Assignment 9.1

Let T_p (peak time) be the time required by the output signal to reach its maximum value. For a second order system:

$$T_p = \frac{\pi}{\omega_n \sqrt{1-\zeta^2}}$$

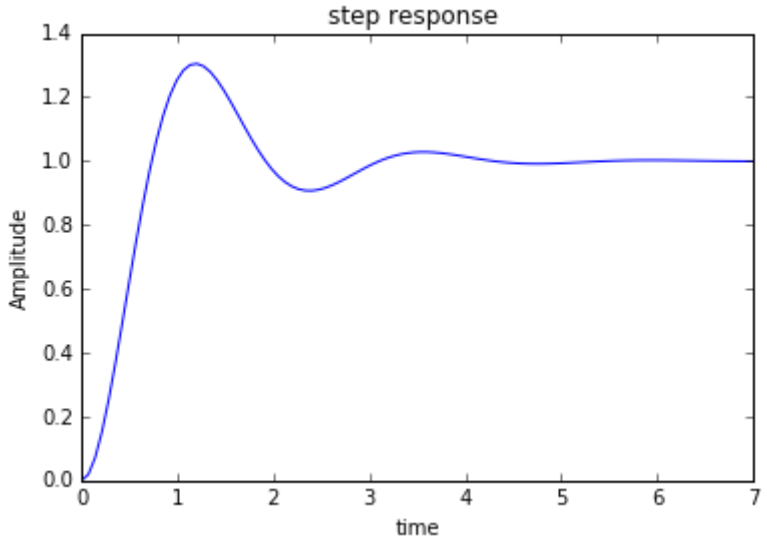
Let $\%OS$ (percentage overshoot) be given by the equation below, where smax is the maximum value of the output signal 's' and sfinal is the final (steady state) value of 's'.

$$\%OS = \frac{s_{max} - s_{final}}{s_{final}} \times 100$$

The damping ratio is related to $\%OS$ as following:

$$\zeta = \frac{-\ln \frac{\%OS}{100}}{\sqrt{\pi^2 + \ln^2(\frac{\%OS}{100})}}$$

1. Consider the step response of a second order system below.Use this response and the equations above to find the (approximate) transfer function expression for the second order system.



2. Simulate your transfer function using python control and verify your result.

From the figure, we can assume that,

$$s_{max} = 1.3$$

$$s_{final} = 1$$

$$T_p = 1.3$$

Overshoots can be computed from :

$$\%OS = \frac{s_{max} - s_{final}}{s_{final}} \times 100$$

Substituting values of s_{max} and s_{final} we get

$$\%OS = 0.3 * 100 = 30\%$$

Damping ratio can be computed from :

$$\zeta = \frac{-\ln \frac{\%OS}{100}}{\sqrt{\pi^2 + \ln^2(\frac{\%OS}{100})}}$$

Substituting value of $\%OS$ in above formula,

$$\zeta = 0.358$$

Similarly ω_n can be computed by using this formula :

$$T_p = \frac{\pi}{\omega_n \sqrt{1-\zeta^2}}$$

substituting values of T_p , ω_n can be computed as :

$$\omega_n = 2.586$$

Therefore, Transfer function of second order system is :

$$T(s) = \frac{6.687}{s^2 + 1.852s + 6.687}$$

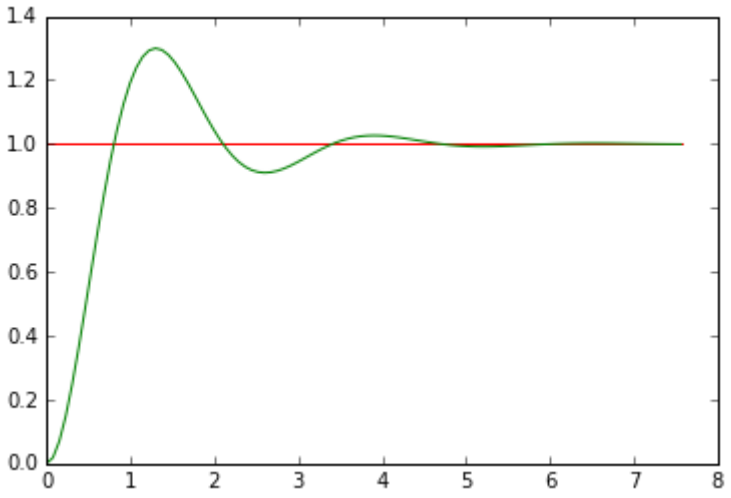
In [1]:

```
# Solution Here
import control
import numpy
import sympy
import matplotlib.pyplot as plt
%matplotlib inline
num = [6.687]
den = [1,1.852,6.687]
transferfunction = control.TransferFunction(num,den)
stepresponse, time = control.matlab.step(transferfunction)
desiredoutput = numpy.ones(len(time))
plt.plot(time,desiredoutput,color='r')
plt.plot(time,stepresponse,color = 'g')
```

```
/home/ramesh/anaconda2/lib/python2.7/site-packages/matplotlib/font_man
ager.py:273: UserWarning: Matplotlib is building the font cache using
 fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list.
 This may take a moment.')
```

Out[1]:

```
[<matplotlib.lines.Line2D at 0x7fb2ea308d50>]
```



Assignment 9.2

A transfer function of a second order system is given in the equation below:

$$T(s) = \frac{24.542}{s^2 + 4s + 24.542}$$

Simulate the step response of this simple system in python. Also simulate the step responses of the following third order systems:

$$T_1(s) = \frac{73.626}{(s+3)(s^2 + 4s + 24.542)}$$

$$T_2(s) = \frac{245.42}{(s+10)(s^2 + 4s + 24.542)}$$

$$T_3(s) = \frac{490.84}{(s+20)(s^2 + 4s + 24.542)}$$

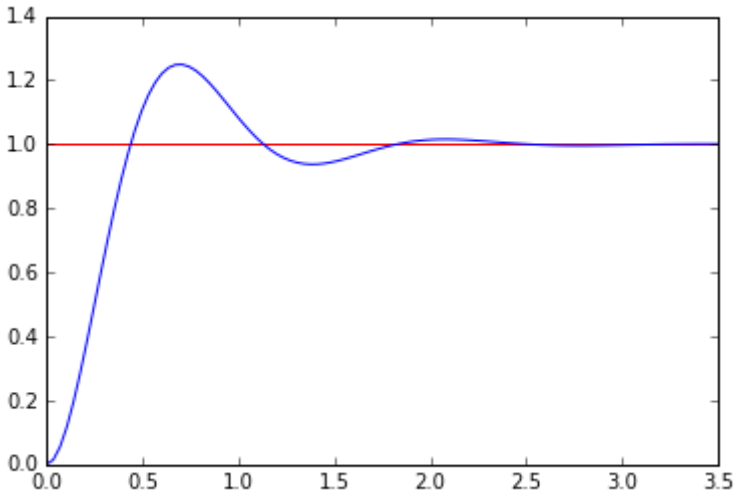
Among the above third order systems which system can be best approximated by T (s) and why?

