ASEN 2003 - Sec. 013 - Lab 6
Rotary Position Control
May 3, 2017

Forrest Barnes [*], Luke Tafur [†], Nicholas Renninger [‡], Yuzhang Chen [§]
University of Colorado - Boulder

The purpose of this lab was to explore the concepts of proportional and derivative control, and then apply those concepts to the control of both a rigid and flexible arm. The objective was to determine the derivative and proportional gains that minimized the settling time of the rigid and flexible arms while still keeping their motor use to a minimum. By using monte-carlo and grid search algorithm based simulations, the best gains were calculated for the rigid and flexible arm systems. The rigid arm system was more straightforward and was found to perform best in simulations with a proportional gain (K_p) of 6.97 and a derivative gain (K_D) of 0.191. During experimentation, the gains were furhter tuned in response to the system's real world performance, and the final gains chosen to best control the rigid arm were $K_p = 7$ and $K_D = 0.3$. The flexible arm's behavior differed a lot from the simulation. While the simulation predicted optimal gain values of $K_{P\theta} = 1.8$, $K_{Pd} = -15$, and $K_{D\theta} = K_{Dd} = 0$, experimental testing yielded $K_{P\theta} = 6.8$, $K_{Pd} = -6.4$, $K_{D\theta} = 1.3$, and $K_{Dd} = 1.1$ as the optimal gains to control the flexible arm system. Not only were the optimal control gains different between model and actual results, but the time response of the system also saw significant deviations from model behavior during experimental testing. Overall, the group gained an understanding of controls and were able to calculate gains that produced desired system characteristics.

Nomenclature

$\mathcal{L}$	=	Laplace Transform
ω_L	=	Angular velocity of arm [rad/s]
ω_m	=	Angular velocity of motor [rad/s]
θ	=	Angle (location) of Arm [rad]
$\Theta_D(s)$	=	$\mathcal{L}[\theta_D]$ - desired pointing angle in the Laplace frequency domain
$\theta_D(t)$	=	Desired Angle (location) of Arm [rad]
$\Theta_L(s)$	=	$\mathcal{L}[\theta_D]$ - measured pointing angle in the Laplace frequency domain
$\theta_L(t)$	=	Measured Angle (location) of Arm [rad]
ζ	=	Damping coefficient [kg m^2/s]
I_m	=	Current through motor [A]
J	=	Moment of inertia of arm [kg m^2]
K_D	=	Derivative control [V s/rad]
K_g	=	Motor to arm gear ratio
K_m	=	Motor speed constant [rad/s/V]
K_P	=	Proportional control [V/rad]
M_0	=	Moment applied to arm [N m]
M_m	=	Torque/Moment created by motor [kg/m/s]
R_m	=	Motor resistance [Ω]
V_{in}	=	Voltage applied to arm by motor [V]

[*]104388401
[†]104923506
[‡]105492876
[§]104420210

Contents

List of Figures

List of Tables

I. Theory and Simulation

The following sections detail the theoretical steps taken to develop a model of both the rigid and flexible arms, and how the gains were chosen for both types of arms. Finally, a control block diagram is presented for the rigid arm system, as an example of their usefulness in describing the control law for a PD-controlled dynamical system.

I.A. Rigid Arm Closed Loop Transfer Function

Beginning with Eqn. 13 from the lab document[1] and substituting the V_{in} from equation 16, equation 17 and 18 were derived as shown in Figure 1.

Eq. 17 Derivation from Eq. 13 & Eq. 16

$$\frac{\Theta_L}{V_{in}} = \frac{K_g K_m / J R_m}{s(s + K_g^2 K_m^2 / J R_m)} \quad (13) \qquad V_{in} = V_{PD} = K_p(\Theta_D - \Theta_L) - s K_d \Theta_L \quad (16)$$

$$\frac{\Theta_L}{K_p(\Theta_D - \Theta_L) - s K_d \Theta_L} = \frac{K_g K_m / J R_m}{s(s + K_g^2 K_m^2 / J R_m)} \cdot \frac{J R_m}{J R_m}$$

$$\rightarrow \Theta_L = \frac{K_g K_m [K_p(\Theta_D - \Theta_L) - s K_d \Theta_L]}{J R_m s^2 + s K_g^2 K_m^2}$$

$$\rightarrow \Theta_L = \frac{K_p K_g K_m \Theta_D - K_p K_m K_g \Theta_L - K_g K_m K_d s \Theta_L}{J R_m s^2 + s K_g^2 K_m^2}$$

$$\rightarrow 0 = \frac{K_p K_g K_m \Theta_D - K_p K_g K_m \Theta_L - K_g K_m K_d s \Theta_L - (J R_m s^2 + s K_g^2 K_m^2)\Theta_L}{J R_m s^2 + s K_g^2 K_m^2}$$

$$\rightarrow \frac{K_p K_g K_m \Theta_L + K_g K_m K_d s \Theta_L + (J R_m s^2 + s K_g^2 K_m^2)\Theta_L}{J R_m s^2 + s K_g^2 K_m^2} = \frac{K_p K_g K_m \Theta_D}{J R_m s^2 + s K_g^2 K_m^2}$$

$$\rightarrow \Theta_L [K_p K_g K_m + K_g K_m K_d s + (J R_m s^2 + s K_g^2 K_m^2)] = K_p K_g K_m \Theta_D$$

$$\rightarrow \frac{\Theta_L}{\Theta_D} = \frac{K_p K_g K_m}{[K_p K_g K_m + K_g K_m K_d s + (J R_m s^2 + s K_g^2 K_m^2)]} \cdot \frac{1/J R_m}{1/J R_m}$$

$$\rightarrow \frac{\Theta_L}{\Theta_D} = \frac{K_p K_g K_m / J R_m}{K_p K_g K_m / J R_m + [(K_g K_m K_d / J R_m) + (K_g^2 K_m^2 / J R_m)] s + s^2}$$

$$\rightarrow \frac{\Theta_L}{\Theta_D} = \frac{K_p K_g K_m / J R_m}{s^2 + (K_g^2 K_m^2 / J R_m + K_d K_g K_m / J R_m) s + K_p K_g K_m / J R_m} \quad (17)$$

Figure 1: Derivation of the closed loop transfer function of the rigid arm.

The closed loop transfer functions for the flexible arm are beyond the scope of this document, but are presented in Figure 2 for completeness.[1] Due to the double mass-spring nature of the flexible arm system, the resultant characteristic polynomials in the denominators of the closed-loop transfer functions for the flexible arm are 4th order, and thus they are much more challenging to use analytically. For this reason, the selection of gains does not come quite as easily as with the second order system that describes the rigid arm system.

$$\frac{\Theta_L}{\Theta_D} = \frac{K_1\left[r_1 s^2 + (q_1 r_2 - r_1 q_2)\right]}{s^4 + \lambda_3 s^3 + \lambda_2 s^2 + \lambda_1 s + \lambda_0}$$

$$\frac{D}{\Theta_D} = \frac{K_1\left[r_2 s^2 + (p_2 r_1 - r_2 p_1)s\right]}{s^4 + \lambda_3 s^3 + \lambda_2 s^2 + \lambda_1 s + \lambda_0}$$

where:

$K_1 = K_{p\theta}$	$\lambda_3 = -p_1 + K_3 r_1 + K_4 r_2$
$K_2 = K_{pd}$	$\lambda_2 = -q_2 + K_1 r_1 + K_2 r_2 + K_4(p_2 r_1 - r_2 p_1)$
$K_3 = K_{D\theta}$	$\lambda_1 = p_1 q_2 - q_1 p_2 + K_3(q_1 r_2 - r_1 q_2) + K_2(p_2 r_1 - r_2 p_1)$
$K_4 = K_{Dd}$	$\lambda_0 = K_1(q_1 r_2 - r_1 q_2)$

Figure 2: Derivation of the closed loop transfer function of the rigid arm.

I.B. Theoretical Selection of Gains

The rigid gains were calculated using the script `findRigidGains.m` in Appendix B by developing a variable-step Monte Carlo simulation to check for proportional and derivative gains that would satisfy the voltage limitations of the motor. The motor was not allowed to operate above $\pm 5V$, for longevity of the motor.

However, another very important limit placed on the motor's voltage has to do with its dead-band, which is characterized by the motor's inability to move due to internal friction below a certain threshold voltage. Without empirical testing, there is no way to determine where exactly the motor's dead band is, so we chose to define the dead-band as any voltage below 1V. We chose to not allow for gain values that would cause the motor to be in its dead-band for extended periods of time, as this would mean that the system would no longer be able to controlled. Most importantly, we do not want to begin controlling the arm with a motor voltage in or near the dead-band, as this would surely result in a complete lack of control over the system. For this reason, the search algorithm employed was limited to search for gain values that would correspond to motor voltages between 1 and 5 volts.

Not only did we want the motor to respond within its operating limits, but we also wanted the motor to respond to the step input quickly, so we limited our search for the optimal gain values to gains that drove the arm to within 5% of θ_D within 0.5 seconds. This requirement can also be notated as the system having a 5% settling time of less than 0.5 seconds. Because the system is second order, we also know some relationships between K_P, K_D, and V_{in} because of the standard form of the denominator in Eqn. 17. Relating the second and third term in the characteristic equation of the closed loop transfer function (the denominator of Eqn. 17) to each other and solving for ζ, then re-arranging to solve for K_D, we get the relation in Eqn. 1:

$$K_D = \frac{2\zeta\sqrt{K_P K_G K_M J R_M} - K_G^2 K_M^2}{K_G K_M} \tag{1}$$

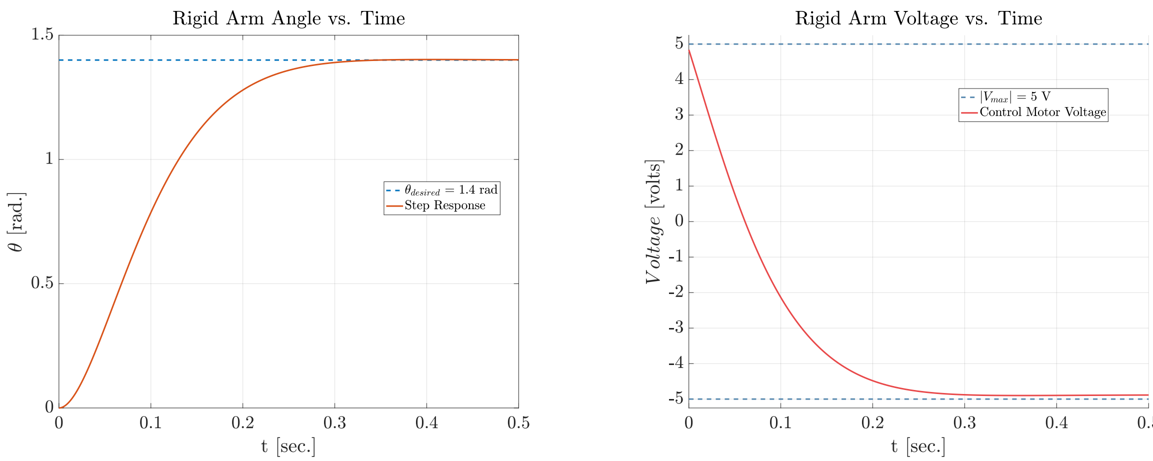
Picking values for K_P and ζ with Eqn. 1 allow for the calculation of values of K_D. Once these values are set, the closed transfer function can be set and evaluated using MATLAB, which gives us the time response of the system (determines $\theta(t)$ numerically). Using the numerically determined $\theta(t)$, we can use Eqn. 2 to determine the V_{in} at every time value, and make sure that the gains chosen do not push the motor over the 5V limit established above, or make the motor respond with too little voltage (dead-band effect described above) over the entire response period, which will in practice make the system completely unresponsive to control inputs.

$$V_{in} = K_P(\theta_D(t) - \theta_D(t)) - K_D\dot{\theta}_D(t) \tag{2}$$

We also can make sure that the system reaches its 5% settling time withing 0.5 seconds. If the gains and damping coefficient chosen create a system that passes all of these tests, we have found a theoretically sound system. The various constraints on the gain values, as well as the mesh grid of viable gain values can be seen in Figure 4.

Now the task becomes to optimize the gains for the fastest rise time subject to the constraints given above. This was done by computing a mesh of viable gain values and then evaluating their fitness using a monte-carlo style grid search algorithm. In the end, this algorithm determined that the optimal system would have a $K_P = 6.97$, $K_D = 0.191$, $\zeta = 0.9$, and a 5% settling time of 0.082 seconds. The performance of the rigid arm system is show in Figures 3a & 3b, both plots demonstrating that the chosen gains keep the system compliant to both settling time and voltage requirements set above.

As $\zeta < 1$, which can be seen by the fact that the gain values we chose are "below" the $\zeta = 1$ line (as seen in Figure 4), the controlled system is slightly under-damped. This was intentionally allowed, as under-damped systems have faster rise times, and we were not looking for 0% overshoot, but rather less than 5% overshoot, so under-damping the system slightly will allow it to perform better under the requirements given. Also, as the real system likely has some damping not accounted for, under-damping the theoretical system will likely result in closer to critical damping in the actual system.



(a) Plot showing the system's step response to a pointing angle command of $\theta_D = 0.7rad$

(b) Plot showing the commanded motor voltage in response to a pointing angle command of $\theta_D = 0.7rad$

Figure 3: Rigid arm time versus angle trials.

