Automated Parameterization of the Joint Space Dynamics of a Robotic Arm

Josh Petersen

Introduction

The goal of my project was to use machine learning to fully automate the parameterization of the joint space dynamics of a robotic arm. When given a set of joint angles, joint velocities, and motor torques, the program would determine the A, B, C and G matrices of the dynamics.

$$A(q)[\ddot{q}] + B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2] + G(q) = \tau$$

This was accomplished in several parts. First, two methodologies were created which could automatically smooth noisy positions, velocities, and accelerations. Next, two methodologies for creating the feature vector for finding the matrices using least squares were developed. Lastly, a structure was setup to combine these elements into one streamlined process.

Automated Position, Velocity, and Acceleration Smoothing

The first goal was to automatically smooth the position, velocity, and acceleration profiles. If this program were to be used in practice on a real robotic arm, smoothing these profiles would allow for better results by reducing the effect of noise in the sensor data. Two methods were developed to accomplish this.

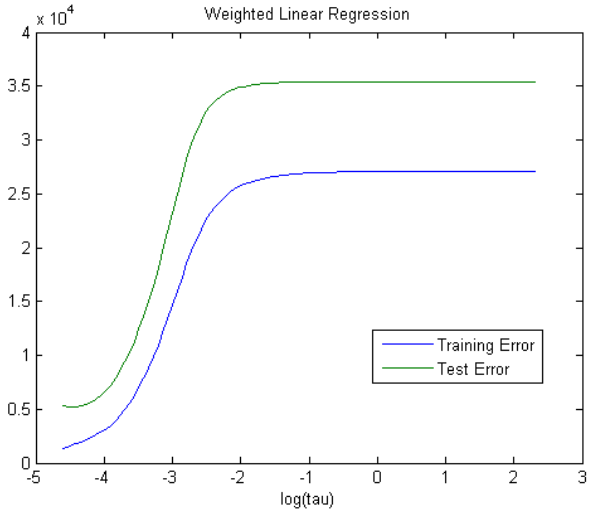
First, weighted linear regression was used. To automatically choose τ such that high bias and high variance was avoided, leave one out cross validation was used. The optimal fit occurs when the training error is close to the test error where the test error is the defined to be the average of the squared differences between the left out data point and the test prediction at that point and the training error is the average of the squared differences between all the training data and the predicted points at those locations.

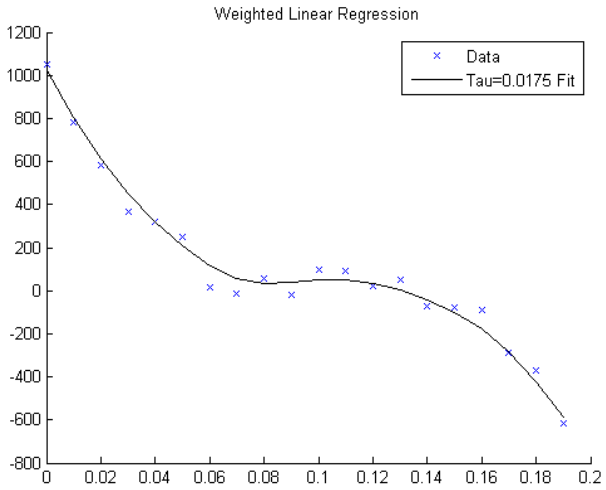
$$min_{\tau} \left| \varepsilon_{Test} - \varepsilon_{Train} \right|$$

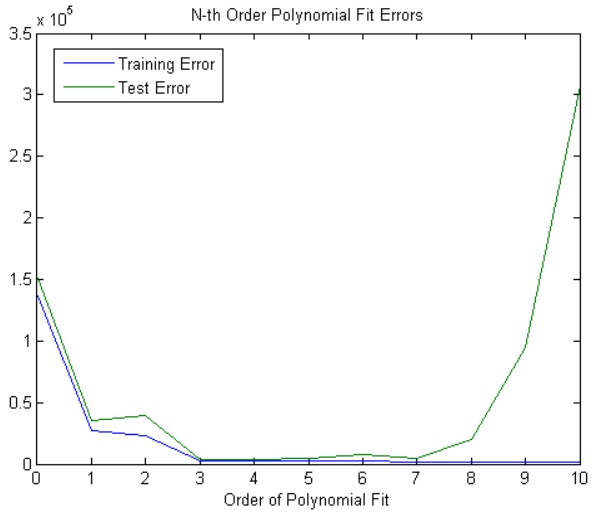
$$\varepsilon_{Test} = \frac{\sum_{i=1}^{m} \left| y^{(i)} - y_p^{(i)} \right|^2}{m} \qquad \varepsilon_{Train} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m, i \neq j} \left| y^{(j)} - y_{p^{(i)}}^{(j)} \right|^2}{m(m-1)}$$

