

PSET from Underactuated Robotics 6.832x edX involving Drake-MATLAB

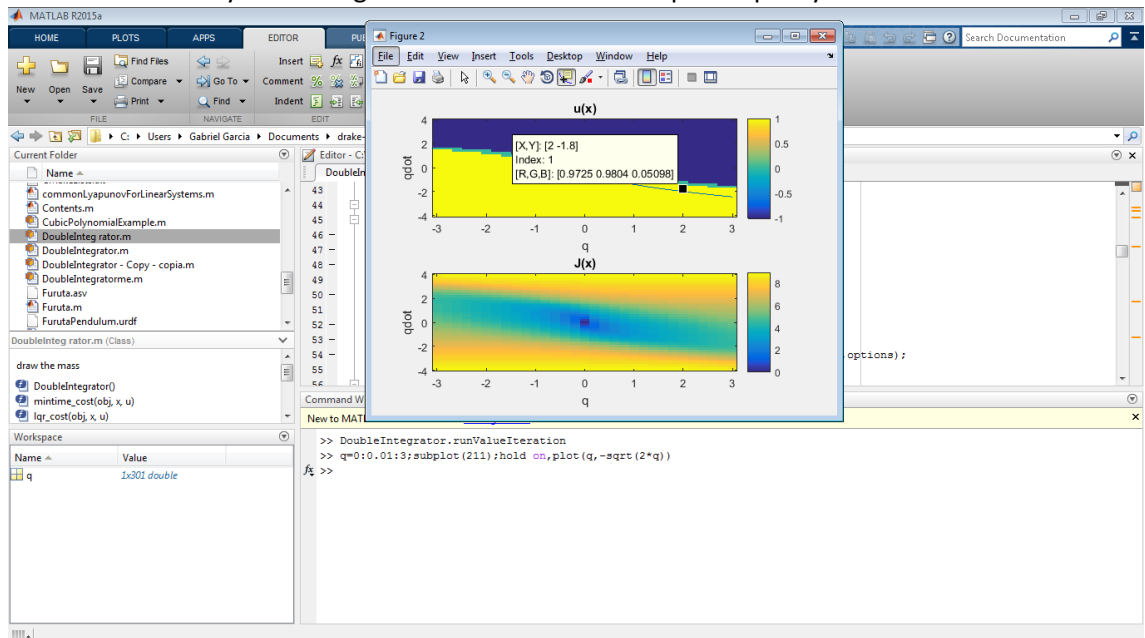
Gabriel Enrique García Chávez

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))
>>
```

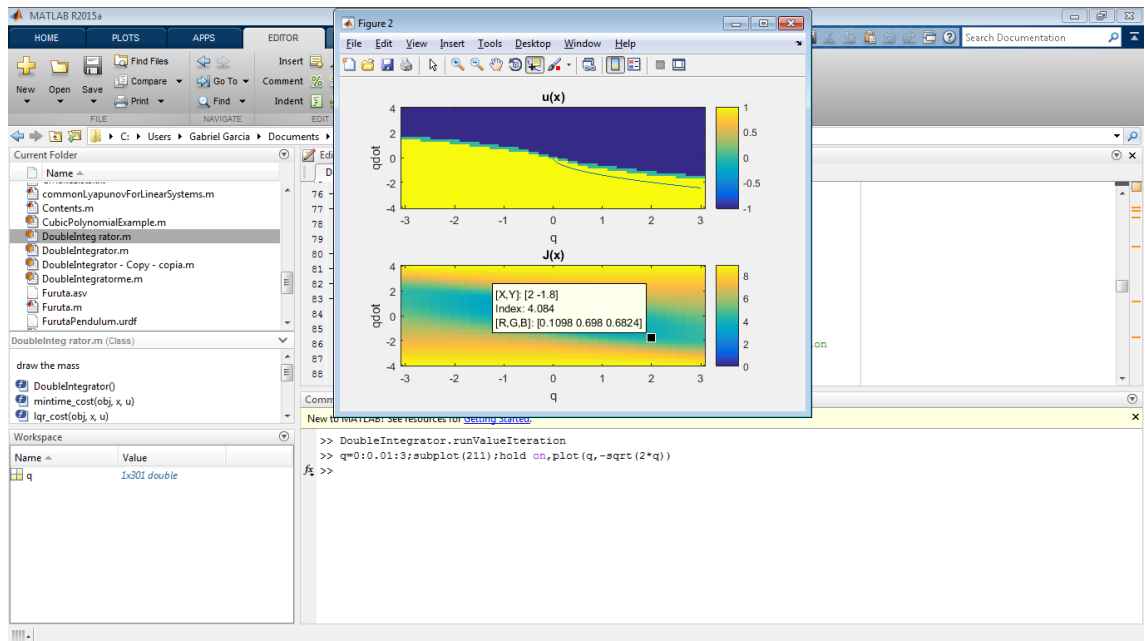
$\dot{q}_0 = -1.8$

- 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?



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

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:

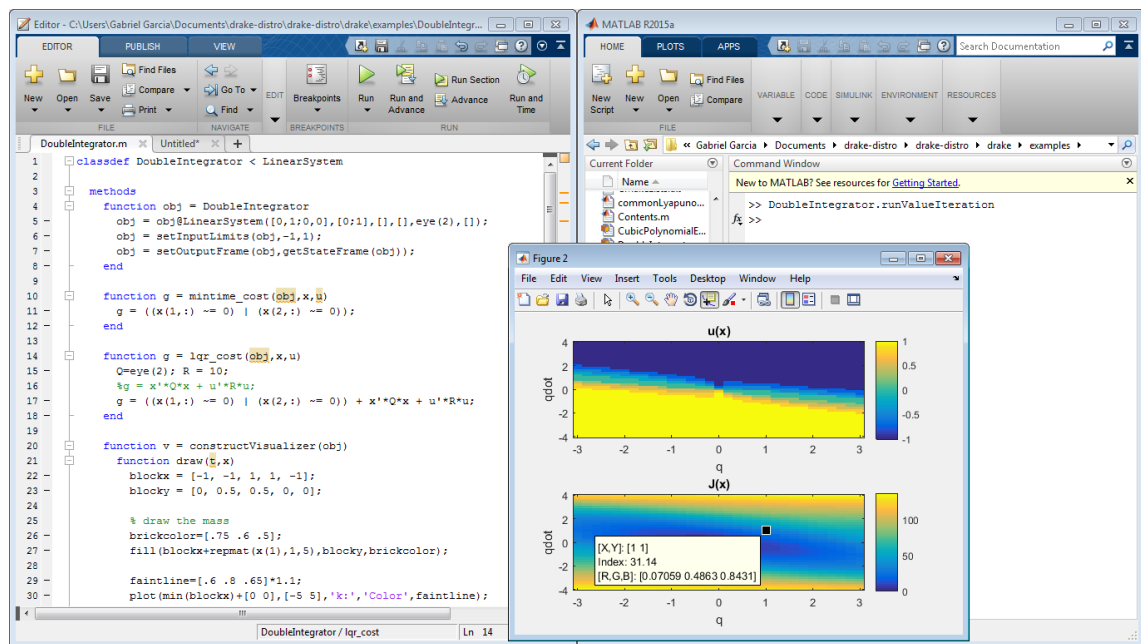
```
xbins = {[-3:.17:3], [-4:.17:4]};
```

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

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

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?