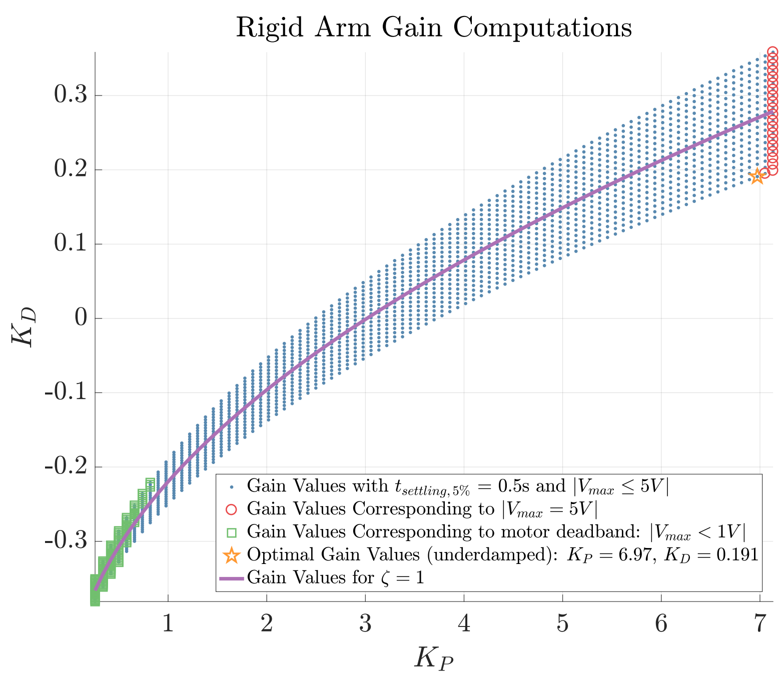


Figure 4: Plot showing the search process and the limits on the gains set by the motor constraints

The flexible gain values were produced with a similar method, but different as the system no longer has any analytic relations between the gains and the voltage, as the flexible system is a fourth-order system.

The gains were no longer inter-related, and thus a full monte-carlo simulation was run to determine gains that worked well. Once again, the dead-band and max voltage constraints were in place, but the 5% settling time requirement was decreased to 1 second. Now, the Monte Carlo simulation found that the gain values that performed well, and within the constraints were $K_{P\theta} = 1.8$, $K_{Pd} = -15$, and $K_{D\theta} = K_{Dd} = 0$. The behavior of the simulated flexible arm system can be seen if Figure 5. However, as will be seen in §II, the theoretically valid gains did not work well in the actual system.

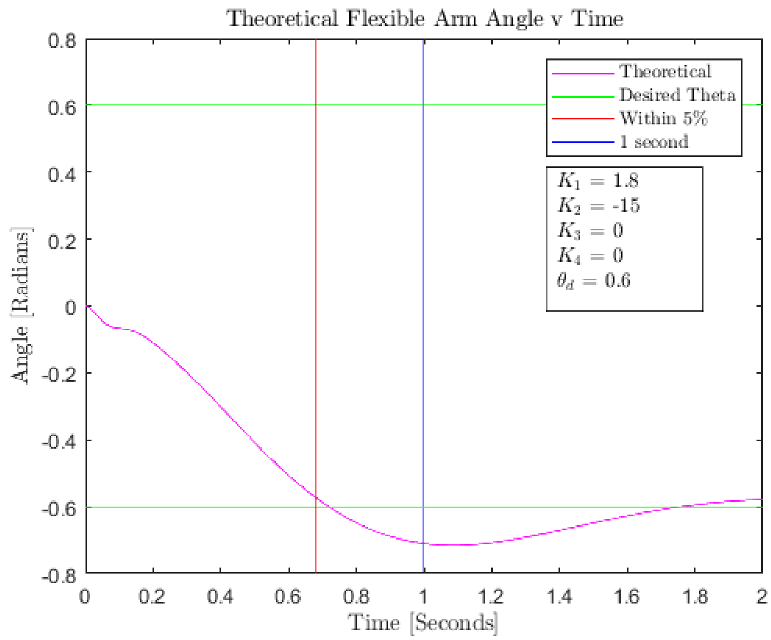


Figure 5: Plot showing the ideal flexible gains that were calculated

I.C. Control Block Diagram

The control block diagram was created once Eqns. 17 & 18 were derived. After deriving the closed loop transfer function, the control block diagram could be created and is shown in Figure 6.

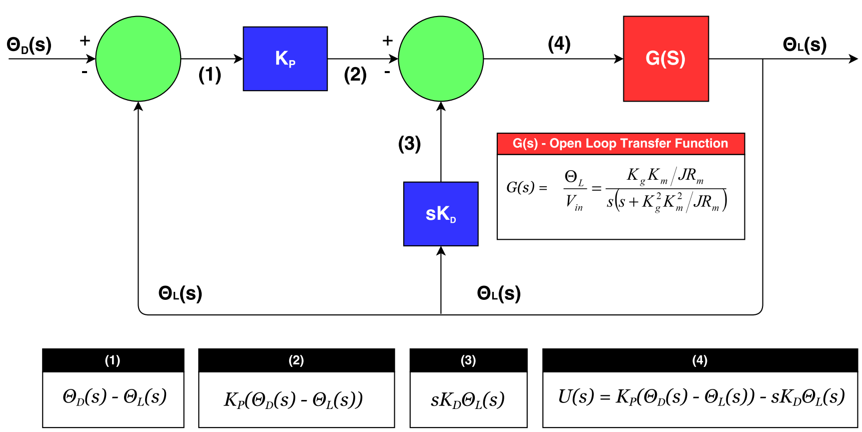
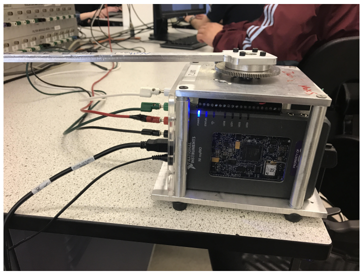


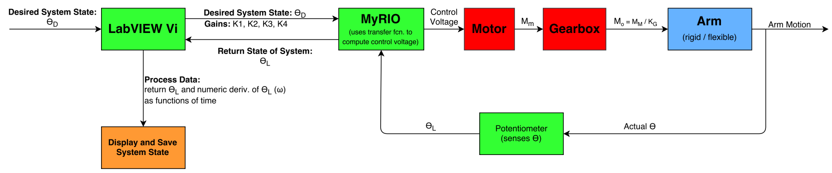
Figure 6: Control block diagram for rigid arm.

II. Experiment

LabVIEW, a MyRIO embedded system, a DC motor, and a potentiometer on the arm itself worked in conjunction to control the arm. The Functional Block diagram describes how the components interacted with each other to control the movement of the arm, as shown in Figure 7.



(a) **Equipment**



(b) Functional Block Diagram, showing how different pieces of the controls puzzle work together to control the arms

Figure 7: Setup

After calculating the most effective gains, tests were performed to see if the gains could actually control the arm within requirements. Ten trials were performed with both the rigid arm and the flexible arm, where the gains corresponding to the best two trials were chosen as the most desirable gains. To collect the data after the trial was performed all that had to be done was click "save data" on the LabVIEW program. The group was able to specify how long back data is saved to ensure that it was saved properly and could be used to analyze the efficacy of the gains chosen theoretically.

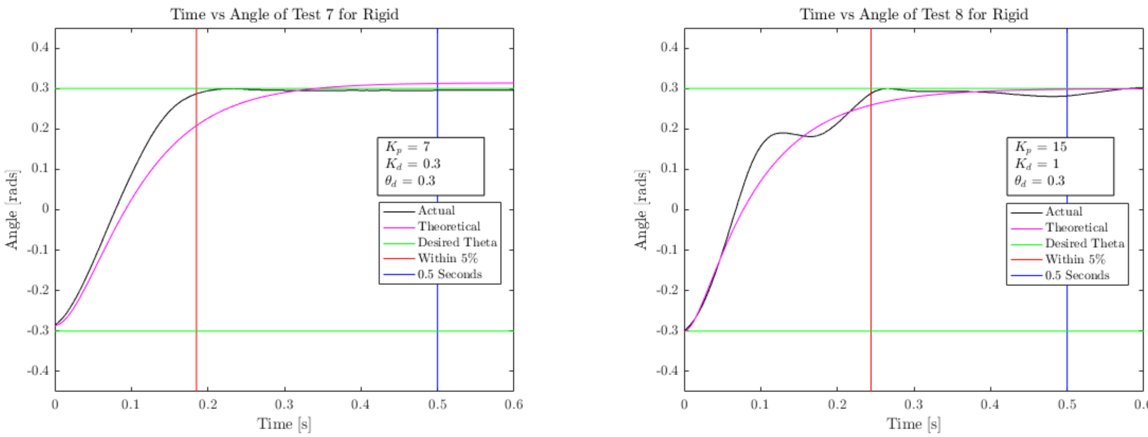
Throughout testing, the program worked relatively well, but the dead-band of the motor was discovered very quickly. Based on our theoretical predictions, there were gains that should ideally work, but after the motor dead-band and other factors such as friction they did not. Several times the angle would get within about 10% of the desired angle, but never actual reach it. The group believes this was caused by the motor's dead band. They also believe that the actual tests took longer than the theoretical on average due to frictional forces not accounted for by our model.

III. Results and Analysis

The rigid arm was the first to be tested using the desired gains the group had calculated, where $K_D = 0.191$ and $K_P = 6.97$. Test 7 was performed using a proportional gain of 7 and a derivative gain of 0.3, and the arm was able to be controlled without any overshoot and reach the desired theta of 0.6 radians in approximately 0.185 seconds. Test 8 was performed using a proportional gain of 15 and a derivative gain of 1, and the arm was also able to be controlled without any overshoot and reach the desired theta of 0.3 radians in approximately 0.244 seconds. Test 7's gains obviously produce a much better system, which validates the method described in §I.B. for determining gains in the rigid arm (shown in Figure 4). Both tests 7 and 8 can be seen in Figure 8 along with all the trials shown in Table 1. Tests 6 and 10 are believed to have not reached their desired angle due to the motor dead-band because they still come very close, but the difference in angle and desired angle is too small for the motor to change. The voltage also never went over 5 volts, as the predicted maximum voltage matched well with that seem experimentally.

Test Number	K_p	K_d	**5% Settling Time**
1	5	0.2	0.232
2	15	0.9	0.247
3	5	0.2	0.249
4	7	0.3	0.183
5	7	0.31	0.195
6	7	0.35	Never Reached
7	7	0.3	0.185
8	15	1	0.244
9	5.5	0.2	0.204
10	10	1	1.246

Table 1: All rigid trials with 5% Settling Time

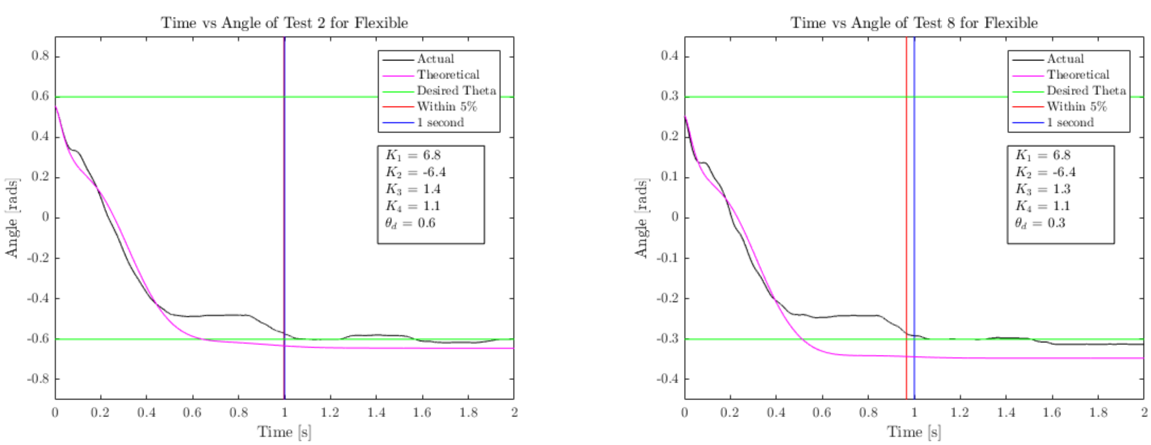


(a) Test 7 for rigid arm with $\theta_d = 0.6$ **(b) Test 7 for rigid arm with $\theta_d = 0.3$**

Figure 8: Rigid arm time versus angle trials.

The flexible arm was tested with the gains that the group had calculated to fit within the criteria, namely $K_{P\theta} = 1.8$, $K_{Pd} = -15$, and $K_{D\theta} = K_{Dd} = 0$, as shown in Figure 5. Many simulations performed with the gains that were calculated performed very well. However, when actually testing, the arm did not perform as expected. The group expected the best gains to appear whenever the derivative gains were 0, with an expected 5% settling time of about 0.568 seconds with a max tip deflection of 0.478 centimeters after 1 second.

In actuality, the best values observed while testing were K_1 = 6.8, K_2 = -6.4, K_3 = 1.3, and K_4 = 1.1. The results for tests 2 and 8 from the flexible arm are shown in Figure 9. The other criterion to pass the requirements was to keep the residual tip vibrations below 0.5 centimeters. As shown in Figure 10, the vibrations at 1 second were well below 0.55 centimeters and the residual vibrations are less than 0.5 centimeters. The difference in actual data observed versus our theoretical predictions is believed to have been caused by the motor not being able to handle not having a derivative gain, as it gets very unstable.