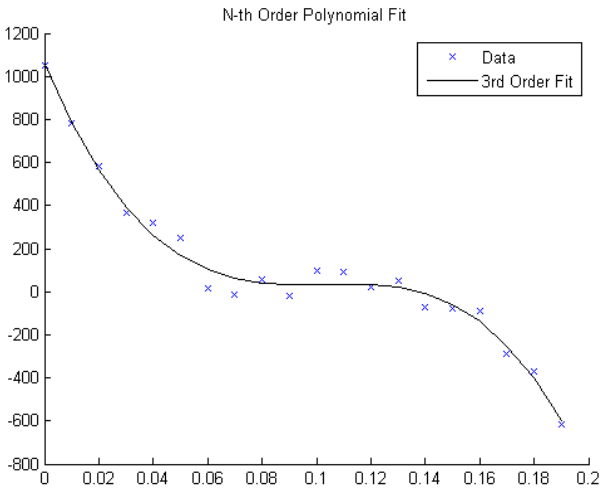
Next, n-th order polynomial fitting was tried using the same criteria. Fits between 0 order to 10th order polynomials were tried and checked using LOOCV.

$$min_{n} \left| \varepsilon_{Test} - \varepsilon_{Train} \right|$$









Above are the results for a set of data points generated from a third order polynomial with some noise added. Additional testing with varying noise revealed that the weighted linear regression was still susceptible to high variance in some cases as discussed in the project milestone. Also, weighted linear regression was much more time intensive than the n-th order polynomial fit in general since it required a larger number of τ to try to fit and computing the value at a particular point required calculations from each point in the data set.

Generating Feature Vectors

Next, two methods were developed to create feature vectors for performing least squares to find the components of the matrices.

First, a function was created that would use the Denavit-Hartenberg Parameters, general center of mass directions for each joint and the direction of gravity to form the matrices in symbolic form and another function was created to turn these matrices into the feature vector.

Second, a method was developed to generate a general feature vector for an n degree of freedom robotic arm. Several properties of the dynamic matrices were used as well as some simplifying assumptions.

The i-th row and column of the mass matrix, $A(q)$, can not contain joint angles at or below the i-th joint angle. Additionally, the Coriolis and centripetal matrices, $B(q)$ and $C(q)$, are simply derivatives of particular components of the $A(q)$ matrix.

To simply the problem further, the auto-generated feature vectors would be restricted to arms consisting of revolute joints only. This restricted the $A(q)$ matrix's components to be linear combinations of cosines of the joint angles. Additionally, a base frame in which the x-direction is perpendicular to gravity was assumed. This led to the $G(q)$ matrix components being linear combinations of cosines of the joint angles.

$$A(q) = \begin{bmatrix} f(q_2,\ldots,q_n) & f(q_2,\ldots,q_n) & f(q_2,..,q_n) & \cdots & f(q_2,\ldots,q_n) \\ f(q_2,\ldots,q_n) & f(q_3,\ldots,q_n) & f(q_3,\ldots,q_n) & \ldots & f(q_3,\ldots,q_n) \\ f(q_2,\ldots,q_n) & f(q_3,\ldots,q_n) & \ddots & & \vdots \\ \vdots & \vdots & & f(q_n) & f(q_n) \\ f(q_2,\ldots,q_n) & f(q_3,\ldots,q_n) & \ldots & f(q_n) & C_n \end{bmatrix}$$

$$f(q_i,\ldots,q_n) = C + \mathrm{a}_1\cos(q_i) + \mathrm{a}_2\cos(q_i + q_{i+1}) + \cdots + a_{n-i}\cos\,(q_i + \cdots + q_n)$$