In [2]:

```
# Solution here
import control
import numpy
import sympy
import matplotlib.pyplot as plt
%matplotlib inline
num = [24.542]
den = [1,4,24.542]
transferfunction = control.TransferFunction(num,den)
stepresponse_t, time = control.matlab.step(transferfunction)
desiredoutput = numpy.ones(len(time))
plt.plot(time,desiredoutput,color='r')
plt.plot(time,stepresponse_t,color = 'b')
```

Out[2]:

```
[<matplotlib.lines.Line2D at 0x7fb31942b9d0>]
```



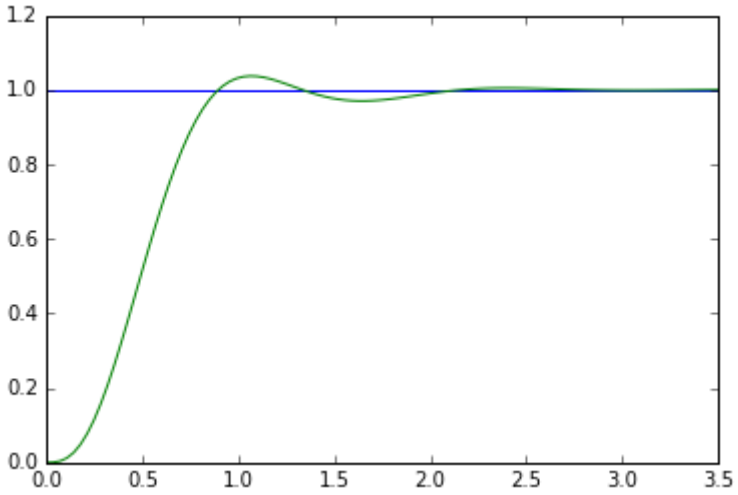
In [3]:

```
#T1(s)
import control
from control import *
import numpy
import sympy
import matplotlib.pyplot as plt
%matplotlib inline
num1 = [73.626]
den1 = [1,3]
num2 = [1]
den2 = [1,4,24.542]
transfer_function1 = control.TransferFunction(num1,den1)
transfer_function2 = control.TransferFunction(num2,den2)
final_transferfunction = control.series(transfer_function1,transfer_function2)
print final_transferfunction
stepresponse_t1,time = step(final_transferfunction)
desired_response = numpy.ones(len(time))
plt.plot(time,desired_response)
plt.plot(time,stepresponse_t1)
```

```
            73.63
------------------------------
s^3 + 7 s^2 + 36.54 s + 73.63
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x7fb31942b0d0>]
```



In [ ]:

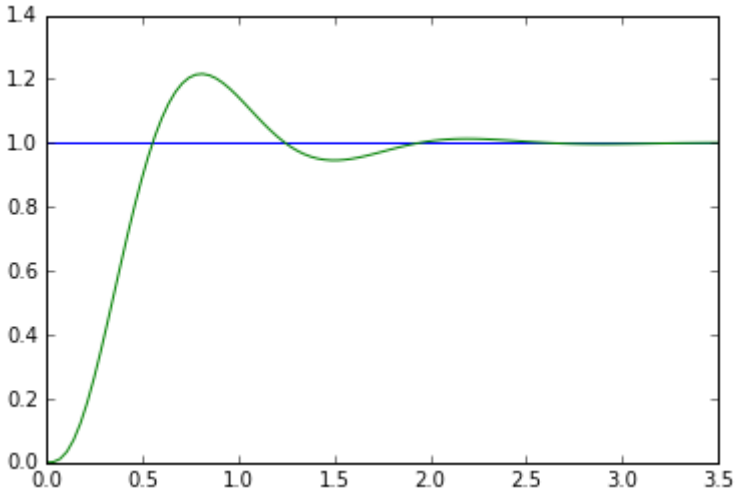
In [4]:

```
#3rd part
#T2(s)
import control
from control import *
import numpy
import sympy
import matplotlib.pyplot as plt
%matplotlib inline
num1 = [1]
den1 = [1,10]
num2 = [245.42]
den2 = [1,4,24.542]
transfer_function1 = control.TransferFunction(num1,den1)
transfer_function2 = control.TransferFunction(num2,den2)
final_transferfunction = control.series(transfer_function1,transfer_function2)
print final_transferfunction
stepresponse_t2,time = step(final_transferfunction)
desired_response = numpy.ones(len(time))
plt.plot(time,desired_response)
plt.plot(time,stepresponse_t2)
```

```
            245.4
------------------------------
s^3 + 14 s^2 + 64.54 s + 245.4
```

Out[4]:

```
[<matplotlib.lines.Line2D at 0x7fb2e76fde50>]
```



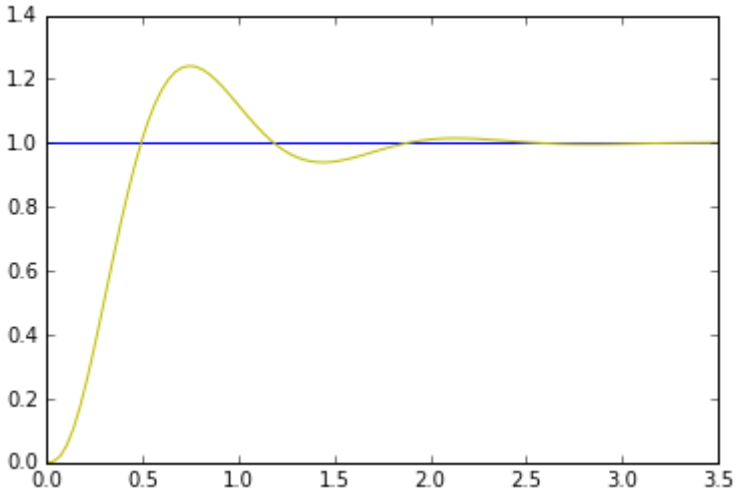
In [5]:

```
#T3
import control
from control import *
import numpy
import sympy
import matplotlib.pyplot as plt
%matplotlib inline
num1 = [1]
den1 = [1,20]
num2 = [490.84]
den2 = [1,4,24.542]
transfer_function1 = control.TransferFunction(num1,den1)
transfer_function2 = control.TransferFunction(num2,den2)
final_transferfunction = control.series(transfer_function1,transfer_function2)
print final_transferfunction
stepresponse_t3,time = step(final_transferfunction)
desired_response = numpy.ones(len(time))
plt.plot(time,desired_response)
plt.plot(time,stepresponse_t3,color = 'y')
```

```
             490.8
------------------------------
s^3 + 24 s^2 + 104.5 s + 490.8
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x7fb2e7851110>]
```



Comparing step response of all provided systems, response of T3(s) can be best approximated to T(s) as transient response of both the systems is almost similar, while difference between responses of T1(s) and T2(s) as compare to T(s) is more than T3(s). Comparison of responses of all systems is shown below on single plot.