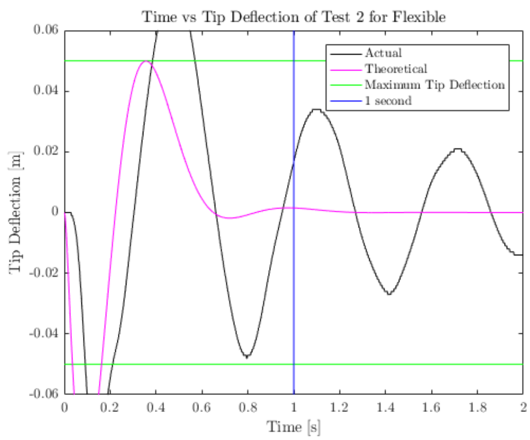
(a) Test 2 for flexible arm with $\theta_d = 0.6$ **(b) Test 8 for flexible arm with $\theta_d = 0.3$**

Figure 9: Flexible arm time versus angle trials.

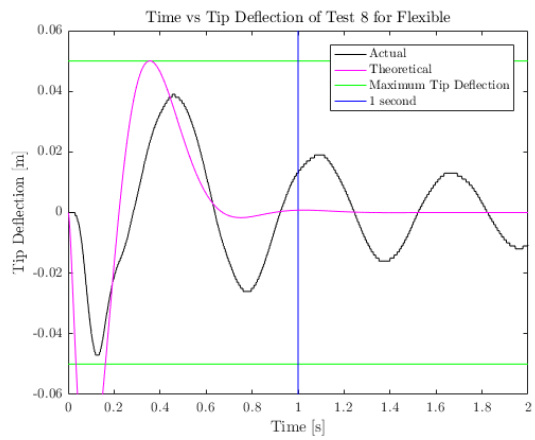
For both the flexible and rigid arm, the theoretical plots appear to match up very well with experimental result, except for the tip displacement. Forces such as friction that are not accounted for in the equations can lead to differences in actual data versus theoretical data. The flexible tip displacement seemed to vary based upon which test stand we used, which may be caused by the weight on the tip being distributed unevenly. We observed some test units with bent arms that impeded the motors' ability to control the arm to one side due to the pre-load in the arm-spring the bend causes. There are many things that could be different for the test stand, but overall they match relatively well. In Figure 9, the actual data cuts off a little before it

Test Number	K_1	K_2	K_3	K_4	Time Within 5%
1	6	-6.4	1.3	1.5	Never Reached
2	6.2	-6.4	1.3	1.5	0.997
3	6.2	-6.4	1.3	1.1	1.570
4	6.5	-6.4	1.3	1.1	1.606
5	6.5	-6.4	1.3	1.1	1.067
6	6.5	-6.4	1.3	1.1	1.036
7	6.8	-6.4	1.3	1.1	1.046
8	6.8	-6.4	1.4	1.1	0.966
9	6.8	-6.4	1.4	1.1	1.038
10	6.8	-6.4	1.4	1.1	1.070

Table 2: All flexible trials with 5% Settling Time



(a) Test 2 flexible arm tip deflection



(b) Test 8 flexible arm tip deflection

Figure 10: Flexible arm time versus tip deflection trials.

reaches the desired angle due to the motor's dead-band where it is not capable of reacting to the applied voltage. The theoretical data also shows that the arms will overshoot, but when using the actual test data it appears that they do not, most likely due to some unaccounted damping in the motor, gearbox, or bearings.

The flexible arm's results differ from the rigid arm based on many factors, but the main one is complexity. The more complex an object is, the harder it is to predict and control its movement. The rigid arm does exactly what is wanted because it is actually well approximated by a second order system. On the other hand, the flexible arm can deform and bend causing the control to not act as intended and lead to undesirable results. The coupled fourth-order system used to model it is not entirely accurate, and represents a much more approximate model than the second-order model of the rigid arm.

Controlling a flexible structure is difficult and to be able to control it properly, integral control would be very helpful. Integral control would allow us to cut out the intermittent steady-state error seen in Figure 9. However, this would then cause integral wind-up delay, which could make it challenging to meet the 5% settling time requirement of 1 second. It also would increase the complexity of analysis, as it would increase the already high order of the system.

The other major implication of trying to model a higher-order system is that the system can no longer be analyzed and gains predicted easily using the characteristic equation of the transfer function. This means that much more sophisticated means of both modeling the system and numerically determining gains must be implemented. In reality, for a flexible, dynamic structure like the flexible arm, a PID tuning package would likely need to be utilized, where the current motion of the system is fed into the tuner to dynamically adjust the gains. This would allow us to account for the variation between units, and even the variation among tests on the same unit, as the flexible arm did not always perform the same between test due its stochastic nature.

IV. Conclusions and Recommendations

Using the test units, we were able to compare theoretical calculations for proportional and derivative gains with experimental result in order to come up with optimal tuning parameters to control a rigid and flexible arm within certain performance constraints. Initially, we inputted random values to the system to observe its reaction. Later, we actually modeled each system analytically and calculated optimal gain values for each system using numeric simulations. The rigid arm system was found to be relatively easy to model, with the theoretical gains we determined working well in the actual system. On the other hand, the flexible arm system is harder to model, as the gains the model suggested did not work as well as gains produced by simple experimentation.

In order to improve the results, we would need to implement smarter gain determination algorithms for the flexible arm system, and possibly introduce integral control. If the calculation of gains could be adjusted based on the performance of the actual hardware, an empirical model combining both theory and the actual system could be developed. This is the sort of modeling that often is done to improve the performance of PD and PID controllers alike.

Improving our understanding of the motor, by experimentally determining its dead-band characteristics, would also improve the performance of our controls laws by improving how we pick gains. If gains could be chosen that neglected the dead-band of the motor as much as possible, our system would have much better performance at its extremes. Alternately, a more powerful motor could be used, which would decrease the importance of dead-band as the motor would be less susceptible to such effects.

Finally, a more sophisticated model of the flexible tip system, incorporating the actual behavior of the system might help to improve our ability to control it. Improving the model may not actually be necessary however, as active PID gain tuning might be enough to compensate for any deficiencies in the model.

References

[1]Nerem, Steven. Rotary Arm_Procedures_2017. CU, 2017. PDF.

Acknowledgments

We would like to take a moment to thank all of those who helped us learn, understand, and accomplish the tasks that come with this lab. Those individuals include, but are not limited to: Trudy Schwartz, Nerem Steven, and Felix.

Appendix A

The primary contributions of each lab member are as follows:

Forrest Barnes:

Luke Tafur: Derivations, Code, Data Analysis, and Report.

Nicholas Renninger: Code, Optimization of Gain Values, and Report.

Yuzhang Chen: Functional Block Diagram.

Appendix B: MATLAB Code

Analyze Experimental Results and Compare with Experimental

```
% Lab 6: Robot Arm
```

```
% Author: Luke Tafur
% Date Created: 4.25.17
% Date Edited: 5.2.17

clear all; close all;

shouldSaveFigures = 0;
set(0, 'defaulttextinterpreter', 'latex');

%declare directory and extract for flexible
gainzFlex = dir('../Data/Real Data/flex*');
for i = 1:10
    dataFlex{i} = load(sprintf('%s/%s', gainzFlex(i).folder, gainzFlex(i).name
        ));
end

%declare directory and extract for rigid
gainzRigid = dir('../Data/Real Data/rigid*');
for i = 1:10
    dataRigid{i} = load(sprintf('%s/%s', gainzRigid(i).folder, gainzRigid(i).
        name));
end

%declare names
name = {'Flexible' 'Rigid'};

%% Flexible
for j = 1:10
    l = 1;
    Kp = [15];
    Kd = [4];

    %split up data
    time = dataFlex{j}(:,1);
    hubAngle = dataFlex{j}(:,2);
    tipDeflection = dataFlex{j}(:,3);
    hubAngularVelocity = dataFlex{j}(:,4);
    tipVelocity = dataFlex{j}(:,5);
    positionReference = dataFlex{j}(:,6);
    outputVoltage = dataFlex{j}(:,7);
    K1 = dataFlex{j}(:,8);
    K2 = dataFlex{j}(:,9);
    K3 = dataFlex{j}(:,10);
    K4 = dataFlex{j}(:,11);

    a=0;

    for k = 1:length(hubAngle)-1000
        if (abs(hubAngle(k+1000) - hubAngle(k+999)) > 0.002) && a==0
            timeStart = k+1000;
            a = 1;
        end
    end
    %% plot time versus theta
    figure(j)
```

```
        actual = plot((time(timeStart:end)-time(timeStart))/1000,hubAngle(timeStart:end),'Color',[0 0 0]);
        hold on

        %define thetad
        if j == 2
            thetad = 0.6;
        else
            thetad = 0.3;
        end
        temp = hubAngle(timeStart+500:end);
        reach5per = find(abs(temp) > thetad*0.95)+timeStart+500;
        if ~isempty(reach5per)
            reach5per = reach5per(1);
            timeTaken = time(reach5per(1)) - time(timeStart);

            %plots data lines
            start = plot([(time(reach5per)-time(timeStart))/1000 (time(reach5per)-time(timeStart))/1000],[thetad*1.5 -thetad*1.5],'Color',[1 0 0]);
            sec1 = plot([1 1],[thetad*1.5 -thetad*1.5],'Color',[0 0 1]);
            td = plot(linspace(0,time(end)/1000,1000),linspace(thetad,thetad,1000),'Color',[0 1 0]);
            plot(linspace(0,time(end)/1000,1000),linspace(-thetad,-thetad,1000),'Color',[0 1 0])

            %label and define limits
            xlabel('Time [s]')
            ylabel('Angle [rads]')

            if (timeStart + 2000) < length(time)
                xend = 2;
            else
                xend = 1;
            end
            xlim([0 xend])
            ylim([thetad*-1.5 thetad*1.5])
            title(sprintf('Time vs Angle of Test %i for %s',j,name{1}))
            saveTitle = ['../Figures/Flexible_TimevAngle' '_Test' num2str(j)];
            saveMeSomeFigs(shouldSaveFigures, saveTitle)

            %% plot time versus tip delfection
            figure(j+10)
            actualtip = plot((time(timeStart:end)-time(timeStart))/1000,tipDeflection(timeStart:end)-tipDeflection(timeStart),'Color',[0 0 0]);
            hold on

            boundsTip = plot(linspace(0,xend,1000),linspace(.05,.05,1000),'Color',[0 1 0]);
            plot(linspace(0,xend,1000),linspace(-.05,-.05,1000),'Color',[0 1 0])
            sec12 = plot([1 1],[-.06 .06],'Color',[0 0 1]);

            xlabel('Time [s]')
            ylabel('Tip Deflection [m]')
            xlim([0 xend])
```

```
        ylim([-.06 .06])
        title(sprintf('Time vs Tip Deflection of Test %i for %s',j,name{l}))
        %% kill plots that don't make the cut
        if time(reach5per) > time(timeStart+1000)
            close(j)
            close(j+10)
        else
            figure(j)
            hold on
            theo = plotTheoretical(thetad,Kp(1),Kd(1),K1(1),K2(1),K3(1),K4(1),3,j,time(timeStart),hubAngle(timeStart));
            legend([actual theo td start sec1],{'Actual','Theoretical','Desired Theta','Within 5\%','1 second'},'interpreter', 'latex')
            set(gca, 'defaulttextinterpreter', 'latex')
            set(gca, 'TickLabelInterpreter', 'latex')
            %% label gainz on graph
            annotation('textbox',...
                [0.675 0.46 .18 0.22],...
                'String',{['$K_1$ = ' num2str(K1(1))] ['$K_2$ = ' num2str(K2(1))] ['$K_3$ = ' num2str(K3(1))] ['$K_4$ = ' num2str(K4(1))] ['$\theta _d$ = ' num2str(thetad)]},...
                'FontSize',10,...
                'EdgeColor',[0 0 0],...
                'LineWidth',.01,...
                'BackgroundColor',[1 1 1],...
                'Color',[0 0 0],'interpreter', 'latex');
            figure(j+10)
            hold on
            theo2 = plotTheoretical(thetad,Kp(1),Kd(1),K1(1),K2(1),K3(1),K4(1),4,j,time(timeStart),hubAngle(timeStart));
            legend([actualtip theo2 boundsTip sec12],{'Actual','Theoretical','Maximum Tip Deflection','1 second'},'interpreter', 'latex')
            set(gca, 'defaulttextinterpreter', 'latex')
            set(gca, 'TickLabelInterpreter', 'latex')
            fprintf('Sick Werking Gainz to get Swole for trial %i are K1 = %1.1f K2 = %1.1f K3 = %1.1f K4 = %1.1f\n',j,K1(1),K2(1),K3(1),K4(1));
            saveTitle = ['../Figures/Flexible_TimevTipDeflection' '_Test' num2str(j)];
            saveMeSomeFigs(shouldSaveFigures, saveTitle)
        end
    else
        fprintf('Redefine upper limits somehow plz for plots for trial %i\n',j)
        close(j)
    end
end

%% Rigid
for j = 1:10
    l = 2;

    %split up data
    time = dataRigid{j}(:,1);
```

```
    hubAngle = dataRigid{j}(:,2);
    tipDeflection = dataRigid{j}(:,3);
    hubAngularVelocity = dataRigid{j}(:,4);
    tipVelocity = dataRigid{j}(:,5);
    positionReference = dataRigid{j}(:,6);
    outputVoltage = dataRigid{j}(:,7);
    K1 = dataRigid{j}(:,8);
    K2 = dataRigid{j}(:,9);
    K3 = dataRigid{j}(:,10);
    K4 = dataRigid{j}(:,11);
    Kp = K1;
    Kd = K3;

    a=0;
    if j ~= 9
        for k = 1:length(hubAngle)-1
            if (abs(hubAngle(k+1) - hubAngle(k)) > 0.002) && a==0
                timeStart = k;
                a = 1;
            end
        end
    else
        for k = 1:length(hubAngle)-101
            if (abs(hubAngle(k+101) - hubAngle(k+100)) > 0.002) && a==0
                timeStart = k+100;
                a = 1;
            end
        end
    end
    %% plot time versus theta
    figure(j+20)
    actual = plot((time(timeStart:timeStart+1000)-time(timeStart))/1000, hubAngle(timeStart:timeStart+1000),'Color',[0 0 0]);
    hold on

    %define thetad
    thetad = 0.3;

    %find when within 5%
    temp = hubAngle(timeStart+100:end);
    reach5per = find(abs(temp) > thetad*0.95)+timeStart+100;
    xend = 0.6;
    if ~isempty(reach5per)
        reach5per = reach5per(1);
        timeTaken = time(reach5per(1)) - time(timeStart);

        %plot lines with info
        start = plot([(time(reach5per)-time(timeStart))/1000 (time(reach5per)-time(timeStart))/1000],[thetad*1.5 -thetad*1.5],'Color',[1 0 0]);
        sec5 = plot([.5 .5],[thetad*1.5 -thetad*1.5],'Color',[0 0 1]);
        td = plot(linspace(0,xend,1000),linspace(thetad,thetad,1000),'Color',[0 1 0]);
        plot(linspace(0,xend,1000),linspace(-thetad,-thetad,1000),'Color',[0 1 0])

