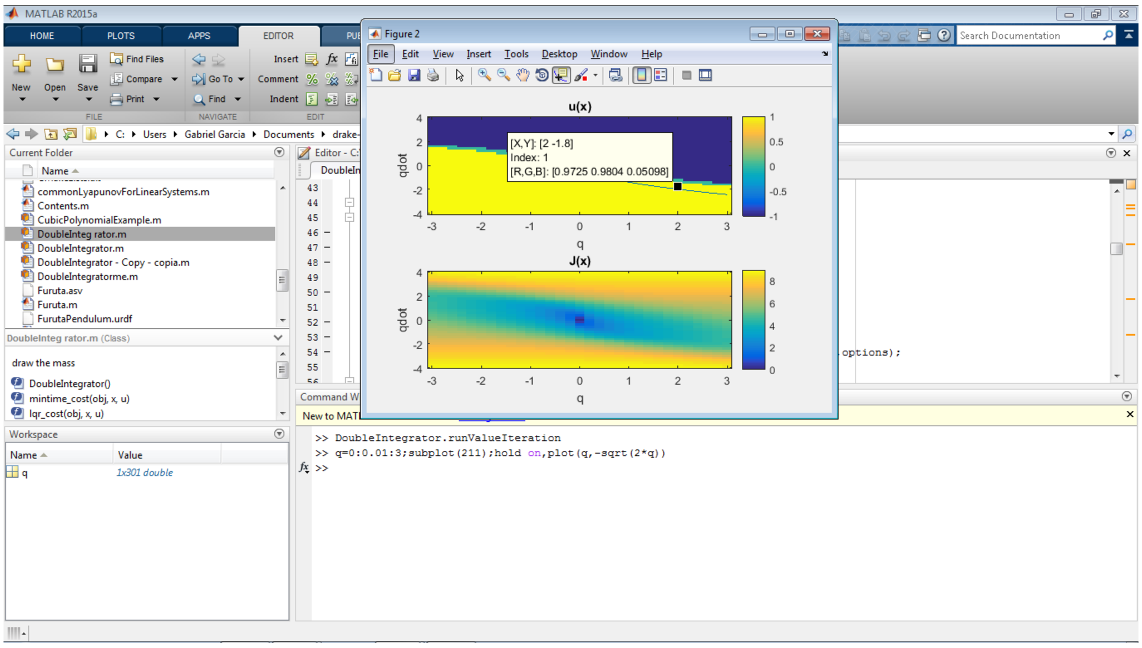
PSET2:

I. Value Iteration (Double Integrator):
- File: “DoubleIntegrator.m” in folder “drake-distro\drake\examples”

(a) Run the value iteration code for the double integrator to compute the optimal policy and optimal cost-to-go for the minimum-time problem. Compare the result to the analytical solution we found in lecture (also available in Example 9.2 in the course notes) by answering the following questions.

1) Find an initial condition of the form $(2, \dot{q}_0)$ such that the value iteration policy takes an action in exactly the wrong direction from the true optimal policy.



```
>> DoubleIntegrator.runValueIteration
>> q=0:0.01:3;subplot(211);hold on,plot(q,-sqrt(2*q))
```

```
[X,Y]: [2 -1.8]
Index: 1
[R,G,B]: [0.9725 0.9804 0.05098]
```

>> DoubleIntegrator.runValueIteration
>> q=0:0.01:3;subplot(211);hold on,plot(q,-sqrt(2*q))

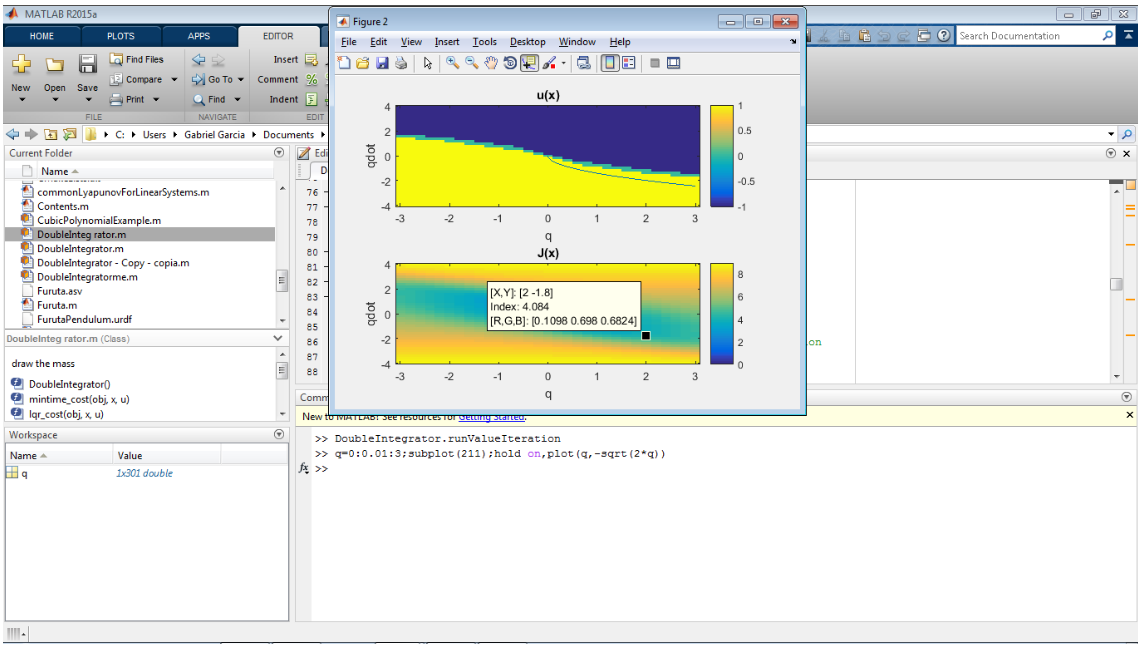
$$\dot{q}_0 = -1.8$$

2) What is the true optimal time-to-go from this state (i.e., for the optimal bang-bang controller derived in class)?

At $(q_0, \dot{q}_0) = (2, -1.8)$ the real optimal action is $u = -1$. Time to go for the $u = -1$ regime is:

$$T(x) = \dot{q} + 2\sqrt{\frac{1}{2}\dot{q}^2 + q} \rightarrow T(x_0) \approx 2.005$$

3) What is the time-to-go from this state estimated by value iteration?



```
[X,Y]: [2 -1.8]
Index: 4.084
[R,G,B]: [0.1098 0.698 0.6824]
```

```
>> DoubleIntegrator.runValueIteration
>> q=0:0.01:3;subplot(211);hold on,plot(q,-sqrt(2*q))
>>
```

$$T_{VI}(x_0) = 4.084$$

4) When implementing value iteration, one needs to be wary of several implementation details. Find a setting of the discretization (i.e., the variable xbins in DoubleIntegrator.m) that causes the code to NOT converge.

We can see the implementation of the Minimum time cost in DoubleIntegrator.m:

```
function g = mintime_cost(obj,x,u)
  g = ((x(1,:) ~= 0) | (x(2,:) ~= 0));
end
```

Also, we can see the stopping condition in the function `valueIteration`, inside of `MarkovDecisionProcess`

```
while (err > converged)
  Jold = J;
  [J,PI] = min(mdp.C+mdp.gamma*reshape(Tstack*J,nS,nA),[],2);
  err = max(abs(Jold-J));
  %if nargin>2, drawfun(J,PI); drawnow; end
end
```

Basically, if we set a discretization of x, that will not contain the origin, the cost will always be one, and in the iteration, J will contain a term that change in one, making err greater or equal than one, and we will stay always in the while loop.

So we can set a discretization like:

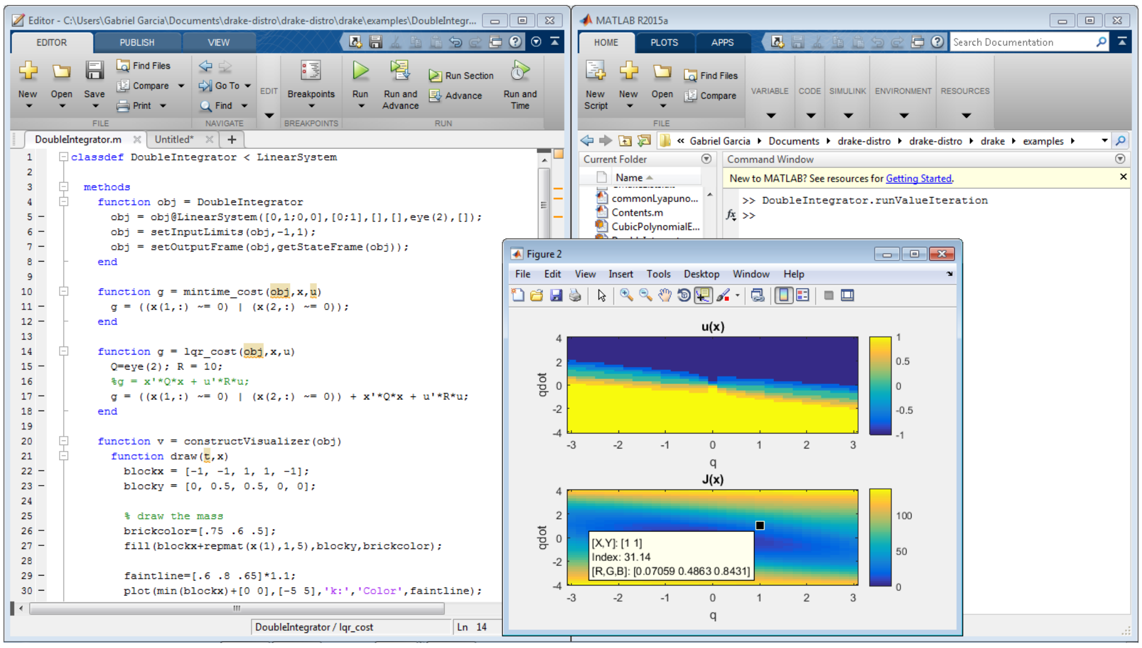
```
xbins = {[-3:.17:3],[-4:.17:4]};
```

(b) Change the cost-to-go function to a combination of the quadratic regulator problem and the minimum-time problem:

$$g(q, \dot{q}, u) = c(q, \dot{q}) + Q_p q^2 + Q_d \dot{q}^2 + Ru^2,$$

where $c(q, \dot{q})$ is 0 when $(q, \dot{q}) = (0, 0)$ and 1 otherwise. Use $Q_p = Q_d = 1, R = 10.$

1) What is the cost-to-go from the point (1.0,1.0) estimated by the value iteration?



```
classdef DoubleIntegrator < LinearSystem

  methods
    function obj = DoubleIntegrator
      obj = obj@LinearSystem([0,1;0,0],[0;1],[],[],eye(2),[]);
      obj = setInputLimits(obj,-1,1);
      obj = setOutputFrame(obj,getStateFrame(obj));
    end

    function g = mintime_cost(obj,x,u)
      g = ((x(1,:) ~= 0) | (x(2,:) ~= 0));
    end

    function g = lqr_cost(obj,x,u)
      Q=eye(2); R = 10;
      %g = x'*Q*x + u'*R*u;
      g = ((x(1,:) ~= 0) | (x(2,:) ~= 0)) + x'*Q*x + u'*R*u;
    end

    function v = constructVisualizer(obj)
      function draw(t,x)
        blockx = [-1, -1, 1, 1, -1];
        blocky = [0, 0.5, 0.5, 0, 0];

        % draw the mass
        brickcolor=[.75 .6 .5];
        fill(blockx+repmat(x(1),1,5),blocky,brickcolor);

        faintline=[.6 .8 .65]*1.1;
        plot(min(blockx)+[0 0],[-5 5],'k:','Color',faintline);
```

```
>> DoubleIntegrator.runValueIteration
>>
```

Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\drake-distro\drake\examples\DoubleIntegr...
DoubleIntegrator.m
Untitled*
DoubleIntegrator / lqr_cost
Ln 14
MATLAB R2015a
« Gabriel Garcia ▸ Documents ▸ drake-distro ▸ drake-distro ▸ drake ▸ examples ▸
Current Folder
Command Window
New to MATLAB? See resources for Getting Started.
Figure 2
u(x)
J(x)
qdot
q
[X,Y]: [1 1]
Index: 31.14
[R,G,B]: [0.07059 0.4863 0.8431]

$J(x_0) = 31.14$

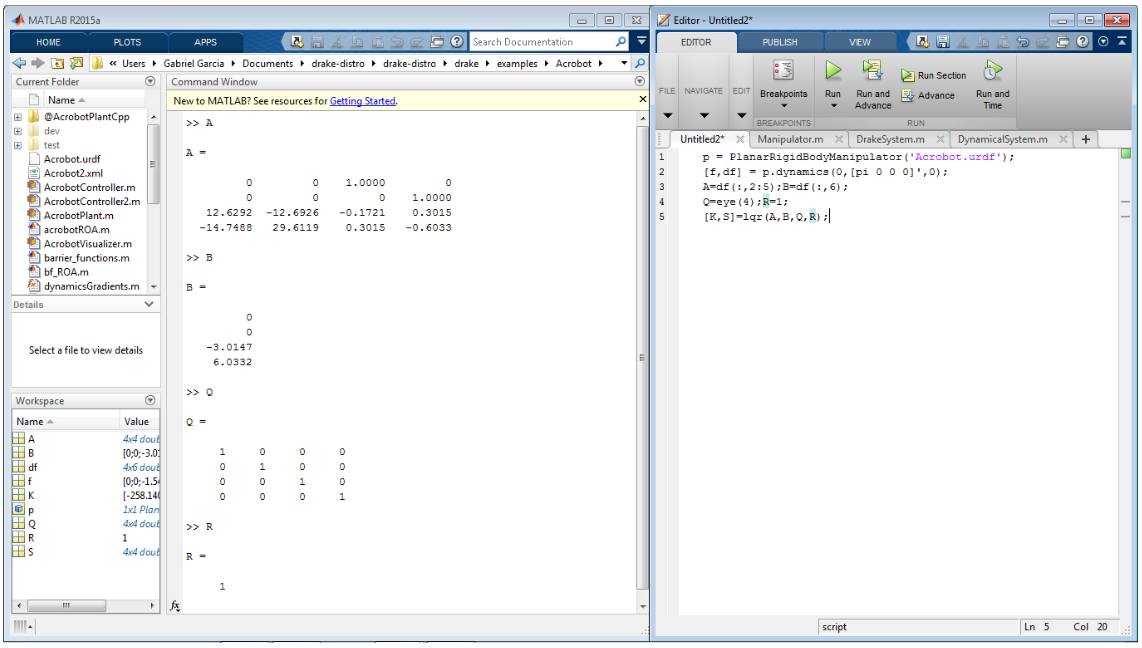
II. Value Iteration (Stochastic Grid World):

(a) Run the value iteration code for the double integrator to compute the optimal policy and optimal cost-to-go for the minimum-time problem. Compare the result to the analytical solution we found in lecture (also available in Example 9.2 in the course notes) by answering the following questions.