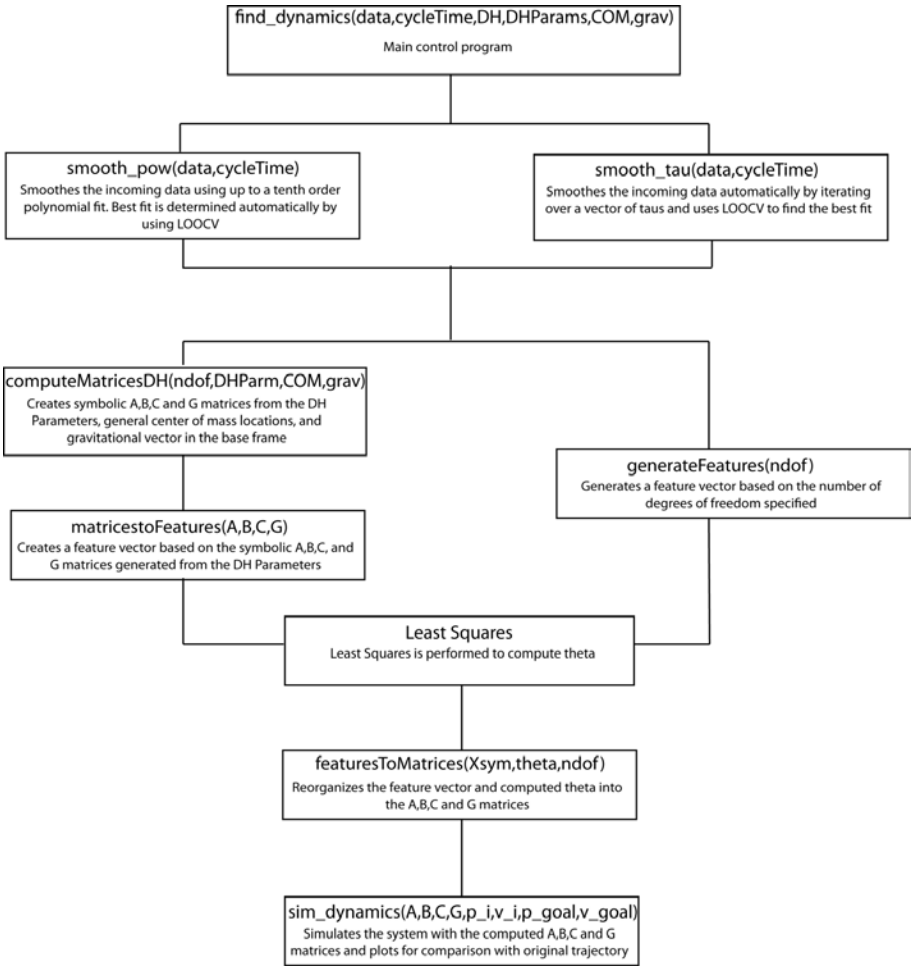
$$B(q) = \begin{bmatrix} 2b_{1,12} & 2b_{1,13} & \ldots & 2b_{1,1n} & 2b_{1,23} & \cdots & 2b_{1,2n} & \ldots & 2b_{1,(n-1)1} \\ 2b_{2,12} & 2b_{2,13} & \ldots & 2b_{2,1n} & 2b_{2,23} & \ldots & 2b_{2,2n} & \cdots & 2b_{2,(n-1)n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 2b_{n,12} & 2b_{n,13} & \cdots & 2b_{n,1n} & 2b_{n,23} & \cdots & 2b_{n,2n} & \cdots & 2b_{2,(n-1)n} \end{bmatrix}$$

$$C(q) = \begin{bmatrix} b_{1,11} & b_{1,22} & \ldots & b_{1,nn} \\ b_{2,11} & b_{2,22} & \ldots & b_{2,nn} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n,11} & b_{n,22} & \ldots & b_{n,nn} \end{bmatrix} \qquad b_{ijk} = \frac{1}{2}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i}\right)$$

$$G(q) = \begin{bmatrix} \mathrm{g}_{11}\cos(q_1) + g_{12}\cos(q_1 + q_2) + \cdots + g_{1n}\cos\,(q_1 + \cdots + q_n) \\ g_{21}\cos(q_1 + q_2) + \cdots + g_{2(n-1)}\cos\,(q_1 + \cdots + q_n) \\ \vdots \\ g_{n1}\cos\,(q_1 + \cdots + q_n) \end{bmatrix}$$