```

```
        %label and set limits
        xlabel('Time [s]')
        ylabel('Angle [rads]')
        xlim([0 xend])
        ylim([thetad*-1.5 thetad*1.5])
        title(sprintf('Time vs Angle of Test %i for %s',j,name{l}))
        %% kill plots that don't make the cut
        if time(reach5per) > time(timeStart+500)
            close(j+20)
        else
            figure(j+20)
            hold on
            theo = plotTheoretical(thetad,Kp(1),Kd(1),K1(1),K2(1),K3(1),K4(1),1,j,time(timeStart),hubAngle(timeStart));
            legend([actual theo td start sec5],{'Actual','Theoretical','Desired Theta','Within 5\%','0.5 Seconds'},'interpreter', 'latex')
            set(gca, 'defaulttextinterpreter', 'latex')
            set(gca, 'TickLabelInterpreter', 'latex')
            %% label gainz on graph
            annotation('textbox',...
                [0.675 0.55 .18 0.13],...
                'String',{['$K_p$ = ' num2str(Kp(1))] ['$K_d$ = ' num2str(K3(1))] ['$\theta _d$ = ' num2str(thetad)]},...
                'FontSize',10,...
                'EdgeColor',[0 0 0],...
                'LineWidth',.01,...
                'BackgroundColor',[1 1 1],...
                'Color',[0 0 0],'interpreter', 'latex');
            saveTitle = ['../Figures/Rigid_TimevAngle' '_Test' num2str(j)];
            saveMeSomeFigs(shouldSaveFigures, saveTitle)
        end

    else
        fprintf('Redefine upper limits somehow plz for plots for trial %i\n',j)
        close(j+20)
    end
end
```

Analyze Rigid Arm System, Optimize Gains

```
%% Uses a variable-step Monte Carlo Simulation to determine the optimal
%% theoretical gains for the rigid arm system, and plots the results. Also
%% generates a plot showing the mesh grid used to search for the optimal
%% values, as well as the practical limits on the system when choosing
%% gains. Automatically saves all figures to a 'Figures' directory:
%% '../Figures'.
%%
%%
%% Last Modified: 5/3/2017
%% Date Created: 4/24/2017
%% Author: Nicholas Renninger

clc
close all
```

```
clear

%%%%%%%%%%%%%%%%%%%%%%%%%% Plot Setup %%%%%%%%%%%%%%%%%%%%%%%%%%
set(0, 'defaulttextinterpreter', 'latex');
saveLocation = '../Figures/';
LINEWIDTH = 2;
MARKERSIZE = 9;
FONTSIZE = 24;
colorVecsOrig = [0.294118 0 0.509804; % indigo
                 0.1 0.1 0.1; % orange red
                 1 0.843137 0; % gold
                 0.180392 0.545098 0.341176; % sea green
                 0.662745 0.662745 0.662745]; % dark grey

% de-saturate background colors. saturation must be from 0-1
saturation = 0.5;
colorVecs = colorVecsOrig * saturation;


markers = {'o','*','s','.','x','d','^','+','v','>','<','p','h'};

shouldSaveFigures = true;

%% Setup
Kg = 48.4; %total gear ratio []
Km = 0.0107; %motor constant [V/(rad/sec) or Nm/amp]
Rm = 3.29; %armature resistance [ohms]

Jh = 0.002; %base inertia [Kg*m^2]
Jl = 0.0015; %load inertia of bar [Kg*m^2]
J = Jh + Jl; %total intertia [Kg*m^2]

% choose range of zeta (damping coef.) values to solve for new gains
zeta_min = 0.9;
zeta_max = 1.1;
num_zeta_to_try = 20;

% define limits on proportional and deriv. gains
K_p_min = 0.1;
K_p_max = 20;

K_d_min = 0;
K_d_max = 1.5;
num_gains_to_try = 250;

% desired pointing angle
theta_des = 0.7; %[rad]

% define range of zeta (damping coef.) values to try
zeta_vec = linspace(zeta_min, zeta_max, num_zeta_to_try);

% define range of K_p values to try for each zeta value
K_p_vec = linspace(K_p_min, K_p_max, num_gains_to_try);

% define limits on time and voltage
```

```
TIME_LIMIT = 0.5; % [s]
V_LIMIT = 5; % [V]
numTimeSteps = 1000;
tspan = linspace(0, 1, numTimeSteps + 1)'; % [s]

% define high and low voltages
error = 0.05;
minV = 1;
maxV = 5;


%% Define New Closed loop system with Varied Gains

% initialize
[acceptable.zeta, acceptable.K_p, ...
 acceptable.K_d,  acceptable.timeToSettle, ...
 acceptable.lowKP, acceptable.lowKD, ...
 acceptable.highKP, acceptable.highKD] = deal(zeros(1,1));

for j = 1:num_zeta_to_try

    curr_zeta = zeta_vec(j);

    for i = 1:num_gains_to_try

        curr_K_p = K_p_vec(i);

        % solve for new K_d
        num = 2* curr_zeta * sqrt(curr_K_p * Kg * Km * J * Rm) - ...
              (Kg^2 * Km^2);

        denom = Kg * Km;
        curr_K_d = num / denom;


        %%% Compute step response of system with new gains
        n1 = (curr_K_p*Kg*Km)/(J*Rm);
        d2 = 1;
        d1 = (curr_K_d*Kg*Km+(Kg^2)*(Km^2))/(J*Rm);
        d0 = (curr_K_p*Kg*Km)/(J*Rm);

        %%% define closed loop transfer fcn
        num = n1;
        den = [d2 d1 d0];
        sysTF = tf(num,den);

        [x, t] = step(sysTF, tspan);
        theta = 2*theta_des*x;

        %%% If system stays within spec, save values of zeta, K_d, and K_p
        Vin = curr_K_p .* (theta_des - theta(1:end-1)) + ...
              curr_K_d.*-1*(diff(theta)./diff(t));
        within5timesRigid = find(abs(2*theta_des*x-theta_des) < 2*theta_des*.05);
        greaterThanVLimit = find(abs(Vin) > V_LIMIT);
```

```matlab

        % save acceptable values
        if  max(abs(Vin)) < (minV + error)
            acceptable.lowKP = [acceptable.lowKP curr_K_p];
            acceptable.lowKD = [acceptable.lowKD curr_K_d];
        elseif max(abs(Vin)) > (maxV - error) && max(abs(Vin)) < (maxV + error)
            acceptable.highKP = [acceptable.highKP curr_K_p];
            acceptable.highKD = [acceptable.highKD curr_K_d];
        end

        % if the voltage is below the limit, and re
        if isempty(greaterThanVLimit) && ~isempty(within5timesRigid)
            timeReachRigid = t(within5timesRigid(1));

            if timeReachRigid < TIME_LIMIT

                acceptable.zeta = [acceptable.zeta, curr_zeta];
                acceptable.K_d = [acceptable.K_d, curr_K_d];
                acceptable.K_p = [acceptable.K_p , curr_K_p];
                acceptable.timeToSettle = [acceptable.timeToSettle, timeReachRigid];

            end

        end

    end
end

% get rid of blank at beginning of each array
acceptable.zeta(1) = [];
acceptable.K_p(1) = [];
acceptable.K_d(1) = [];
acceptable.timeToSettle(1) = [];

[acceptable.timeToSettle, sortedIDX] = sortrows(acceptable.timeToSettle');

% pull out the gains that led to the fastest time to settle
acceptable.K_p = acceptable.K_p(sortedIDX);
acceptable.K_d = acceptable.K_d(sortedIDX);
acceptable.zeta = acceptable.zeta(sortedIDX);

% fastest time to settle
K_d_fast = acceptable.K_d(1);
K_p_fast = acceptable.K_p(1);

% slowest time to settle
K_d_slow = acceptable.K_d(end);
K_p_slow = acceptable.K_p(end);

fprintf('fast K_d: %0.3g, slow K_d: %0.3g\n', K_d_fast, K_d_slow)
fprintf('fast K_p: %0.3g, slow K_p: %0.3g\n', K_p_fast, K_p_slow)
fprintf('zeta for fast gain values: %0.3g\n', acceptable.zeta(1))

```

```
%% plot fastest gains model - theta vs time

%%% Compute step response of system with new gains
n1 = (K_p_fast*Kg*Km)/(J*Rm);
d2 = 1;
d1 = (K_d_fast*Kg*Km+(Kg^2)*(Km^2))/(J*Rm);
d0 = (K_p_fast*Kg*Km)/(J*Rm);

%%% define closed loop transfer fcn
num = n1;
den = [d2 d1 d0];
sysTF = tf(num,den);

[x, t] = step(sysTF, tspan);
theta = 2*theta_des*x;

% plot where the step should end up at
t_vec = linspace(0, max(t), 100);
max_step_vec = ones(1, 100) * theta_des * 2;


titleString = sprintf('Rigid Arm - Theta vs Time');

% setup plot saving
saveTitle = cat(2, saveLocation, sprintf('%s.pdf', titleString));

hFig = figure('name', titleString);
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)


plot(t_vec, max_step_vec, '--', 'linewidth', LINEWIDTH)
hold on
plot(t, theta, 'linewidth', LINEWIDTH)


colormap('linspecer')
xlabel('t [sec.]')
xlim([0 0.5])
ylabel('$\theta$ [rad.]')
title('Rigid Arm Angle vs. Time')
within5timesRigid = find(abs(2*theta_des*x-theta_des) < 2*theta_des*.05);
timeReachRigid = t(within5timesRigid(1));
fprintf('Time rigid reaches within 5%%: %0.3g sec. \n',timeReachRigid)
legend({sprintf('$\\theta_{desired}$ = %0.3g rad', theta_des*2), ...
       'Step Response'}, 'interpreter', 'latex', ...
       'location', 'best')

set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')

h = gca;
leg = h.Legend;
titleStruct = h.Title;
```

```
set(titleStruct, 'FontWeight', 'bold')
set(gca, 'FontSize', FONTSIZE)
set(leg, 'FontSize', round(FONTSIZE * 0.7))

grid on

% setup and save figure as .pdf
saveMeSomeFigs(shouldSaveFigures, saveTitle)


%% voltage plot

% plot where the step should end up at
t_vec = linspace(0, max(t), 100);
max_volt_vec = ones(1, 100) * V_LIMIT;

titleString = sprintf('Rigid Arm - Voltage vs Time');

% setup plot saving
saveTitle = cat(2, saveLocation, sprintf('%s.pdf', titleString));

hFig = figure('name', titleString);
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)

colors = linspecer(2);

hold on
plot(t_vec, -max_volt_vec, '--', 'linewidth', LINEWIDTH, ...
     'color', colors(1, :));
p1 = plot(t_vec, max_volt_vec, '--', 'linewidth', LINEWIDTH, ......
     'color', colors(1, :));
Vin = K_p_fast .* (theta_des - theta(1:end-1)) + ...
      K_d_fast .* -1 * (diff(theta) ./ diff(t));
p2 = plot(t(1:end-1), Vin, 'linewidth', LINEWIDTH, ...
     'color', colors(2, :));


xlabel('t [sec.]')
xlim([0 0.5])
ylim([-V_LIMIT*1.05, V_LIMIT*1.05])
ylabel('$Voltage$ [volts]')
title('Rigid Arm Voltage vs. Time')
within5timesRigid = find(abs(2*theta_des*x-theta_des) < 2*theta_des*.05);
timeReachRigid = t(within5timesRigid(1));
fprintf('Max Voltage for rigid: +%0.3gV and -%0.3gV \n', ...
        abs(max(Vin)), abs(min(Vin)))

leg_str = {sprintf('$| V_{max} |$ = %0.3g V', V_LIMIT), ...
           'Control Motor Voltage'};
legend([p1 p2], leg_str, 'interpreter', 'latex', ...
       'location', 'best')

set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
```

```matlab

h = gca;
leg = h.Legend;
titleStruct = h.Title;
set(titleStruct, 'FontWeight', 'bold')
set(gca, 'FontSize', FONTSIZE)
set(leg, 'FontSize', round(FONTSIZE * 0.7))

grid on


% setup and save figure as .pdf
saveMeSomeFigs(shouldSaveFigures, saveTitle)

%% plot all of the acceptable values and format

K_d = acceptable.K_d;
K_p = acceptable.K_p;

zeta = 1;

%%% get K_p and K_d for zeta = 1
for i = 1:num_gains_to_try
    curr_K_p = K_p_vec(i);

    % solve for new K_d
    num = 2* zeta * sqrt(curr_K_p * Kg * Km * J * Rm) - ...
          (Kg^2 * Km^2);

    denom = Kg * Km;
    K_d_zetaOne(i) = num / denom;
end


%%% Ploting
titleString = sprintf('Rigid Arm Gain Determination');

% setup plot saving
saveTitle = cat(2, saveLocation, sprintf('%s.pdf', titleString));

hFig = figure('name', titleString);
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)
colors = linspecer(5);

hold on

p1 = plot(K_p, K_d, '.', 'markersize', MARKERSIZE, 'color', colors(1,:));
hold on
p2 = plot(acceptable.highKP, acceptable.highKD, 'bo', ...
          'markersize', MARKERSIZE, 'color', colors(2, :), ...
          'linewidth', LINEWIDTH * 0.6);
p3 = plot(acceptable.lowKP, acceptable.lowKD, 'rs', ...
          'markersize', MARKERSIZE, 'color', colors(3, :), ...
          'linewidth', LINEWIDTH * 0.6);
```

```
p4 = plot(K_p_fast, K_d_fast, 'pg', 'markersize', MARKERSIZE*1.5, ...
          'color', colors(4, :), 'linewidth', LINEWIDTH*0.8);
p5 = plot(K_p_vec, K_d_zetaOne, 'linewidth', LINEWIDTH * 1.6, ...
          'color', colors(5, :));

xlim([min(K_p), max(K_p)])
ylim([min(K_d), max(K_d)])


xlabel('$K_P$')
ylabel('$K_D$')
title('Rigid Arm Gain Computations')

leg_str = {'Gain Values with $t_{settling, \, 5\%}$ = 0.5s and $|V_{max} \leq 5V|$', ...
           'Gain Values Corresponding to $|V_{max} = 5V|$', ...
           'Gain Values Corresponding to motor deadband: $|V_{max} < 1V|$', ...
           sprintf(['Optimal Gain Values (underdamped): ', ...
                    '$K_P = %0.3g$, $K_D = %0.3g$'], K_p_fast, K_d_fast), ...
           'Gain Values for $\zeta = 1$'};
legend([p1 p2 p3 p4 p5], leg_str, 'interpreter', 'latex', ...
       'location', 'best')

set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')

h = gca;
leg = h.Legend;
titleStruct = h.Title;
set(titleStruct, 'FontWeight', 'bold')
set(gca, 'FontSize', FONTSIZE)
set(leg, 'FontSize', round(FONTSIZE * 0.7))

grid on


% setup and save figure as .pdf
saveMeSomeFigs(shouldSaveFigures, saveTitle)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     Luke Tafur                                          %
%     Robot Control Arm                                   %
%     Date Created: 4.18.2017                             %
%     Date Modified: 4.20.2017                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;tic
clear all; close all;
%% set parameters
%set rules
maxDeflect = 5;
maxTimeRigid = 0.5;
maxTimeFlex = 1;
maxVoltRigid = 5;
maxVoltFlex = 5;

```

```
%desired angle
thetad = .3;

%rigid arm
Kp = 14;
Kd = 0.6;

%flexible arm
%working-ish values
%[6 -6.4 1.3 1.5] too much tip deflect tho
%
%6.8 -6.4 1.4 1.1
%
%[4.2 -15 1.2 0.1] 0.999 [s] 5.06 [cm]
%[3.9 -15 1.1 0.1] 0.993 [s] 4.89 [cm]
%[3.6 -15 0.9 0.1] 0.860 [s] 4.88 [cm]
%[3.0 -15 0.6 0.1] 0.761 [s] 4.67 [cm]
%[2.5 -15 0.4 0.1] 0.750 [s] 4.36 [cm]
%[2.0 -15 0.2 0.1] 0.752 [s] 4.01 [cm]
%[1.8 -15 0.0 0.0] 0.678 [s] 4.09 [cm] NOT SURE WHY 0 DER CONTROL IS
%[1.2 -15 0.0 0.0] 0.919 [s] 2.79 [cm] DOING WELL. DID THEY LIE TO US??
K1 = 6.8; %Kptheta *0 - 10*
K2 = -6.4; %Kpd *-15 - 0*
K3 = 1.4; %KDtheta *0 - 1.5*
K4 = 1.1; %KDd *0 -1.5*

Kg = 48.4; %total gear ratio []
Km = 0.0107; %motor constant [V/(rad/sec) or Nm/amp]
Rm = 3.29; %armature resistance [ohms]

Jh = 0.002; %base inertia [Kg*m^2]
Jl = 0.0015; %load inertia of bar [Kg*m^2]
J = Jh+Jl; %total intertia [Kg*m^2]
L = 0.45; %link length [m]

%closed loop transfer function for rigid
n1 = (Kp*Kg*Km)/(J*Rm); %numerator constant term
d2 = 1; %denominator s^2 term
d1 = (Kd*Kg*Km+(Kg^2)*(Km^2))/(J*Rm); %denominator s term
d0 = (Kp*Kg*Km)/(J*Rm);%denominator constant term

Marm = 0.06; %link mass of the ruler [Kg]
Ja = (Marm*L^2)/3; %link rigid body inertia [Kg*m^2]
Mtip = 0.05; %tip mass [Kg]
Jm = Mtip*L^2; %tip mass inertia [Kg*m^2]
Jl = Ja + Jm; %total inertia [Kg*m^2]
Fc = 1.8; %natural frequency [Hz]
Karm = ((2*pi*Fc)^2)*(Jl); %flexible link stiffness

%constants from lab doc pg. 6
r1 = (Kg*Km)/(Jh*Rm);
r2 = (-1*Kg*Km*L)/(Jh*Rm);
p1 = (-1*(Kg^2)*(Km^2))/(Jh*Rm);
p2 = ((Kg^2)*(Km^2)*L)/(Jh*Rm);
q1 = Karm/(L*Jh);
```

```matlab
q2 = (-1*Karm*(Jh+Jl))/(Jl*Jh);

%constants from lab doc pg. 7
lambda3 = -1*p1+K3*r1+K4*r2;
lambda2 = -1*q2+K1*r1+K2*r2+K4*(p2*r1-r2*p1);
lambda1 = p1*q2-q1*p2+K3*(q1*r2-r1*q2)+K2*(p2*r1-r2*p1);
lambda0 = K1*(q1*r2-r1*q2);

tspan = (0:0.001:1.5)';

%% Closed loop system Rigid
num = n1;
den = [d2 d1 d0];
sysTF = tf(num,den)

%% Step response
% thetad = desired arm angle (scalar)
[x,t] = step(sysTF,tspan);
theta= 2*thetad*x;
figure(1);
clf;
plot(t,theta)
xlabel('Time [Seconds]')
xlim([0 0.5])
ylabel('Angle [Radians]')
title('Theoretical Rigid Arm Angle v Time')
within5timesRigid = find(abs(2*thetad*x-2*thetad)<2*thetad*.05);
timeReachRigid = t(within5timesRigid(1));
fprintf('Time rigid reaches within 5%% is %f \n',timeReachRigid)

%voltage
figure(2);
Vin = Kp.*(thetad - theta(1:end-1))+Kd.*-1*(diff(theta)./diff(t));
plot(t(1:end-1),Vin)
xlabel('Time [Seconds]')
xlim([0 0.5])
ylabel('Voltage [Volts]')
title('Theoretical Rigid Arm Voltage v Time')
fprintf('Max Voltage for rigid is +%f and -%f \n',abs(max(Vin)),abs(min(Vin)))

%% Flexible
%angle
num = [K1*r1 0 K1*(q1*r2-r1*q2)];
den = [1 lambda3 lambda2 lambda1 lambda0];
sysTFflexAngle = tf(num,den)

[x2,t2] = step(sysTFflexAngle,tspan);
theta2= 2*thetad*x2;
figure(3);
clf;
plot(t2,theta2)
xlabel('Time [Seconds]')
ylabel('Angle [Radians]')
title('Theoretical Flexible Arm Angle v Time')
within5timesFlex = find(abs(2*thetad*x2-2*thetad)<2*thetad*.05);
```

```
timeReachFlex = t2(within5timesFlex(1));
fprintf('Time flexible reaches within 5%% is %f \n',timeReachFlex)

%tip deflection
num = [K1*r2 K1*(p2*r1-r2*p1) 0];
den = [1 lambda3 lambda2 lambda1 lambda0];
sysTFflexDisplacement = tf(num,den)

[x3,t3] = step(sysTFflexDisplacement,tspan);
figure(4);
clf;
plot(t3,x3)
xlabel('Time [Seconds]')
ylabel('Displacement [Meters]')
title('Theoretical Flexible Arm Tip Displacement v Time')
fprintf('Max tip deflection is +%f [cm] and -%f [cm]\n',abs(max(x3)*100),abs(min(x3)*100))

%voltage
figure(5);
Vin2 = K1*(thetad - theta2(1:end-1))-K3*(diff(theta2)./diff(t2))-K2*x3(1:end-1)-K4*(diff(x3)./diff(t3));
plot(t2(1:end-1),Vin2)
xlabel('Time [Seconds]')
ylabel('Voltage [Volts]')
title('Theoretical Flexible Arm Voltage v Time')
fprintf('Max Voltage for flexible is +%f and -%f \n\n',abs(max(Vin2)),abs(min(Vin2)))

%%check criteria
if timeReachRigid < maxTimeRigid
    cprintf('green','Passes rigid time test!\n')
    check1 = 1;
else
    cprintf('red','Fails rigid time test!\n')
    check1 = 0;
end

if abs(max(Vin)) > maxVoltRigid || abs(min(Vin)) > maxVoltRigid
    cprintf('red','Fails rigid voltage test!\n')
    check2 = 0;
else
    cprintf('green','Passes rigid voltage test!\n')
    check2 = 1;
end

if timeReachFlex < maxTimeFlex
    cprintf('green','Passes flex time test!\n')
    check3 = 1;
else
    cprintf('red','Fails flex time test!\n')
    check3 = 0;
end

if abs(max(x3(1000:1500))*100) > maxDeflect || abs(min(x3(1000:1500))*100) >
```

```
maxDeflect
    cprintf('red','Fails tip deflection test!\n')
    check4 = 0;
else
    cprintf('green','Passes tip deflection test!\n')
    check4 = 1;
end

if abs(max(Vin2)) > maxVoltFlex || abs(min(Vin2)) > maxVoltFlex
    cprintf('red','Fails flex voltage test!\n')
    check5 = 0;
else
    cprintf('green','Passes flex voltage test!\n')
    check5 = 1;
end

if check1 && check2
    fprintf('\nWorking model for rigid with values Kp:%f and Kd:%f\n',Kp,Kd)
else
    fprintf('\nBad rigid model\n')
end

if check3 && check4 && check5
    fprintf('Working model for flexible with values K1:%f, K2:%f, K3:%f, and K4:%f\n\n',K1,K2,K3,K4)
else
    fprintf('Bad flexible model\n\n')
end

toc
```

Plot Theoretical

```
% Lab 6: Robot Arm
% Author: Luke Tafur
% Date Created: 4.13.17
% Date Edited: 5.2.17

function h = plotTheoretical(thetad,Kp,Kd,K1,K2,K3,K4,plotNum,j,timeStart,angleStart)
%% set constantst
Kg = 48.4; %total gear ratio []
Km = 0.0107; %motor constant [V/(rad/sec) or Nm/amp]
Rm = 3.29; %armature resistance [ohms]

Jh = 0.002; %base inertia [Kg*m^2]
Jl = 0.0015; %load inertia of bar [Kg*m^2]
J = Jh+Jl; %total intertia [Kg*m^2]
L = 0.45; %link length [m]

%closed loop transfer function for rigid
n1 = (Kp*Kg*Km)/(J*Rm); %numerator constant term
d2 = 1; %denominator s^2 term
d1 = (Kd*Kg*Km+(Kg^2)*(Km^2))/(J*Rm); %denominator s term
d0 = (Kp*Kg*Km)/(J*Rm);%denominator constant term

```

```
Marm = 0.06; %link mass of the ruler [Kg]
Ja = (Marm*L^2)/3; %link rigid body inertia [Kg*m^2]
Mtip = 0.05; %tip mass [Kg]
Jm = Mtip*L^2; %tip mass inertia [Kg*m^2]
Jl = Ja + Jm; %total inertia [Kg*m^2]
Fc = 1.8; %natural frequency [Hz]
Karm = ((2*pi*Fc)^2)*(Jl); %flexible link stiffness

%constants from lab doc pg. 6
r1 = (Kg*Km)/(Jh*Rm);
r2 = (-1*Kg*Km*L)/(Jh*Rm);
p1 = (-1*(Kg^2)*(Km^2))/(Jh*Rm);
p2 = ((Kg^2)*(Km^2)*L)/(Jh*Rm);
q1 = Karm/(L*Jh);
q2 = (-1*Karm*(Jh+Jl))/(Jl*Jh);

%constants from lab doc pg. 7
lambda3 = -1*p1+K3*r1+K4*r2;
lambda2 = -1*q2+K1*r1+K2*r2+K4*(p2*r1-r2*p1);
lambda1 = p1*q2-q1*p2+K3*(q1*r2-r1*q2)+K2*(p2*r1-r2*p1);
lambda0 = K1*(q1*r2-r1*q2);

tspan = (0:0.001:3)';

if plotNum == 1
    %% Closed loop system Rigid
    num = n1;
    den = [d2 d1 d0];
    sysTF = tf(num,den);

    %% Step response
    % thetad = desired arm angle (scalar)
    [x,t] = step(sysTF,tspan);
    theta= 2*thetad*x;
    if j == 1 || j == 2 || j ==9
        angle = angleStart - theta;
    else
        angle = angleStart + theta;
    end
    h = plot(t,angle,'Color',[1 0 1]);
    % xlabel('Time [Seconds]')
    % xlim([0 0.5])
    % ylabel('Angle [Radians]')
    % title('Theoretical Rigid Arm Angle v Time')
    % within5timesRigid = find(abs(2*thetad*x-2*thetad)<2*thetad*.05);
    % timeReachRigid = t(within5timesRigid(1));
    % fprintf('Time rigid reaches within 5%% is %f \n',timeReachRigid)

elseif plotNum == 2
    %voltage
    Vin = Kp.*(thetad - theta(1:end-1))+Kd.*-1*(diff(theta)./diff(t));
    plot(t(1:end-1),Vin)
    xlabel('Time [Seconds]')
    xlim([0 0.5])
    ylabel('Voltage [Volts]')
```

```
    title('Theoretical Rigid Arm Voltage v Time')
    fprintf('Max Voltage for rigid is +%f and -%f \n',abs(max(Vin)),abs(min(Vin)))

elseif plotNum == 3
    %% Flexible
    %angle
    num = [K1*r1 0 K1*(q1*r2-r1*q2)];
    den = [1 lambda3 lambda2 lambda1 lambda0];
    sysTFflexAngle = tf(num,den);

    [x2,t2] = step(sysTFflexAngle,tspan);
    theta2= 2*thetad*x2;
    h = plot(t2,angleStart - theta2,'Color',[1 0 1]);
    % xlabel('Time [Seconds]')
    % ylabel('Angle [Radians]')
    % title('Theoretical Flexible Arm Angle v Time')
    % within5timesFlex = find(abs(2*thetad*x2-2*thetad)<2*thetad*.05);
    % timeReachFlex = t2(within5timesFlex(1));
    % fprintf('Time flexible reaches within 5%% is %f \n',timeReachFlex)

elseif plotNum == 4
    %tip deflection
    num = [K1*r2 K1*(p2*r1-r2*p1) 0];
    den = [1 lambda3 lambda2 lambda1 lambda0];
    sysTFflexDisplacement = tf(num,den);

    [x3,t3] = step(sysTFflexDisplacement,tspan);
    h = plot(t3,x3,'Color',[1 0 1]);
    % xlabel('Time [Seconds]')
    % ylabel('Displacement [Meters]')
    % title('Theoretical Flexible Arm Tip Displacement v Time')
    % fprintf('Max tip deflection is +%f [cm] and -%f [cm]\n',abs(max(x3)*100),abs(min(x3)*100))

elseif plotNum == 5
    %voltage
    Vin2 = K1*(thetad - theta2(1:end-1))-K3*(diff(theta2)./diff(t2))-K2*x3(1:end-1)-K4*(diff(x3)./diff(t3));
    plot(t2(1:end-1),Vin2)
    xlabel('Time [Seconds]')
    ylabel('Voltage [Volts]')
    title('Theoretical Flexible Arm Voltage v Time')
    fprintf('Max Voltage for flexible is +%f and -%f \n\n',abs(max(Vin2)),abs(min(Vin2)))
end
end
```

Helper Functions

```
function saveMeSomeFigs(shouldSaveFigures, saveTitle)

    %% saveMeSomeFigs(shouldSaveFigures, saveTitle)
    %%
    %% Takes a boolean toggle (shouldSaveFigures) and a string with the
    %% save name (saveTitle), including the path, that you want to save
```

```
    %%% the current figure as. Example saveTitle looks like the following:
    %%%
    %%%     saveTitle = '../Figures/Velocity vs Time.pdf' - this would save
    %%%     a figure named 'Velocity vs Time.pdf' (as a .pdf) to a folder
    %%%     called 'Figures' up one directoy from the code's working
    %%%     directory.
    %%%
    %%% Uses the gca function to pull the current figure, normalize
    %%% and scale it to the deafult paper size, and save it as a .pdf.
    %%%
    %%%
    %%% Examples function call:
    %%%
    %%% x = linspace(0, 2*pi, 100);
    %%% y = sin(x);
    %%% plot(x, y)
    %%% title('saveMeSomeFigs Test')
    %%%
    %%% saveTitle = 'saveMeSomeFigs Test Plot.pdf';
    %%% shouldSaveFigures = true;
    %%% saveMeSomeFigs(shouldSaveFigures, saveTitle)
    %%%
    %%% Last Modified: 5/3/2017
    %%% Date Created: 2/10/2017
    %%% Author: Nicholas Renninger

    % setup and save figure as .pdf
    if shouldSaveFigures
        curr_fig = gcf;
        set(curr_fig, 'PaperOrientation', 'landscape');
        set(curr_fig, 'PaperUnits', 'normalized');
        set(curr_fig, 'PaperPosition', [0 0 1 1]);
        [fid, errmsg] = fopen(saveTitle, 'w+');

        if fid < 1 % check if file is already open.
            error('Error Opening File in fopen: \n%s', errmsg);
        end

        fclose(fid);
        print(gcf, '-dpdf', saveTitle);
    end

end
```

```
% function lineStyles = linspecer(N)
% This function creates an Nx3 array of N [R B G] colors
% These can be used to plot lots of lines with distinguishable and nice
% looking colors.
%
% lineStyles = linspecer(N);  makes N colors for you to use: lineStyles(ii,:)
%
% colormap(linspecer); set your colormap to have easily distinguishable
%                      colors and a pleasing aesthetic
%
% lineStyles = linspecer(N,'qualitative'); forces the colors to all be
```

```
distinguishable (up to 12)
% lineStyles = linspecer(N,'sequential'); forces the colors to vary along a spectrum
%
% % Examples demonstrating the colors.
%
% LINE COLORS
% N=6;
% X = linspace(0,pi*3,1000);
% Y = bsxfun(@(x,n)sin(x+2*n*pi/N), X.', 1:N);
% C = linspecer(N);
% axes('NextPlot','replacechildren', 'ColorOrder',C);
% plot(X,Y,'linewidth',5)
% ylim([-1.1 1.1]);
%
% SIMPLER LINE COLOR EXAMPLE
% N = 6; X = linspace(0,pi*3,1000);
% C = linspecer(N)
% hold off;
% for ii=1:N
%      Y = sin(X+2*ii*pi/N);
%      plot(X,Y,'color',C(ii,:),'linewidth',3);
%      hold on;
% end
%
% COLORMAP EXAMPLE
% A = rand(15);
% figure; imagesc(A); % default colormap
% figure; imagesc(A); colormap(linspecer); % linspecer colormap
%
%   See also NDHIST, NHIST, PLOT, COLORMAP, 43700-cubehelix-colormaps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% by Jonathan Lansey, March 2009-2013 ? Lansey at gmail.com                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% credits and where the function came from
% The colors are largely taken from:
% http://colorbrewer2.org and Cynthia Brewer, Mark Harrower and The Pennsylvania State University
%
%
% She studied this from a phsychometric perspective and crafted the colors
% beautifully.
%
% I made choices from the many there to decide the nicest once for plotting
% lines in Matlab. I also made a small change to one of the colors I
% thought was a bit too bright. In addition some interpolation is going on
% for the sequential line styles.
%
%
%%

function lineStyles=linspecer(N,varargin)

if nargin==0 % return a colormap
```

```
    lineStyles = linspecer(128);
    return;
end

if ischar(N)
    lineStyles = linspecer(128,N);
    return;
end

if N<=0 % its empty, nothing else to do here
    lineStyles=[];
    return;
end

% interperet varagin
qualFlag = 0;
colorblindFlag = 0;

if ~isempty(varargin)>0 % you set a parameter?
    switch lower(varargin{1})
        case {'qualitative','qua'}
            if N>12 % go home, you just can't get this.
                warning('qualitiative is not possible for greater than 12 items, please reconsider');
            else
                if N>9
                    warning(['Default may be nicer for ' num2str(N) ' for clearer colors use: whitebg(''black''); ']);
                end
            end
            qualFlag = 1;
        case {'sequential','seq'}
            lineStyles = colorm(N);
            return;
        case {'white','whitefade'}
            lineStyles = whiteFade(N);return;
        case 'red'
            lineStyles = whiteFade(N,'red');return;
        case 'blue'
            lineStyles = whiteFade(N,'blue');return;
        case 'green'
            lineStyles = whiteFade(N,'green');return;
        case {'gray','grey'}
            lineStyles = whiteFade(N,'gray');return;
        case {'colorblind'}
            colorblindFlag = 1;
        otherwise
            warning(['parameter ''' varargin{1} ''' not recognized']);
    end
end
% *.95
% predefine some colormaps
set3 = colorBrew2mat({[141, 211, 199];[ 255, 237, 111];[ 190, 186, 218];[ 251, 128, 114];[ 128, 177, 211];[ 253, 180, 98];[ 179, 222, 105];[ 188, 128, 189];[ 217, 217, 217];[ 204, 235, 197];[ 252, 205, 229];[ 255, 255,
```

```
    179]}');
set1JL = brighten(colorBrew2mat({[228, 26, 28];[ 55, 126, 184]; [ 77, 175,
    74];[ 255, 127, 0];[ 255, 237, 111]*.85;[ 166, 86, 40];[ 247, 129, 191];[
    153, 153, 153];[ 152, 78, 163]}'));
set1 = brighten(colorBrew2mat({[ 55, 126, 184]*.85;[228, 26, 28];[ 77, 175,
    74];[ 255, 127, 0];[ 152, 78, 163]}),.8);

% colorblindSet = {[215,25,28];[253,174,97];[171,217,233];[44,123,182]};
colorblindSet = {[215,25,28];[253,174,97];[171,217,233]*.8;[44,123,182]*.8};

set3 = dim(set3,.93);

if colorblindFlag
    switch N
        %     sorry about this line folks. kind of legacy here because I used
            to
        %     use individual 1x3 cells instead of nx3 arrays
        case 4
            lineStyles = colorBrew2mat(colorblindSet);
        otherwise
            colorblindFlag = false;
            warning('sorry unsupported colorblind set for this number, using
                regular types');
    end
end
if ~colorblindFlag
    switch N
        case 1
            lineStyles = { [  55, 126, 184]/255};
        case {2, 3, 4, 5 }
            lineStyles = set1(1:N);
        case {6 , 7, 8, 9}
            lineStyles = set1JL(1:N)';
        case {10, 11, 12}
            if qualFlag % force qualitative graphs
                lineStyles = set3(1:N)';
            else % 10 is a good number to start with the sequential ones.
                lineStyles = cmap2linspecer(colorm(N));
            end
        otherwise % any old case where I need a quick job done.
            lineStyles = cmap2linspecer(colorm(N));
    end
end
lineStyles = cell2mat(lineStyles);

end

% extra functions
function varIn = colorBrew2mat(varIn)
for ii=1:length(varIn) % just divide by 255
    varIn{ii}=varIn{ii}/255;
end
end

function varIn = brighten(varIn,varargin) % increase the brightness
```

```
if isempty(varargin),
    frac = .9;
else
    frac = varargin{1};
end

for ii=1:length(varIn)
    varIn{ii}=varIn{ii}*frac+(1-frac);
end
end

function varIn = dim(varIn,f)
    for ii=1:length(varIn)
        varIn{ii} = f*varIn{ii};
    end
end

function vOut = cmap2linspecer(vIn) % changes the format from a double array
    to a cell array with the right format
vOut = cell(size(vIn,1),1);
for ii=1:size(vIn,1)
    vOut{ii} = vIn(ii,:);
end
end
%%
% colorm returns a colormap which is really good for creating informative
% heatmap style figures.
% No particular color stands out and it doesn't do too badly for colorblind
    people either.
% It works by interpolating the data from the
% 'spectral' setting on http://colorbrewer2.org/ set to 11 colors
% It is modified a little to make the brightest yellow a little less bright.
function cmap = colorm(varargin)
n = 100;
if ~isempty(varargin)
    n = varargin{1};
end

if n==1
    cmap =  [0.2005    0.5593    0.7380];
    return;
end
if n==2
     cmap =  [0.2005    0.5593    0.7380;
              0.9684    0.4799    0.2723];
            return;
end

frac=.95; % Slight modification from colorbrewer here to make the yellows in
    the center just a bit darker
cmapp = [158, 1, 66; 213, 62, 79; 244, 109, 67; 253, 174, 97; 254, 224, 139;
    255*frac, 255*frac, 191*frac; 230, 245, 152; 171, 221, 164; 102, 194, 165;
     50, 136, 189; 94, 79, 162];
x = linspace(1,n,size(cmapp,1));
```

```
xi = 1:n;
cmap = zeros(n,3);
for ii=1:3
    cmap(:,ii) = pchip(x,cmapp(:,ii),xi);
end
cmap = flipud(cmap/255);
end

function cmap = whiteFade(varargin)
n = 100;
if nargin>0
    n = varargin{1};
end

thisColor = 'blue';

if nargin>1
    thisColor = varargin{2};
end
switch thisColor
    case {'gray','grey'}
        cmapp =
            [255,255,255;240,240,240;217,217,217;189,189,189;150,150,150;115,115,115;82,82

    case 'green'
        cmapp =
            [247,252,245;229,245,224;199,233,192;161,217,155;116,196,118;65,171,93;35,139,6

    case 'blue'
        cmapp =
            [247,251,255;222,235,247;198,219,239;158,202,225;107,174,214;66,146,198;33,113

    case 'red'
        cmapp =
            [255,245,240;254,224,210;252,187,161;252,146,114;251,106,74;239,59,44;203,24,29

    otherwise
        warning(['sorry your color argument ' thisColor ' was not recognized'
            ]);
end

cmap = interpomap(n,cmapp);
end

% Eat a approximate colormap, then interpolate the rest of it up.
function cmap = interpomap(n,cmapp)
    x = linspace(1,n,size(cmapp,1));
    xi = 1:n;
    cmap = zeros(n,3);
    for ii=1:3
        cmap(:,ii) = pchip(x,cmapp(:,ii),xi);
    end
    cmap = (cmap/255); % flipud??
end
```

```
function count = cprintf(style,format,varargin)
% CPRINTF displays styled formatted text in the Command Window
%
% Syntax:
%    count = cprintf(style,format,...)
%
% Description:
%    CPRINTF processes the specified text using the exact same FORMAT
%    arguments accepted by the built-in SPRINTF and FPRINTF functions.
%
%    CPRINTF then displays the text in the Command Window using the
%    specified STYLE argument. The accepted styles are those used for
%    Matlab's syntax highlighting (see: File / Preferences / Colors /
%    M-file Syntax Highlighting Colors), and also user-defined colors.
%
%    The possible pre-defined STYLE names are:
%
%       'Text'                 - default: black
%       'Keywords'             - default: blue
%       'Comments'             - default: green
%       'Strings'              - default: purple
%       'UnterminatedStrings'  - default: dark red
%       'SystemCommands'       - default: orange
%       'Errors'               - default: light red
%       'Hyperlinks'           - default: underlined blue
%
%       'Black','Cyan','Magenta','Blue','Green','Red','Yellow','White'
%
%    STYLE beginning with '-' or '_' will be underlined. For example:
%          '-Blue' is underlined blue, like 'Hyperlinks';
%          '_Comments' is underlined green etc.
%
%    STYLE beginning with '*' will be bold (R2011b+ only). For example:
%          '*Blue' is bold blue;
%          '*Comments' is bold green etc.
%    Note: Matlab does not currently support both bold and underline,
%          only one of them can be used in a single cprintf command. But of
%          course bold and underline can be mixed by using separate commands.
%
%    STYLE also accepts a regular Matlab RGB vector, that can be underlined
%    and bolded: -[0,1,1] means underlined cyan, '*[1,0,0]' is bold red.
%
%    STYLE is case-insensitive and accepts unique partial strings just
%    like handle property names.
%
%    CPRINTF by itself, without any input parameters, displays a demo
%
% Example:
%    cprintf;   % displays the demo
%    cprintf('text',   'regular black text');
%    cprintf('hyper',  'followed %s','by');
%    cprintf('key',    '%d colored', 4);
%    cprintf('-comment','& underlined');
%    cprintf('err',    'elements\n');
%    cprintf('cyan',   'cyan');
```

```
%    cprintf('_green', 'underlined green');
%    cprintf(-[1,0,1], 'underlined magenta');
%    cprintf([1,0.5,0],'and multi-\nline orange\n');
%    cprintf('*blue',  'and *bold* (R2011b+ only)\n');
%    cprintf('string');  % same as fprintf('string') and cprintf('text','string')
%
% Bugs and suggestions:
%    Please send to Yair Altman (altmany at gmail dot com)
%
% Warning:
%    This code heavily relies on undocumented and unsupported Matlab
%    functionality. It works on Matlab 7+, but use at your own risk!
%
%    A technical description of the implementation can be found at:
%    <a href="http://undocumentedmatlab.com/blog/cprintf/">http://UndocumentedMatlab.com/blog/cprintf/</a>
%
% Limitations:
%    1. In R2011a and earlier, a single space char is inserted at the
%       beginning of each CPRINTF text segment (this is ok in R2011b+).
%
%    2. In R2011a and earlier, consecutive differently-colored multi-line
%       CPRINTFs sometimes display incorrectly on the bottom line.
%       As far as I could tell this is due to a Matlab bug. Examples:
%         >> cprintf('-str','under\nline'); cprintf('err','red\n'); % hidden 'red', unhidden '_'
%         >> cprintf('str','regu\nlar'); cprintf('err','red\n'); % underline red (not purple) 'lar'
%
%    3. Sometimes, non newline ('\n')-terminated segments display unstyled
%       (black) when the command prompt chevron ('>>') regains focus on the
%       continuation of that line (I can't pinpoint when this happens).
%       To fix this, simply newline-terminate all command-prompt messages.
%
%    4. In R2011b and later, the above errors appear to be fixed. However,
%       the last character of an underlined segment is not underlined for
%       some unknown reason (add an extra space character to make it look better)
%
%    5. In old Matlab versions (e.g., Matlab 7.1 R14), multi-line styles
%       only affect the first line. Single-line styles work as expected.
%       R14 also appends a single space after underlined segments.
%
%    6. Bold style is only supported on R2011b+, and cannot also be underlined.
%
% Change log:
%    2015-06-24: Fixed a few discoloration issues (some other issues still remain)
%    2015-03-20: Fix: if command window isn't defined yet (startup) use standard fprintf as suggested by John Marozas
%    2012-08-09: Graceful degradation support for deployed (compiled) and non-desktop applications; minor bug fixes
%    2012-08-06: Fixes for R2012b; added bold style; accept RGB string (non-
```

```
numeric) style
%   2011-11-27: Fixes for R2011b
%   2011-08-29: Fix by Danilo (FEX comment) for non-default text colors
%   2011-03-04: Performance improvement
%   2010-06-27: Fix for R2010a/b; fixed edge case reported by Sharron; CPRINTF with no args runs the demo
%   2009-09-28: Fixed edge-case problem reported by Swagat K
%   2009-05-28: corrected nargout behavior suggested by Andreas G b
%   2009-05-13: First version posted on <a href="http://www.mathworks.com/matlabcentral/fileexchange/authors/27420">MathWorks File Exchange</a>
%
% See also:
%   sprintf, fprintf

% License to use and modify this code is granted freely to all interested, as long as the original author is
% referenced and attributed as such. The original author maintains the right to be solely associated with this work.

% Programmed and Copyright by Yair M. Altman: altmany(at)gmail.com
% $Revision: 1.10 $  $Date: 2015/06/24 01:29:18 $

  persistent majorVersion minorVersion
  if isempty(majorVersion)
      %v = version; if str2double(v(1:3)) <= 7.1
      %majorVersion = str2double(regexprep(version,'^(\d+).*','$1'));
      %minorVersion = str2double(regexprep(version,'^\d+\.(\d+).*','$1'));
      %[a,b,c,d,versionIdStrs]=regexp(version,'^(\d+)\.(\d+).*');  %#ok unused
      v = sscanf(version, '%d.', 2);
      majorVersion = v(1); %str2double(versionIdStrs{1}{1});
      minorVersion = v(2); %str2double(versionIdStrs{1}{2});
  end

  % The following is for debug use only:
  %global docElement txt el
  if ~exist('el','var') || isempty(el),  el=handle([]);  end  %#ok mlint short-circuit error ("used before defined")
  if nargin<1, showDemo(majorVersion,minorVersion); return;  end
  if isempty(style),  return;  end
  if all(ishandle(style)) && length(style)~=3
      dumpElement(style);
      return;
  end

  % Process the text string
  if nargin<2, format = style; style='text';  end
  %error(nargchk(2, inf, nargin, 'struct'));
  %str = sprintf(format,varargin{:});

  % In compiled mode
  try useDesktop = usejava('desktop'); catch, useDesktop = false; end
  if isdeployed | ~useDesktop %#ok<OR2> - for Matlab 6 compatibility
      % do not display any formatting - use simple fprintf()
      % See: http://undocumentedmatlab.com/blog/bold-color-text-in-the-command-window/#comment-103035
```

```
        % Also see: https://mail.google.com/mail/u/0/?ui=2&shva=1#all/1390a26e7ef4aa4d
        % Also see: https://mail.google.com/mail/u/0/?ui=2&shva=1#all/13a6ed3223333b21
        count1 = fprintf(format,varargin{:});
    else
        % Else (Matlab desktop mode)
        % Get the normalized style name and underlining flag
        [underlineFlag, boldFlag, style, debugFlag] = processStyleInfo(style);

        % Set hyperlinking, if so requested
        if underlineFlag
            format = ['<a href="">' format '</a>'];

            % Matlab 7.1 R14 (possibly a few newer versions as well?)
            % have a bug in rendering consecutive hyperlinks
            % This is fixed by appending a single non-linked space
            if majorVersion < 7 || (majorVersion==7 && minorVersion <= 1)
                format(end+1) = ' ';
            end
        end

        % Set bold, if requested and supported (R2011b+)
        if boldFlag
            if (majorVersion > 7 || minorVersion >= 13)
                format = ['<strong>' format '</strong>'];
            else
                boldFlag = 0;
            end
        end

        % Get the current CW position
        cmdWinDoc = com.mathworks.mde.cmdwin.CmdWinDocument.getInstance;
        lastPos = cmdWinDoc.getLength;

        % If not beginning of line
        bolFlag = 0;  %#ok
        %if docElement.getEndOffset - docElement.getStartOffset > 1
            % Display a hyperlink element in order to force element separation
            % (otherwise adjacent elements on the same line will be merged)
            if majorVersion<7 || (majorVersion==7 && minorVersion<13)
                if ~underlineFlag
                    fprintf('<a href=""> </a>');  %fprintf('<a href=""> </a>\b');
                elseif format(end)~=10  % if no newline at end
                    fprintf(' ');  %fprintf(' \b');
                end
            end
            %drawnow;
            bolFlag = 1;
        %end

        % Get a handle to the Command Window component
        mde = com.mathworks.mde.desk.MLDesktop.getInstance;
        cw = mde.getClient('Command Window');
```

```
        % Fix: if command window isn't defined yet (startup), use standard fprintf()
        if (isempty(cw))
           count1 = fprintf(format,varargin{:});
           if nargout
               count = count1;
           end
           return;
        end

        xCmdWndView = cw.getComponent(0).getViewport.getComponent(0);

        % Store the CW background color as a special color pref
        % This way, if the CW bg color changes (via File/Preferences),
        % it will also affect existing rendered strs
        com.mathworks.services.Prefs.setColorPref('CW_BG_Color',xCmdWndView.getBackground);

        % Display the text in the Command Window
        % Note: fprintf(2,...) is required in order to add formatting tokens, which
        % ^^^^  can then be updated below (no such tokens when outputting to stdout)
        count1 = fprintf(2,format,varargin{:});

        % Repaint the command window
        %awtinvoke(cmdWinDoc,'remove',lastPos,1);   % TODO: find out how to remove the extra '_'
        drawnow;  % this is necessary for the following to work properly (refer to Evgeny Pr in FEX comment 16/1/2011)
        xCmdWndView.repaint;
        %hListeners = cmdWinDoc.getDocumentListeners; for idx=1:numel(hListeners), try hListeners(idx).repaint; catch, end, end

        docElement = cmdWinDoc.getParagraphElement(lastPos+1);
        if majorVersion<7 || (majorVersion==7 && minorVersion<13)
            if bolFlag && ~underlineFlag
                % Set the leading hyperlink space character ('_') to the bg color, effectively hiding it
                % Note: old Matlab versions have a bug in hyperlinks that need to be accounted for...
                %disp(' '); dumpElement(docElement)
                setElementStyle(docElement,'CW_BG_Color',1+underlineFlag,majorVersion,minorVersion); %+getUrlsFix(docElement));
                %disp(' '); dumpElement(docElement)
                el(end+1) = handle(docElement);  % #ok used in debug only
            end

            % Fix a problem with some hidden hyperlinks becoming unhidden...
            fixHyperlink(docElement);
            %dumpElement(docElement);
        end

        % Get the Document Element(s) corresponding to the latest fprintf
```

```
operation
        while docElement.getStartOffset < cmdWinDoc.getLength
            % Set the element style according to the current style
            if debugFlag, dumpElement(docElement); end
            specialFlag = underlineFlag | boldFlag;
            setElementStyle(docElement,style,specialFlag,majorVersion,minorVersion);
            if debugFlag, dumpElement(docElement); end
            docElement2 = cmdWinDoc.getParagraphElement(docElement.getEndOffset+1);
            if isequal(docElement,docElement2), break; end
            docElement = docElement2;
        end
        if debugFlag, dumpElement(docElement); end

        % Force a Command-Window repaint
        % Note: this is important in case the rendered str was not '\n'-terminated
        xCmdWndView.repaint;

        % The following is for debug use only:
        el(end+1) = handle(docElement);  %#ok used in debug only
        %elementStart  = docElement.getStartOffset;
        %elementLength = docElement.getEndOffset - elementStart;
        %txt = cmdWinDoc.getText(elementStart,elementLength);
    end

    if nargout
        count = count1;
    end
    return;  % debug breakpoint

% Process the requested style information
function [underlineFlag,boldFlag,style,debugFlag] = processStyleInfo(style)
    underlineFlag = 0;
    boldFlag = 0;
    debugFlag = 0;

    % First, strip out the underline/bold markers
    if ischar(style)
        % Styles containing '-' or '_' should be underlined (using a no-target hyperlink hack)
        %if style(1)=='-'
        underlineIdx = (style=='-') | (style=='_');
        if any(underlineIdx)
            underlineFlag = 1;
            %style = style(2:end);
            style = style(~underlineIdx);
        end

        % Check for bold style (only if not underlined)
        boldIdx = (style=='*');
        if any(boldIdx)
            boldFlag = 1;
            style = style(~boldIdx);
```

```
        end
        if underlineFlag && boldFlag
            warning('YMA:cprintf:BoldUnderline','Matlab does not support both bold & underline')
        end

        % Check for debug mode (style contains '!')
        debugIdx = (style=='!');
        if any(debugIdx)
            debugFlag = 1;
            style = style(~debugIdx);
        end

        % Check if the remaining style sting is a numeric vector
        %styleNum = str2num(style); %#ok<ST2NM>  % not good because style='text' is evaled!
        %if ~isempty(styleNum)
        if any(style==' ' | style==',' | style==';')
            style = str2num(style); %#ok<ST2NM>
        end
    end

    % Style = valid matlab RGB vector
    if isnumeric(style) && length(style)==3 && all(style<=1) && all(abs(style)>=0)
        if any(style<0)
            underlineFlag = 1;
            style = abs(style);
        end
        style = getColorStyle(style);

    elseif ~ischar(style)
        error('YMA:cprintf:InvalidStyle','Invalid style - see help section for a list of valid style values')

    % Style name
    else
        % Try case-insensitive partial/full match with the accepted style names
        matlabStyles = {'Text','Keywords','Comments','Strings','UnterminatedStrings','SystemCommands','Errors'};
        validStyles  = [matlabStyles, ...
                        'Black','Cyan','Magenta','Blue','Green','Red','Yellow','White', ...
                        'Hyperlinks'];
        matches = find(strncmpi(style,validStyles,length(style)));

        % No match - error
        if isempty(matches)
            error('YMA:cprintf:InvalidStyle','Invalid style - see help section for a list of valid style values')

        % Too many matches (ambiguous) - error
        elseif length(matches) > 1
            error('YMA:cprintf:AmbigStyle','Ambiguous style name - supply extra characters for uniqueness')
```

```

      % Regular text
      elseif matches == 1
          style = 'ColorsText';  % fixed by Danilo, 29/8/2011

      % Highlight preference style name
      elseif matches <= length(matlabStyles)
          style = ['Colors_M_' validStyles{matches}];

      % Color name
      elseif matches < length(validStyles)
          colors = [0,0,0; 0,1,1; 1,0,1; 0,0,1; 0,1,0; 1,0,0; 1,1,0; 1,1,1];
          requestedColor = colors(matches-length(matlabStyles),:);
          style = getColorStyle(requestedColor);

      % Hyperlink
      else
          style = 'Colors_HTML_HTMLLinks';  % CWLink
          underlineFlag = 1;
      end
  end

% Convert a Matlab RGB vector into a known style name (e.g., '[255,37,0]')
function styleName = getColorStyle(rgb)
  intColor = int32(rgb*255);
  javaColor = java.awt.Color(intColor(1), intColor(2), intColor(3));
  styleName = sprintf('[%d,%d,%d]',intColor);
  com.mathworks.services.Prefs.setColorPref(styleName,javaColor);

% Fix a bug in some Matlab versions, where the number of URL segments
% is larger than the number of style segments in a doc element
function delta = getUrlsFix(docElement)  %#ok currently unused
  tokens = docElement.getAttribute('SyntaxTokens');
  links  = docElement.getAttribute('LinkStartTokens');
  if length(links) > length(tokens(1))
      delta = length(links) > length(tokens(1));
  else
      delta = 0;
  end

% fprintf(2,str) causes all previous '_'s in the line to become red - fix this
function fixHyperlink(docElement)
  try
      tokens = docElement.getAttribute('SyntaxTokens');
      urls   = docElement.getAttribute('HtmlLink');
      urls   = urls(2);
      links  = docElement.getAttribute('LinkStartTokens');
      offsets = tokens(1);
      styles  = tokens(2);
      doc = docElement.getDocument;

      % Loop over all segments in this docElement
      for idx = 1 : length(offsets)-1
          % If this is a hyperlink with no URL target and starts with ' ' and is collored as an error (red)...
```

```
            if strcmp(styles(idx).char,'Colors_M_Errors')
                character = char(doc.getText(offsets(idx)+docElement.getStartOffset,1));
                if strcmp(character,' ')
                    if isempty(urls(idx)) && links(idx)==0
                        % Revert the style color to the CW background color (i.e., hide it!)
                        styles(idx) = java.lang.String('CW_BG_Color');
                    end
                end
            end
        end
    catch
        % never mind...
    end

% Set an element to a particular style (color)
function setElementStyle(docElement,style,specialFlag, majorVersion, minorVersion)
    %global tokens links urls urlTargets % for debug only
    global oldStyles
    if nargin<3, specialFlag=0; end
    % Set the last Element token to the requested style:
    % Colors:
    tokens = docElement.getAttribute('SyntaxTokens');
    try
        styles = tokens(2);
        oldStyles{end+1} = cell(styles);

        % Correct edge case problem
        extraInd = double(majorVersion>7 || (majorVersion==7 && minorVersion>=13)); % =0 for R2011a-, =1 for R2011b+
        %{
        if ~strcmp('CWLink',char(styles(end-hyperlinkFlag))) && ...
            strcmp('CWLink',char(styles(end-hyperlinkFlag-1)))
           extraInd = 0;%1;
        end
        hyperlinkFlag = ~isempty(strmatch('CWLink',tokens(2)));
        hyperlinkFlag = 0 + any(cellfun(@(c)(~isempty(c)&&strcmp(c,'CWLink')),cell(tokens(2))));
        %}

        jStyle = java.lang.String(style);
        if numel(styles)==4 && isempty(char(styles(2)))
            % Attempt to fix discoloration issues - NOT SURE THAT THIS IS OK! - 24/6/2015
            styles(1) = jStyle;
        end
        styles(end-extraInd) = java.lang.String('');
        styles(end-extraInd-specialFlag) = jStyle; % #ok apparently unused but in reality used by Java
        if extraInd
            styles(end-specialFlag) = jStyle;
        end

