TUTORIAL: Using MATLAB for Feedback Analysis

by Rodrigo Carrasco Sch.

One important aspect in the design of amplifiers and PLL is to analyze the effects of the feedback loop in the overall system. A well designed feedback loop can help to exchange bandwidth for gain, obtain useful gain from amplifiers or make the PLL work, while a bad can give you an oscillator instead of an amplifier (or maybe that was what you wanted).

There are several tools that can be used for analyzing and designing a system, like Bode plots or Root Locus diagrams. All these methods are available on MATLAB in the Control Toolbox, and can be adapted for using them in the design of feedbacks for amplifiers, oscillators and PLLs.

The following tutorial gives a look into some of the tools that can be used for design and simulation of different feedback systems, assuming that single input – single output (SISO) systems are used (which is always the case for the different applications that are discussed in class).

Creation of Transfer Functions

There are several equivalent ways to create a transfer function in MATLAB, which must be stored in SYS type variables for later use. The following two methods are the more useful ones:

Transfer Function Model

This model is used when the transfer function is in the following format:

$$T(s)=\frac{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0}{b_m s^m + b_{m-1}s^{m-1} + \ldots + b_1 s + b_0}$$

For creating this model the `tf()` function is used, having as input the coefficients of the numerator and denominator polynomials.
Ex. (this will store your model on T): `T=tf([a_n a_n-1 ... a_1 a_0], [b_m b_m-1 ... b_1 b_0])`

Zero – Pole – Gain Model

This model stores the transfer function in the following format:

$$T(s)=\frac{K(s+z_1)(s+z_2)\ldots(s+z_n)}{(s+p_1)(s+p_2)\ldots(s+p_m)}$$

This will create a model with zeros on z_1, z_2, …, z_n and poles on p_1, p_2, …, p_m. The function used for this is `zpk()` that has as input two vectors, one containing the zeros and another the poles, followed by the gain K.
Ex.: `T=zpk([z_1 z_2 ... z_n], [p_1 p_2 ... p_m], K)`

You can transform one model to the other by using the desired function with the model as an input.
Ex. Suppose that T stores a TF model, then `H=zpk(T)` will store on H the ZPK transformation of the TF model stored on T.

For the rest of this tutorial we will define $T = 1x10^4 \frac{\left(s/1x10^4 + 1\right)}{s\left(s/2x10^3 + 1\right)}$.

Obtaining Poles and Zeros

Poles and zeros of an LTI model can be obtained using functions `pole(T)` and `zero(T)`. These functions give a vector containing the poles or zeros of the LTI model, as output.
Ex.:

```
>>zero(T)
ans =
      -10000
>>pole(T)
ans =
           0
       -2000
```

Furthermore, these poles and zeros can be plotted using the function `pzmap(T)`. As figure 1 shows, by clicking over a pole or a zero the value and some extra data can be obtained.

