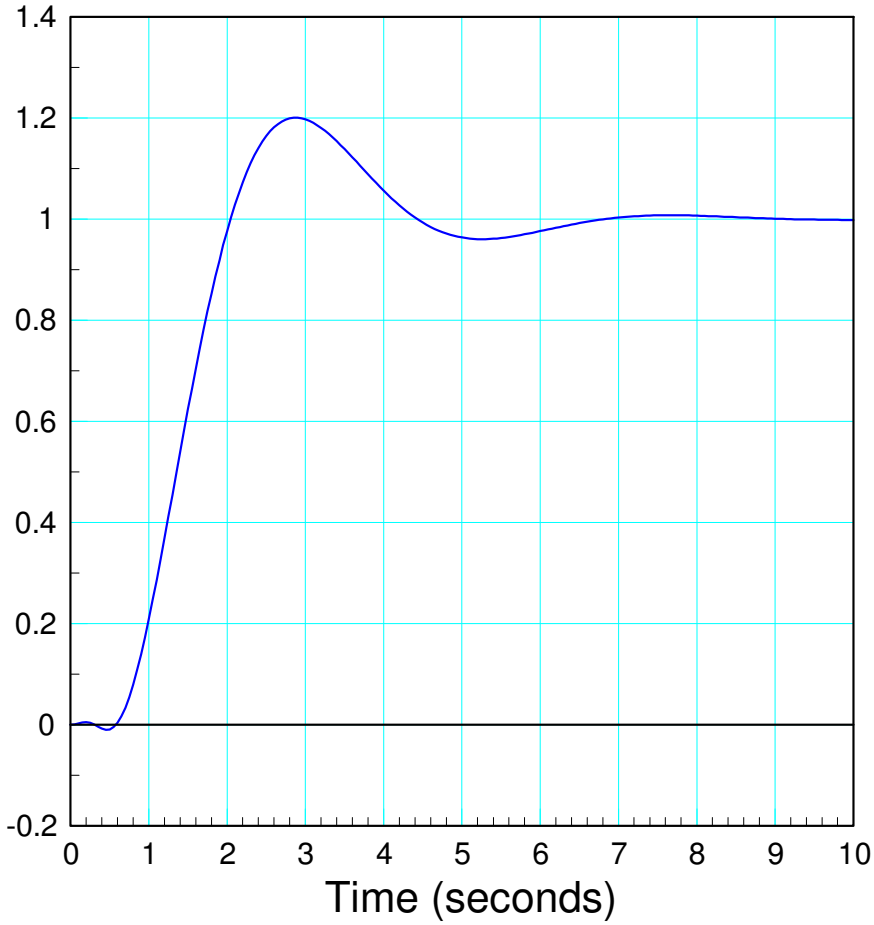
- Wobble for 0 < t < 1/2 second due to Pade approximation:

```
G = zpk([],[0,-5,-10],100);
[num,den] = pade(0.5, 2);
Delay = tf(num,den)

      s^2 - 12 s + 48
      ---------------
      s^2 + 12 s + 48

k = 0.4484;

Gcl = G*Delay*k / (1 + G*Delay*k);
Gcl = minreal(Gcl);
t = [0:0.01:10]';
y = step(Gcl,t);
plot(t,y);
```

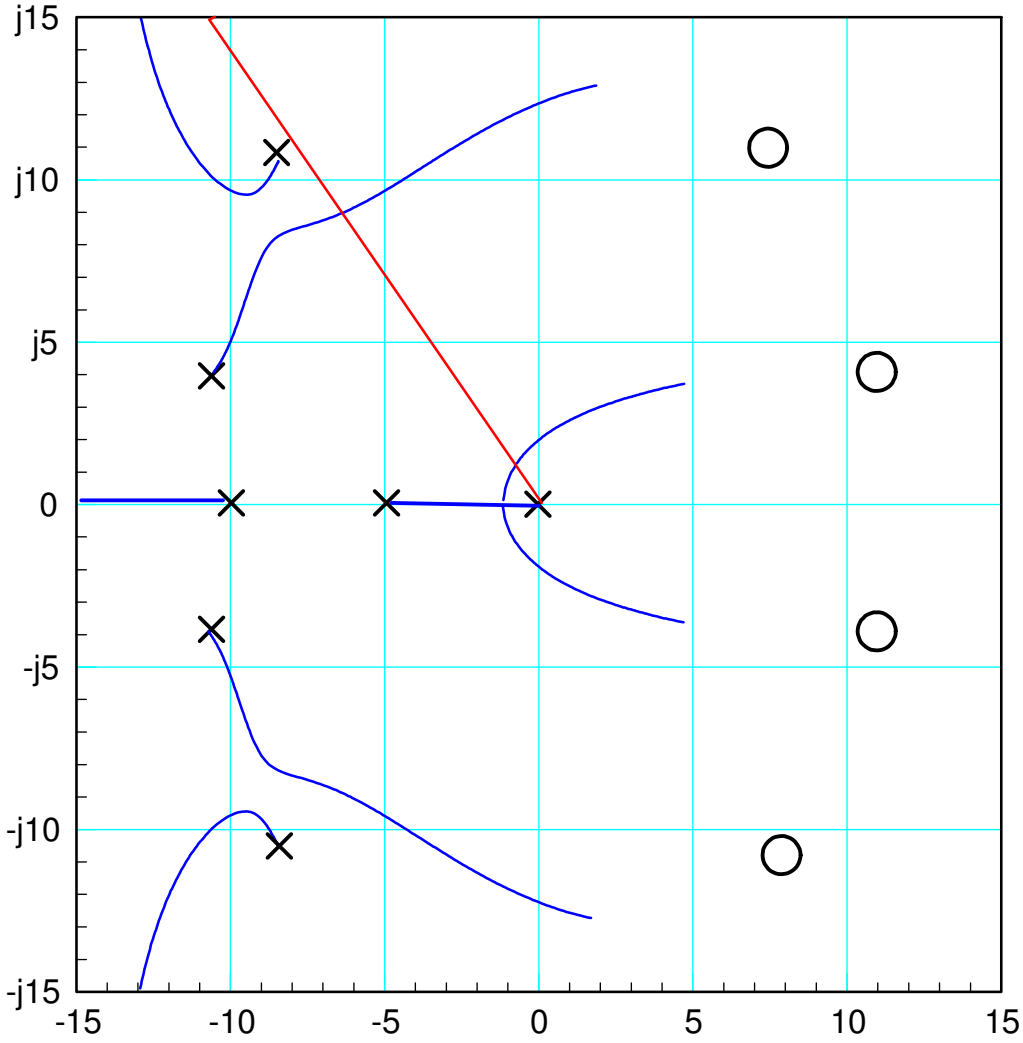


# Problem: 4th-order Pade Approximation

$$e^{-0.5s} \approx \left( \frac{(s-8.42\pm j10.63)(s-11.58\pm j3.47)}{(s+8.42\pm j10.63)(s+11.58\pm j3.47)} \right)$$

## Root Locus:

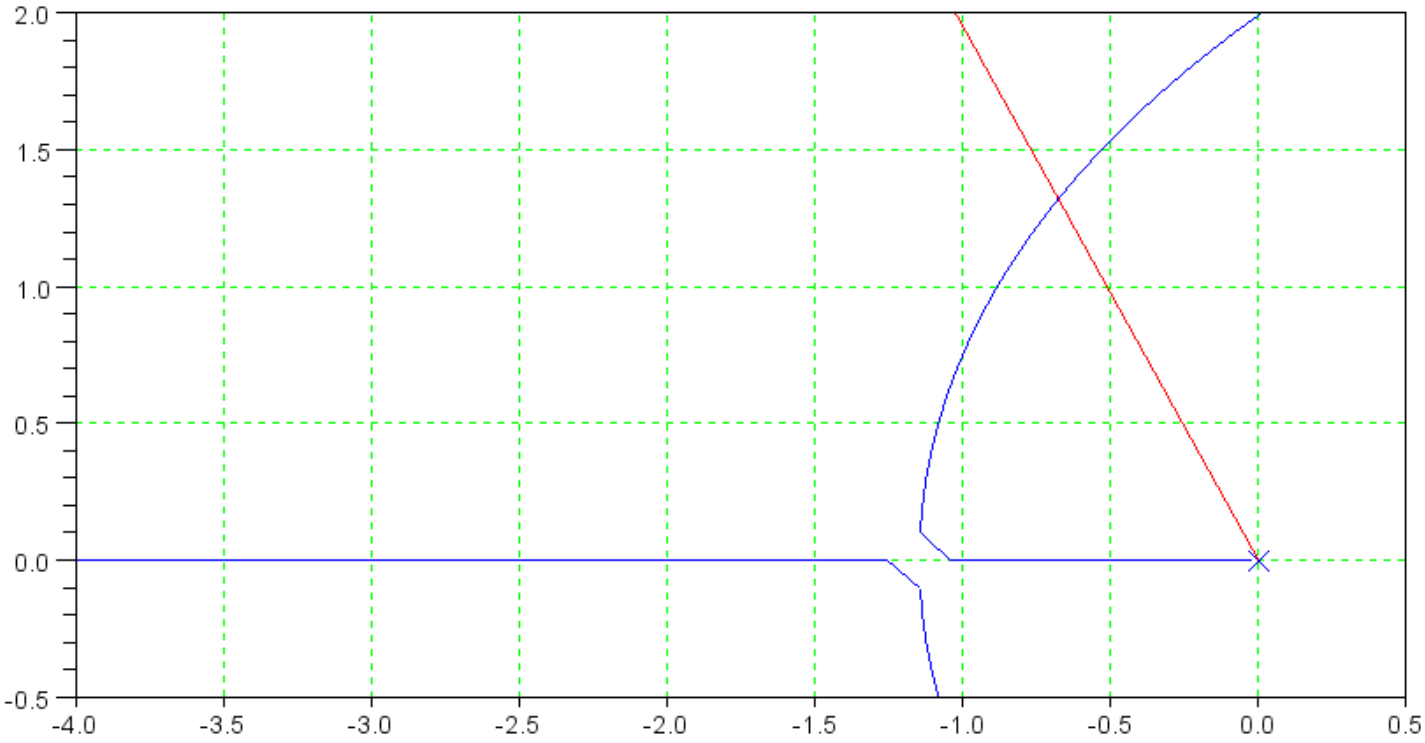
```
[num,den] = pade(0.5, 4);
Delay = tf(num,den)
k = logspace(-2,2,1000)';
G = zpk([],[0,-5,-10],100)
rlocus(G*Delay,k);
```



Zooming in on the dominant pole:

s = -0.6666 + j1.3333

k = 0.4566



# Option #2: Numerical Solution

Any point on the root locus satisfies

$$\angle(GK) = 180^0$$

Search along the damping line until the angles add up 180 degrees

For 20% overshoot

$$angle\left(\frac{100\cdot e^{-0.5s}}{s(s+5)(s+10)}\right)_{s=\alpha(-1+j2)} = 180^0$$

```
s = -1 + j*2;
G = zpk([],[0,-5,-10],100);
evalfr(G,s) * exp(-0.5*s)

ans =  -1.5006 + 0.9728i

s = s * 0.9;
evalfr(G,s) * exp(-0.5*s)

ans =  -1.7267 + 0.7338i
```

This method doesn't have a name and might not be taught anywhere else other than NDSU.

- It isn't really root locus design, since you're not drawing the root locus
- It sort of is root locus design, since you're finding the point on the root locus which intersects the desired damping line. That's the only point you care about anyway, so you don't need (and won't use) the rest of the root loucs.
- It's a lot easier and more accurate since you using $e^{-sT}$ to model a delay, not an approximation. (Typing in four poles and four zeros for the 4th-order model was a bit of a pain. Typing in $e^{-sT}$ was easy.)

In Matlab, iterate until the angle of G * delay is zero. Taking your initial guess as

$s = -0.5 + j$

then iterating by scaling s each step results in

```
s = 0.5 * (-1 + j*2);
evalfr(G,s) * exp(-s*T)

  - 2.5039068 - 0.7297865i

s = s * 1.1;                10% larger
evalfr(G,s) * exp(-s*T)

  - 2.4062416 - 0.4851018i   good: complex portion is getting smaller.

s = s * 1.1;              add another 10%
evalfr(G,s) * exp(-s*T)

  - 2.303687 - 0.2437231i        better:  complex portion is getting smaller

s = s * 1.1;              add another 10%
evalfr(G,s) * exp(-s*T)

 - 2.1923504 - 0.0045661i         close:  complex portion is almost zero
```

( time passes )

```
s = 0.666717 * (-1 +j*2)

  - 0.666717 + 1.333434i

evalfr(G,s) * exp(-s*T)

  - 2.1901017 + 0.0000002i            close enough
```

If you use this method, the result is the exact solution

| Method | s | k |
|---|---|---|
| 2nd-Order Pade | -0.6646 + j1.3293 | 0.4484 |
| 4th-Order Pade | -0.6666 + j1.3333 | 0.4566 |
| exp(-sT) (exact) | -0.6667 + j1.3334 | 0.4566 |

## Positives

- Works
- Doesn't need Pade approximations
- Exact solution

## Negatives

- Not a built in function in Matlab