$$J(x_0) = 31.14$$

II. Value Iteration (Stochastic Grid World):

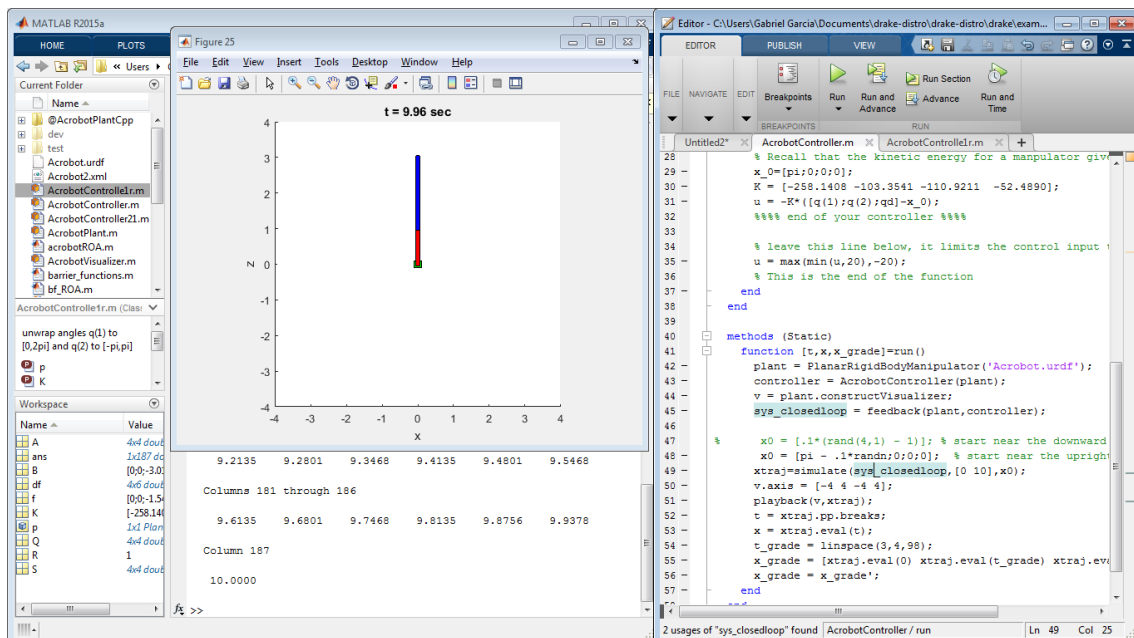
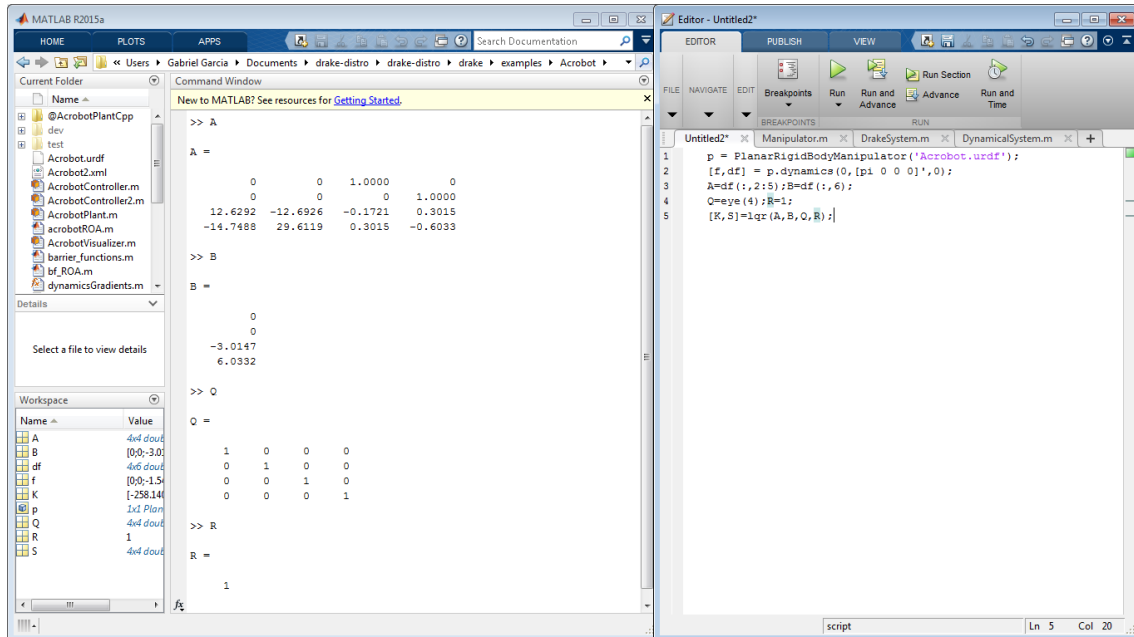
(a) Run the value iteration code for the double integrator to compute the optimal policy and optimal cost-to-go for the minimum-time problem. Compare the result to the analytical solution we found in lecture (also available in Example 9.2 in the course notes) by answering the following questions.

PSET3:

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

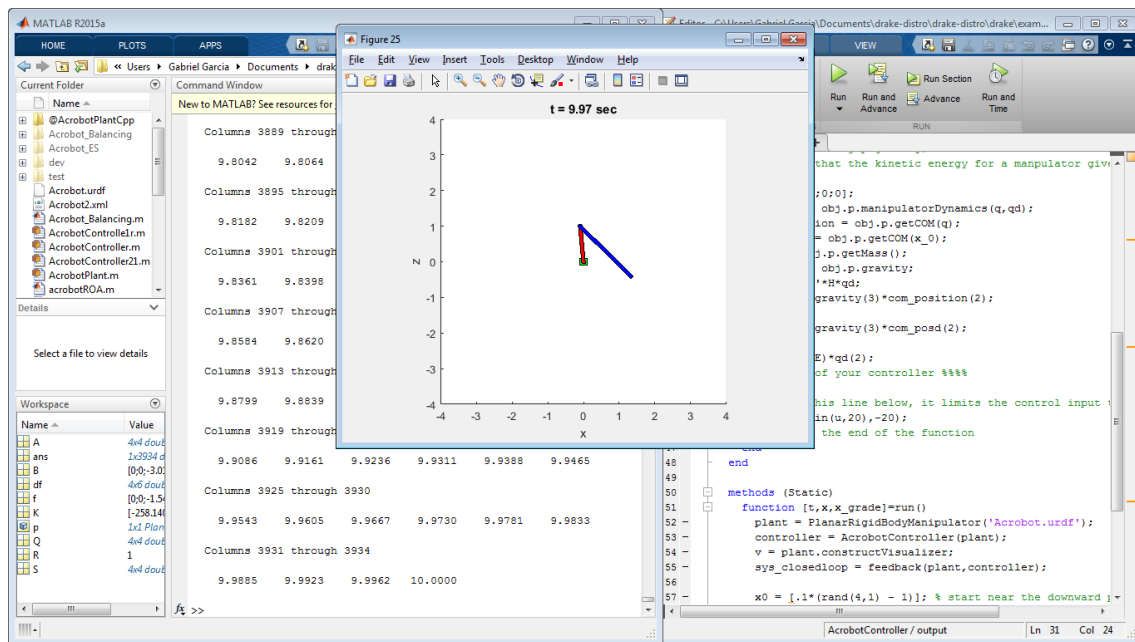
I. Acrobot Balancing:

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



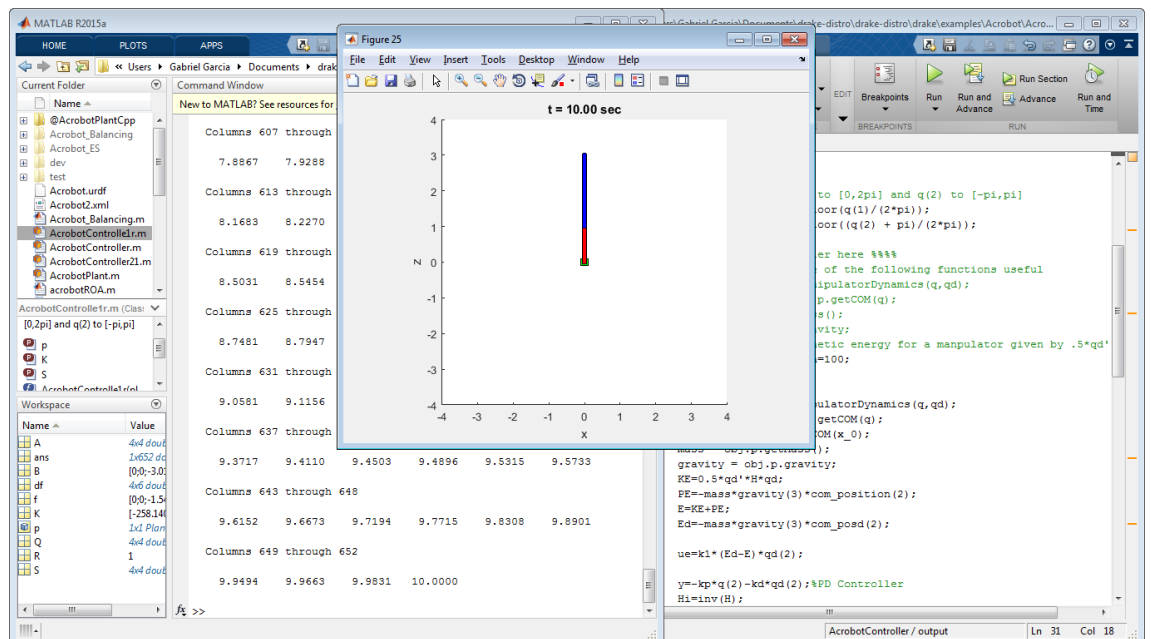
II. Acrobot Energy Shaping:

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



III. Acrobot Swingup

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



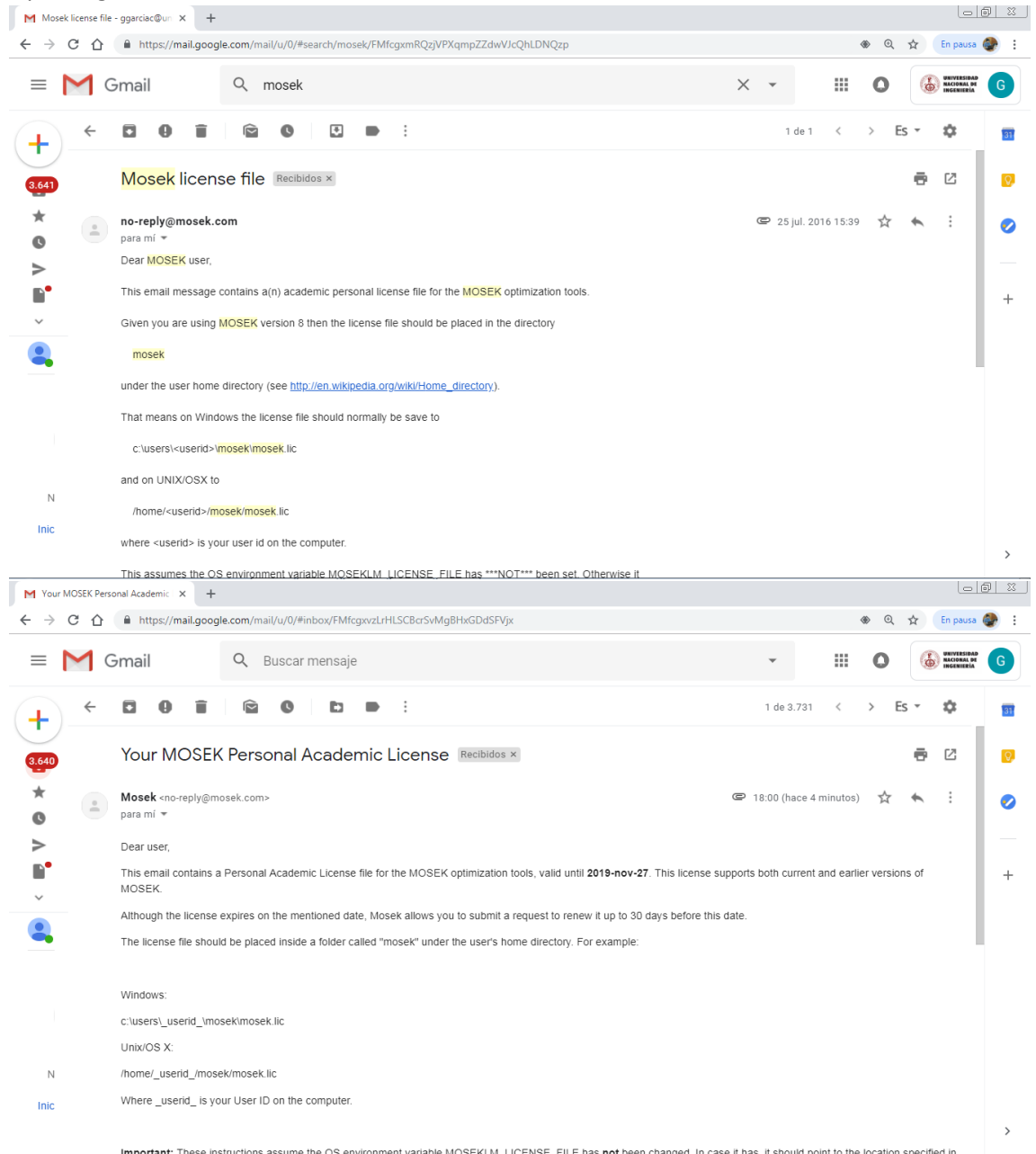
Note that we have added K and S as properties to the class.