Code Structure

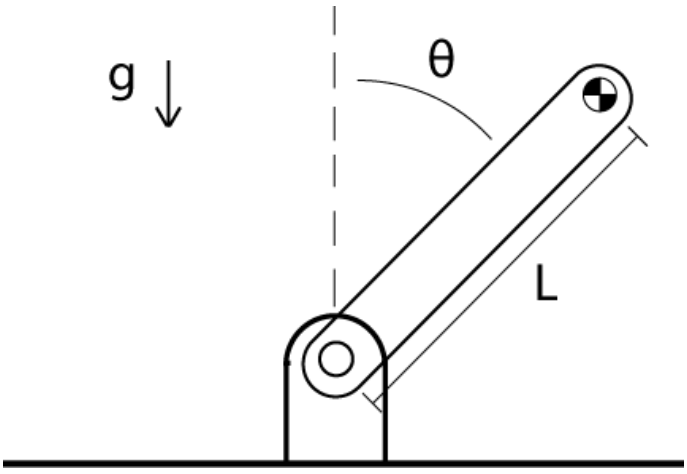
A generalized code structure was then setup to allow any of these methodologies to be used in combination. The user can pass in the data and the options desired for smoothing and feature vector creation and the program returns the matrices found and plots of the smoothed velocity, the smoothed acceleration, and simulated motion from the resulting computed matrices.

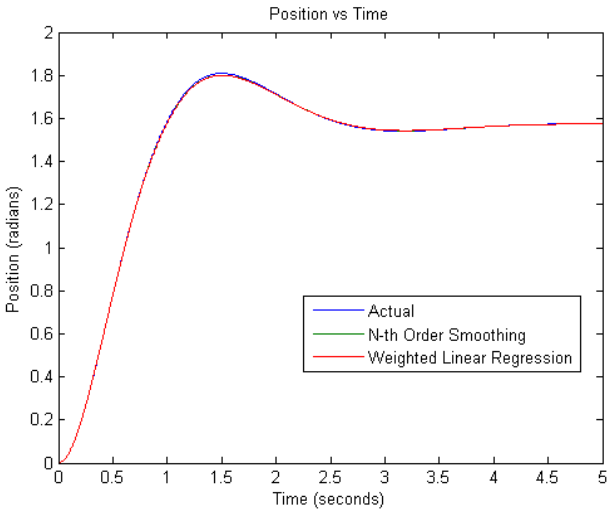


<u>One Degree of Freedom Results</u>

After the basic structure for the program was setup, a one degree of freedom robotic arm was simulated to test the effects of the differences between the two smoothing methods. Noise was added to the simulated position and velocity data and the feature vector in both cases was created using the specified DH parameters.

Simulation Equations of Motion	N-th Order Polynomial Fit	Weighted Linear Regression
$2\ddot{\theta} + 9.8\cos\theta = \tau$	$1.9727\ddot{\theta} + 9.9633\cos\theta = \tau$	$1.9697\ddot{\theta} + 9.9745\cos\theta = \tau$





While both result it close approximations of the simulated equations of motion, the n-th order polynomial fit slightly out performs the weighted linear regression in this case. Weighted linear regression also took a much longer time than the n-th order polynomial fit.

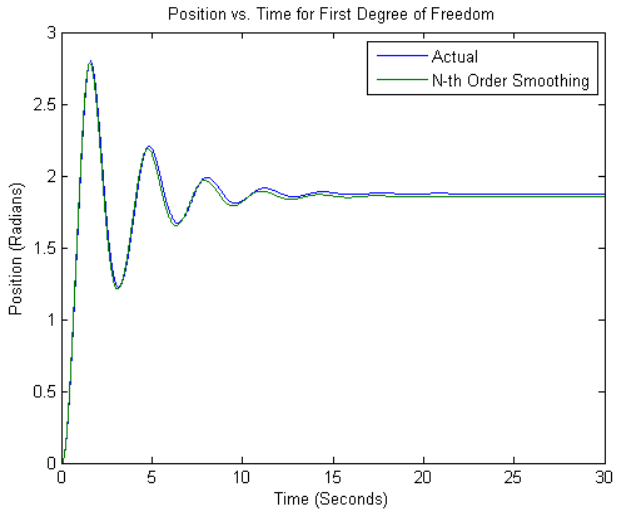
The test revealed another important aspect of finding the matrices. Initially, the simulated trajectory inputted into the program was a more ideal trajectory from a controls point of view, with no overshoot or oscillation. The calculated parameters were not as close to the actual values as in the above case. Further testing with higher degree of freedom systems and varying gains revealed that to get more accurate results from this methodology, the trajectory coming in must have all the modes activated. That is, more oscillation helped to get better parameter estimations.