PSET3:

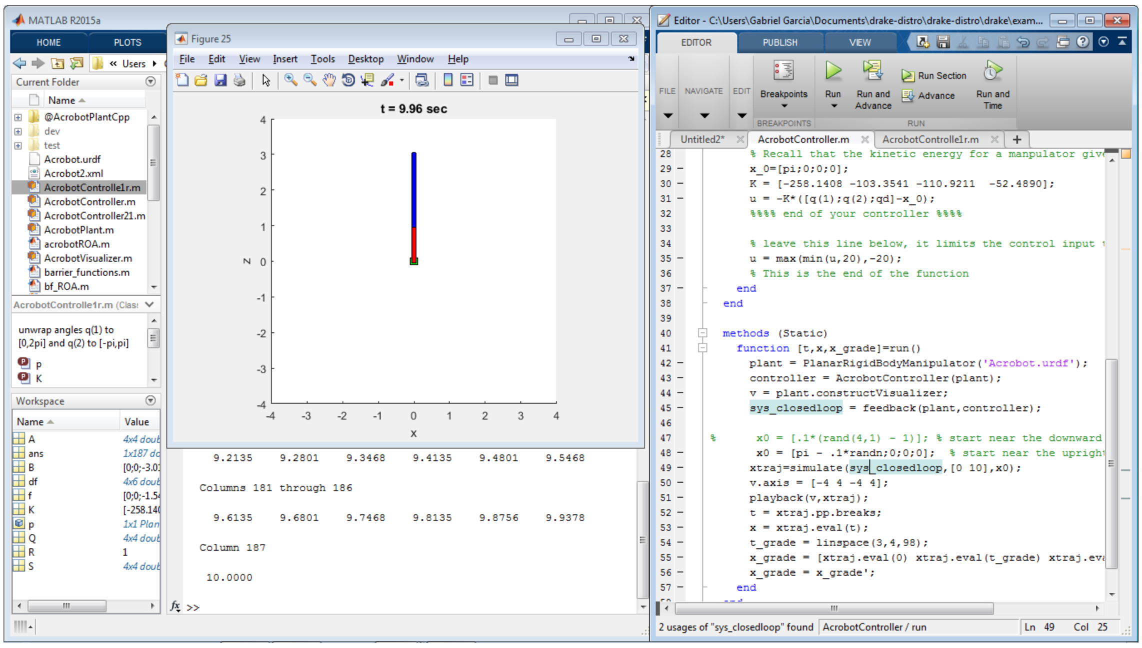
- Alternative file: “AcrobotController.m” in folder “drake-distro\drake\examples\Acrobot”

I. Acrobot Balancing:
   - files inside: “Acrobot_Balancing/” in folder “drake-distro\drake\examples\Acrobot”



```
>> A

A =

         0         0    1.0000         0
         0         0         0    1.0000
   12.6292  -12.6926   -0.1721    0.3015
  -14.7488   29.6119    0.3015   -0.6033

>> B

B =

         0
         0
   -3.0147
    6.0332

>> Q

Q =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> R

R =

     1
```

```
p = PlanarRigidBodyManipulator('Acrobot.urdf');
[f,df] = p.dynamics(0,[pi 0 0 0]',0);
A=df(:,2:5);B=df(:,6);
Q=eye(4);R=1;
[K,S]=lqr(A,B,Q,R);
```

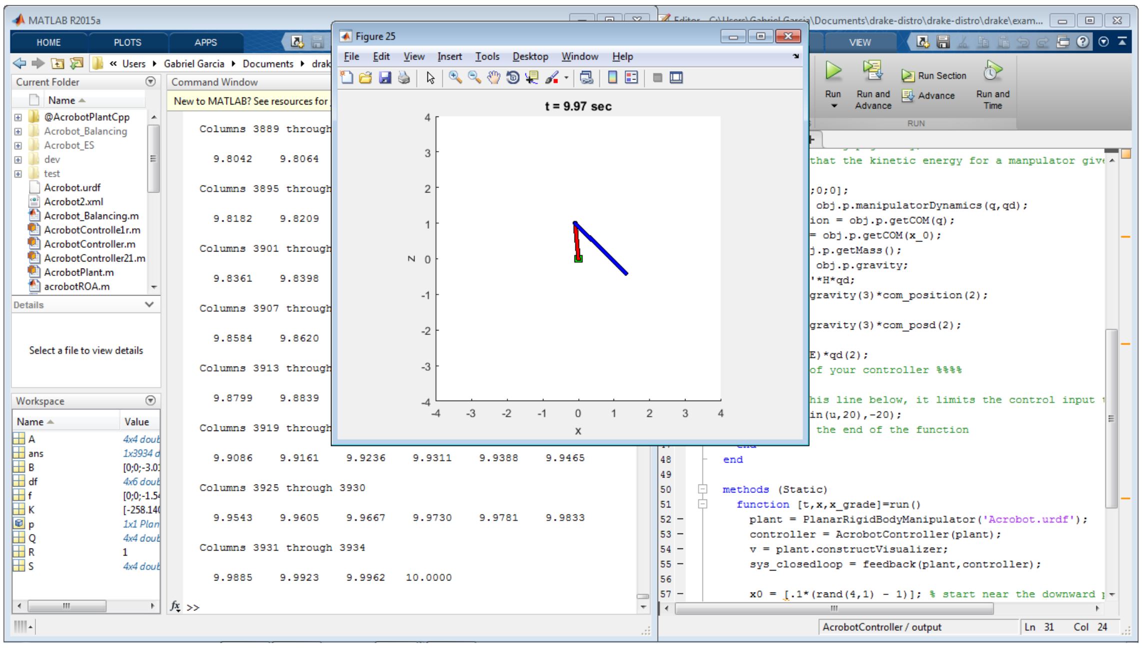


```
  9.2135    9.2801    9.3468    9.4135    9.4801    9.5468

Columns 181 through 186

  9.6135    9.6801    9.7468    9.8135    9.8756    9.9378

Column 187

 10.0000

>>
```

```
      % Recall that the kinetic energy for a manpulator give
      x_0=[pi;0;0;0];
      K = [-258.1408 -103.3541 -110.9211  -52.4890];
      u = -K*([q(1);q(2);qd]-x_0);
      %%%% end of your controller %%%%

      % leave this line below, it limits the control input t
      u = max(min(u,20),-20);
      % This is the end of the function
    end
  end

  methods (Static)
    function [t,x,x_grade]=run()
      plant = PlanarRigidBodyManipulator('Acrobot.urdf');
      controller = AcrobotController(plant);
      v = plant.constructVisualizer;
      sys_closedloop = feedback(plant,controller);

%       x0 = [.1*(rand(4,1) - 1)]; % start near the downward
       x0 = [pi - .1*randn;0;0;0];  % start near the upright
      xtraj=simulate(sys_closedloop,[0 10],x0);
      v.axis = [-4 4 -4 4];
      playback(v,xtraj);
      t = xtraj.pp.breaks;
      x = xtraj.eval(t);
      t_grade = linspace(3,4,98);
      x_grade = [xtraj.eval(0) xtraj.eval(t_grade) xtraj.eva
      x_grade = x_grade';
    end
```

2 usages of "sys_closedloop" found | AcrobotController / run | Ln 49 Col 25

II. Acrobot Energy Shaping:
   - files inside: “Acrobot_ES/” in folder “drake-distro\drake\examples\Acrobot”

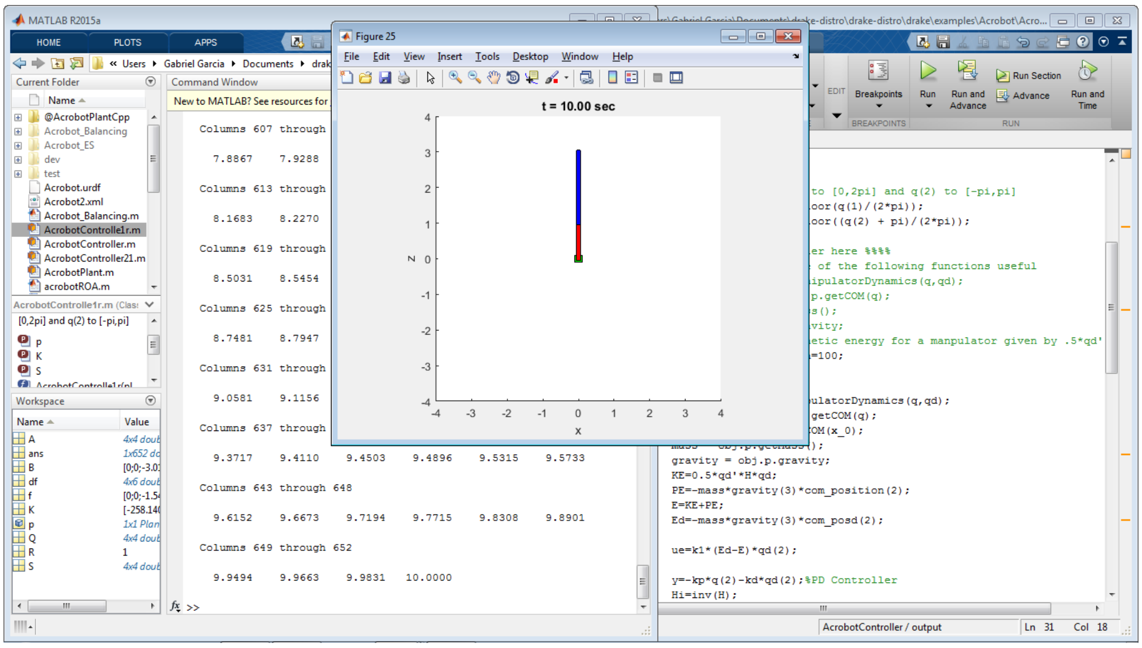


```
Columns 3889 through

  9.8042    9.8064

Columns 3895 through

  9.8182    9.8209

Columns 3901 through

  9.8361    9.8398

Columns 3907 through

  9.8584    9.8620

Columns 3913 through

  9.8799    9.8839

Columns 3919 through

  9.9086    9.9161    9.9236    9.9311    9.9388    9.9465

Columns 3925 through 3930

  9.9543    9.9605    9.9667    9.9730    9.9781    9.9833

Columns 3931 through 3934

  9.9885    9.9923    9.9962   10.0000

>>
```

```
that the kinetic energy for a manpulator give

;0;0];
 obj.p.manipulatorDynamics(q,qd);
ion = obj.p.getCOM(q);
= obj.p.getCOM(x_0);
j.p.getMass();
 obj.p.gravity;
'*H*qd;
gravity(3)*com_position(2);

gravity(3)*com_posd(2);

E)*qd(2);
of your controller %%%%

his line below, it limits the control input
in(u,20),-20);
 the end of the function

    end

    methods (Static)
      function [t,x,x_grade]=run()
        plant = PlanarRigidBodyManipulator('Acrobot.urdf');
        controller = AcrobotController(plant);
        v = plant.constructVisualizer;
        sys_closedloop = feedback(plant,controller);

        x0 = [.1*(rand(4,1) - 1)]; % start near the downward
```

III. Acrobot Swingup

- files inside: “Acrobot_Swing-up/” in folder “drake-distro\drake\examples\Acrobot”



```
Columns 607 through

  7.8867    7.9288

Columns 613 through

  8.1683    8.2270

Columns 619 through

  8.5031    8.5454

Columns 625 through

  8.7481    8.7947

Columns 631 through

  9.0581    9.1156

Columns 637 through

  9.3717    9.4110    9.4503    9.4896    9.5315    9.5733

Columns 643 through 648

  9.6152    9.6673    9.7194    9.7715    9.8308    9.8901

Columns 649 through 652

  9.9494    9.9663    9.9831   10.0000

>>
```

```
to [0,2pi] and q(2) to [-pi,pi]
oor(q(1)/(2*pi));
oor((q(2) + pi)/(2*pi));

er here %%%%
 of the following functions useful
ipulatorDynamics(q,qd);
p.getCOM(q);
s();
vity;
etic energy for a manpulator given by .5*qd'
=100;


ulatorDynamics(q,qd);
getCOM(q);
OM(x_0);
);
gravity = obj.p.gravity;
KE=0.5*qd'*H*qd;
PE=-mass*gravity(3)*com_position(2);
E=KE+PE;
Ed=-mass*gravity(3)*com_posd(2);

ue=k1*(Ed-E)*qd(2);

y=-kp*q(2)-kd*qd(2);%PD Controller
Hi=inv(H);
```

Note that we have added K and S as properties to the class.

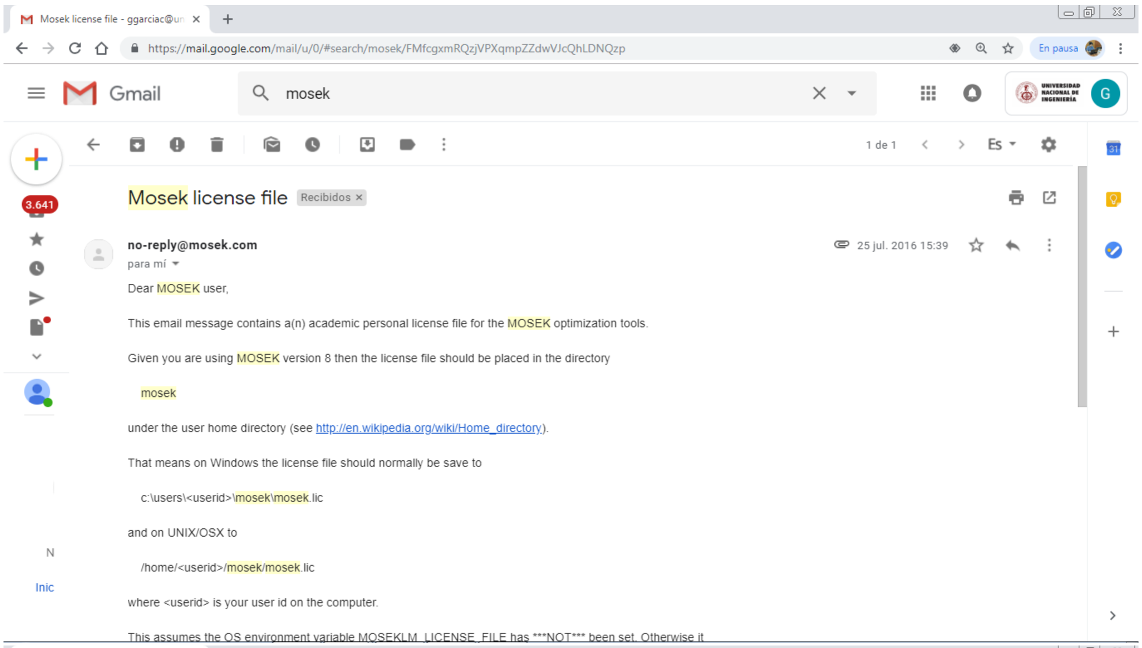
For more information we can take a look to the sections 3.6.3 and 3.6.4, and its paper references of the Curse Notes Textbooks

[23] Chung Choo Chung and John Hauser. Nonlinear control of a swinging pendulum. Automatica, 31(6):851–862, June 1995.

[94] Xin Xin and M. Kaneda. New analytical results of the energy based swinging up control of the acrobot. In Proceedings of the 43rd IEEE Conference on Decision and Control (CDC), volume 1, pages 704 – 709. IEEE, Dec 2004.

PSET4:

Updating Mosek Academic License.


Mosek license file - ggarciac@un
https://mail.google.com/mail/u/0/#search/mosek/FMfcgxmRQzjVPXqmpZZdwVJcQhLDNQzp
Gmail
mosek
1 de 1
Recibidos

Mosek license file

no-reply@mosek.com
para mí

25 jul. 2016 15:39

Dear MOSEK user,

This email message contains a(n) academic personal license file for the MOSEK optimization tools.

Given you are using MOSEK version 8 then the license file should be placed in the directory

mosek

under the user home directory (see http://en.wikipedia.org/wiki/Home_directory).