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.



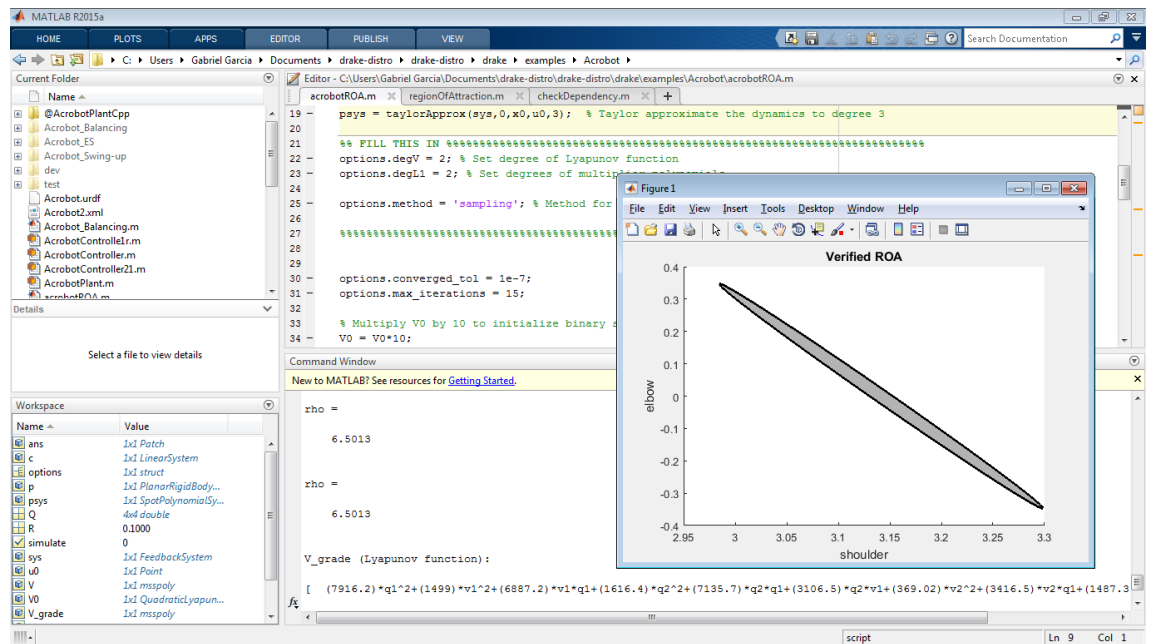
In startup.m we should add:

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

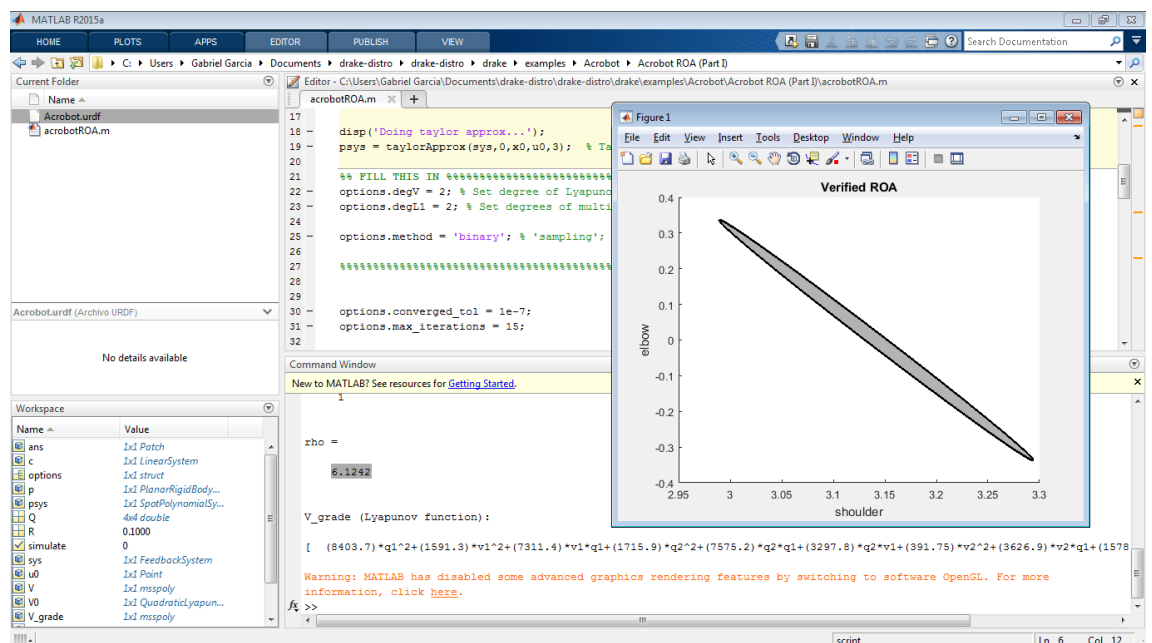
The license is copied to %USERPROFILE%\mosek\mosek.lic

I. Acrobot Region of Attraction (Part I):

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



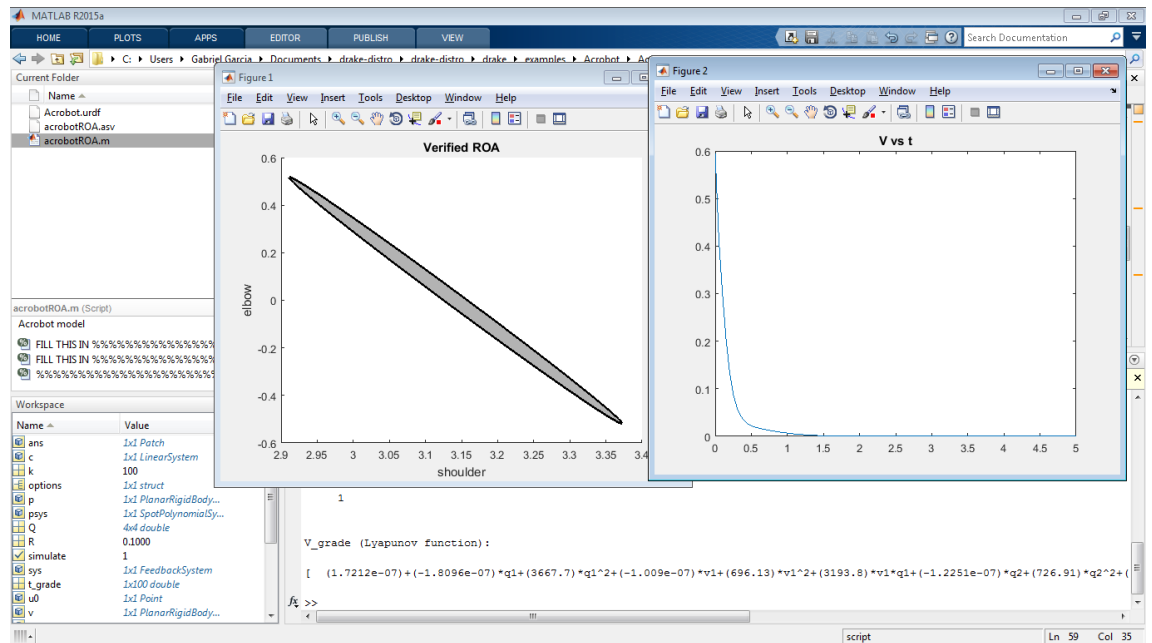
$$\rho = 6.5013$$



$$\rho = 6.1242$$

II. Acrobot Region of Attraction (Part II):

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



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.

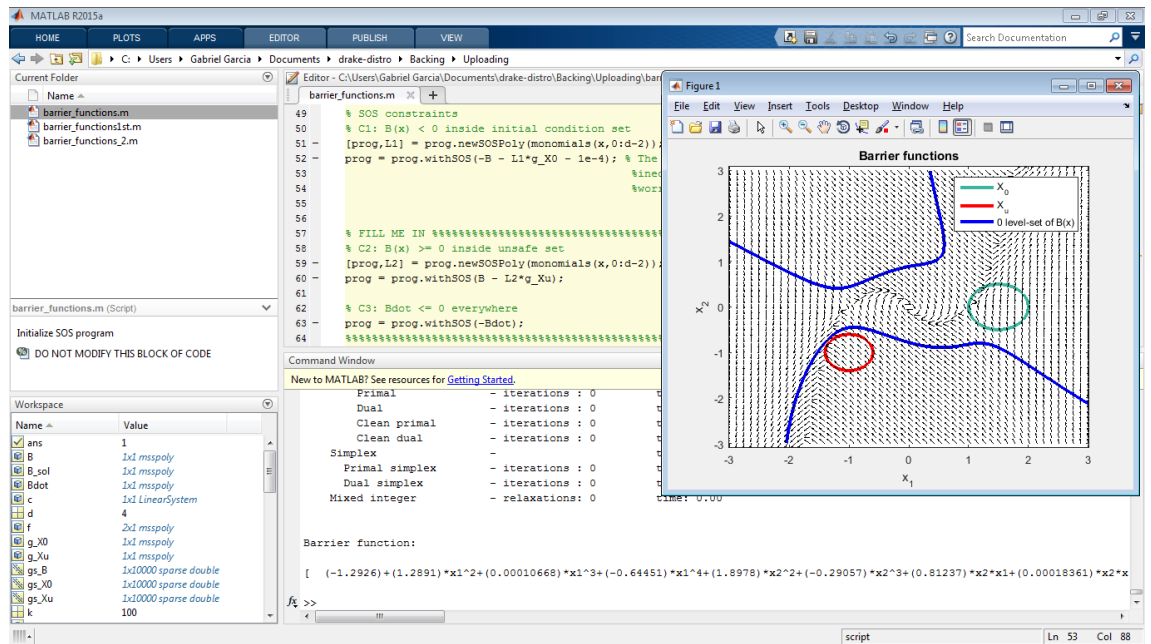
III. Barrier Functions:

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

We have the polynomial dynamical system:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_1 + x_1^3 - x_2\end{aligned}$$