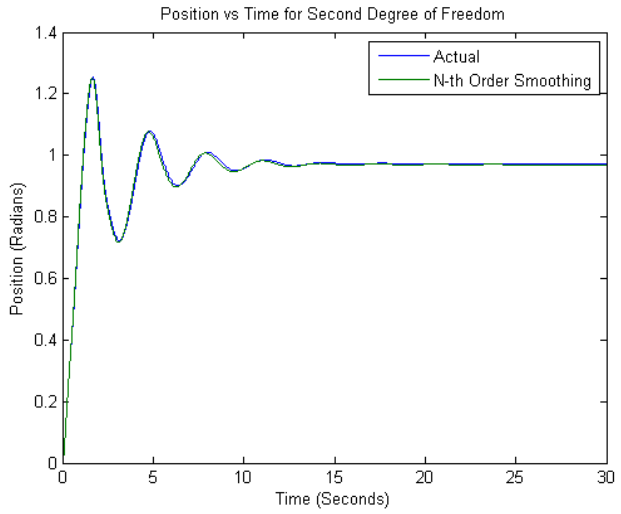
Two Degree of Freedom Results

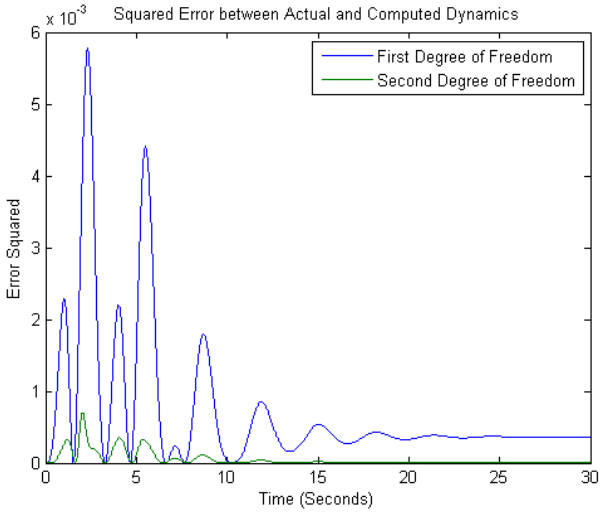
To test the auto-generated feature vector, a two degree of freedom, planar revolute joint arm was simulated. The following equations are the simulation equation of motion and the computed equation of motion respectively.

$$\begin{bmatrix} 5+2\cos q_2 & 2+\cos q_2 \\ 2+\cos q_2 & 2 \end{bmatrix}\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -2\sin q_2 \\ 0 \end{bmatrix}\dot{q}_1\dot{q}_2 + \begin{bmatrix} 0 & -\sin q_2 \\ \sin q_2 & 0 \end{bmatrix}\begin{bmatrix} \dot{q}_1^{\,2} \\ \dot{q}_2^{\,2} \end{bmatrix} + \begin{bmatrix} 19.6\cos q_1 + 9.8\cos(q_1+q_2) \\ 9.8\cos(q_1+q_2) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

$$\begin{bmatrix} 4.93+1.93\cos q_2 & 1.93+.996\cos q_2 \\ 1.93+.990\cos q_2 & 1.96 \end{bmatrix}\begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -1.98\sin q_2 \\ 0 \end{bmatrix}\dot{q}_1\dot{q}_2 + \begin{bmatrix} 0 & -.996\sin q_2 \\ .981\sin q_2 & 0 \end{bmatrix}\begin{bmatrix} \dot{q}_1^{\,2} \\ \dot{q}_2^{\,2} \end{bmatrix} + \begin{bmatrix} 19.1\cos q_1 + 9.39\cos(q_1+q_2) \\ 9.71\cos(q_1+q_2) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$







Other cases in which extra terms existed that were not in the simulated dynamic model performed well but not as well as this case. The extra terms tended to have very small coefficients that affected the resulting simulation to a greater degree than that above but still not large enough to cause significant deviation.

Further Work

One thing to notice is that the mass matrix is not symmetric in the two degree of freedom example. For simulation, the mass matrix was made symmetric using the worse estimate but further work could improve up the current process by ensuring the mass matrix is symmetric. Additionally, since the $B(q)$ and $C(q)$ matrices are derivatives of the $A(q)$ matrix, a more accurate set of equations may be able to be found by try to learn just $A(q)$ and then derivating to find $B(q)$ and $C(q)$. Further work could also be done to ensure that trajectories coming into the program activate all modes. Commanding oscillatory torques and recording the output to be put into the currently developed may lead to better matrix estimations and would lead to a more completely automated system.