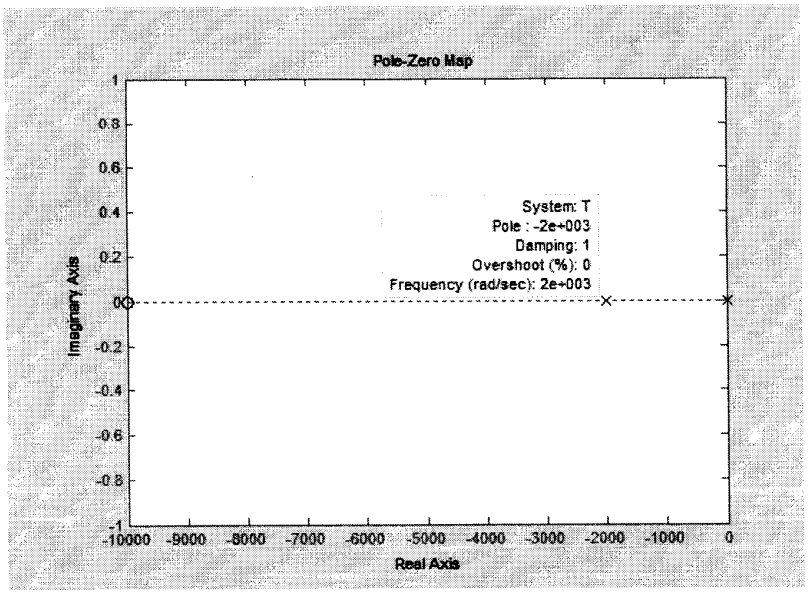

Figure 1: Poles and Zeros of T

This function can also be used for obtaining the values of the poles and zeros by storing them on variables.

```
>>[Poles, Zeros]=pzmap(T)
Poles =
           0
      -2000
Zeros =
      -10000
```

Bode, Nyquist and Nichols Plots

Once you have your transfer function and you know where its poles and zeros lie, most probably you will be interested in the frequency response and other characteristics of the model.

Bode plots are the most used way to show the frequency response of a certain system. This can be obtained using the function `bode(T)`.

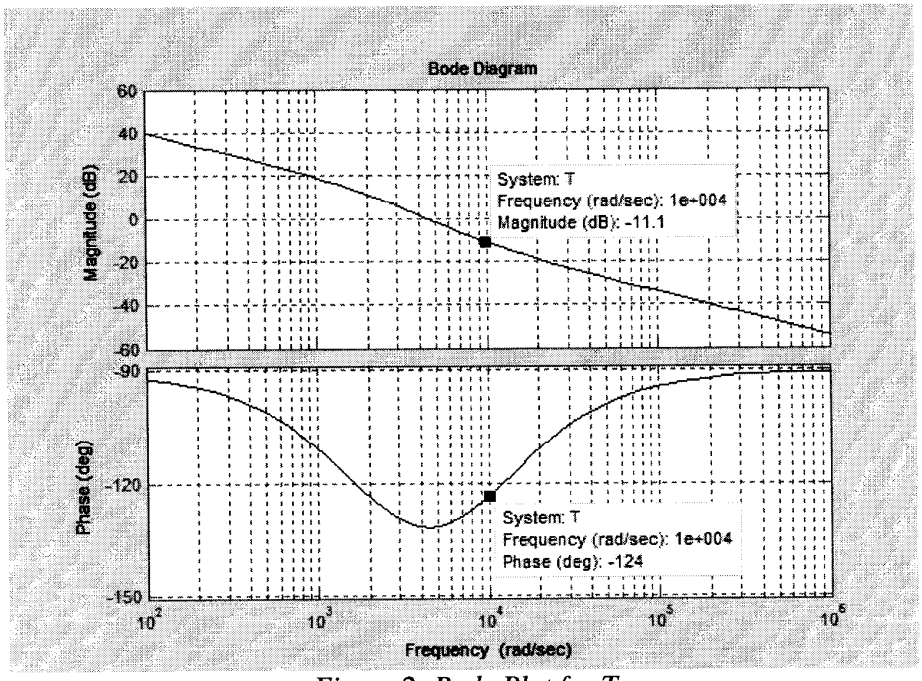

Figure 2: Bode Plot for T

The same with most of the plots you can obtain in the control toolbox, by clicking over the curve you can obtain data from that point. The limits of the plot can be arranged by adding the limits when calling the bode function: `bode(T, {wmin wmax})`. This function always plots the amplitude in dB, the phase in degrees and the frequency in radians per second.

You can store this data into variables without plotting by adding the desired variables: `[Mag, Phase, w]=bode(T)`. The `w` element at the output can be omitted if you

don't want to store that value. It is important to notice that if the function is used in this way then the magnitude will not be given in dB.

For plotting just the magnitude of the frequency response the function `bodemag(T)` can be used. The same as with the `bode()` function; the output can be stored on variables.

This plot can be used for stability analysis if the phase and gain margins can be obtained. The phase margin is defined as the degrees you have of margin to get to 180° when the system gain is 0. In a similar way, the gain margin is defined as the margin in dB you have when the phase shift is 180°. An easy way of obtaining this information is by using the `margin(T)` function. This will generate a bode plot that shows the phase and gain margins.