We implement the conditions for the quadratic program to solve.



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 $-B - \lambda_1 g_{X_0} - \varepsilon$ SOS means:

$$-B - \lambda_1 g_{X_0} - \varepsilon \geq 0 \rightarrow -\lambda_1 g_{X_0} - \varepsilon \geq B$$

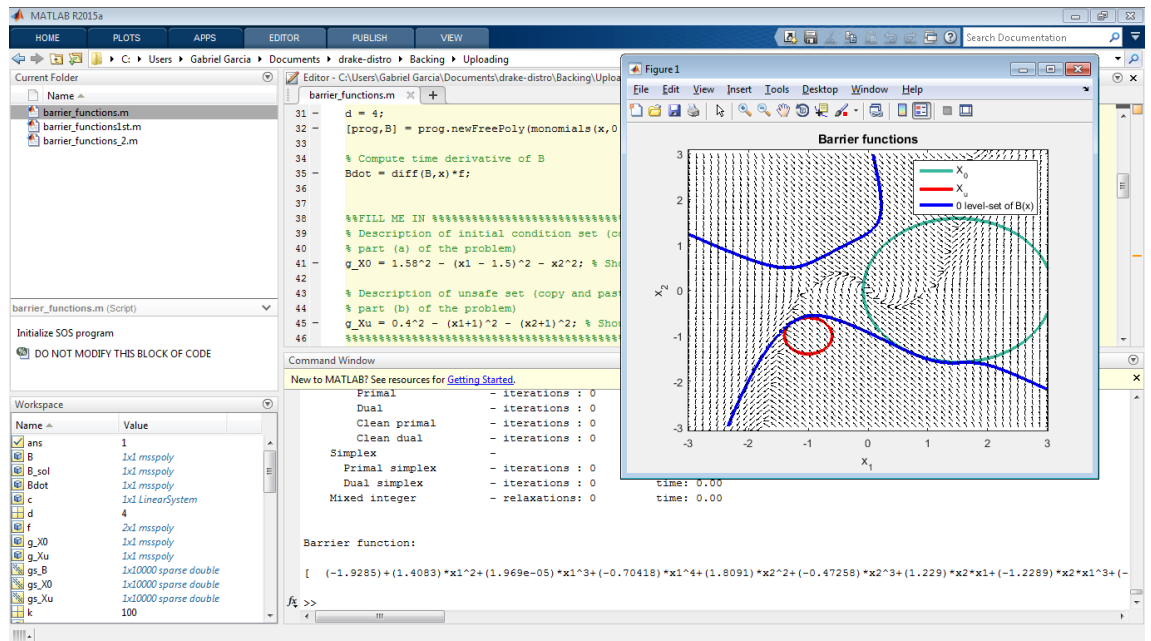
In particular, in the region X_0 we have:

$$g_{X_0} \geq 0 \rightarrow 0 \geq -\lambda_1 g_{X_0}$$

$$-\varepsilon \geq -\lambda_1 g_{X_0} - \varepsilon \geq B$$

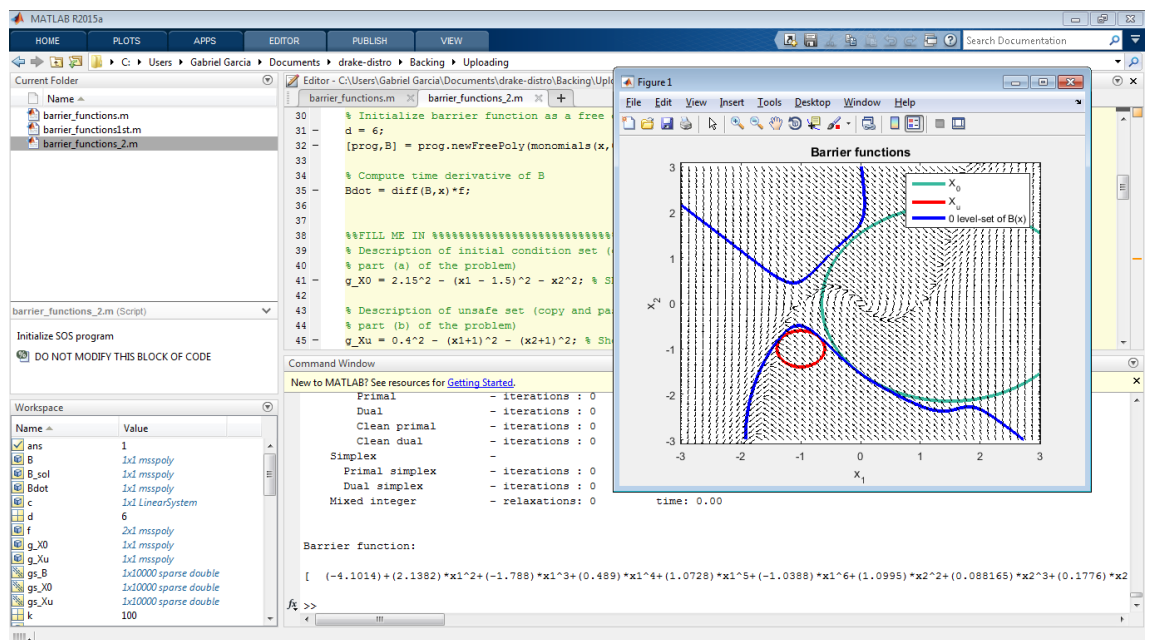
So we have $-\varepsilon \geq B$, or in particular, $0 > B(x) \forall x \in X_0$. Let's remember that for good performance, we need that the degree of B will be the same as the degree of $\lambda_1 g_{X_0}$. Given that g is a quadratic polynomial, the degree of λ_1 is two less than the degree of B .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



$$r_{max} = 1.58$$

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.