```

```
        oldStyles{end} = [oldStyles{end} cell(styles)];
    catch
        % never mind for now
    end

    % Underlines (hyperlinks):
    %{
    links = docElement.getAttribute('LinkStartTokens');
    if isempty(links)
        %docElement.addAttribute('LinkStartTokens',repmat(int32(-1),length(tokens(2)),1));
    else
        %TODO: remove hyperlink by setting the value to -1
    end
    %}

    % Correct empty URLs to be un-hyperlinkable (only underlined)
    urls = docElement.getAttribute('HtmlLink');
    if ~isempty(urls)
        urlTargets = urls(2);
        for urlIdx = 1 : length(urlTargets)
            try
                if urlTargets(urlIdx).length < 1
                    urlTargets(urlIdx) = [];  % '' => []
                end
            catch
                % never mind...
                a=1;  %#ok used for debug breakpoint...
            end
        end
    end

    % Bold: (currently unused because we cannot modify this immutable int32 numeric array)
    %{
    try
        %hasBold = docElement.isDefined('BoldStartTokens');
        bolds = docElement.getAttribute('BoldStartTokens');
        if ~isempty(bolds)
            %docElement.addAttribute('BoldStartTokens',repmat(int32(1),length(bolds),1));
        end
    catch
        % never mind - ignore...
        a=1;  %#ok used for debug breakpoint...
    end
    %}

    return;  % debug breakpoint

% Display information about element(s)
function dumpElement(docElements)
    %return;
    disp(' ');
    numElements = length(docElements);
```

```
    cmdWinDoc = docElements(1).getDocument;
    for elementIdx = 1 : numElements
        if numElements > 1,  fprintf('Element #%d:\n',elementIdx);  end
        docElement = docElements(elementIdx);
        if ~isjava(docElement),  docElement = docElement.java;  end
        %docElement.dump(java.lang.System.out,1)
        disp(docElement)
        tokens = docElement.getAttribute('SyntaxTokens');
        if isempty(tokens),  continue;  end
        links = docElement.getAttribute('LinkStartTokens');
        urls  = docElement.getAttribute('HtmlLink');
        try bolds = docElement.getAttribute('BoldStartTokens'); catch, bolds = []; end
        txt = {};
        tokenLengths = tokens(1);
        for tokenIdx = 1 : length(tokenLengths)-1
            tokenLength = diff(tokenLengths(tokenIdx+[0,1]));
            if (tokenLength < 0)
                tokenLength = docElement.getEndOffset - docElement.getStartOffset - tokenLengths(tokenIdx);
            end
            txt{tokenIdx} = cmdWinDoc.getText(docElement.getStartOffset+tokenLengths(tokenIdx),tokenLength).char;  %#ok
        end
        lastTokenStartOffset = docElement.getStartOffset + tokenLengths(end);
        try
            txt{end+1} = cmdWinDoc.getText(lastTokenStartOffset, docElement.getEndOffset-lastTokenStartOffset).char; %#ok
        catch
            txt{end+1} = ''; %#ok<AGROW>
        end
        %cmdWinDoc.uiinspect
        %docElement.uiinspect
        txt = strrep(txt',sprintf('\n'),'\n');
        try
            data = [cell(tokens(2)) m2c(tokens(1)) m2c(links) m2c(urls(1)) cell(urls(2)) m2c(bolds) txt];
            if elementIdx==1
                disp('    SyntaxTokens(2,1) - LinkStartTokens - HtmlLink(1,2) - BoldStartTokens - txt');
                disp('=================================================================================');
            end
        catch
            try
                data = [cell(tokens(2)) m2c(tokens(1)) m2c(links) txt];
            catch
                disp([cell(tokens(2)) m2c(tokens(1)) txt]);
                try
                    data = [m2c(links) m2c(urls(1)) cell(urls(2))];
                catch
                    % Mtlab 7.1 only has urls(1)...
                    data = [m2c(links) cell(urls)];
                end
```

```
            end
        end
        disp(data)
    end

% Utility function to convert matrix => cell
function cells = m2c(data)
  %datasize = size(data);  cells = mat2cell(data,ones(1,datasize(1)),ones(1,datasize(2)));
  cells = num2cell(data);

% Display the help and demo
function showDemo(majorVersion,minorVersion)
  fprintf('cprintf displays formatted text in the Command Window.\n\n');
  fprintf('Syntax: count = cprintf(style,format,...);  click <a href="matlab:help cprintf">here</a> for details.\n\n');
  url = 'http://UndocumentedMatlab.com/blog/cprintf/';
  fprintf(['Technical description: <a href="' url '">' url '</a>\n\n']);
  fprintf('Demo:\n\n');
  boldFlag = majorVersion>7 || (majorVersion==7 && minorVersion>=13);
  s = ['cprintf(''text'',    ''regular black text'');' 10 ...
       'cprintf(''hyper'',   ''followed %s'',''by'');' 10 ...
       'cprintf(''key'',     ''%d colored'',' num2str(4+boldFlag) ');' 10 ...
       'cprintf(''-comment'',''& underlined'');' 10 ...
       'cprintf(''err'',     ''elements:\n'');' 10 ...
       'cprintf(''cyan'',    ''cyan'');' 10 ...
       'cprintf(''_green'',  ''underlined green'');' 10 ...
       'cprintf(-[1,0,1], ''underlined magenta'');' 10 ...
       'cprintf([1,0.5,0], ''and multi-\nline orange\n'');' 10];
   if boldFlag
       % In R2011b+ the internal bug that causes the need for an extra space
       % is apparently fixed, so we must insert the sparator spaces manually ...
       % On the other hand, 2011b enables *bold* format
       s = [s 'cprintf(''*blue'',   ''and *bold* (R2011b+ only)\n'');' 10];
       s = strrep(s, ''')',' '')');
       s = strrep(s, ''',5)',' '',5)');
       s = strrep(s, '\n ','\n');
   end
   disp(s);
   eval(s);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TODO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% - Fix: Remove leading space char (hidden underline '_')
% - Fix: Find workaround for multi-line quirks/limitations
% - Fix: Non-\n-terminated segments are displayed as black
% - Fix: Check whether the hyperlink fix for 7.1 is also needed on 7.2 etc.
% - Enh: Add font support
```