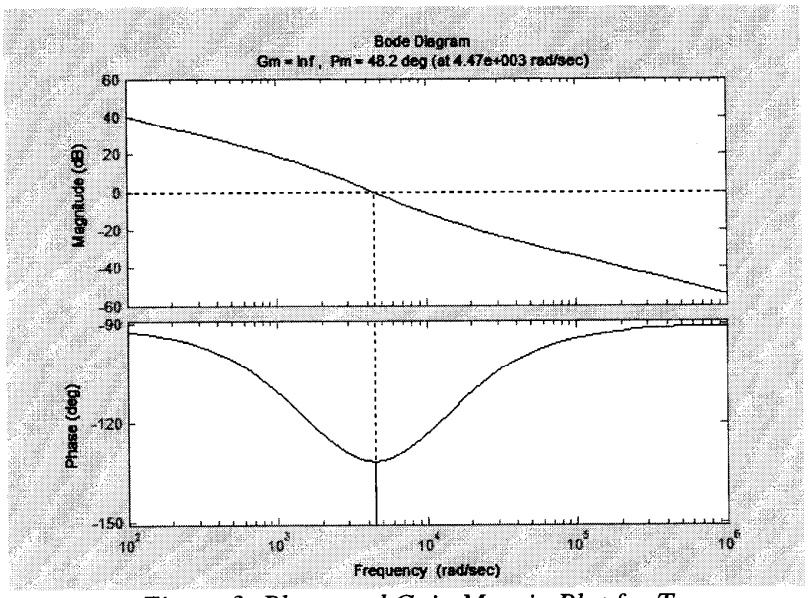

Figure 3: Phase and Gain Margin Plot for T

These values can be stored without generating a bode plot by adding the output variables: `[Gm, Pm, wgm, wpm]=margin(T)`. You have to be careful, because in this case the gain margin will not be given in dB.

Another way of obtaining the margins for a given LTI model is to use the `allmargin` function. In this case the margins can be stored on an output structure.

```
>>allmargin(T)
ans =
    GMFrequency: [1x0 double]
     GainMargin: [1x0 double]
    PMFrequency: 4.4721e+003
    PhaseMargin: 48.1897
    DMFrequency: 4.4721e+003
    DelayMargin: 1.8807e-004
         Stable: 1
```

Another important plot used for stability analysis is Nyquist plot. A system is stable if (and only if) the contour of the Nyquist diagram does not encircle (-1,0) when there are no poles on the right hand side of the s-plane, or when the number of counterclockwise encirclements of (-1,0) is equal to the number of poles that are on the right side of the s-plane.

This kind of plots can be created using the `nyquist(T)` function.

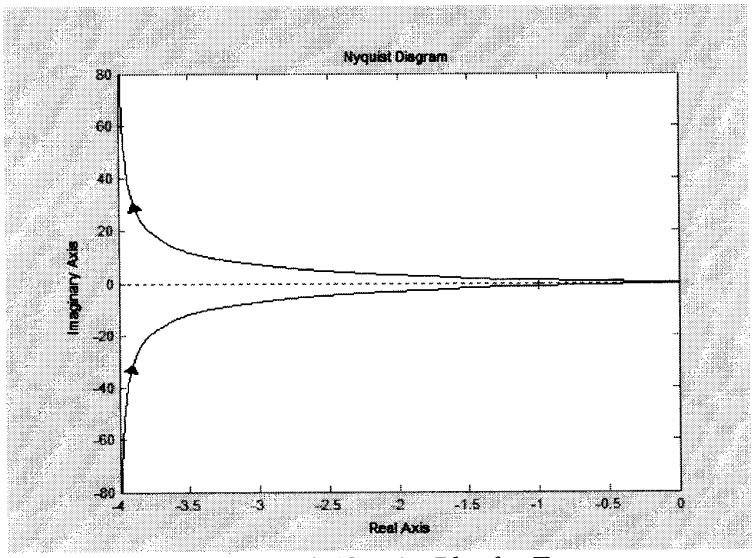

Figure 4: Nyquist Plot for T

Finally there is another way to obtain the phase and gain margins from a plot, and that is by using the Nichols diagrams. These graph the gain of the system versus the phase shift for every frequency.