That means on Windows the license file should normally be save to

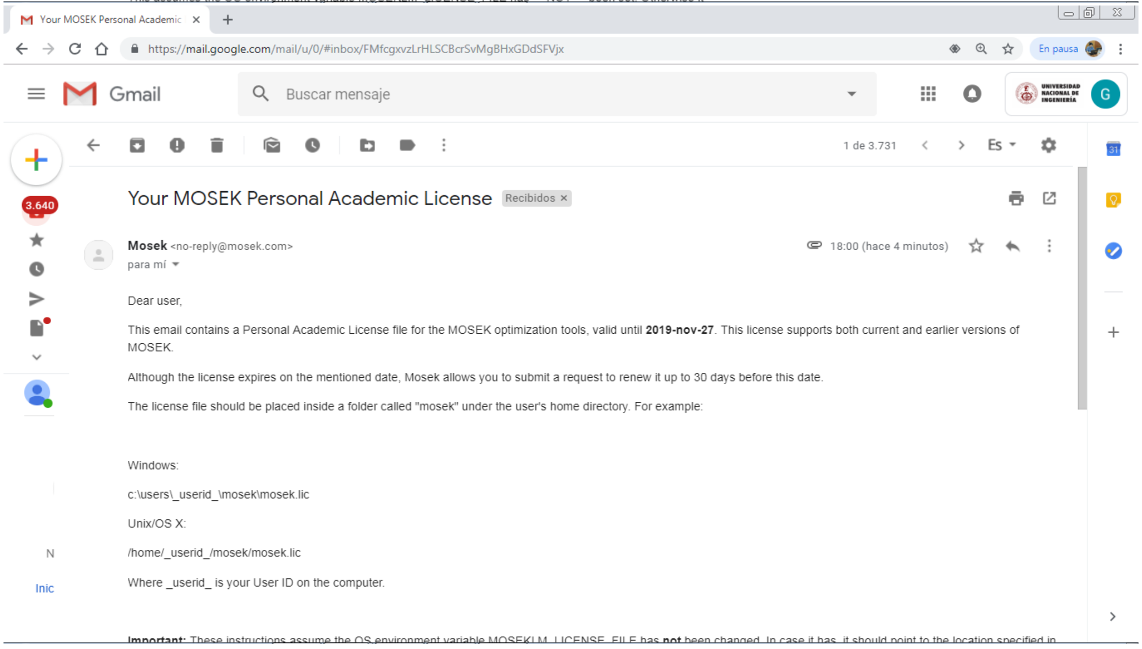
c:\users\<userid>\mosek\mosek.lic

and on UNIX/OSX to

/home/<userid>/mosek/mosek.lic

where <userid> is your user id on the computer.

This assumes the OS environment variable MOSEKLM_LICENSE_FILE has ***NOT*** been set. Otherwise it


Your MOSEK Personal Academic
https://mail.google.com/mail/u/0/#inbox/FMfcgxvzLrHLSCBcrSvMgBHxGDdSFVjx
Gmail
Buscar mensaje
1 de 3.731
Recibidos

Your MOSEK Personal Academic License

Mosek <no-reply@mosek.com>
para mí

18:00 (hace 4 minutos)

Dear user,

This email contains a Personal Academic License file for the MOSEK optimization tools, valid until **2019-nov-27**. This license supports both current and earlier versions of MOSEK.

Although the license expires on the mentioned date, Mosek allows you to submit a request to renew it up to 30 days before this date.

The license file should be placed inside a folder called "mosek" under the user's home directory. For example:

Windows:

c:\users_userid_\mosek\mosek.lic

Unix/OS X:

/home/_userid_/mosek/mosek.lic

Where _userid_ is your User ID on the computer.

Important: These instructions assume the OS environment variable MOSEKLM_LICENSE_FILE has **not** been changed. In case it has, it should point to the location specified in

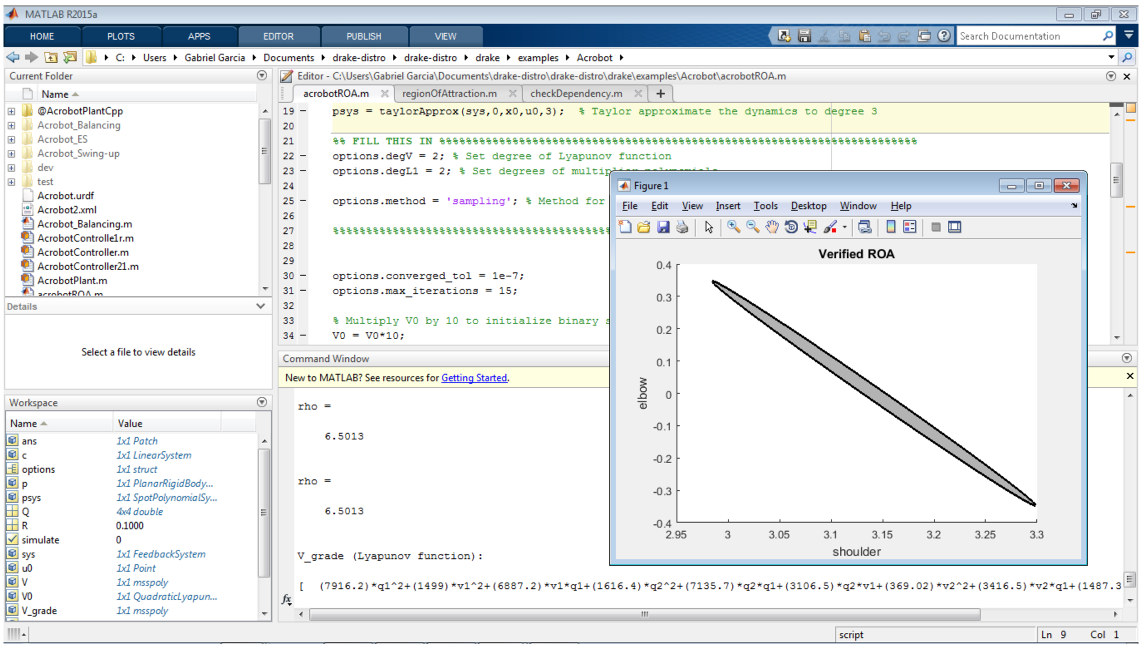
In startup.m we should add:

```
addpath('C:\Program Files\Mosek\8\tools\platform\win64x86\bin')
addpath('C:\Program Files\Mosek\8\toolbox\r2014a')
```

The license is copied to %USERPROFILE%\mosek\mosek.lic

I. Acrobot Region of Attraction (Part I):

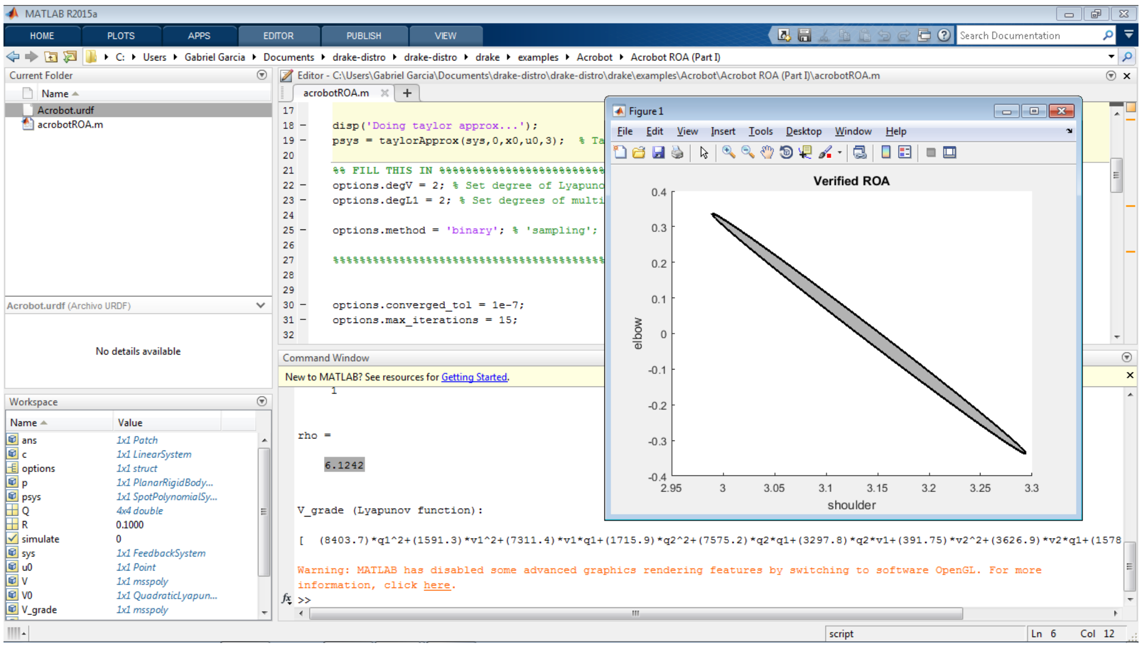
- File: “Acrobot ROA (Part I)/ acrobotROA.m” in folder “drake-distro\drake\examples\Acrobot”


Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\drake-distro\drake\examples\Acrobot\acrobotROA.m

```
    psys = taylorApprox(sys,0,x0,u0,3);  % Taylor approximate the dynamics to degree 3

    %% FILL THIS IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    options.degV = 2; % Set degree of Lyapunov function
    options.degL1 = 2; % Set degrees of multip

    options.method = 'sampling'; % Method for

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    options.converged_tol = 1e-7;
    options.max_iterations = 15;

    % Multiply V0 by 10 to initialize binary s
    V0 = V0*10;
```

```
rho =

    6.5013


rho =

    6.5013


V_grade (Lyapunov function):

[  (7916.2)*q1^2+(1499)*v1^2+(6887.2)*v1*q1+(1616.4)*q2^2+(7135.7)*q2*q1+(3106.5)*q2*v1+(369.02)*v2^2+(3416.5)*v2*q1+(1487.3
```

$$\rho = 6.5013$$


Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\drake-distro\drake\examples\Acrobot\Acrobot ROA (Part I)\acrobotROA.m

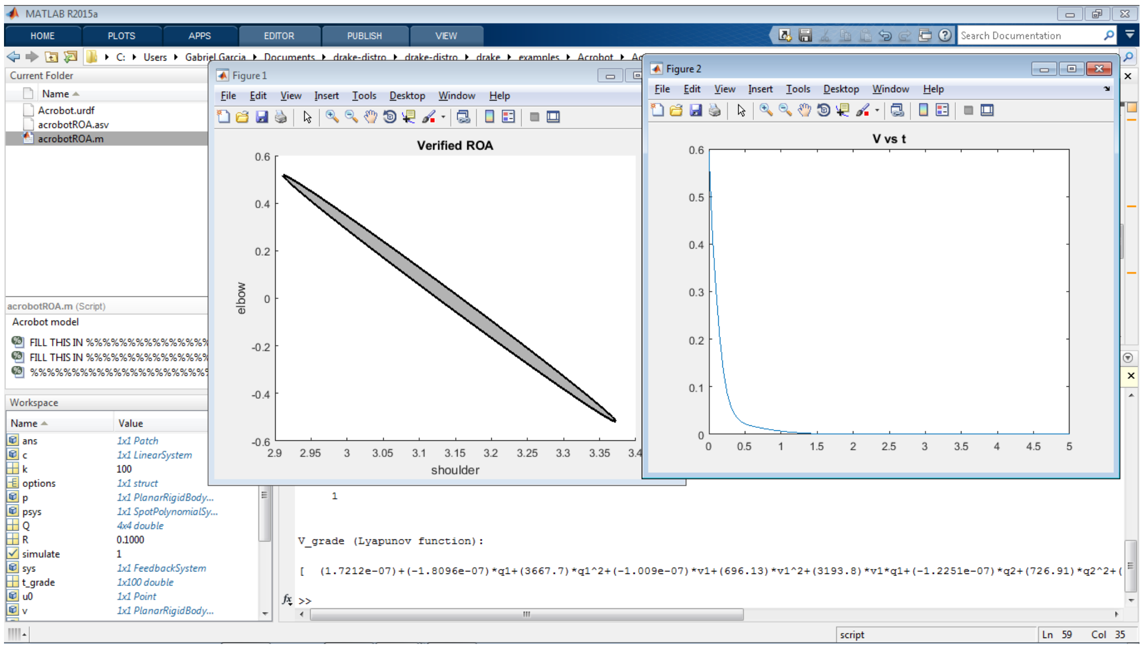
```

    disp('Doing taylor approx...');
    psys = taylorApprox(sys,0,x0,u0,3);  % Ta

    %% FILL THIS IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    options.degV = 2; % Set degree of Lyapuno
    options.degL1 = 2; % Set degrees of multi

    options.method = 'binary'; % 'sampling';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    options.converged_tol = 1e-7;
    options.max_iterations = 15;

```

```
     1


rho =

    6.1242


V_grade (Lyapunov function):

[  (8403.7)*q1^2+(1591.3)*v1^2+(7311.4)*v1*q1+(1715.9)*q2^2+(7575.2)*q2*q1+(3297.8)*q2*v1+(391.75)*v2^2+(3626.9)*v2*q1+(1578

Warning: MATLAB has disabled some advanced graphics rendering features by switching to software OpenGL. For more
information, click here.
>>
```

$$\rho = 6.1242$$

II. Acrobot Region of Attraction (Part II):

- File: “Acrobot ROA (Part II)/ acrobotROA.m” in folder “drake-distro\drake\examples\Acrobot”


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
C: ▸ Users ▸ Gabriel Garcia ▸ Documents ▸ drake-distro ▸ drake-distro ▸ drake ▸ examples ▸ Acrobot ▸ Ac
Current Folder
Name
Acrobot.urdf
acrobotROA.asv
acrobotROA.m
acrobotROA.m (Script)
Acrobot model
FILL THIS IN %%%%%%%%%%%%%%
FILL THIS IN %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
Figure 1
File Edit View Insert Tools Desktop Window Help
Verified ROA
elbow
shoulder
Figure 2
File Edit View Insert Tools Desktop Window Help
V vs t
script
Ln 59 Col 35

Workspace

Name	Value
ans	1x1 Patch
c	1x1 LinearSystem
k	100
options	1x1 struct
p	1x1 PlanarRigidBody...
psys	1x1 SpotPolynomialSy...
Q	4x4 double
R	0.1000
simulate	1
sys	1x1 FeedbackSystem
t_grade	1x100 double
u0	1x1 Point
v	1x1 PlanarRigidBody...

```
     1


V_grade (Lyapunov function):

[   (1.7212e-07)+(-1.8096e-07)*q1+(3667.7)*q1^2+(-1.009e-07)*v1+(696.13)*v1^2+(3193.8)*v1*q1+(-1.2251e-07)*q2+(726.91)*q2^2+(

>>
```

The function plotFunnel plots the one-level set of V. We can pick a value inside of the gray region. We chose $x_0 = (3{,}0.3{,}0{,}0)$ and we can see the decay of V to zero.

