**CoSMo 2013**

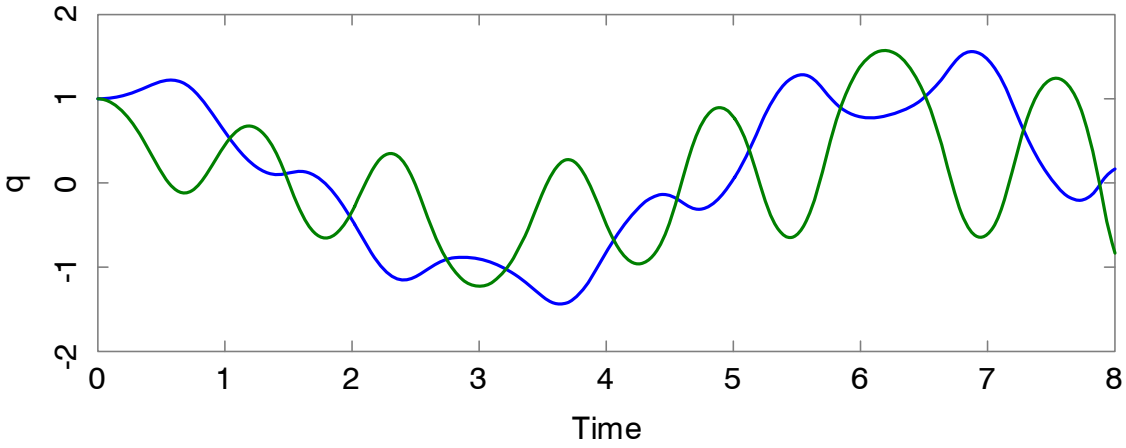
**TUTORIAL**

You job is to simulate a 2-joint arm, design a feedback-linearization controller to steer the arm to a target, and compute a roughly time-optimal policy to move the arm to **0**.

The arm has these state dynamics

$$\boldsymbol{u} = \begin{bmatrix} 0.536 + 0.36\cos(q_2) & 0.2 + 0.18\cos(q_2) \\ 0.2 + 0.18\cos(q_2) & 0.2 \end{bmatrix} \ddot{\boldsymbol{q}} + 0.18\sin(q_2) \begin{bmatrix} -\dot{q}_2 & -(\dot{q}_1 + \dot{q}_2) \\ \dot{q}_1 & 0 \end{bmatrix} \dot{\boldsymbol{q}}$$

where $\boldsymbol{q}$ is a 2-element vector containing the shoulder and elbow angles, in that order. In your code, you will have to rearrange this formula into control-canonical form. Integrate the dynamics using Euler's method with $\Delta t = 0.01$ s. To check whether you have done all this correctly, simulate 8 s of motion, starting from $\boldsymbol{x} = (1, 1, 0, 0)^T$, with the policy $\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{q}$, and plot $\boldsymbol{q}$ versus time The result should look like this:




Next, program a policy to steer the arm to a jumping target, taking Feedback_linearization.m as your guide. Simulate 8 s of reaching, and make the target joint angles jump simultaneously within the range [–1, 1] every 1 s. Assume your controller already has exact knowledge of the state-dynamics functions $\boldsymbol{f}()$ and $\boldsymbol{G}()$, so there is no need to learn them or distinguish in your code between f, G and f_est, G_est. Choose your desired dynamics so both joints always reach their target angles within 1 s. In your graph, plot shoulder and elbow angles (as

solid lines) and their targets (as dashed lines) versus time. As far as possible, name your variables as in these pages, the notes, and the sample code, e.g. `q`, `q_star`, `q_vel`, `q_acc`, `Dt`, `f`, `G`, `u`.

Finally, program a roughly time-optimal controller for the arm, using HG_TO_1DOF.m as your guide. To keep your code brief, put the code that computes ***f*** and ***G*** into a function m-file called sys_arm.m, much as I did in HG_TO_1DOF.m. Use the same `hpr` (for your initial, feedback-linearization policy), `Q`, `r_`, and `phi_` as in HG_TO_1DOF.m, except that you will need more features. During the learning, test your current policy after every 5000 examples, by computing the cost it accumulates on a movement from `x_test` = [-0.5; 0.2; 0; 0] to [0; 0; 0; 0]. Plot your policies' performance on this test movement in a 3-panel figure, the top panel showing shoulder and elbow angles versus time, the middle panel showing the accumulating cost versus time, and the bottom panel showing ***u*** versus *t*. As far as possible, name your variables as in this question, the notes, and the sample code, e.g. `x`, `x_vel`, `WX`, `w`, `W`, `phi`.