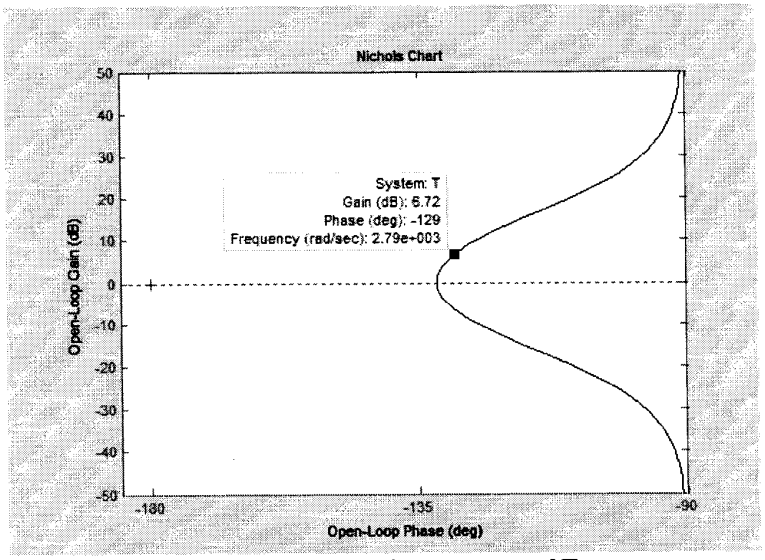

Figure 5: Nichols Diagram of T

Creating a Feedback

Now that the open loop has been analyzed the effect of the feedback loop must me determined to see if the complete system meet the desired specifications. The first thing for doing this is calculating the transfer function of the closed loop system.

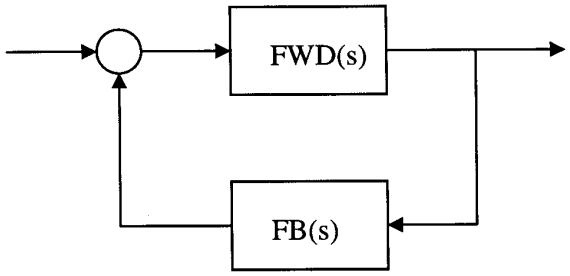

Figure 6: Closed Loop System

This simple SISO feedback system can be calculated using the feedback function, as `feedback(FWD, FB)`. The function assumes that a negative feedback is used. In case a positive feedback is wanted a third input parameter must be added when calling the function: `feedback(FWD, FB, +1)`.

For example, in a PLL the feedback loop is just 1 so the transfer function for the closed loop system can be obtained by:

```
>> H=feedback(T, 1)
Transfer function:
       s + 10000
------------------------
0.0005 s^2 + 2 s + 10000
```

All the tools showed before can be used with H for analyzing its behavior.

Determining the Gain, Damping Factor, and Natural Frequency

When a closed loop is going to be used, the poles of the system will move on the pole-zero diagrams depending on the gain of the loop. The location of these poles for different gains is called the root locus of the system, and it can let you see if for some gain the system becomes unstable or perhaps the oscillations start.

For obtaining the root locus of a open loop system the function `rlocus(T)` can be used. This function assumes a negative feedback with FB(s)=k the gain of the feedback loop.