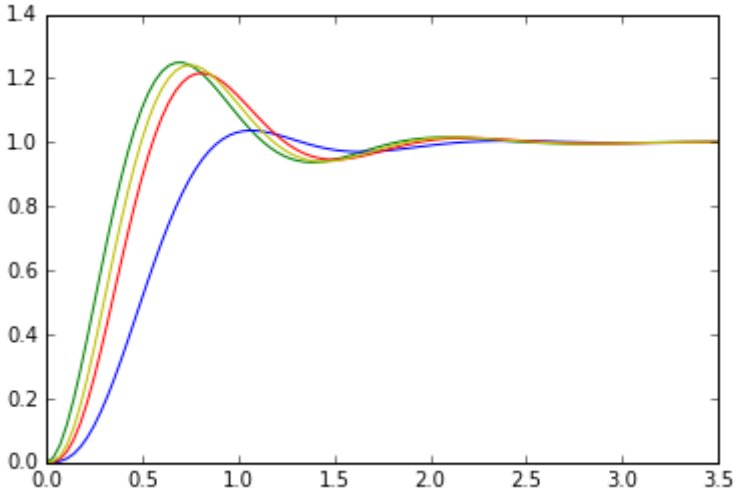
In [6]:

```
desired_response = numpy.ones(len(time))

plt.plot(time,stepresponse_t,color = 'g')
plt.plot(time,stepresponse_t1,color='b')
plt.plot(time,stepresponse_t2,color = 'r')
plt.plot(time,stepresponse_t3,color = 'y')
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x7fb2e74b5850>]
```



Assignment 9.3

The system in Fig. 2 is a unity feedback system with a compensator.

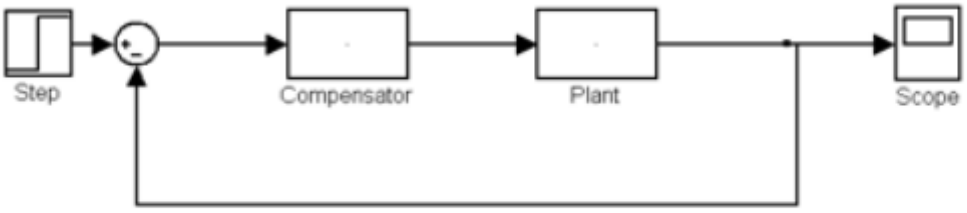


FIGURE 2. Unity feedback system with a compensator.

Let the transfer function of the block named 'plant' be given by following expression:

$$G(s) = \frac{1}{(s+1)(s+2)(s+10)}$$

1. Let the gain of the compensator be 100 and simulate the system.
2. Let the complete transfer function of the 'compensator' be C(s), where:

   $$C(s) = \frac{100(s+0.1)}{s}$$

   Simulate the system and compare your result with the result of part (1).
3. A model of the system is shown in

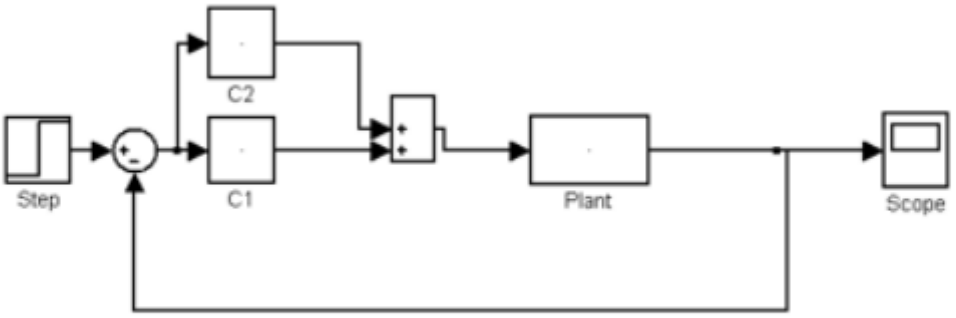


FIGURE 3. Unity feedback system with a compensator.

. Let K_1 = 100 and K_2 = 10. Simulate the system with following transfer functions for C_1 and C_2 :

$$C_1(s) = K_1$$

$$C_2(s) = \frac{K_2}{s}$$

Compare the result with the result of part (2). How do $C_1(s)$ and $C_2(s)$ relate to $C(s)$ in part 2?

4. Let the transfer function of the compensator in Fig.2 be following:

$$C(s) = \frac{100(s + 0.111)}{(s + 0.01)}$$

Simulate the system and compare the result with the result of part (2).

5. Briefly summarize your results (part (1) to (4)) in plain english by commenting on the compensator(s) and their role in the system response.

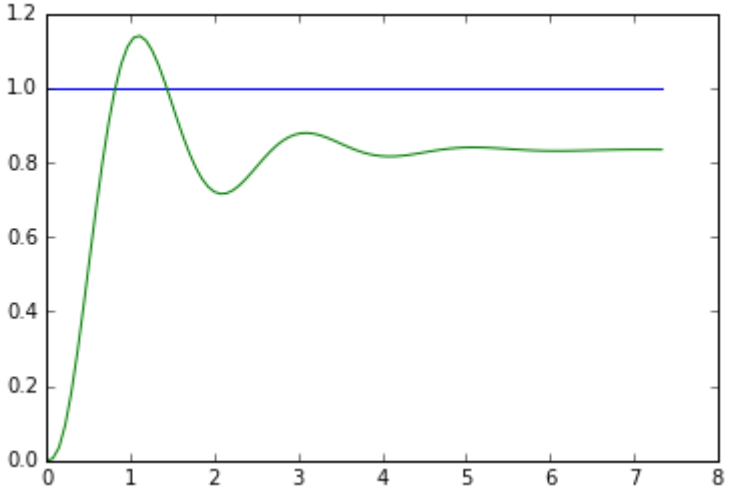
In [24]:

```
# Solution here
# Part 1
import control
import numpy
from control import *
import matplotlib.pyplot as plt
compensator_num = [100]
compensator_den = [1]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,13,32,20]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback_1 = control.feedback(G,1)
step_response_part1, time1 = step(feedback_1)
desired_response = numpy.ones(len(time1))
plt.plot(time1,desired_response)
plt.plot(time1,step_response_part1)
```

```
          100
------------------------
s^3 + 13 s^2 + 32 s + 20
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x7fb2e684bf10>]
```



In [ ]:

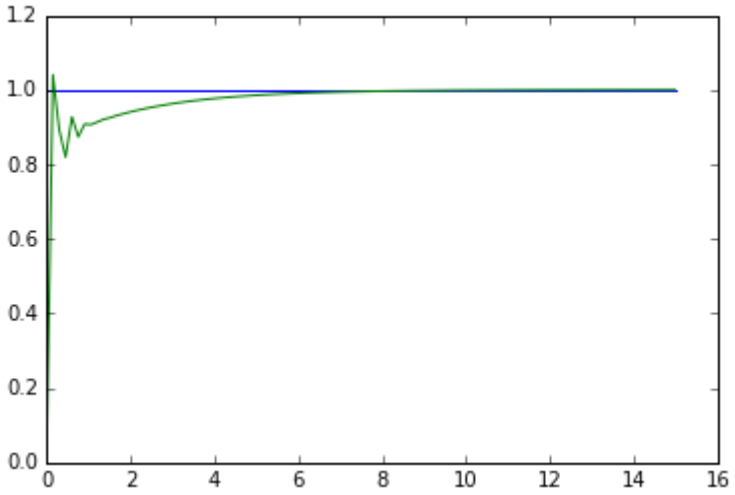
In [26]:

```
# part 2
# Solution here
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [100,10]
compensator_den = [1,0]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,13,32,20]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback_2 = control.feedback(G,1)
step_response_part2, time2 = step(feedback_2)
desired_response = numpy.ones(len(time))
plt.plot(time,desired_response)
plt.plot(time,step_response_part2)
```

```
         100 s + 10
----------------------------
s^4 + 13 s^3 + 32 s^2 + 20 s
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x7fb2e6663710>]
```

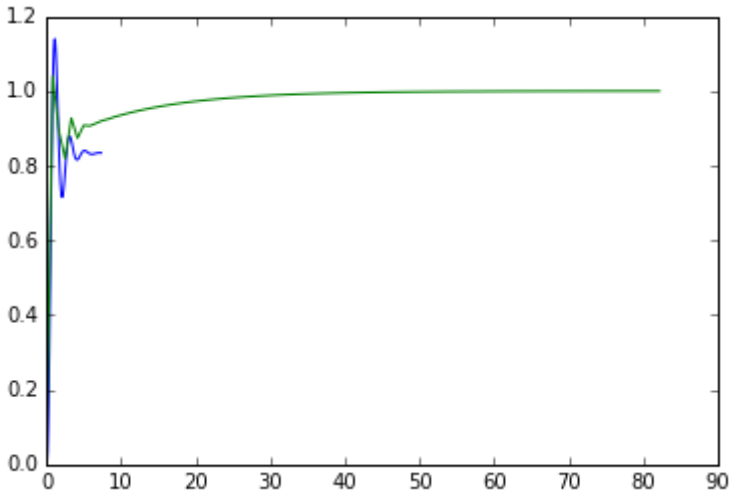


In [29]:

```
#Comparing responses of 1 and 2.
plt.plot(time1,step_response_part1)
plt.plot(time2,step_response_part2)
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x7fb2e63da950>]
```



By looking at the responses of 1 and 2, it is clear that both the system has same number of poles with same values but system 1 has larger overshoots than system 2. Moreover, rise time and settling time of 2 is much lesser than 1. Lastly, by adding compensator, it is clear that we can approach to the desired response while on the other hand, 1 is no way approaching the desired response.

In [ ]:

In [ ]:

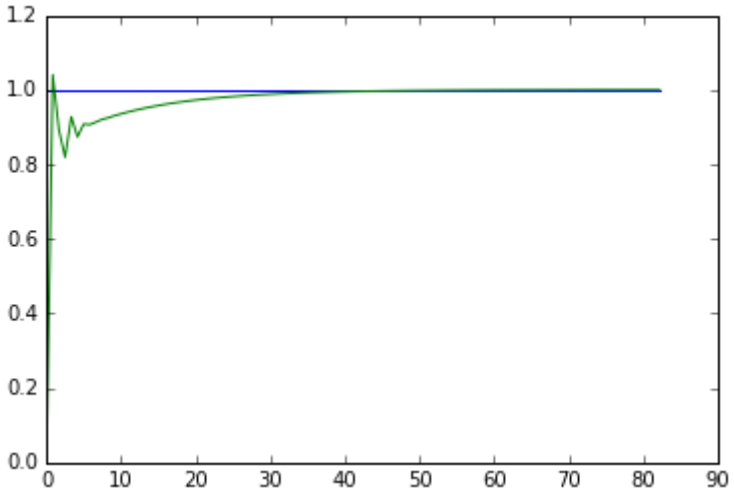
In [25]:

```
# part 3
# Solution here
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [100,10]
compensator_den = [1,0]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,13,32,20]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response, time3 = step(feedback)
desired_response = numpy.ones(len(time3))
plt.plot(time3,desired_response)
plt.plot(time3,step_response)
```

```
         100 s + 10
----------------------------
s^4 + 13 s^3 + 32 s^2 + 20 s
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x7fb2e6735690>]
```



Responses of system 2 and 3 are exactly same as transfer function of both the system is same. On the other hand, $C_1(s)$ and $C_2(s)$ are parallel in C(s). Therefore, $C(s) = C_1(s) + C_2(s)$

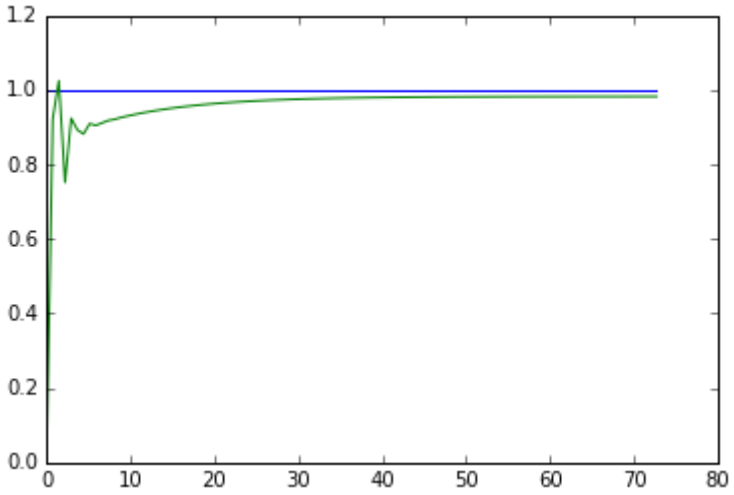
In [31]:

```
# part 4
# Solution here
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [100,11.1]
compensator_den = [1,0.01]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,13,32,20]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_4, time4 = step(feedback)
desired_response = numpy.ones(len(time4))
plt.plot(time4,desired_response)
plt.plot(time4,step_response_4)
```

```
                100 s + 11.1
-------------------------------------------
s^4 + 13.01 s^3 + 32.13 s^2 + 20.32 s + 0.2
```