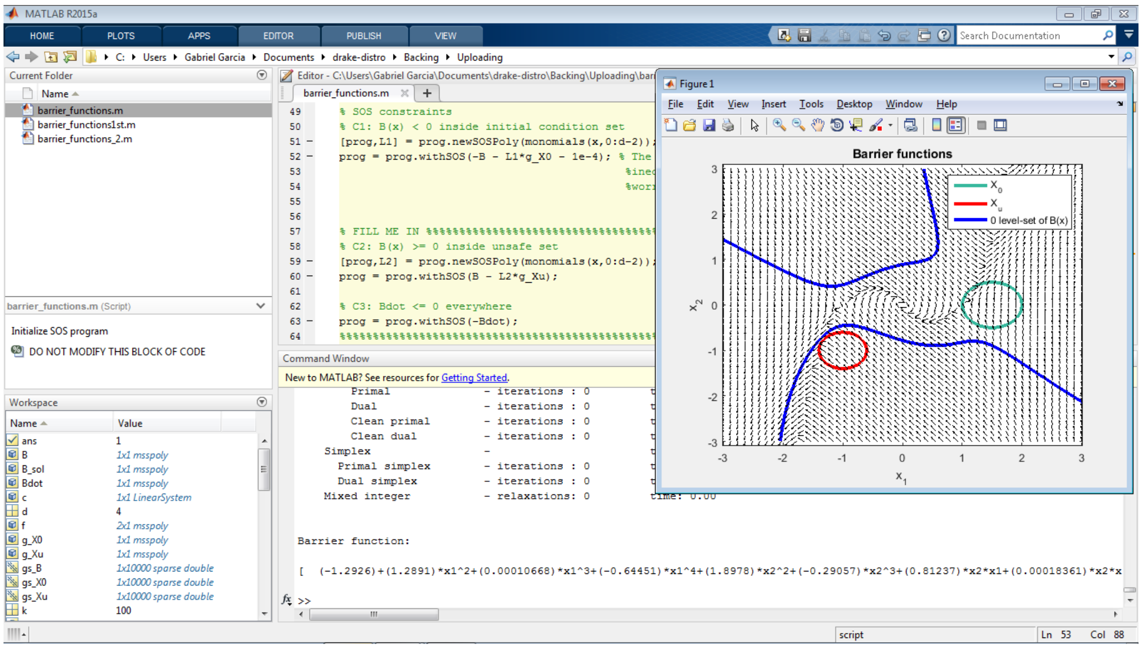
III. Barrier Functions:

- File “barrier_functions.m” in folder “drake-distro\drake\examples\Acrobot”
-

We have the polynomial dynamical system:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_1 + x_1^3 - x_2$$

We implement the conditions for the quadratic program to solve.



```
% SOS constraints
% C1: B(x) < 0 inside initial condition set
[prog,L1] = prog.newSOSPoly(monomials(x,0:d-2));
prog = prog.withSOS(-B - L1*g_X0 - 1e-4); % The
                                           %ined
                                           %worr


% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% C2: B(x) >= 0 inside unsafe set
[prog,L2] = prog.newSOSPoly(monomials(x,0:d-2));
prog = prog.withSOS(B - L2*g_Xu);

% C3: Bdot <= 0 everywhere
prog = prog.withSOS(-Bdot);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        Primal                 - iterations : 0
        Dual                   - iterations : 0
        Clean primal           - iterations : 0
        Clean dual             - iterations : 0
    Simplex                    -
      Primal simplex           - iterations : 0
      Dual simplex             - iterations : 0
    Mixed integer              - relaxations: 0          time: 0.00


Barrier function:

[  (-1.2926)+(1.2891)*x1^2+(0.00010668)*x1^3+(-0.64451)*x1^4+(1.8978)*x2^2+(-0.29057)*x2^3+(0.81237)*x2*x1+(0.00018361)*x2*x

>>
```

MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
C: ▸ Users ▸ Gabriel Garcia ▸ Documents ▸ drake-distro ▸ Backing ▸ Uploading
Current Folder
Name
barrier_functions.m
barrier_functions1st.m
barrier_functions_2.m
barrier_functions.m (Script)
Initialize SOS program
DO NOT MODIFY THIS BLOCK OF CODE
Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\Backing\Uploading\bar
barrier_functions.m
Command Window
New to MATLAB? See resources for Getting Started.
Figure 1
File Edit View Insert Tools Desktop Window Help
Barrier functions
X0
Xu
0 level-set of B(x)
x1
x2
script
Ln 53 Col 88

Name	Value
ans	1
B	1x1 msspoly
B_sol	1x1 msspoly
Bdot	1x1 msspoly
c	1x1 LinearSystem
d	4
f	2x1 msspoly
g_X0	1x1 msspoly
g_Xu	1x1 msspoly
gs_B	1x10000 sparse double
gs_X0	1x10000 sparse double
gs_Xu	1x10000 sparse double
k	100

Basically, the stub code is looking for quartic polynomials for the barrier function B, and quadratic polynomials for λ lagrange multipliers. Note that $\lambda \geq 0 \ \forall x \in \mathbb{R}^2$

Setting $-B - \lambda_1 g_{X0} - \varepsilon$ SOS means:

$$-B - \lambda_1 g_{X0} - \varepsilon \geq 0 \rightarrow -\lambda_1 g_{X0} - \varepsilon \geq B$$

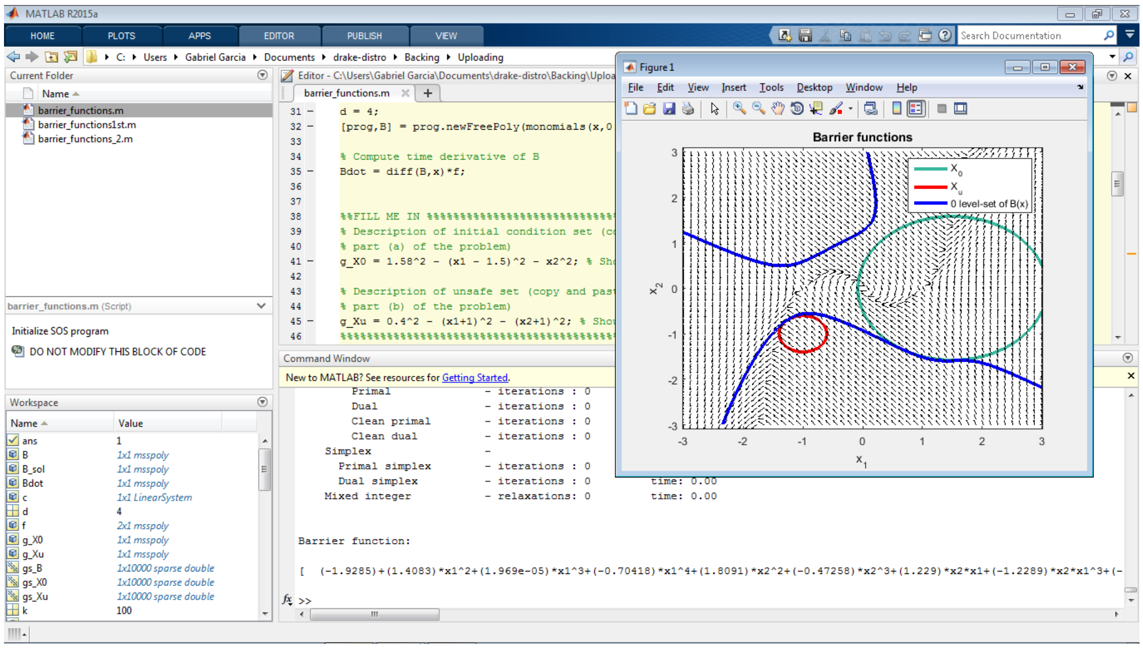
In particular, in the region X_0 we have:

$$g_{X0} \geq 0 \rightarrow 0 \geq -\lambda_1 g_{X0}$$

$$-\varepsilon \geq -\lambda_1 g_{X0} - \varepsilon \geq B$$

So we have $-\varepsilon \geq B$, or in particular, $0 > B(x) \ \forall x \in X_0$. Let's remember that for good performance, we need that the degree of B will be the same as the degree of $\lambda_1 g_{X0}$. Given that g is a quadratic polynomial, the degree of λ_1 is two less than the degree of B. This applies, for higher even degrees of B. The same applies for SOS in the region X_u.

- Increase the radius of the initial condition set (without changing the point about which it is centered) to see how large you can make it and still guarantee that trajectories that start in it remain safe:



Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\Backing\Uploa

barrier_functions.m

```
    d = 4;
    [prog,B] = prog.newFreePoly(monomials(x,0

    % Compute time derivative of B
    Bdot = diff(B,x)*f;


    %%FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Description of initial condition set (co
    % part (a) of the problem)
    g_X0 = 1.58^2 - (x1 - 1.5)^2 - x2^2; % Sho

    % Description of unsafe set (copy and past
    % part (b) of the problem)
    g_Xu = 0.4^2 - (x1+1)^2 - (x2+1)^2; % Show
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Workspace

Name	Value
ans	1
B	1x1 msspoly
B_sol	1x1 msspoly
Bdot	1x1 msspoly
c	1x1 LinearSystem
d	4
f	2x1 msspoly
g_X0	1x1 msspoly
g_Xu	1x1 msspoly
gs_B	1x10000 sparse double
gs_X0	1x10000 sparse double
gs_Xu	1x10000 sparse double
k	100

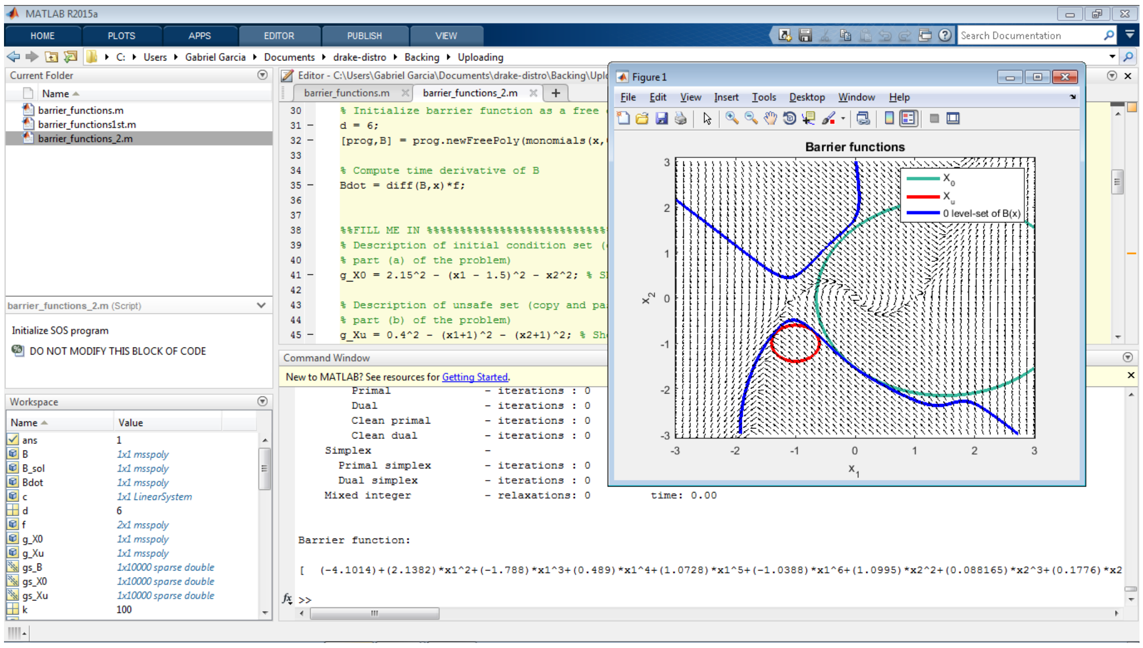
Command Window

New to MATLAB? See resources for Getting Started.

```
        Primal                - iterations : 0
        Dual                  - iterations : 0
        Clean primal          - iterations : 0
        Clean dual            - iterations : 0
    Simplex                   -
      Primal simplex          - iterations : 0
      Dual simplex            - iterations : 0          time: 0.00
    Mixed integer             - relaxations: 0          time: 0.00


Barrier function:

[   (-1.9285)+(1.4083)*x1^2+(1.969e-05)*x1^3+(-0.70418)*x1^4+(1.8091)*x2^2+(-0.47258)*x2^3+(1.229)*x2*x1+(-1.2289)*x2*x1^3+(-

>>
```

$$r_{max} = 1.58$$

If we increase the degree of B, we can get greater values of the maximum radius for the initial conditions. This is because the degree of a multivalued polynomial is related with the numbers of "folding" or inflexion points for the 0-level set of $B(x)$. We get a $r_{max} = 2.15$ for $d = 6$. The new degree of λ is 4.



Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\Backing\Uplo

barrier_functions.m | barrier_functions_2.m

```
    % Initialize barrier function as a free
    d = 6;
    [prog,B] = prog.newFreePoly(monomials(x,

    % Compute time derivative of B
    Bdot = diff(B,x)*f;


    %%FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Description of initial condition set (
    % part (a) of the problem)
    g_X0 = 2.15^2 - (x1 - 1.5)^2 - x2^2; % Sh

    % Description of unsafe set (copy and pa
    % part (b) of the problem)
    g_Xu = 0.4^2 - (x1+1)^2 - (x2+1)^2; % Sh
```

Workspace

Name	Value
ans	1
B	1x1 msspoly
B_sol	1x1 msspoly
Bdot	1x1 msspoly
c	1x1 LinearSystem
d	6
f	2x1 msspoly
g_X0	1x1 msspoly
g_Xu	1x1 msspoly
gs_B	1x10000 sparse double
gs_X0	1x10000 sparse double
gs_Xu	1x10000 sparse double
k	100

Command Window

New to MATLAB? See resources for Getting Started.

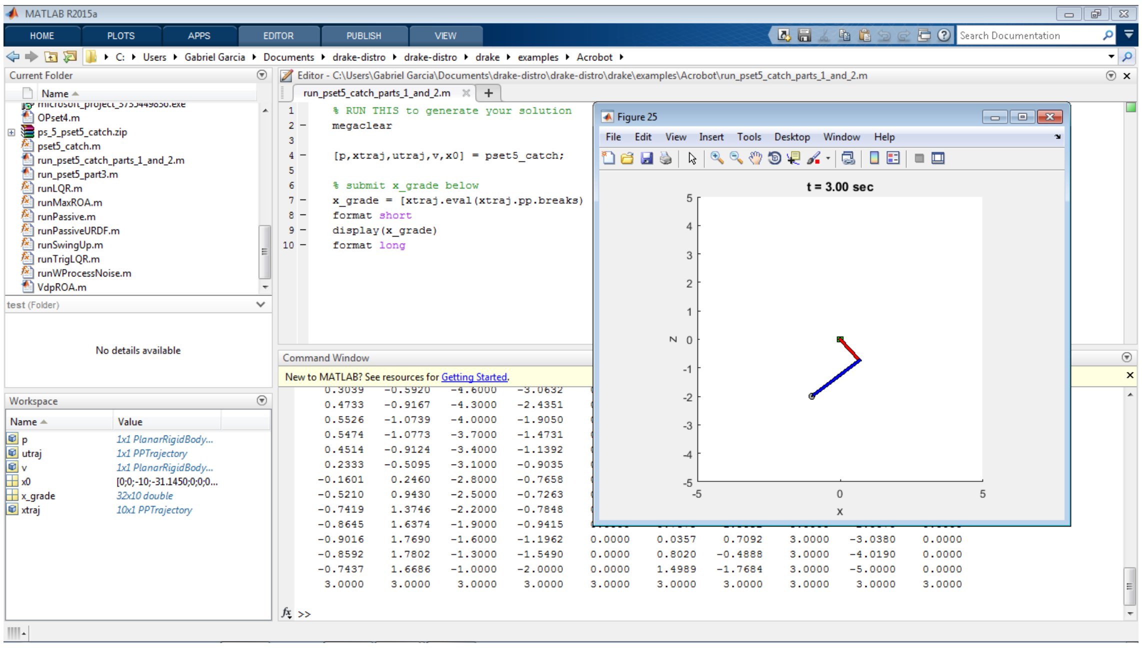
```
        Primal                - iterations : 0
        Dual                  - iterations : 0
        Clean primal          - iterations : 0
        Clean dual            - iterations : 0
    Simplex                   -
      Primal simplex          - iterations : 0
      Dual simplex            - iterations : 0
    Mixed integer             - relaxations: 0          time: 0.00


Barrier function:

[   (-4.1014)+(2.1382)*x1^2+(-1.788)*x1^3+(0.489)*x1^4+(1.0728)*x1^5+(-1.0388)*x1^6+(1.0995)*x2^2+(0.088165)*x2^3+(0.1776)*x2

>>
```

PSET5:

- files inside: "PSET5/" in folder "drake-distro\drake\examples\Acrobot"

I. Catch the Falling Ball:

Part 1: We initialize randomly the values of x_init_vec and u_init_vec, and Direct Collocation will do the work. We also set the final state condition, this is, the position of the ball (x_3, x_4), and the extreme of the acrobot $\left(hand_{pos}(1), hand_{pos}(2)\right)$ must match in both components. We also take the gradient.