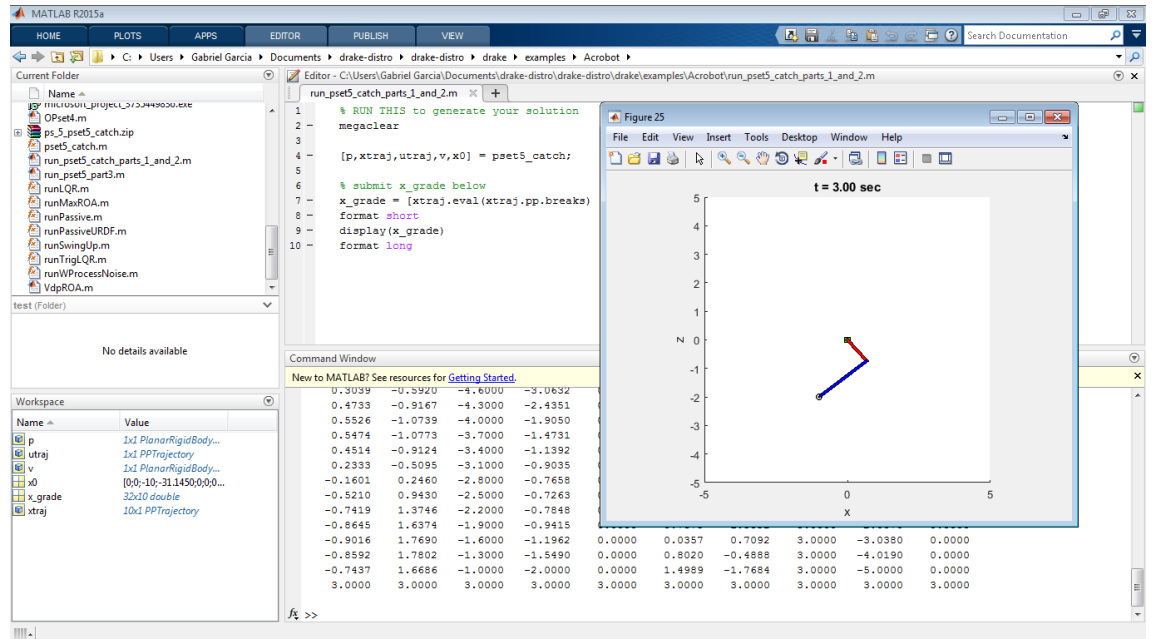


PSET5:

- 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 $(hand_{pos}(1), hand_{pos}(2))$ must match in both components. We also take the gradient.



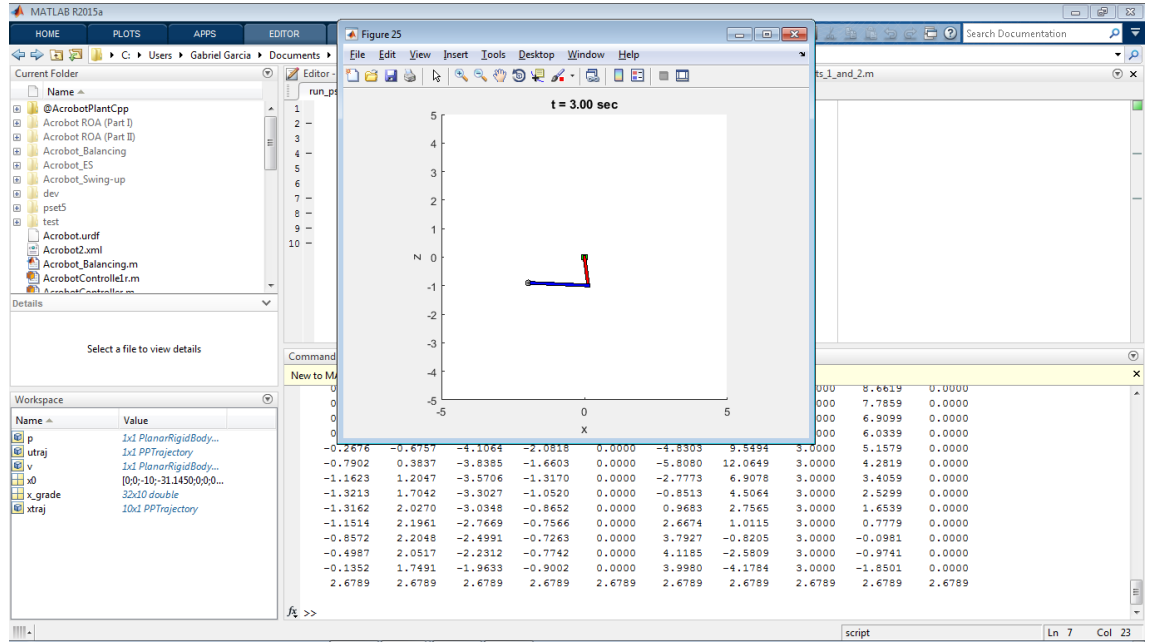
Part 2: We must add an expression to the final cost that represents the height of the ball. The simplest way to do it is set the expression to:

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

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.



Part 3:

We have the cost we wish to add:

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

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.

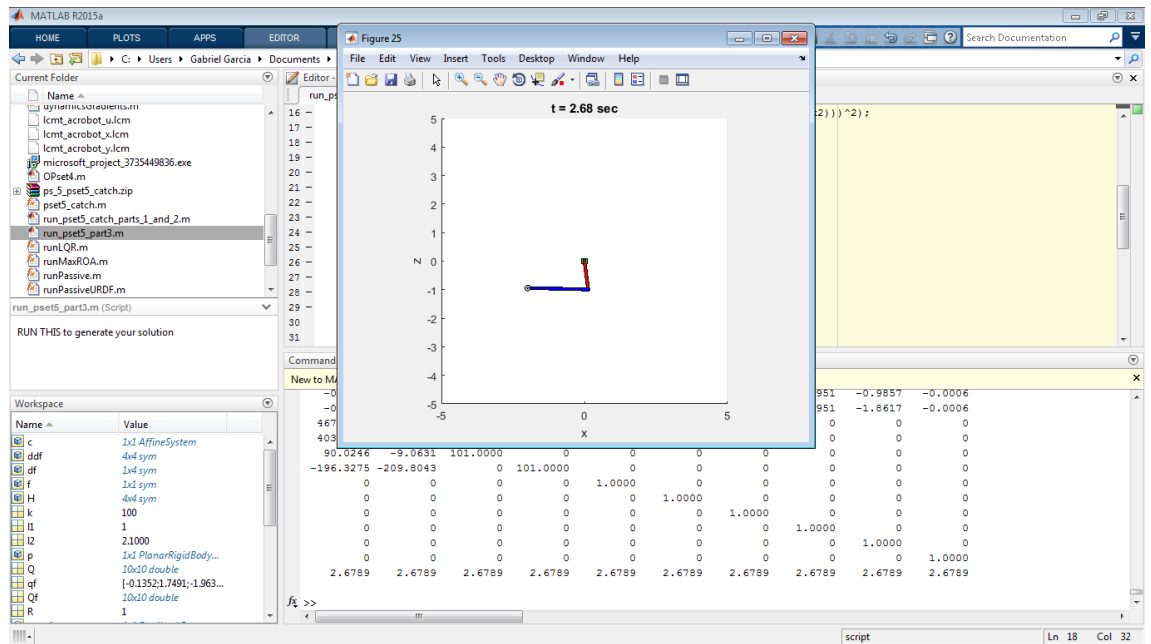
$$x_f = x_{traj}(t_f) \text{ in part. } t_f = 3$$

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

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

We have $f(x_f) = \frac{\partial f}{\partial x^T} \Big|_{x=x_f} = 0$, so we can take $\frac{\partial^2 f}{\partial x^2} \Big|_{x=x_f}$ 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.

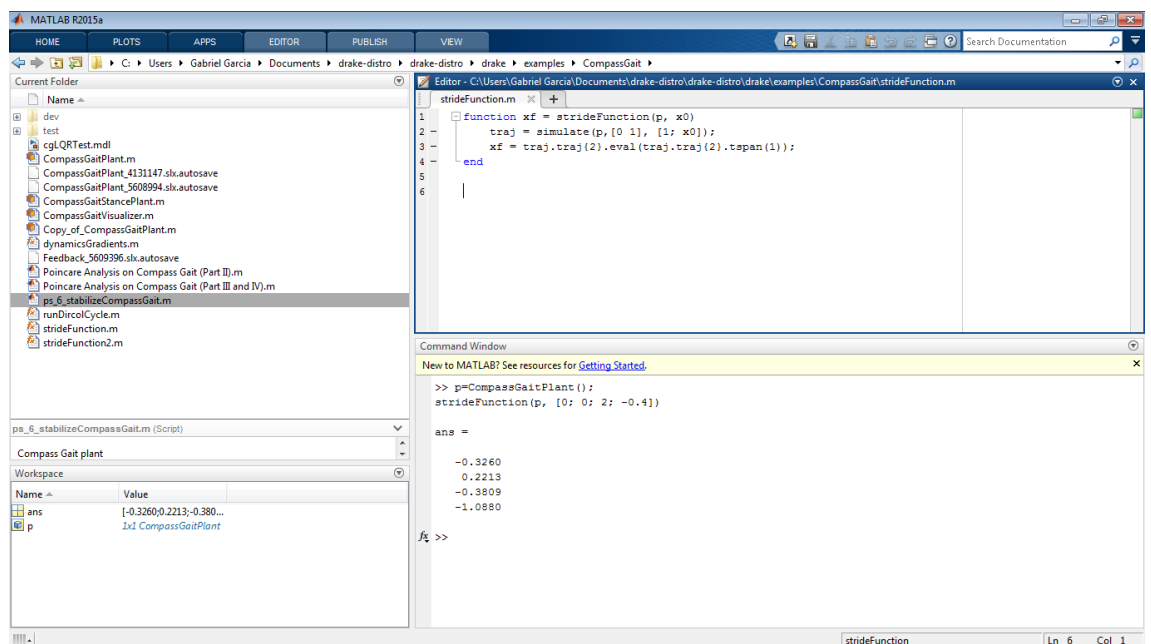


PSET6:

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

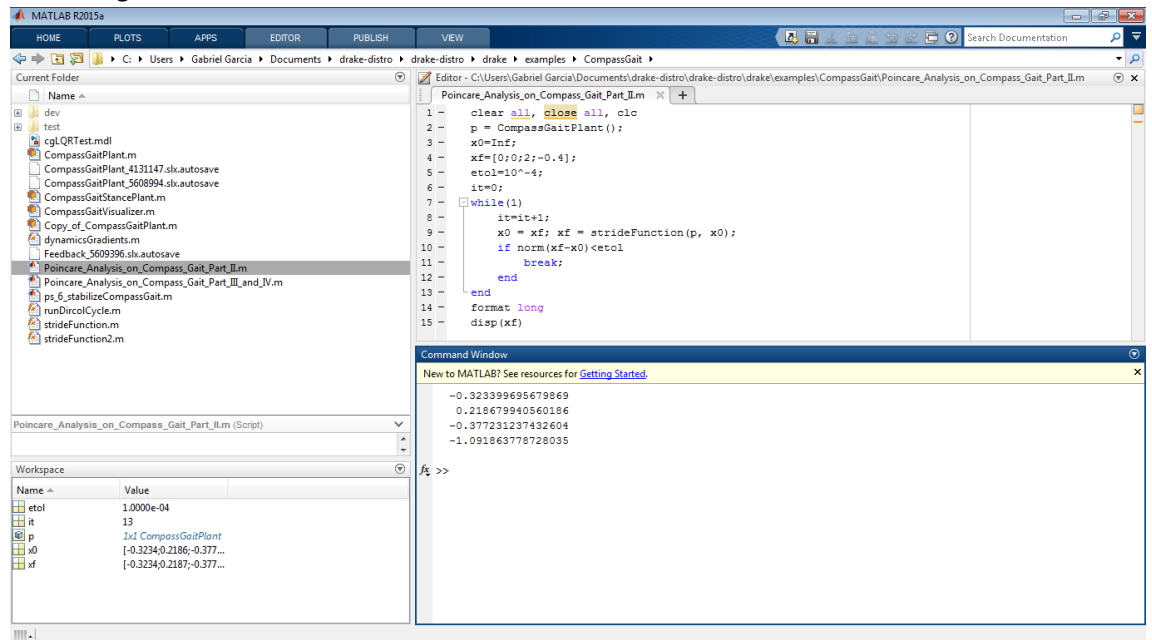
I. Poincare Analysis (Compass Gait) Part I:

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.



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.



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

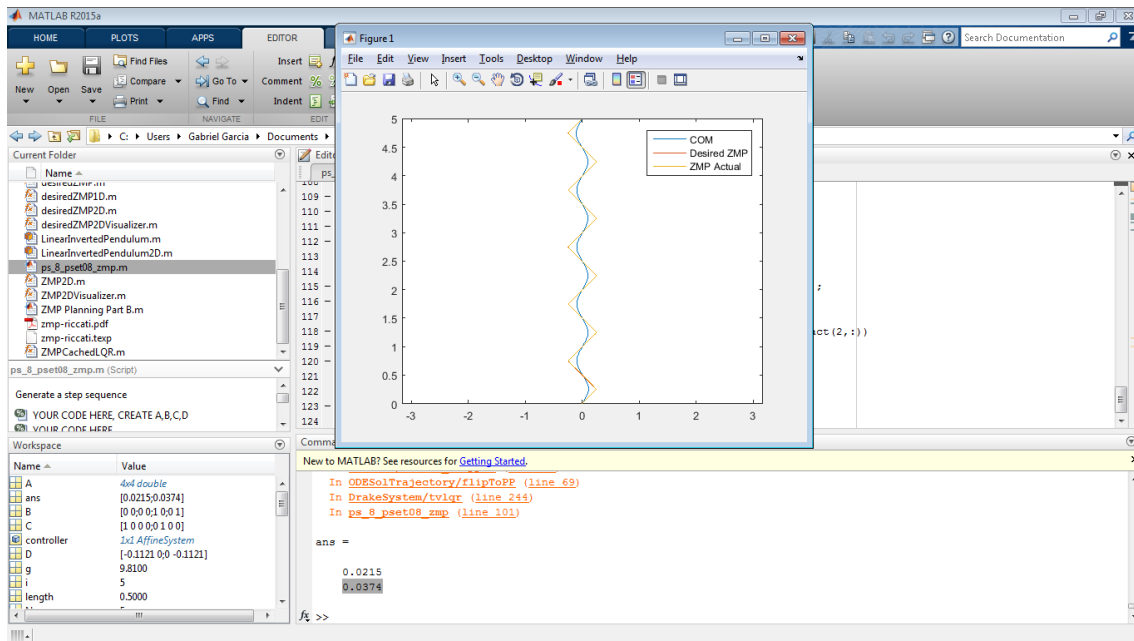
In order to compute the matrix of the linearization of the Poincare map around the “initial” point of the limit cycle, we take a look at the map:

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

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

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

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.



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:

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

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:

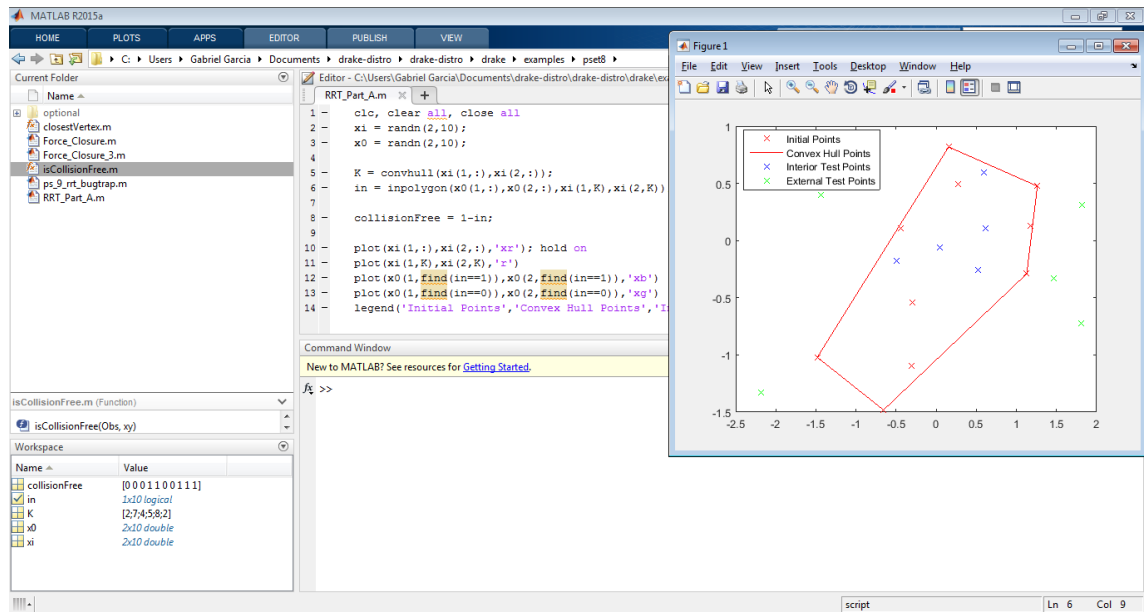
$$\begin{aligned} A &= G \\ b &= \mathbf{0}_3 \end{aligned}$$

We can write for simplicity:

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

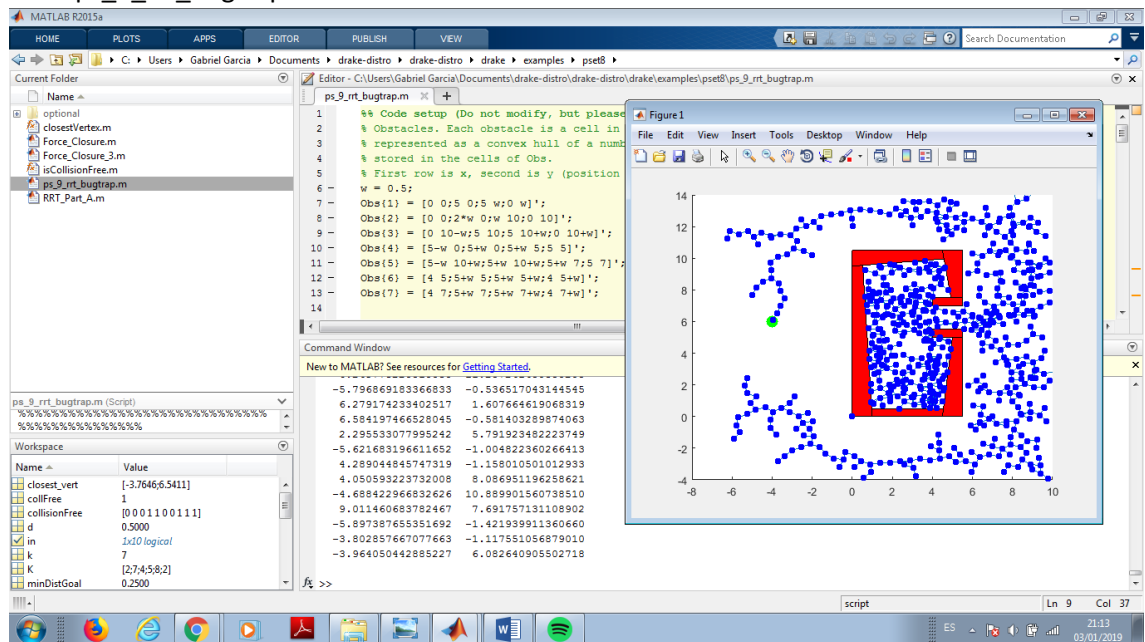
Where the superscript "a" means augment one component to the vector (the new component is 0, and the new vectors will have dimension 3).

- III. RRT:
 - a) RRT (Part A)
 - Run "RRT_Part_A.m"



b) RRT (Part B)

- Run "`ps_9_rrt_bugtrap.m`"



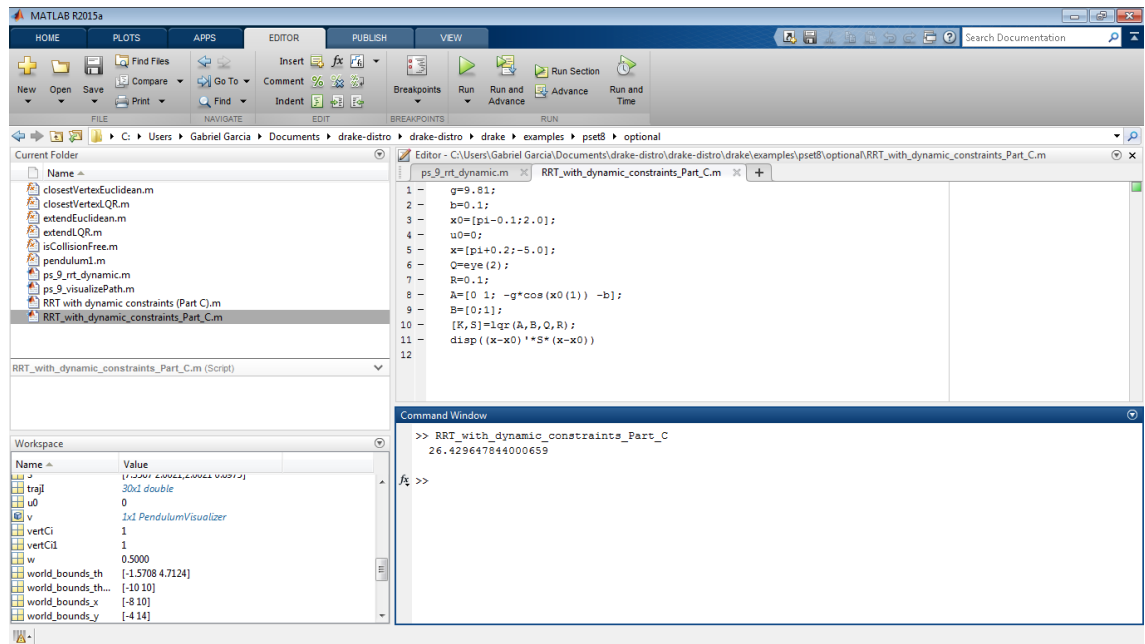
IV. Optional: RRT with dynamic constraints:

- Folder "Optional" inside pset8

a) RRT with dynamic constraints (Part A)

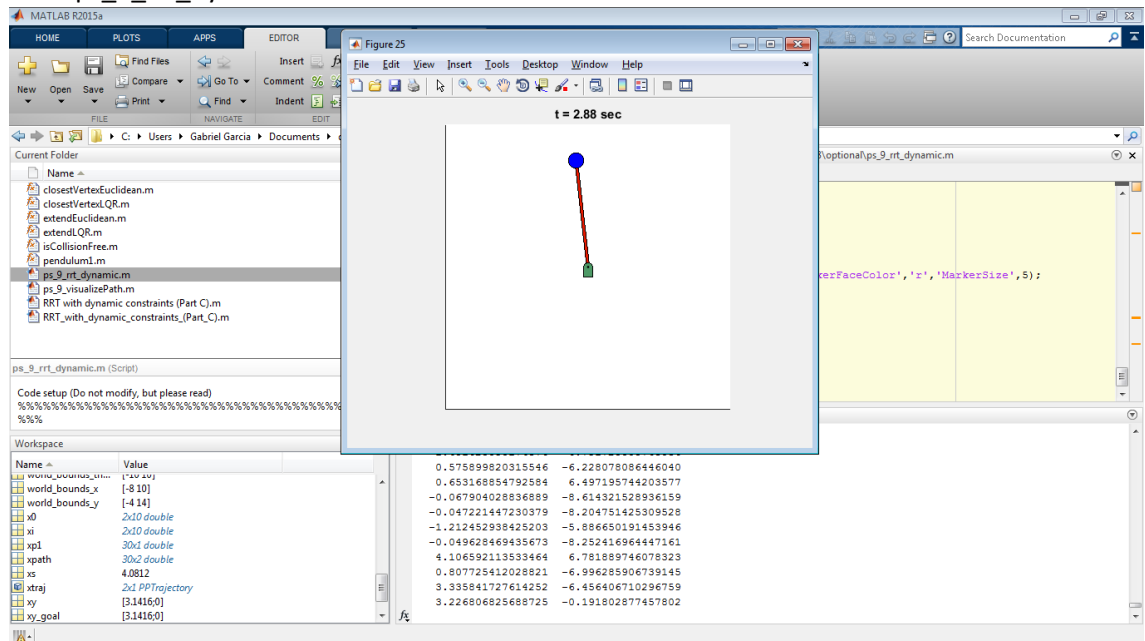
- Run "`ps_9_rrt_dynamic.m`" commenting line 29 "`%method = 'lqr'`" to avoid "`method = 'euclidean';`" be overwritten and run. Results is the same as Part D, but slower

b) RRT with dynamic constraints (Part C)



c) RRT with dynamic constraints (Part D)

- Run “ps_9_rrt_dynamic.m”



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

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

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

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 writing the full matrix. Results are the same as the answer provided in the course.

Model Predictive Control, Part 1

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

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

$$x[0] = x_0$$

$$x[N] = \mathbf{0}^{4 \times 1}$$

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

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

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

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

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

$$\begin{bmatrix} 0_{4N} & I_4 & 0_{2N} \end{bmatrix} z = \begin{bmatrix} 0_4 \end{bmatrix}$$

$$B_{MPC2} = \begin{bmatrix} 0_{4N} & I_4 & 0_{2N} \end{bmatrix}$$

$$c_2 = 0_4$$

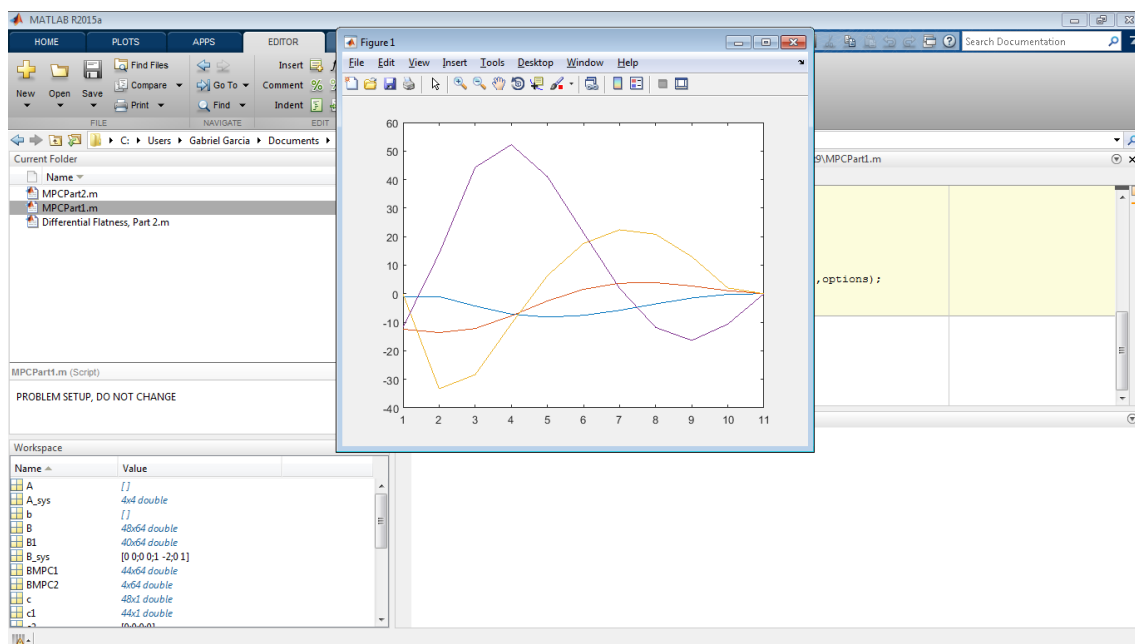
$$B_{MPC} z = \begin{bmatrix} B_{MPC1} \\ B_{MPC2} \end{bmatrix} z = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = c$$

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

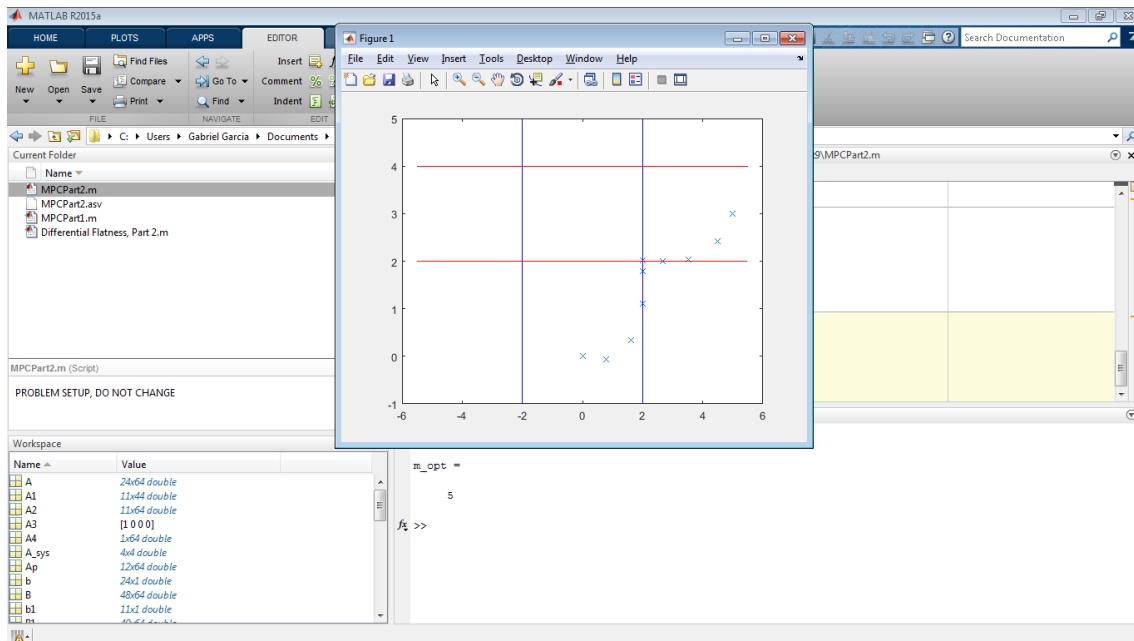
$$H = 2 \begin{bmatrix} 0.01I_{4(N+1)} & 0 \\ 0 & I_{2N} \end{bmatrix}$$

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

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



Model Predictive Control, Part 2



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

$$T_f = 5$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

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

The image shows the MATLAB R2015a interface. The top menu bar includes HOME, PLOTS, APPS, EDITOR, PUBLISH, and VIEW. The toolbar contains icons for file operations (New, Open, Save, Compare, Print), navigation (Find Files, Go To, Find), editing (Insert, Comment, Indent), and execution (Run, Run Section, Run and Advance, Run and Time). The current folder is 'C:\Users\Gabriel Garcia\Documents\drake-distro\drake\examples\pset9'. The editor window displays the script 'Differential_Flatness_Part_2.m' with the following code:

```

26 B = eval(B);
27 c = [y_0; yd_0; ydd_0; y_g];
28
29 % SOLVE QP, "a_coeffs" and "b_coeffs" below will be graded!
30 [z,fval,exitflag] = quadprog(H,f,A,b,B,c);

```

The Command Window shows the output of the script:

```

exitflag =

     1

a_coeffs =

    0.108718425462351
    0.031253014298113
   -0.461044379661286
    0.149244012520981
   -0.008955840557832

b_coeffs =

   -0.937452104074465
   -0.570130754365491
   -0.301425074917253
    0.182483327839208
   -0.010948999433878

```

The Workspace window shows the following variables:

Name	Value
A	[]
A1	11x44 double
A2	11x64 double
A3	[1 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
c	5x5 double