Out[31]:

```
[<matplotlib.lines.Line2D at 0x7fb2e6382410>]
```

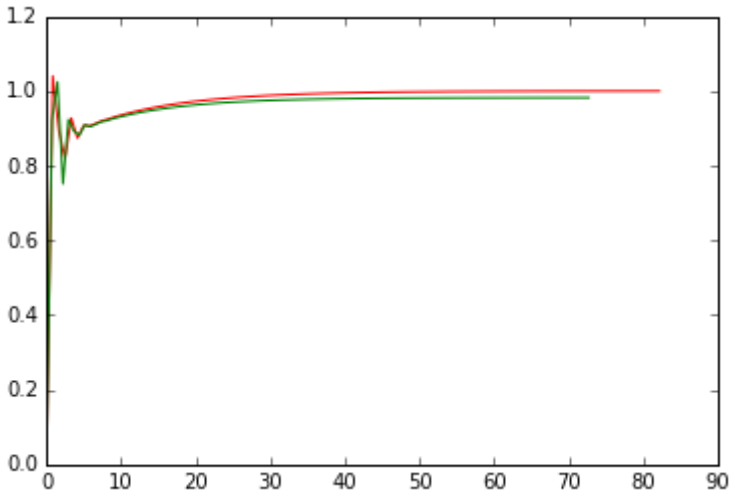


In [33]:

```
#Summaries
plt.plot(time2,step_response_part2,color='r')
plt.plot(time4,step_response_4,color = 'g')
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x7fb2e60624d0>]
```



By comparing result of system 2 and system 4, it can be concluded that,peak value of system 2 is little higher than system 4 and system 2 achieves desired response i.e. 1 , while system 4 doesnot approaches to desired response. While both the system are stable and transient response of both the system is almost similar.

Compensator plays vital role in the control system. A compensator can be used in control system in order to achieve the desired performance of system. It can compensate a unstable system to stable. Furthermore, it can also reduces the overshoots of the system.Moreover, this can also change the performace of system by adding poles or zeros in the system.

Therefore, in above example, we can see that initially system has no compensator so desired response was not achieved, but after inserting compensator our output seems improved due to less number of overshoot, less transient response and desired response is achieved easily.

Assignment 9.4

Again consider the system in

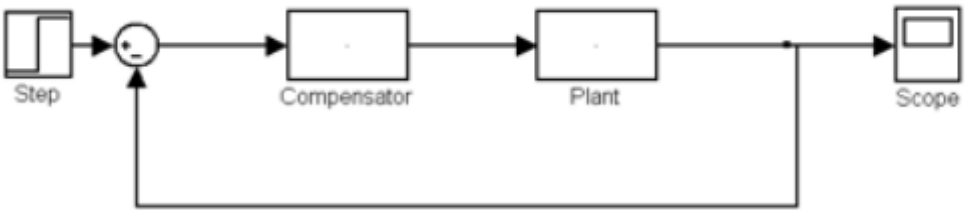


FIGURE 2. Unity feedback system with a compensator.

. Let the transfer function of the plant be G(s), where:

$$G(s) = \frac{1}{s(s+4)(s+6)}$$

1) Let the gain of the compensator be 25. Simulate.

2) Let the complete transfer function of the compensator block be C(s), where:

$$C(s) = 25(s + 3.006)$$

Simulate the system and compare your result with the result of part (1).

3) See

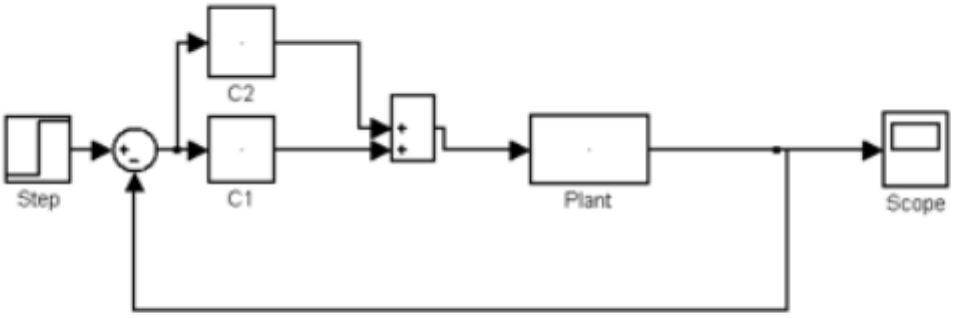


FIGURE 3. Unity feedback system with a compensator.

. Let K_1 = 75.15 and K_2 = 25. Simulate the system with following transfer functions for C_1 and C_2 :

$$C_1(s) = K_1$$
$$C_2(s) = K_2 s$$

Compare the result with the result of part (2). How do $C_1(s)$ and $C_2(s)$ relate to C(s) in part 2?

4) Briefly summarize your results (part (1) to (3)) in plain english by commenting on the compensator(s) and their role in the system response.

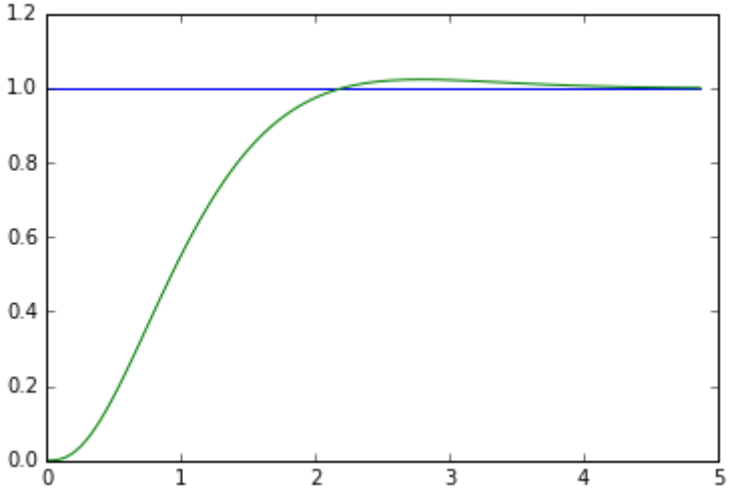
In [34]:

```
# Solution here
# Part 1
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [25]
compensator_den = [1]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,10,24,0]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_1, time1 = step(feedback)
desired_response = numpy.ones(len(time1))
plt.plot(time1,desired_response)
plt.plot(time1,step_response_1)
```

```
         25
--------------------
s^3 + 10 s^2 + 24 s
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x7fb2e6056d50>]
```



In [35]:

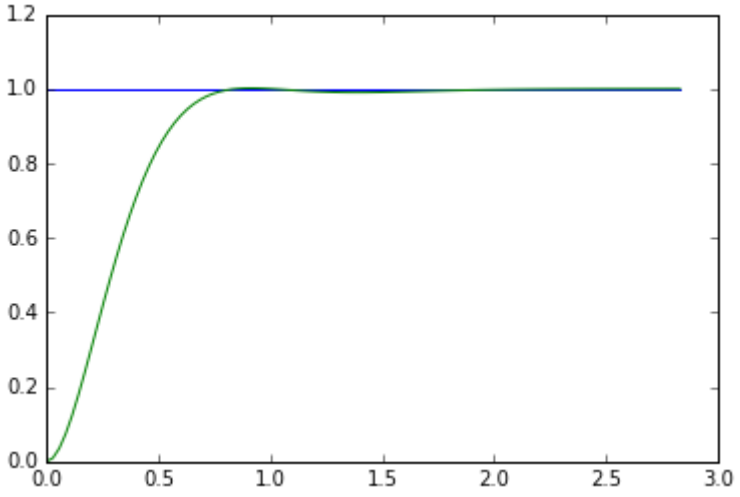
```
# Solution here
# Part 2
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [25,75.15]
compensator_den = [1]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,10,24,0]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
print feedback
step_response_2, time2 = step(feedback)
desired_response = numpy.ones(len(time2))
plt.plot(time2,desired_response)
plt.plot(time2,step_response_2)
```

```
    25 s + 75.15
-------------------
s^3 + 10 s^2 + 24 s


        25 s + 75.15
----------------------------
s^3 + 10 s^2 + 49 s + 75.15
```

Out[35]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5ee9250>]
```

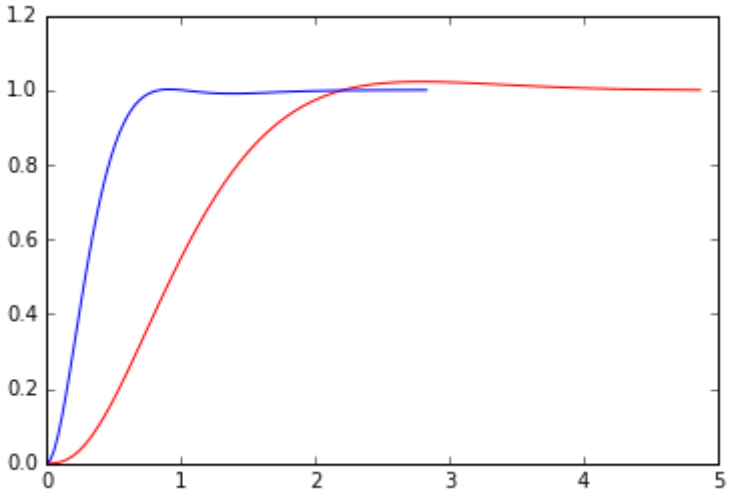


In [36]:

```
plt.plot(time1,step_response_1,color='r')
plt.plot(time2,step_response_2)
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5f75a90>]
```



By comparing above two system responses, It is clear that by adding zero with the help of compensator in system 2, system response has largely improved as transient response is less and system achieves stability fast. While system 1, only has gain therefore transient response is not better as compared to system 2.

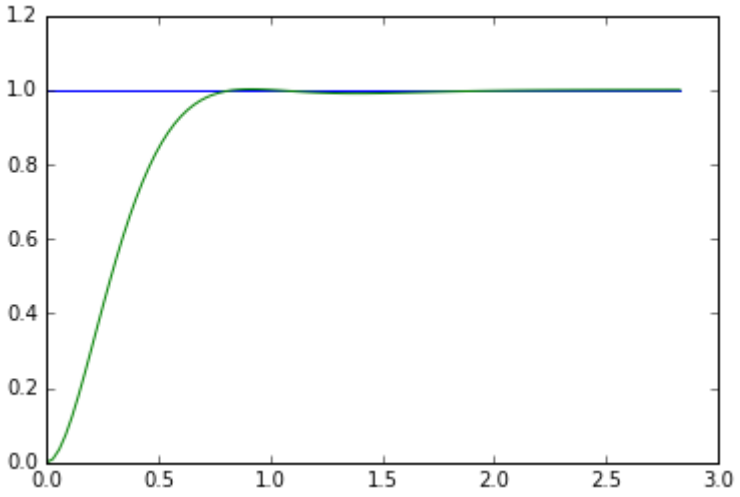
In [37]:

```
# Solution here
# Part 3
#C1 and C2 are to parallel
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [25,75.15]
compensator_den = [1]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1]
#solve denominator part manually we have
den_plant = [1,10,24,0]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_3, time3 = step(feedback)
desired_response = numpy.ones(len(time3))
plt.plot(time3,desired_response)
plt.plot(time3,step_response_3)
```

```
    25 s + 75.15
--------------------
s^3 + 10 s^2 + 24 s
```

Out[37]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5e92c90>]
```

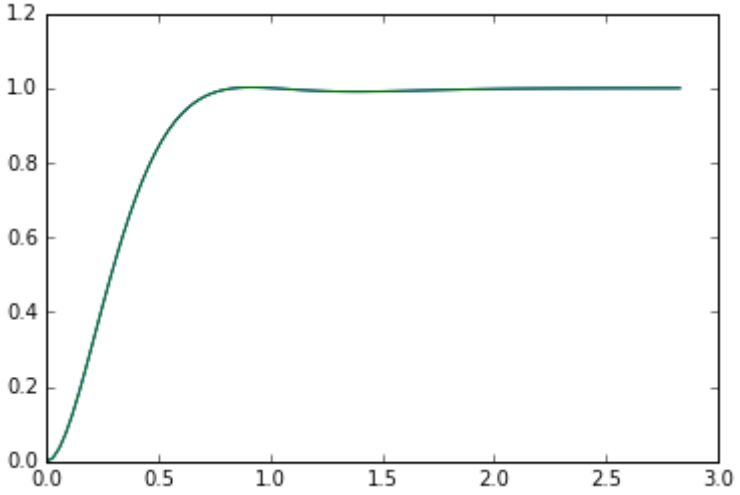


In [38]:

```
plt.plot(time2,step_response_2)
plt.plot(time3,step_response_3)
```