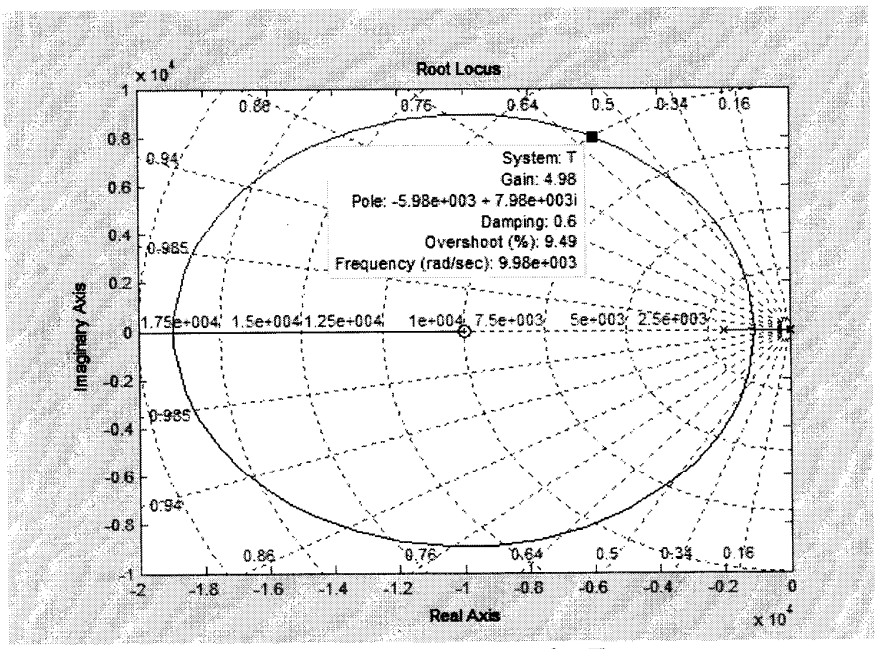

Figure 7: Root Locus for T

Figure 7 can show you all the possible poles locations for different values of K. This can be used for example in designing a PLL, in such a way that the closed loop has a damping factor of $\xi = 0.6$. As it is observed on figure 7, this happens when the gain is 4.98.

When clicking over the root locus you can obtain a lot of data from that point, such as the damping factor, the overshoot, the gain or the natural frequency the system will have. This data can aid in the design of the system.

Another way of determining the gain needed for having a certain damping factor and natural frequency is by using the function `sgrid(z, wn)` where z is the damping factor and wn the natural frequency. This function plots the position in which the poles must be for obtaining the desired characteristics, and is used after using `pzmap` or `rlocus`. Remember that each pair of complex poles will have its own damping factor and natural frequency, so if you want to set each to a certain value, the z and wn parameters of sgrid must be an array of values.

Ex: after using `rlocus(T)`, you can use `sgrid(0.6, 9.98e3)`

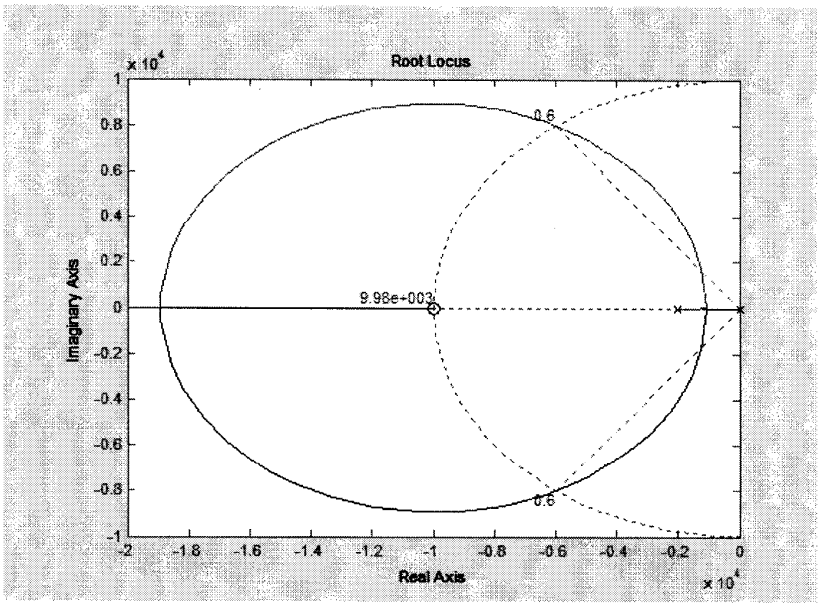

Figure 8: Using sgrid

If you already have a closed loop system and you are interested on obtaining the damping factor and natural frequency you can use the function `damp`.

```
>>damp(H)
        Eigenvalue                Damping      Freq. (rad/s)

 -2.00e+003 + 4.00e+003i      4.47e-001       4.47e+003
 -2.00e+003 - 4.00e+003i      4.47e-001       4.47e+003
```

This shows that for a gain of 1 (which was used to calculate H) the damping factor will be 0.447 and the natural frequency will be 4.47×10^3 [rad/sec].

Step Response

Finally, to see what will happen on your system when a step change will be applied to the reference the step response is used. By knowing the damping factor and natural frequency you can have a certain idea of what will that response be, but for determining the real thing the function `step(H)` is used.

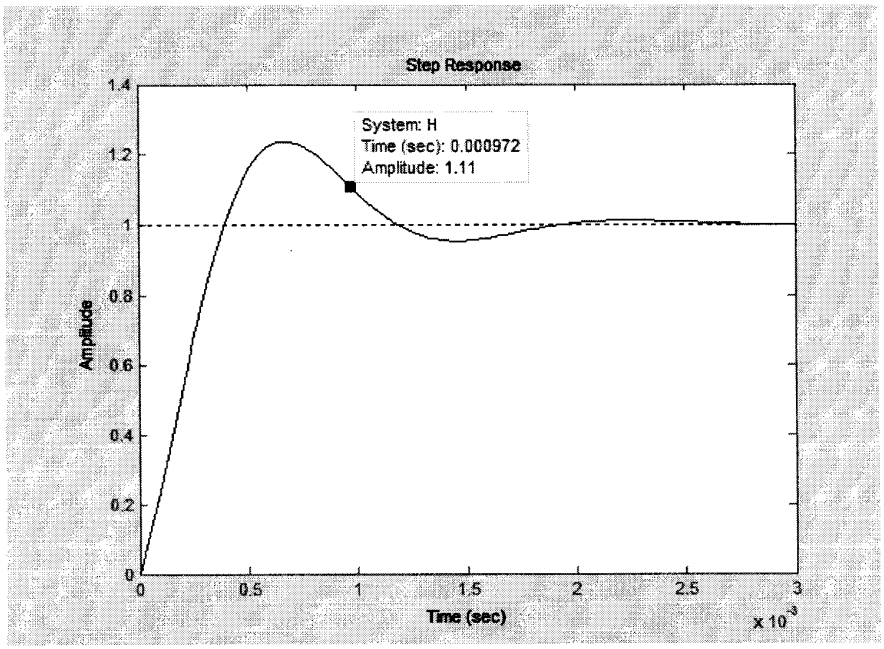

Figure 9: Step Response for H

The final time of the simulation can be changed by adding a second input parameter when calling the function: `step(H, tend)`. As in most of the diagrams that can be created, the output values can also be stores by adding the needed output variables `[Amp, t]=step(H)`.

In the case of PLL design for synthesizers more important than the step response of the system is the response to a ramp. This is because the reference in this case is a frequency and not a phase so your input is $\frac{\Delta\omega}{s^2}$ and not $\frac{\Delta\varphi}{s}$. An easy way to adapt the existing functions is by adding the needed 1/s term to the close loop transfer function. This is done by multiplying the original function by $\frac{1}{s}$ and then using the step response command over the new transfer function.

Ex:

```
>> st=tf([1], [1 0])
Transfer function:
1
-
s
>> Hp=H*st
Transfer function:
          s + 10000
----------------------------
0.0005 s^3 + 2 s^2 + 10000 s
>> step(Hp)
```

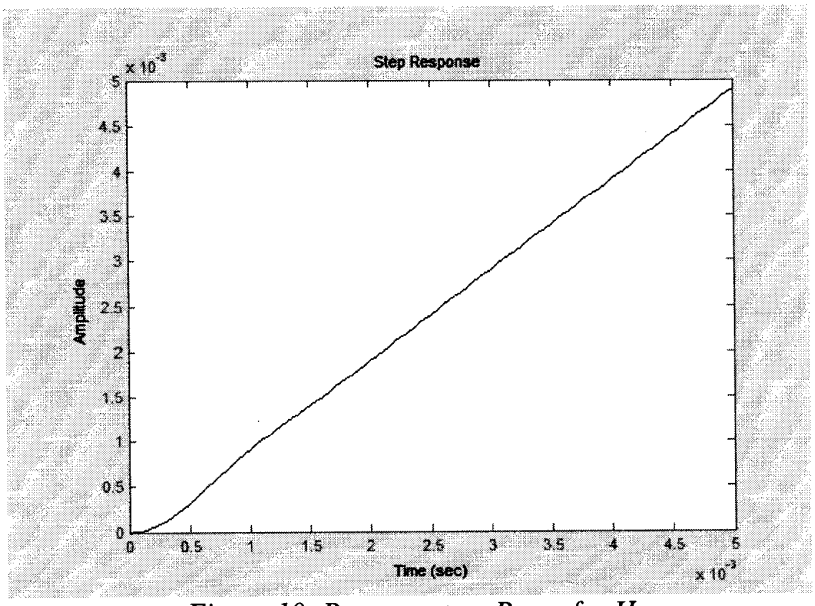

Figure 10: Response to a Ramp for H

More interesting is to see how the error is affected by a change in the input. Thought there are no MATLAB functions that can calculate this, you can do it by writing the transfer function for the error given a certain input, which is: $E(s)=\frac{e(s)}{R(s)}=\frac{1}{1+T(s)}$. Then the step response (or ramp response) can be applied to E(s) to see what happens to the error after a change in the reference occurs.

```
>> E=1/(1+T)
Transfer function:
     0.0005 s^2 + s
------------------------
0.0005 s^2 + 2 s + 10000
>> Ep=E*st
Transfer function:
       0.0005 s^2 + s
----------------------------
0.0005 s^3 + 2 s^2 + 10000 s
>> step(Ep)
```

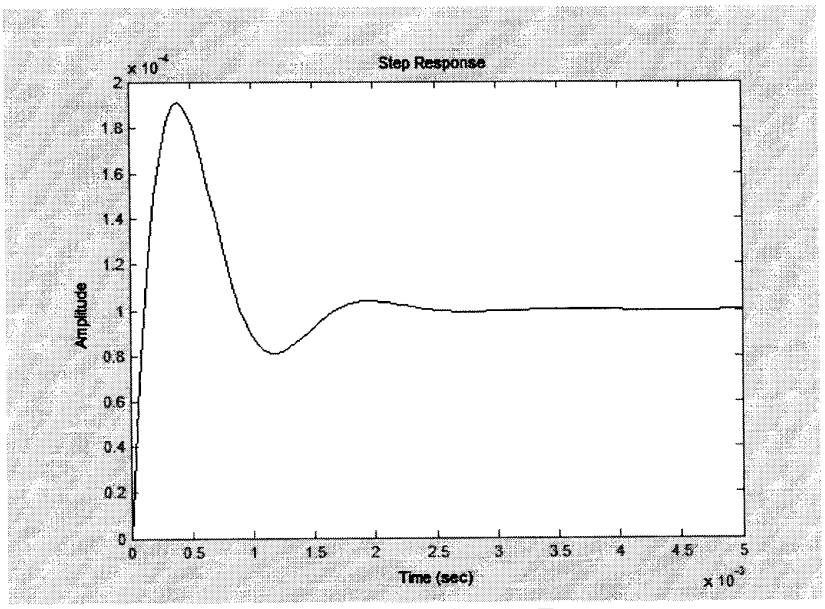

Figure 11: Error for a Step in Frequency

As it can be observed on figure 11, a transfer function such as T will not be good for a PLL synthesizer because it has a permanent error when a step in frequency is applied.

RLTOOL

The `rltool(T)` command will open a user interface where you can add poles and zeros for your T transfer function, showing in real time the effects this has over the bode plot, step response and other important characteristics. Very useful for design and study of the effects of certain changes on your system.