```
% RUN THIS to generate your solution
megaclear

[p,xtraj,utraj,v,x0] = pset5_catch;

% submit x_grade below
x_grade = [xtraj.eval(xtraj.pp.breaks)
format short
display(x_grade)
format long
```

```
    0.3039   -0.5920   -4.6000   -3.0632
    0.4733   -0.9167   -4.3000   -2.4351
    0.5526   -1.0739   -4.0000   -1.9050
    0.5474   -1.0773   -3.7000   -1.4731
    0.4514   -0.9124   -3.4000   -1.1392
    0.2333   -0.5095   -3.1000   -0.9035
   -0.1601    0.2460   -2.8000   -0.7658
   -0.5210    0.9430   -2.5000   -0.7263
   -0.7419    1.3746   -2.2000   -0.7848
   -0.8645    1.6374   -1.9000   -0.9415
   -0.9016    1.7690   -1.6000   -1.1962    0.0000    0.0357    0.7092    3.0000   -3.0380    0.0000
   -0.8592    1.7802   -1.3000   -1.5490    0.0000    0.8020   -0.4888    3.0000   -4.0190    0.0000
   -0.7437    1.6686   -1.0000   -2.0000    0.0000    1.4989   -1.7684    3.0000   -5.0000    0.0000
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000

>>
```

Name	Value
p	1x1 PlanarRigidBody...
utraj	1x1 PPTrajectory
v	1x1 PlanarRigidBody...
x0	[0;0;-10;-31.1450;0;0;0...
x_grade	32x10 double
xtraj	10x1 PPTrajectory

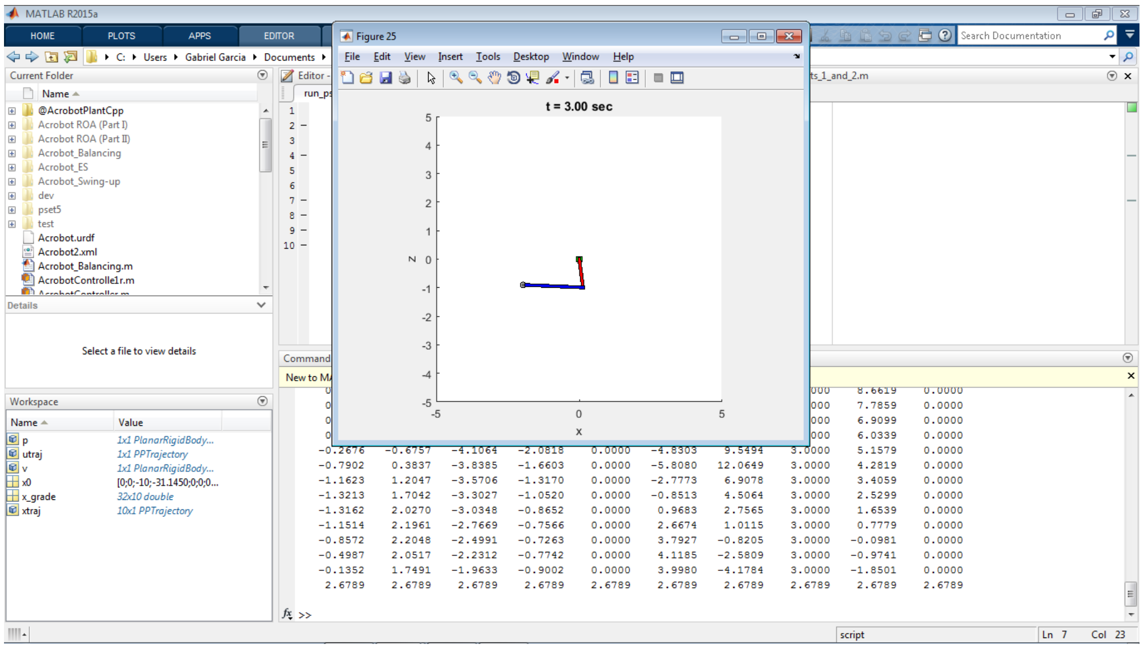
Part 2: We must add an expression to the final cost that represents the height of the ball. The simplest way to do it is set the expression to:

$$f = -kz_{ball} = -kx(4)$$

So we have as cost:

$$J = -kz_{ball}(T) + \int_0^T u^2 dt$$

Note the negative sign, regarding the higher point of the trajectory. We set k to 500.



Name	Value
p	1x1 PlanarRigidBody...
utraj	1x1 PPTrajectory
v	1x1 PlanarRigidBody...
x0	[0;0;-10;-31.1450;0;0;0...
x_grade	32x10 double
xtraj	10x1 PPTrajectory

```
         0                                                                         000    8.6619    0.0000
         0                                                                         000    7.7859    0.0000
         0                                                                         000    6.9099    0.0000
         0                                                                         000    6.0339    0.0000
   -0.2676   -0.6757   -4.1064   -2.0818    0.0000   -4.8303    9.5494    3.0000    5.1579    0.0000
   -0.7902    0.3837   -3.8385   -1.6603    0.0000   -5.8080   12.0649    3.0000    4.2819    0.0000
   -1.1623    1.2047   -3.5706   -1.3170    0.0000   -2.7773    6.9078    3.0000    3.4059    0.0000
   -1.3213    1.7042   -3.3027   -1.0520    0.0000   -0.8513    4.5064    3.0000    2.5299    0.0000
   -1.3162    2.0270   -3.0348   -0.8652    0.0000    0.9683    2.7565    3.0000    1.6539    0.0000
   -1.1514    2.1961   -2.7669   -0.7566    0.0000    2.6674    1.0115    3.0000    0.7779    0.0000
   -0.8572    2.2048   -2.4991   -0.7263    0.0000    3.7927   -0.8205    3.0000   -0.0981    0.0000
   -0.4987    2.0517   -2.2312   -0.7742    0.0000    4.1185   -2.5809    3.0000   -0.9741    0.0000
   -0.1352    1.7491   -1.9633   -0.9002    0.0000    3.9980   -4.1784    3.0000   -1.8501    0.0000
    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789

fx >>
```

Part 3:

We have the cost we wish to add:

$$f(x) = \frac{1}{2}\left(\left(x_{ball} - hand_{pos}(1)\right)^2 + \left(z_{ball} - hand_{pos}(2)\right)^2\right)$$

This is, the distance from the ball to the hand scaled. There is a nonlinear term there, so we can make a taylor expansion and pick the quadratic term. The problem now is chose the value of the linearization point. We know the trajectory computed by Direct Collocation, so we simply evaluate that trajectory at the time when the acrobat catch the falling ball. That will be the linearization point.

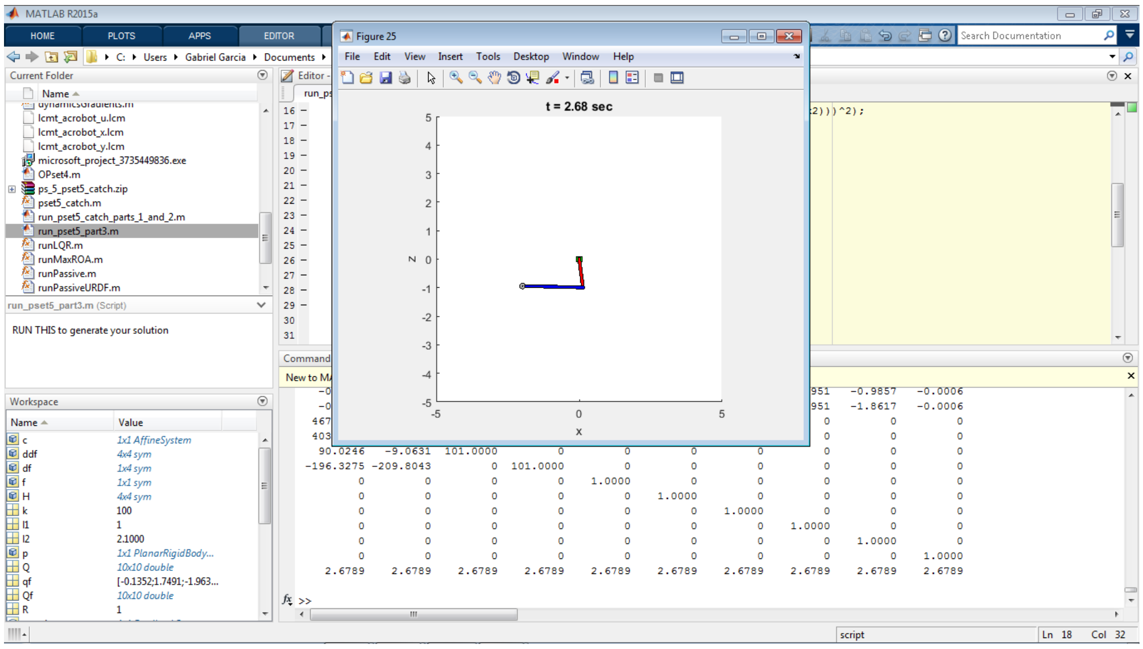
$$x_f = x_{traj}(t_f) \text{ in part. } t_f = 3$$

$$f(x) \approx f(x_f) + \left.\frac{\partial f}{\partial x^T}\right|_{x=x_f}(x - x_f) + \frac{1}{2}(x - x_f)^T \left.\frac{\partial^2 f}{\partial x^2}\right|_{x=x_f}(x - x_f)$$

$$f = \left(x_3 - hand_{pos}(1)\right)^2 + \left(x_4 - hand_{pos}(2)\right)^2$$

We have $f(x_f) = \left.\frac{\partial f}{\partial x^T}\right|_{x=x_f} = 0$, so we can take $\left.\frac{\partial^2 f}{\partial x^2}\right|_{x=x_f}$ as the cost Q_f scaled by a constant k, in our case $k = 100$.

Note that the Time Variant LQR solver ask you to change some extra code if we left Q_f as positive semidefinite, so we just add an Identity matrix to run directly the code.



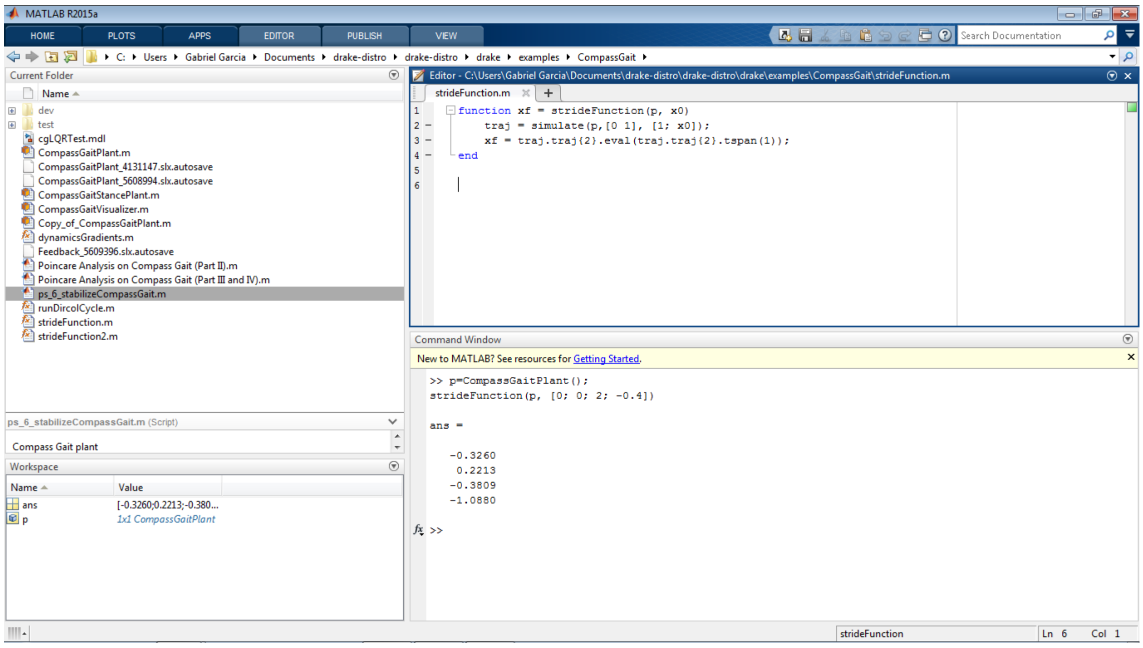
RUN THIS to generate your solution

```
  -0                                                              951   -0.9857   -0.0006
  -0                                                              951   -1.8617   -0.0006
 467                                                                0         0         0
 403                                                                0         0         0
   90.0246   -9.0631  101.0000         0         0         0         0         0         0         0
 -196.3275 -209.8043         0  101.0000         0         0         0         0         0         0
         0         0         0         0    1.0000         0         0         0         0         0
         0         0         0         0         0    1.0000         0         0         0         0
         0         0         0         0         0         0    1.0000         0         0         0
         0         0         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0    1.0000
    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789    2.6789

>>
```

PSET6:

- files inside: “PSET6/” in folder “drake-distro\drake\examples\CompassGait”

I. Poincare Analysis (Compass Gait) Part I:
We define a Stride Function, which will take the role of the Poincare section for finding a limit cycle for a passive Compass Gait. We take advantage of the fact that the simulation of the system is a hybrid trajectory, redefined when occurs a guard event.

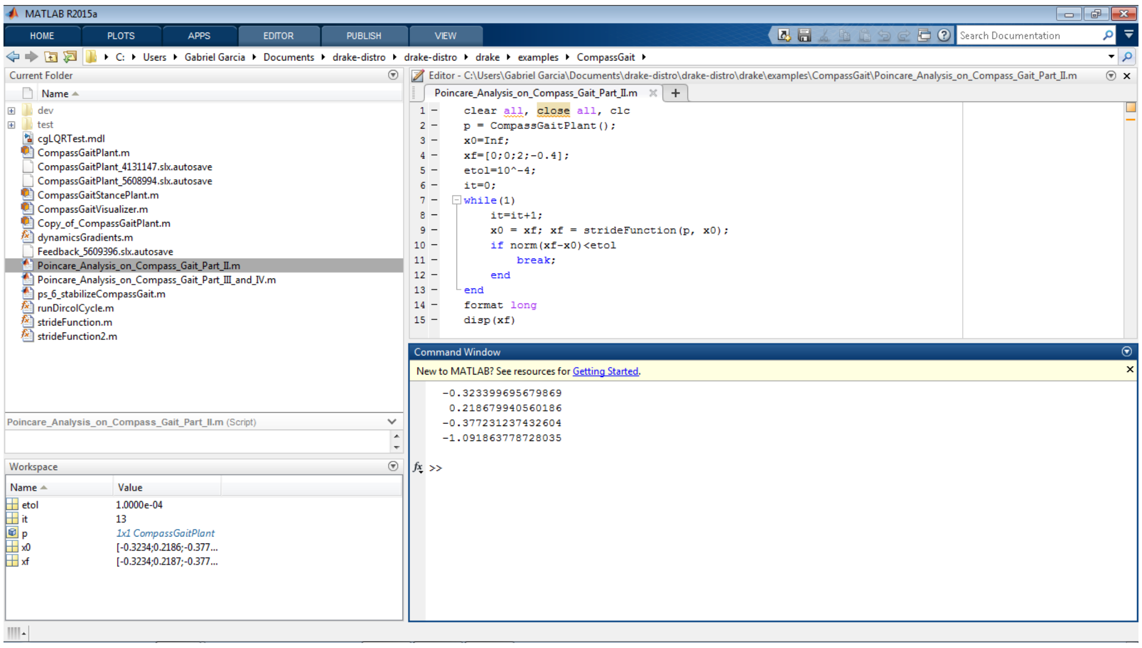


```
function xf = strideFunction(p, x0)
    traj = simulate(p,[0 1], [1; x0]);
    xf = traj.traj{2}.eval(traj.traj{2}.tspan(1));
end
```

```
>> p=CompassGaitPlant();
strideFunction(p, [0; 0; 2; -0.4])

ans =

   -0.3260
    0.2213
   -0.3809
   -1.0880

>>
```

II. Poincare Analysis (Compass Gait) Part II:

We find a limit cycle, repeating the stride function, until the state of the system will not change.



```
clear all, close all, clc
p = CompassGaitPlant();
x0=Inf;
xf=[0;0;2;-0.4];
etol=10^-4;
it=0;
while(1)
    it=it+1;
    x0 = xf; xf = strideFunction(p, x0);
    if norm(xf-x0)<etol
        break;
    end
end
format long
disp(xf)
```

```
  -0.323399695679869
   0.218679940560186
  -0.377231237432604
  -1.091863778728035

>>
```

Workspace

Name	Value
etol	1.0000e-04
it	13
p	1x1 CompassGaitPlant
x0	[-0.3234;0.2186;-0.377...
xf	[-0.3234;0.2187;-0.377...

III. Poincare Analysis (Compass Gait) Part III and IV:

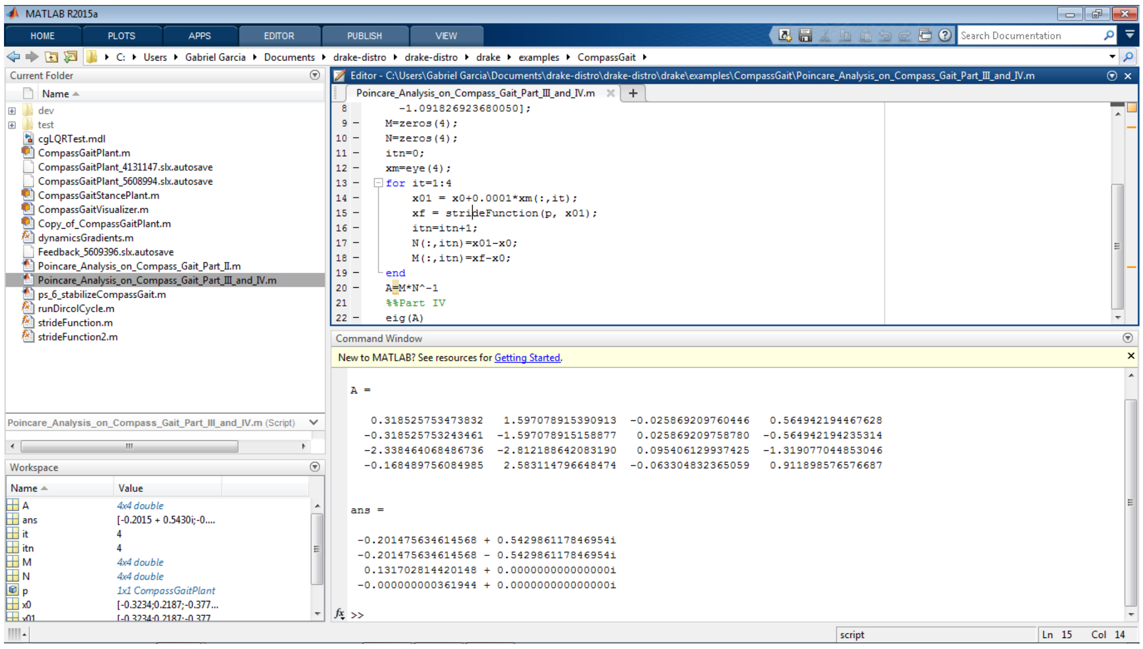
In order to compute the matrix of the linearization of the Poincare map around the "initial" point of the limit cycle, we take a look at the map:

$$x[n+1] = P(x[n]), x^* = P(x^*)$$

$$x[n+1] \approx x^* + \frac{\partial P}{\partial x}(x[n] - x^*)$$

$$x[n+1] - x^* \approx A(x[n] - x^*)$$

If we simulate the system for a lot of values around the fixed point, we can solve the last equation considering A as an approximation. Look that we need at least 4 points in different directions, and if we take more than that, we can find an A by using an pseudoinverse matrix. For now, we use 4 points in different directions around the fixed point. We also take the eigenvalues of A, realizing that the magnitude of them is less than 1, implying that the limit cycle is stable.



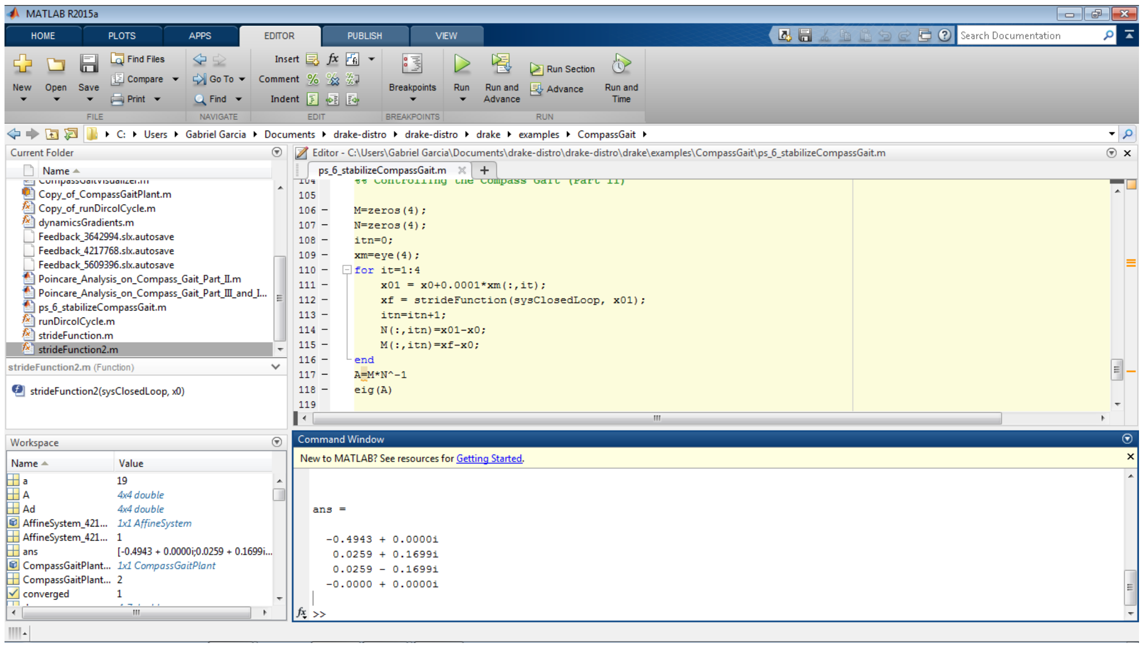
```
        -1.091826923680050];
    M=zeros(4);
    N=zeros(4);
    itn=0;
    xm=eye(4);
    for it=1:4
        x01 = x0+0.0001*xm(:,it);
        xf = strideFunction(p, x01);
        itn=itn+1;
        N(:,itn)=x01-x0;
        M(:,itn)=xf-x0;
    end
    A=M*N^-1
    %%Part IV
    eig(A)
```

```
A =

   0.318525753473832   1.597078915390913  -0.025869209760446   0.564942194467628
  -0.318525753243461  -1.597078915158877   0.025869209758780  -0.564942194235314
  -2.338464068486736  -2.812188642083190   0.095406129937425  -1.319077044853046
  -0.168489756084985   2.583114796648474  -0.063304832365059   0.911898576576687


ans =

 -0.201475634614568 + 0.542986117846954i
 -0.201475634614568 - 0.542986117846954i
  0.131702814420148 + 0.000000000000000i
 -0.000000000361944 + 0.000000000000000i
```

IV. Optional: Controlling the Compass Gait:

- File: “ps_6_stabilizeCompassGait.m”

Floquet Multipliers of closed-loop system:



```
    %% Controlling the Compass Gait (Part II)

    M=zeros(4);
    N=zeros(4);
    itn=0;
    xm=eye(4);
    for it=1:4
        x01 = x0+0.0001*xm(:,it);
        xf = strideFunction(sysClosedLoop, x01);
        itn=itn+1;
        N(:,itn)=x01-x0;
        M(:,itn)=xf-x0;
    end
    A=M*N^-1
    eig(A)
```

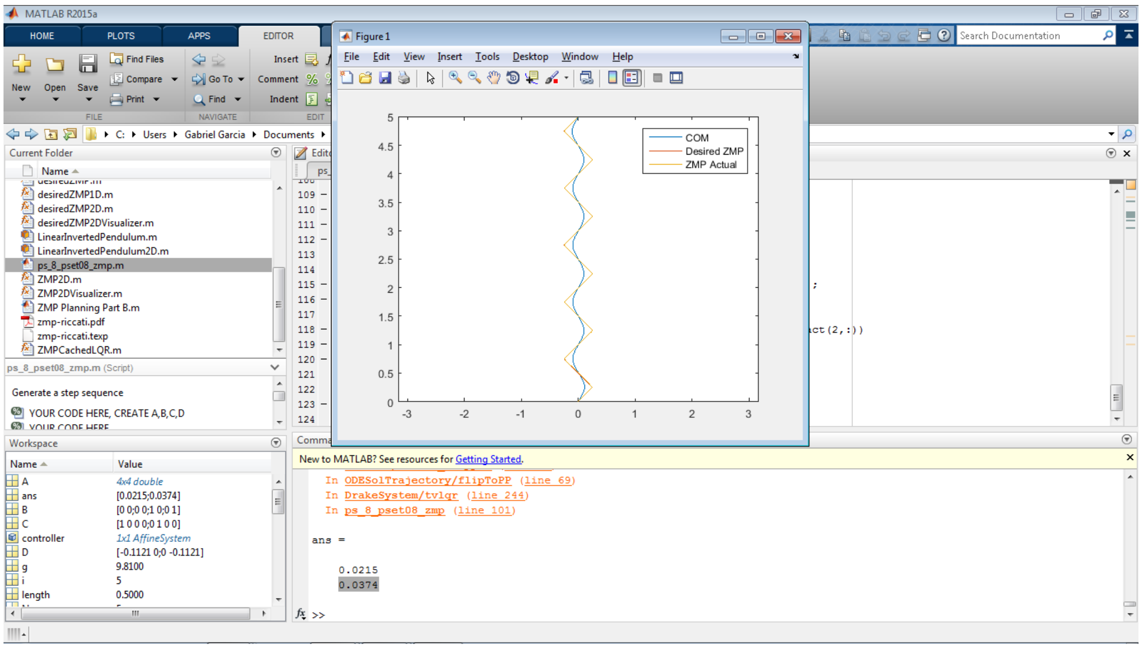
```
ans =

  -0.4943 + 0.0000i
   0.0259 + 0.1699i
   0.0259 - 0.1699i
  -0.0000 + 0.0000i
```

PSET7:

- Files “PSET7/” in folder “drake-distro\drake\examples\ZMP”

I. ZMP Part C:

- Run “ps_8_pset08_zmp” (it’s only the name misspelled, it must be ps_7_pset07_zmp)



Current Folder

```
Name
desiredZMP1D.m
desiredZMP2D.m
desiredZMP2DVisualizer.m
LinearInvertedPendulum.m
LinearInvertedPendulum2D.m
ps_8_pset08_zmp.m
ZMP2D.m
ZMP2DVisualizer.m
ZMP Planning Part B.m
zmp-riccati.pdf
zmp-riccati.texp
ZMPCachedLQR.m
```

ps_8_pset08_zmp.m (Script)

Generate a step sequence

YOUR CODE HERE, CREATE A,B,C,D

Workspace

Name	Value
A	4x4 double
ans	[0.0215;0.0374]
B	[0 0;0 0;1 0;0 1]
C	[1 0 0 0;0 1 0 0]
controller	1x1 AffineSystem
D	[-0.1121 0;0 -0.1121]
g	9.8100
i	5
length	0.5000

New to MATLAB? See resources for Getting Started.

```
  In ODESolTrajectory/flipToPP (line 69)
  In DrakeSystem/tvlqr (line 244)
  In ps_8_pset08_zmp (line 101)

ans =

    0.0215
    0.0374

>>
```

PSET8:

- Files "PSET8" in folder "drake-distro\drake\examples\PSET8"

II. Force Closure:

Let's note that is unnecessary to put the condition $f_{i,z} > 0$ as a restriction. In fact, it is a consequence of:

$$|f_{i,x}| < \mu f_{i,z} \forall i$$

$$f_{i,z} > \frac{1}{\mu}|f_{i,x}| \geq 0 \rightarrow f_{i,z} > 0$$

We are going to minimize the slack variable $\gamma = c^T z \leq 0$, with

$$z = \begin{bmatrix} \gamma \\ f \end{bmatrix}, c = \begin{bmatrix} 1 \\ \mathbf{0_{2n}} \end{bmatrix}$$

The solution is going to be less than zero iff the conditions for force closure holds. If there is no force closure the solution will be 0.

So for now it's enough to set the following conditions:

$$|f_{i,x}| < \mu f_{i,z} \forall i \leftrightarrow \gamma < 0$$

Suppose we set:

$$\mu f_{i,z} - |f_{i,x}| > -\gamma \geq 0 \ \forall i \leftrightarrow \gamma < 0$$

$$\mu f_{i,z} - |f_{i,x}| > -\gamma \geq 0 \ \forall i \leftrightarrow \gamma < 0$$

Nota that if $\gamma = 0$ we have a contradiction so there is no solution for $|f_{i,x}| < \mu f_{i,z} \forall i$, this is what we are looking for.

$$0 > -\gamma \pm f_{i,x} - \mu f_{i,z} \; \forall i$$

So we wish to have: $Cz \leq d$. So we set:

$$C = \begin{bmatrix} -\mathbf{1}_{2n\times 1} & \begin{matrix} I_n \otimes [1 & -\mu] \\ I_n \otimes [-1 & -\mu] \end{matrix} \\ -1 & \mathbf{0}_{1\times 2n} \end{bmatrix}, d = \begin{bmatrix} \mathbf{0}_{2n\times 1} \\ 10 \end{bmatrix}$$

Where the column of $\mathbf{1}_{2n\times 1}$ is for take the value of γ, the Kronecker product is done in order to make a simplification for the condition $\pm f_{i,x} - \mu f_{i,z}$ for all i. (Positive and Negative values of $f_{i,x}$ are used instead of $|f_{i,x}|$ for linearity)

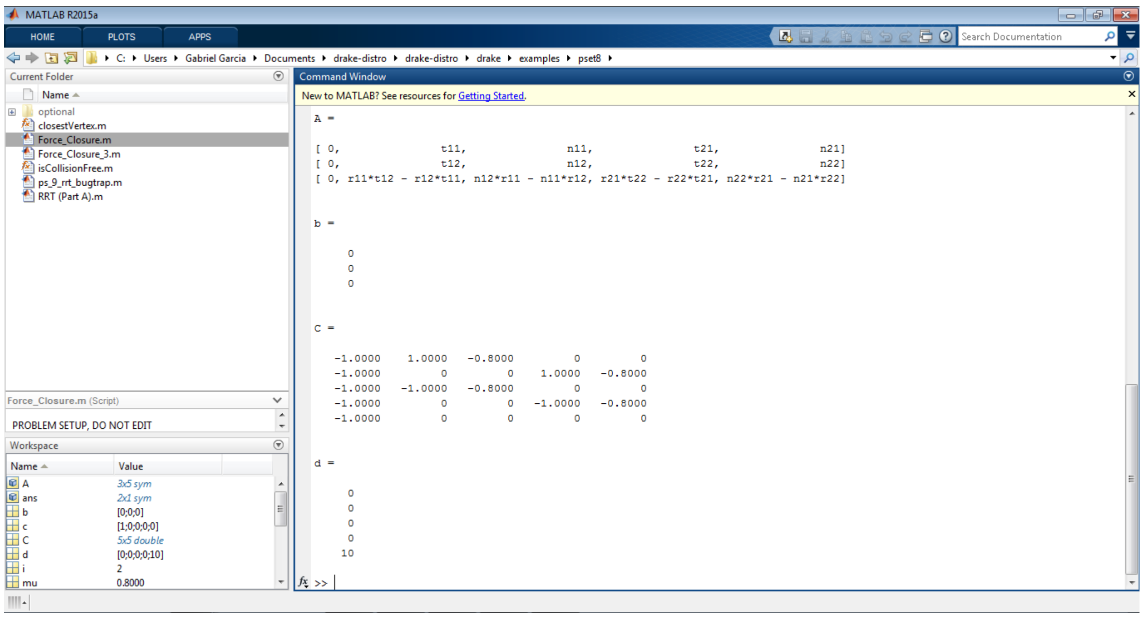
Respect to the equalities $Az = b$, we have:

$$A = G$$
$$b = \mathbf{0_3}$$

We can write for simplicity:

$$G = [\mathbf{0_3} \quad r_1^a \times t_1^a + t_1^a \quad r_1^a \times n_1^a + n_1^a \quad \cdots \quad r_n^a \times t_n^a + t_n^a \quad r_n^a \times n_n^a + n_n^a]$$

Where the superscript "a" means augment one component to the vector (the new component is 0, and the new vectors will have dimension 3).


MATLAB R2015a
HOME
PLOTS
APPS
Search Documentation
C: ▸ Users ▸ Gabriel Garcia ▸ Documents ▸ drake-distro ▸ drake-distro ▸ drake ▸ examples ▸ pset8
Current Folder
Name
Force_Closure.m (Script)
Workspace
Command Window
New to MATLAB? See resources for Getting Started.

```
optional
closestVertex.m
Force_Closure.m
Force_Closure_3.m
isCollisionFree.m
ps_9_rrt_bugtrap.m
RRT (Part A).m
```

```
PROBLEM SETUP, DO NOT EDIT
```

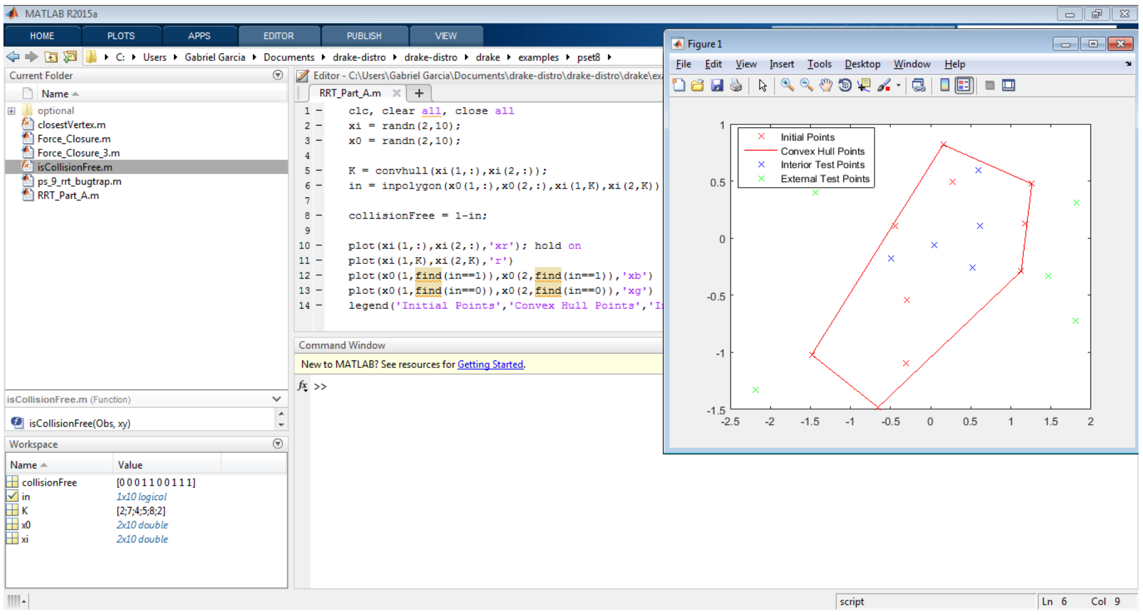
Name	Value
A	3x5 sym
ans	2x1 sym
b	[0;0;0]
c	[1;0;0;0;0]
C	5x5 double
d	[0;0;0;0;10]
i	2
mu	0.8000

```
A =

[ 0,               t11,               n11,               t21,               n21]
[ 0,               t12,               n12,               t22,               n22]
[ 0, r11*t12 - r12*t11, n12*r11 - n11*r12, r21*t22 - r22*t21, n22*r21 - n21*r22]


b =

     0
     0
     0


C =

   -1.0000    1.0000   -0.8000         0         0
   -1.0000         0         0    1.0000   -0.8000
   -1.0000   -1.0000   -0.8000         0         0
   -1.0000         0         0   -1.0000   -0.8000
   -1.0000         0         0         0         0


d =

     0
     0
     0
     0
    10

fx >>
```

III. RRT:

a) RRT (Part A)

- Run "RRT_Part_A.m"

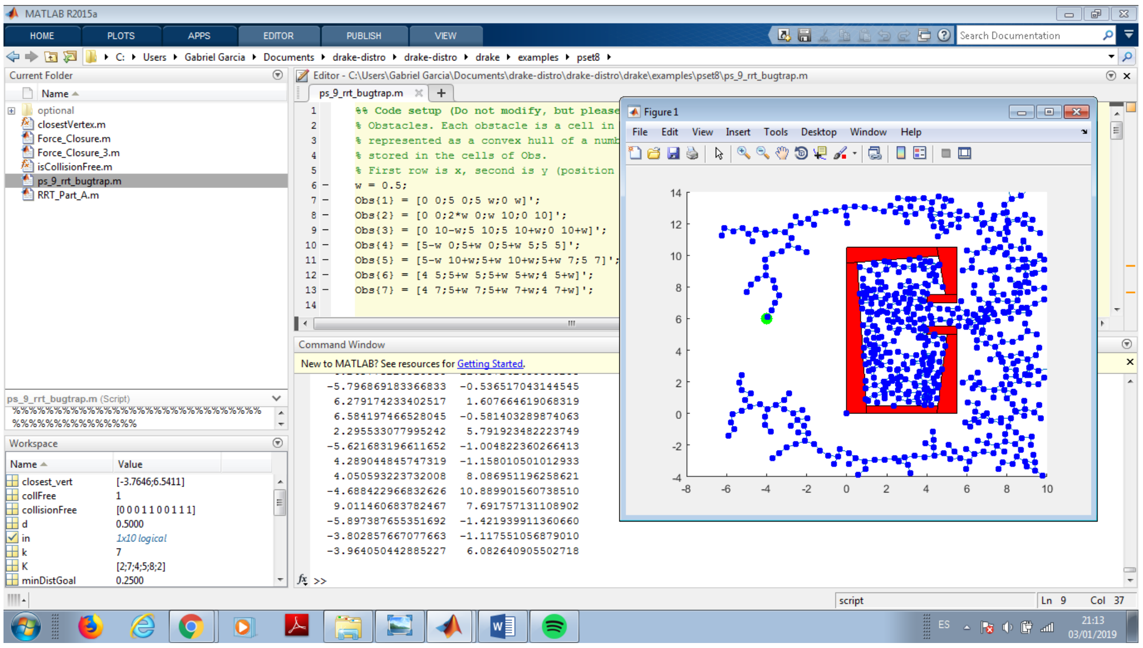


```
clc, clear all, close all
xi = randn(2,10);
x0 = randn(2,10);

K = convhull(xi(1,:),xi(2,:));
in = inpolygon(x0(1,:),x0(2,:),xi(1,K),xi(2,K))

collisionFree = 1-in;

plot(xi(1,:),xi(2,:),'xr'); hold on
plot(xi(1,K),xi(2,K),'r')
plot(x0(1,find(in==1)),x0(2,find(in==1)),'xb')
plot(x0(1,find(in==0)),x0(2,find(in==0)),'xg')
legend('Initial Points','Convex Hull Points','I
```

Name	Value
collisionFree	[0 0 0 1 1 0 0 1 1 1]
in	1x10 logical
K	[2;7;4;5;8;2]
x0	2x10 double
xi	2x10 double

b) RRT (Part B)

- Run “ps_9_rrt_bugtrap.m”



```
%% Code setup (Do not modify, but please
% Obstacles. Each obstacle is a cell in
% represented as a convex hull of a numb
% stored in the cells of Obs.
% First row is x, second is y (position
w = 0.5;
Obs{1} = [0 0;5 0;5 w;0 w]';
Obs{2} = [0 0;2*w 0;w 10;0 10]';
Obs{3} = [0 10-w;5 10;5 10+w;0 10+w]';
Obs{4} = [5-w 0;5+w 0;5+w 5;5 5]';
Obs{5} = [5-w 10+w;5+w 10+w;5+w 7;5 7]';
Obs{6} = [4 5;5+w 5;5+w 5+w;4 5+w]';
Obs{7} = [4 7;5+w 7;5+w 7+w;4 7+w]';
```

```
  -5.796869183366833  -0.536517043144545
   6.279174233402517   1.607664619068319
   6.584197466528045  -0.581403289874063
   2.295533077995242   5.791923482223749
  -5.621683196611652  -1.004822360266413
   4.289044845747319  -1.158010501012933
   4.050593223732008   8.086951196258621
  -4.688422966832626  10.889901560738510
   9.011460683782467   7.691757131108902
  -5.897387655351692  -1.421939911360660
  -3.802857667077663  -1.117551056879010
  -3.964050442885227   6.082640905502718
```

Name	Value
closest_vert	[-3.7646;6.5411]
collFree	1
collisionFree	[0 0 0 1 1 0 0 1 1 1]
d	0.5000
in	1x10 logical
k	7
K	[2;7;4;5;8;2]
minDistGoal	0.2500

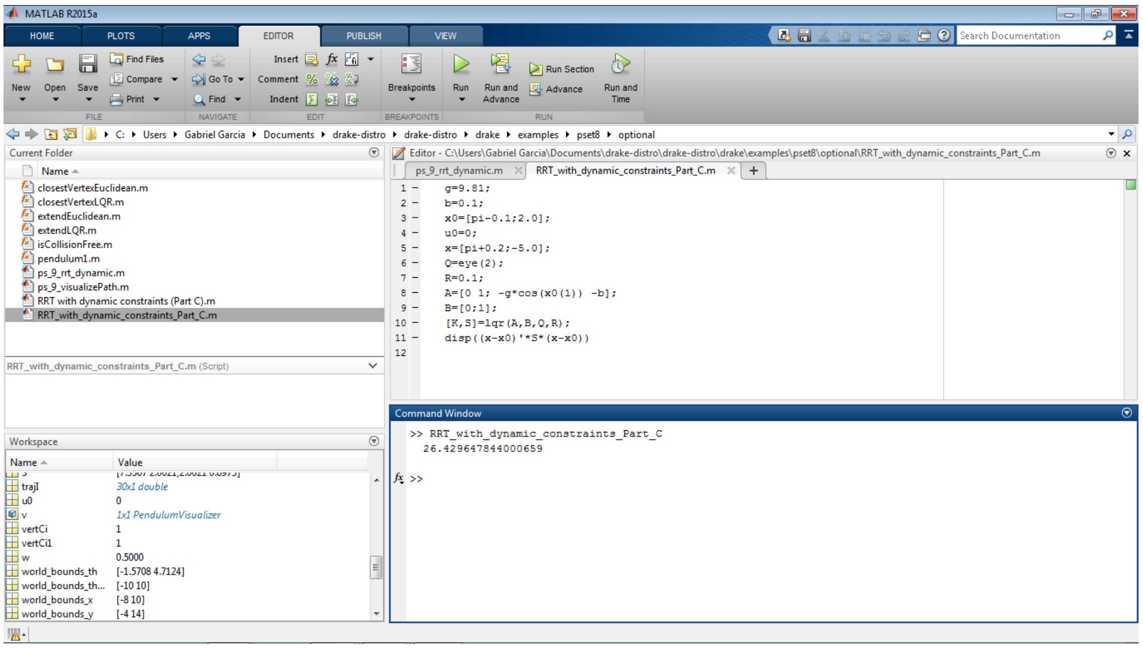
IV. Optional: RRT with dynamic constraints:

- Folder “Optional” inside pset8

a) RRT with dynamic constraints (Part A)

- Run “ps_9_rrt_dynamic.m” commenting line 29 “`%method = 'lqr'`” to avoid “`method = 'euclidean';`” be overwrited and run. Results is the same as Part D, but slower

b) RRT with dynamic constraints (Part C)

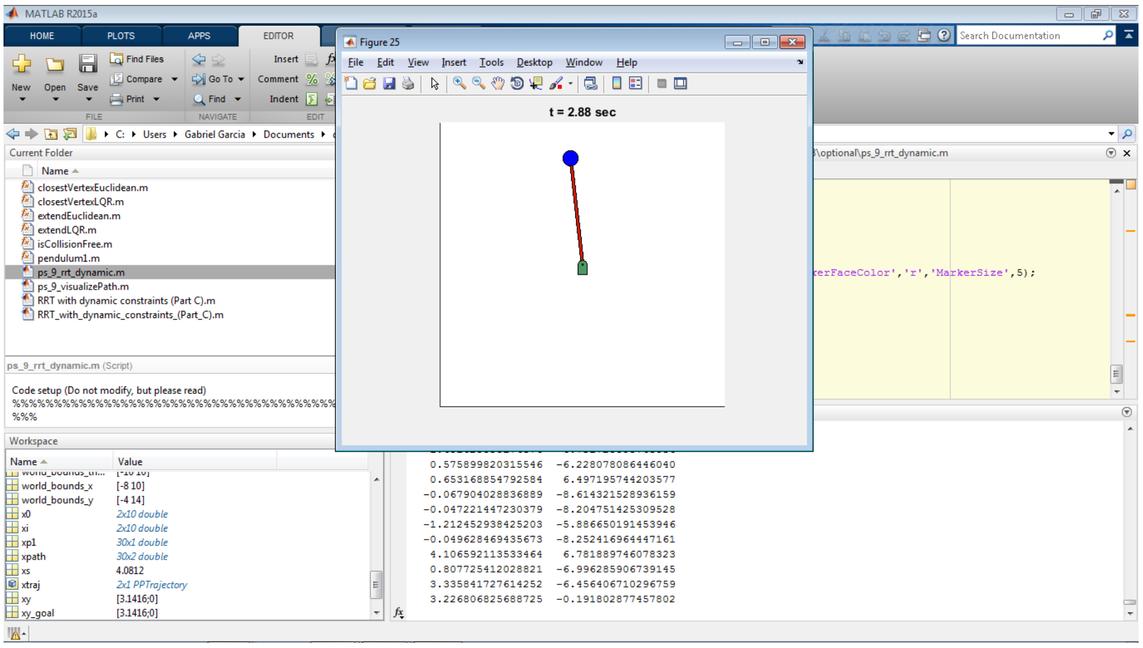


```
g=9.81;
b=0.1;
x0=[pi-0.1;2.0];
u0=0;
x=[pi+0.2;-5.0];
Q=eye(2);
R=0.1;
A=[0 1; -g*cos(x0(1)) -b];
B=[0;1];
[K,S]=lqr(A,B,Q,R);
disp((x-x0)'*S*(x-x0))
```

```
>> RRT_with_dynamic_constraints_Part_C
  26.429647844000659

>>
```

c) RRT with dynamic constraints (Part D)

- Run "ps_9_rrt_dynamic.m"



```
   0.575899820315546  -6.228078086446040
   0.653168854792584   6.497195744203577
  -0.067904028836889  -8.614321528936159
  -0.047221447230379  -8.204751425309528
  -1.212452938425203  -5.886650191453946
  -0.049628469435673  -8.252416964447161
   4.106592113533464   6.781889746078323
   0.807725412028821  -6.996285906739145
   3.335841727614252  -6.456406710296759
   3.226806825688725  -0.191802877457802
```

$$f(x,u) = \begin{bmatrix} x_2 \\ -g\sin x_1 - bx_2 + u \end{bmatrix}$$

$$A = \frac{\partial f}{\partial x^T} = \begin{bmatrix} 0 & 1 \\ -g\cos x_1 & -b \end{bmatrix}$$

$$B = \frac{\partial f}{\partial u} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

PSET 9:

- Files "PSET9" in folder "drake-distro\drake\examples\PSET9"
- Note: I particularly write this on my own using Kronecker product because is by far easier than wirting the full matrix. Results are the same as the answer provided in the course.

Model Predictive Control, Part 1

$$J = \sum_{k=1}^{N} (\|u[k-1]\|^2 + 0.01\|x[k]\|^2)$$

$$x[k+1] = Ax[k] + Bu[k]$$

$$x[0] = x_0$$

$$x[N] = \mathbf{0}^{4x1}$$

$$x[k+1] - Ax[k] - Bu[k] = 0$$

$$z = \begin{bmatrix} x[0] \\ \vdots \\ x[N] \\ u[0] \\ \vdots \\ u[N-1] \end{bmatrix} \in \mathbb{R}^{6N+4}$$

$$\begin{bmatrix} I_{4(N+1)} & 0_{2N} \end{bmatrix} z - \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ A & 0 & \cdots & 0 & 0 & B & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 & 0 & 0 & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A & 0 & 0 & 0 & \cdots & B \end{bmatrix} z = \begin{bmatrix} x_0 \\ 0_{4N} \end{bmatrix}$$

$$B_{MPC1} = \begin{bmatrix} I_{4(N+1)} & 0_{2N} \end{bmatrix} - \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ A & 0 & \cdots & 0 & 0 & B & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 & 0 & 0 & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A & 0 & 0 & 0 & \cdots & B \end{bmatrix}$$

$$c_1 = \begin{bmatrix} x_0 \\ 0_{4N} \end{bmatrix}$$

$$[0_{4N} \quad I_4 \quad 0_{2N}]z = [0_4]$$

$$B_{MPC2} = [0_{4N} \quad I_4 \quad 0_{2N}]$$

$$c_2 = 0_4$$

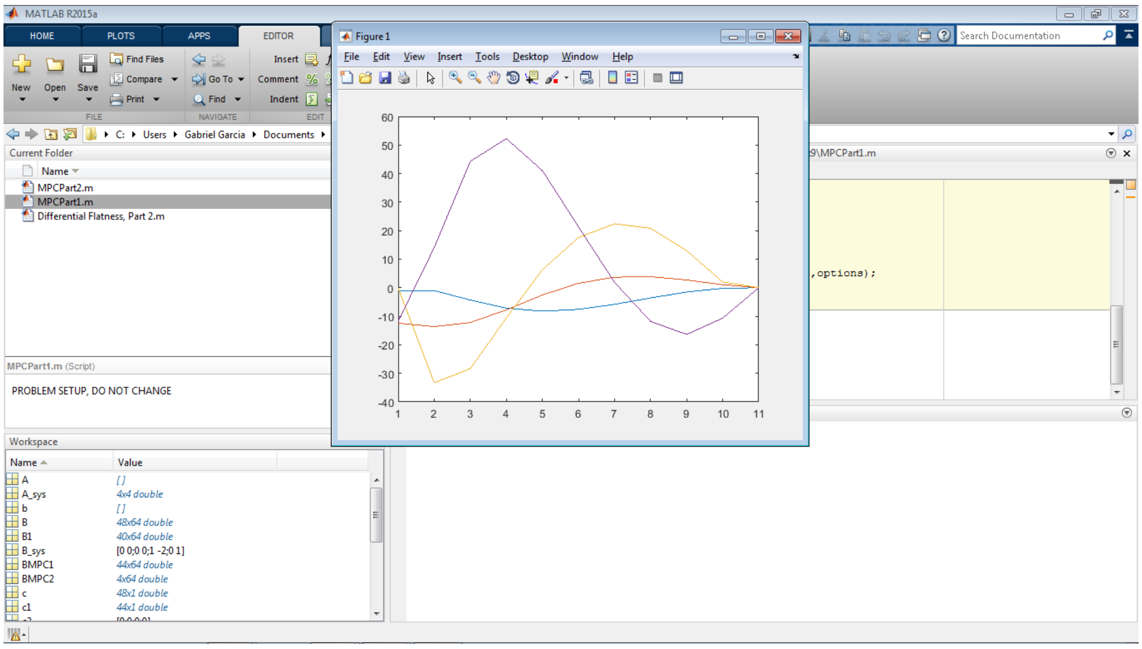
$$B_{MPC}z = \begin{bmatrix} B_{MPC1} \\ B_{MPC2} \end{bmatrix} z = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = c$$

$$J_2 = \sum_{k=1}^{N} (\|u[k-1]\|^2 + 0.01\|x[k]\|^2) + 0.01\|x[0]\|^2 = .5z^THz + f^Tz$$

$$H = 2\begin{bmatrix} 0.01I_{4(N+1)} & 0 \\ 0 & I_{2N} \end{bmatrix}$$

$$f = 0_{6N+4}$$

*Note that adding the term $0.01\|x[0]\|^2$ will not affect the optimization ($x[0]$ is subject to a constant value).



MPCPart1.m (Script)

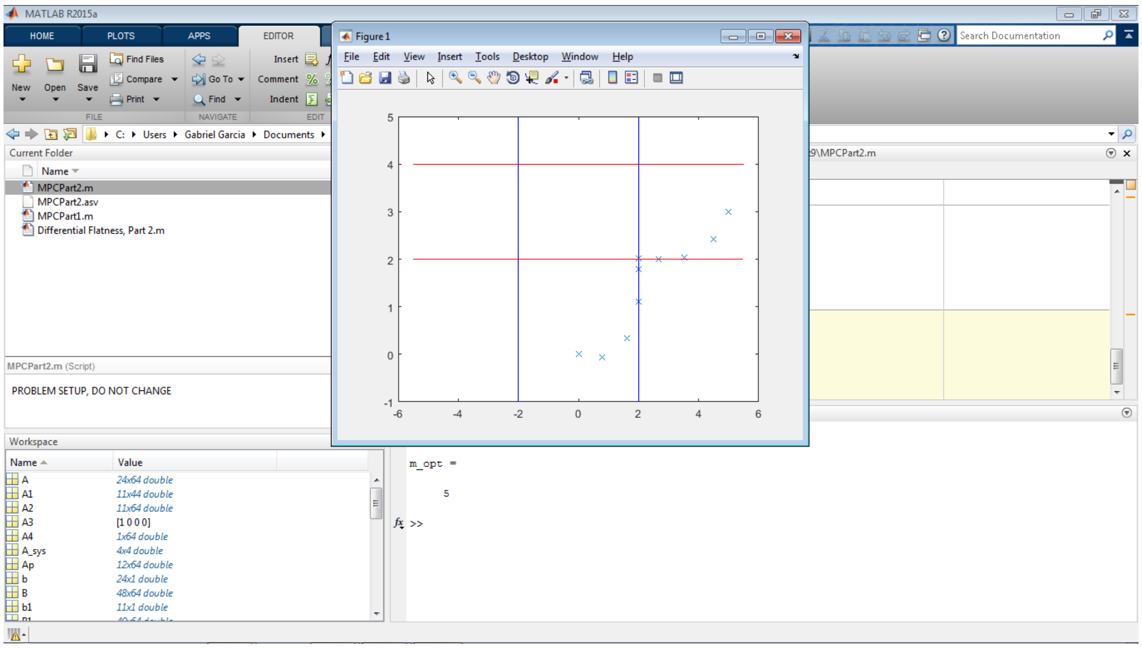
PROBLEM SETUP, DO NOT CHANGE

```
,options);
```

Workspace

Name	Value
A	[]
A_sys	4x4 double
b	[]
B	48x64 double
B1	40x64 double
B_sys	[0 0;0 0;1 -2;0 1]
BMPC1	44x64 double
BMPC2	4x64 double
c	48x1 double
c1	44x1 double

Model Predictive Control, Part 2



MPCPart2.m (Script)

PROBLEM SETUP, DO NOT CHANGE

Workspace

Name	Value
A	24x64 double
A1	11x44 double
A2	11x64 double
A3	[1 0 0 0]
A4	1x64 double
A_sys	4x4 double
Ap	12x64 double
b	24x1 double
B	48x64 double
b1	11x1 double

```
m_opt =

     5

>>
```

Note that we are in the border of the line of the restriction. This is very common in optimal control, optimality is often in the boundary conditions. “m” optimal is 5.

Differential Flatness, Part 2

$$T_f = 5$$

$$a = [a_0 \quad \cdots \quad a_N]^T$$

$$b = [b_0 \quad \cdots \quad b_N]^T$$

$$z = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$m(t) = [1 \quad t^1 \quad \cdots \quad t^N]^T$$

$$v(t) = m^{(3)}(t)$$

$$y_1 = a^T m(t)$$

$$y_2 = b^T m(t)$$

$$y_1^{(3)} = a^T v(t)$$

$$y_2^{(3)} = b^T v(t)$$

$$J = \int_0^{T_f} \left\| y^{(3)} \right\|^2 dt$$

$$J = \int_0^{T_f} \left(a^T v(t)\right)^2 + \left(b^T v(t)\right)^2 dt$$

$$J = \int_0^{T_f} a^T v v^T a + b^T v v^T b \, dt$$

$$J = \int_0^{T_f} z^T \begin{bmatrix} v(t)v(t)^T & 0 \\ 0 & v(t)v(t)^T \end{bmatrix} z \, dt$$

$$J = z^T \left(\int_0^{T_f} \begin{bmatrix} v(t)v(t)^T & 0 \\ 0 & v(t)v(t)^T \end{bmatrix} \, dt \right) z$$

$$H = 2 \left(\int_0^{T_f} \begin{bmatrix} v(t)v(t)^T & 0 \\ 0 & v(t)v(t)^T \end{bmatrix} \, dt \right)$$

s.t.

$$y(0) = y_0$$

$$\dot{y}(0) = \dot{y}_0$$

$$\ddot{y}(0) = \ddot{y}_0$$

$$y(T_f) = y_g$$

$$y_1 = a^T m(t) = m(t)^T a$$

$$y_2 = b^T m(t) = m(t)^T b$$

$$y(0) = y_0$$

$$y_{01} = m(0)^T [I_{N+1} \, 0_{N+1}] z$$

$$y_{02} = m(0)^T [0_{N+1} \, I_{N+1}] z$$

$$y(0) = y_0 = \begin{bmatrix} m(0)^T & 0 \\ 0 & m(0)^T \end{bmatrix} z = I_2 \otimes m(0)^T z$$

Same for:

$$\dot{y}(0) = \dot{y}_0 = I_2 \otimes m'(0)^T z$$

$$\ddot{y}(0) = \ddot{y}_0 = I_2 \otimes m''(0)^T z$$

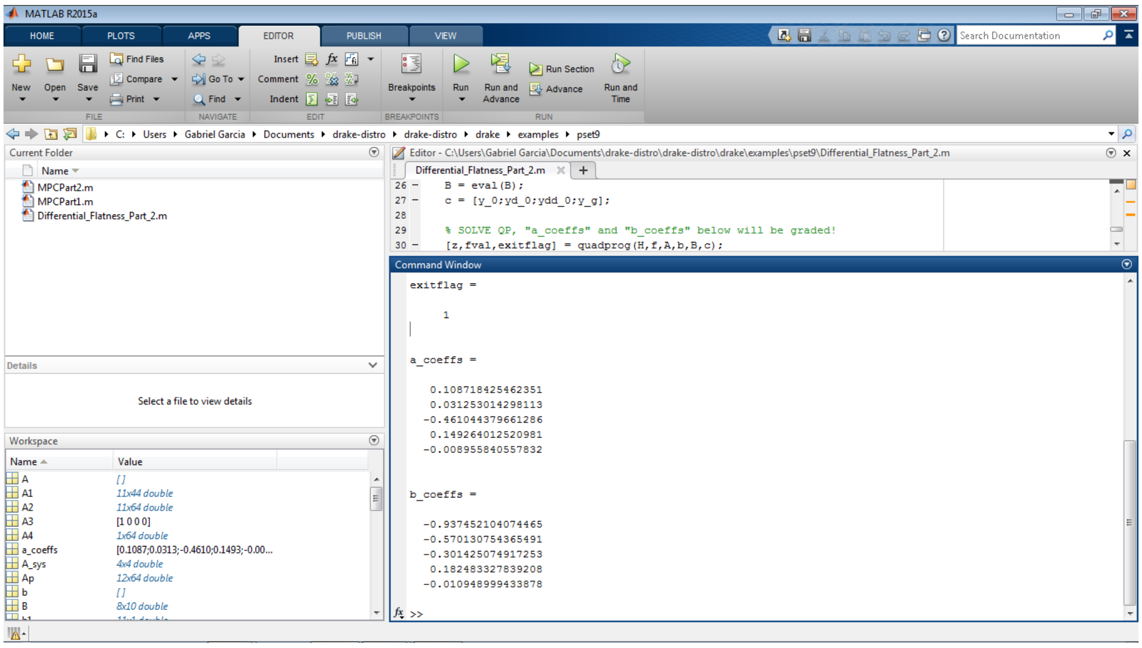
$$y(T_f) = y_g = I_2 \otimes m(T_f)^T z$$

$$m(t) = [1 \quad t^1 \quad \cdots \quad t^n]^T$$

$$v(t) = m^{(3)}(t)$$

$$y_1 = a^T m(t)$$

$$y_2 = b^T m(t)$$


MATLAB R2015a
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
C: ▸ Users ▸ Gabriel Garcia ▸ Documents ▸ drake-distro ▸ drake-distro ▸ drake ▸ examples ▸ pset9
Current Folder
Name
MPCPart2.m
MPCPart1.m
Differential_Flatness_Part_2.m
Details
Select a file to view details
Workspace
Editor - C:\Users\Gabriel Garcia\Documents\drake-distro\drake-distro\drake\examples\pset9\Differential_Flatness_Part_2.m
Differential_Flatness_Part_2.m
Command Window

```
    B = eval(B);
    c = [y_0;yd_0;ydd_0;y_g];

    % SOLVE QP, "a_coeffs" and "b_coeffs" below will be graded!
    [z,fval,exitflag] = quadprog(H,f,A,b,B,c);
```

```
exitflag =

     1


a_coeffs =

   0.108718425462351
   0.031253014298113
  -0.461044379661286
   0.149264012520981
  -0.008955840557832


b_coeffs =

  -0.937452104074465
  -0.570130754365491
  -0.301425074917253
   0.182483327839208
  -0.010948999433878

>>
```

Name	Value
A	[]
A1	11x44 double
A2	11x64 double
A3	[1 0 0 0]
A4	1x64 double
a_coeffs	[0.1087;0.0313;-0.4610;0.1493;-0.00...
A_sys	4x4 double
Ap	12x64 double
b	[]
B	8x10 double