Out[38]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5d34d10>]
```



Responses of system 2 and 3 are exactly same as transfer function of both the system is same. On the other hand, $C_1(s)$ and $C_2(s)$ are parallel in C(s). Therefore, $C(s) = C_1(s) + C_2(s)$

By comparing above two system responses, It is clear that by adding zero with the help of compensator in system 3, system response has largely improved as transient response is less and system achieves stability fast. While system 1, only has gain therefore transient response is not better as compared to system 3.

Compensator plays vital role in the control system. A compensator can be used in control system in order to achieve the desired performance of system. It can compensate a unstable system to stable. Furthermore, it can also reduces the overshoots of the system.Moreover, this can also change the performace of system by adding poles or zeros in the system.

Assignment 9.5

Once again consider the system in

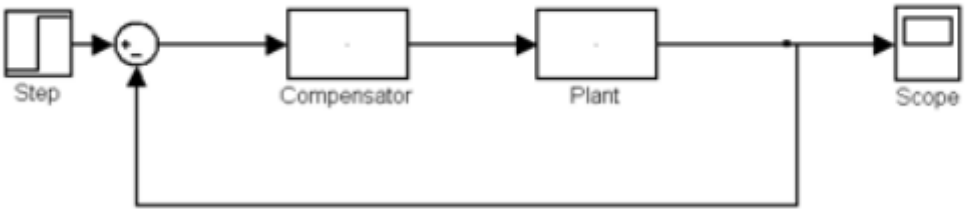


FIGURE 2. Unity feedback system with a compensator.

. Let the transfer function of the plant be G(s), where:

$$G(s) = \frac{s+8}{(s+3)(s+6)(s+10)}$$

1) Let the gain of the compensator be 121.5. Simulate.

2) Let the complete transfer function of the compensator block be C(s), where:

$$C(s) = \frac{4.6(s + 55.92)(s + 0.5)}{s}$$

Simulate the system and compare your result with the result of part (1).

3) See

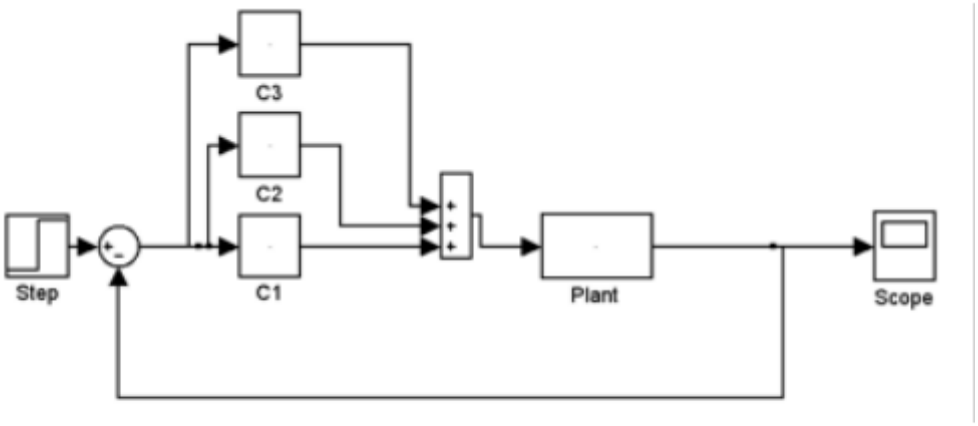


FIGURE 4. Unity feedback system with a compensator.

. Let K_1 = 259.532, K_2 = 128.616 and K_3 = 4.6. Simulate the system with following transfer functions for C_1, C_2 and C_3:

$$C_1(s) = K_1$$
$$C_2(s) = \frac{K_2}{s}$$
$$C_3(s) = K_3 s$$

Compare the result with the result of part (2). How do $C_1(s)$, $C_2(s)$ and C_3 relate to C(s) in part 2?

4) Briefly summarize your results (part (1) to (3)) in plain english by commenting on the compensator(s) and their role in the system response.

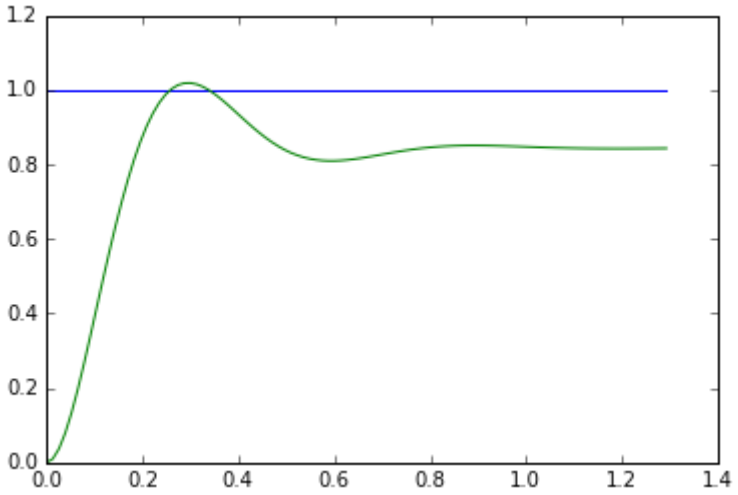
In [39]:

```
# Solution here
# Part 1
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [121.5]
compensator_den = [1]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1,8]
#solve denominator part manually we have
den_plant = [1,19,108,180]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_1, time1 = step(feedback)
desired_response = numpy.ones(len(time1))
plt.plot(time1,desired_response)
plt.plot(time1,step_response_1)
```

```
       121.5 s + 972
--------------------------
s^3 + 19 s^2 + 108 s + 180
```

Out[39]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5c78d10>]
```



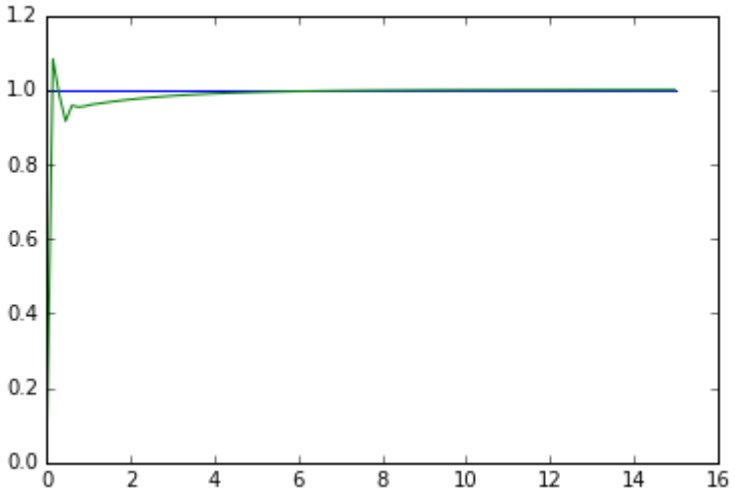
In [40]:

```
# Solution here
# Part 2
import control
from control import *
import matplotlib.pyplot as plt
compensator_num = [4.6,259.532,128.616]
compensator_den = [1,0]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1,8]
#solve denominator part manually we have
den_plant = [1,19,108,180]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_2, time2 = step(feedback)
desired_response = numpy.ones(len(time2))
plt.plot(time2,desired_response)
plt.plot(time2,step_response_2)
```

```
4.6 s^3 + 296.3 s^2 + 2205 s + 1029
-----------------------------------
   s^4 + 19 s^3 + 108 s^2 + 180 s
```

Out[40]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5c17310>]
```

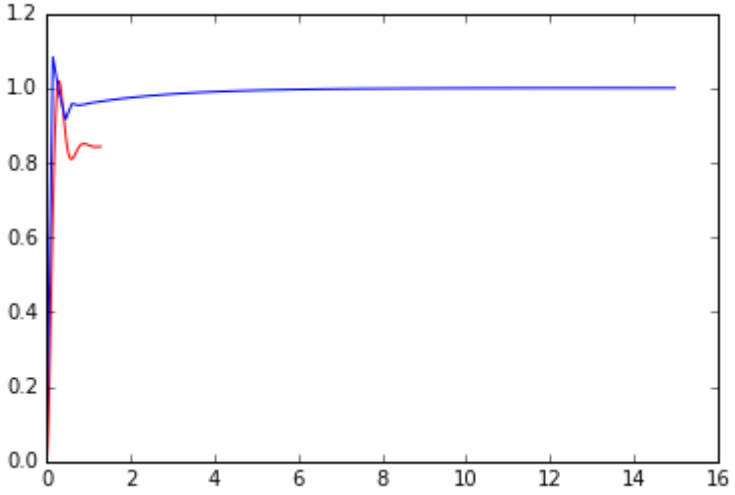


In [41]:

```
plt.plot(time1,step_response_1,color='r')
plt.plot(time2,step_response_2)
```

Out[41]:

```
[<matplotlib.lines.Line2D at 0x7fb2e629ab90>]
```



By looking at the responses of 1 and 2, it is clear that both the system has same number of poles with same values but system 2 has larger overshoot value than system 1. Lastly, by adding compensator, it is clear that system 2 approaches to the desired response while on the other hand, system 1 is not approaching the desired response.

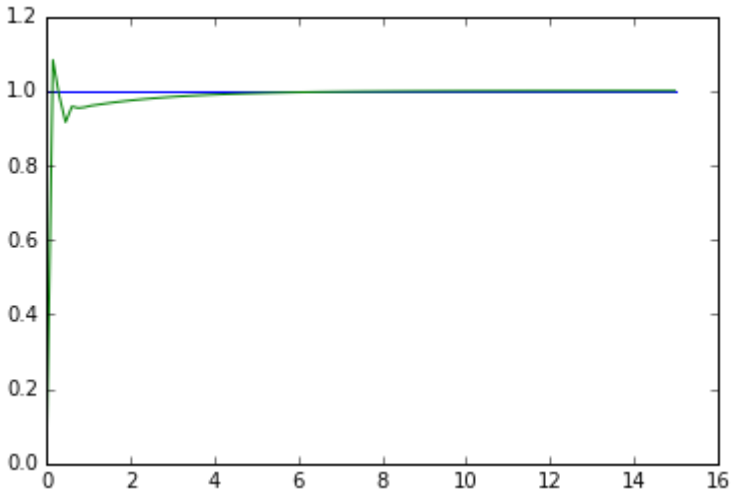
In [42]:

```
# Solution here
# Part 3
import control
from control import *
#c1,c2, and c3 are parallel
import matplotlib.pyplot as plt
compensator_num = [4.6,259.532,128.616]
compensator_den = [1,0]
compensator = control.TransferFunction(compensator_num,compensator_den)
num_plant = [1,8]
#solve denominator part manually we have
den_plant = [1,19,108,180]
plant = control.TransferFunction(num_plant,den_plant)
G = control.series(compensator,plant)
print G
feedback = control.feedback(G,1)
step_response_3, time3 = step(feedback)
desired_response = numpy.ones(len(time3))
plt.plot(time3,desired_response)
plt.plot(time3,step_response_3)
```

```
4.6 s^3 + 296.3 s^2 + 2205 s + 1029
-----------------------------------
   s^4 + 19 s^3 + 108 s^2 + 180 s
```

Out[42]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5b06d90>]
```

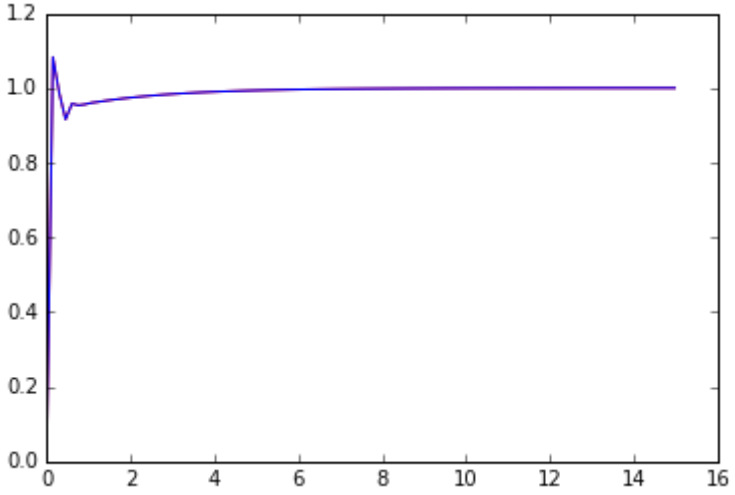


In [43]:

```
plt.plot(time2,step_response_2,color='r')
plt.plot(time3,step_response_3)
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x7fb2e5c22a90>]
```



Responses of system 2 and 3 are exactly same as transfer function of both the system is same. On the other hand, $C_1(s)$, $C_2(s)$, and $C_3(s)$ are parallel in C(s). Therefore, $C(s) = C_1(s) + C_2(s) + C_3(s)$

By comparing above two system responses, It is clear that by adding zero with the help of compensator in system 3, system response has largely improved as transient response is less and system achieves stability fast. While system 1, only has gain therefore transient response is not better as compared to system 3.

Compensator plays vital role in the control system. A compensator can be used in control system in order to achieve the desired performance of system. It can compensate a unstable system to stable. Furthermore, it can also reduces the overshoots of the system.Moreover, this can also change the performace of system by adding poles or zeros in the system.

Assignment 9.6

A second order system is given by the following transfer function:

$$G(s) = \frac{361}{s^2 + 16s + 361}$$

Without the help of python , using the formulas in Assignment 9.1 find T_p and $\%OS$ for this system.

The standard form of second order system is :

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

By comparing given equation with standard form, we have :

$$\omega_n^2 = 361$$

Therefore,

$$\omega_n = 19$$

Similarly

$$2\zeta\omega_n s = 16s$$

Therefore, ζ can be computed as :

$$\zeta = \frac{16}{2 * 19} = 0.42$$

As we know, T_p is defined as :

$$T_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$$

substituting values for ω_n and ζ in above equation,

$$T_p = \frac{\pi}{19\sqrt{1-\zeta^2}} = 0.182$$

As we know, ζ is related to %OS as :

$$\zeta = \frac{-\ln\frac{\%OS}{100}}{\sqrt{\pi^2 + \ln^2(\frac{\%OS}{100})}}$$

Therfore %OS can be defined as :

$$\%OS = 100 \cdot e^{\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right)}$$

substitute ζ in above equation we get

$$\%OS = 100 * 0.2336$$

$$\%OS = 23.36\%$$

In [ ]: