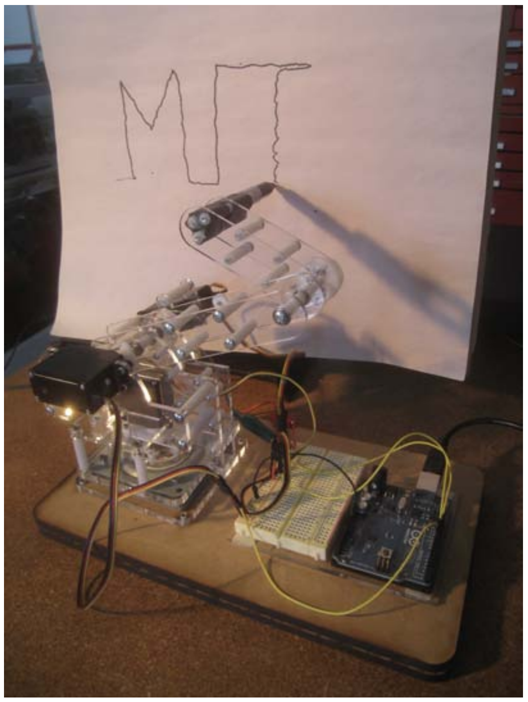
Problem Set 5 — 2.086 Spring 2012

Released: *Monday, 7 May*
Due: *Thursday, 17 May, at 2:30 PM, by hardcopy at the beginning of class.*

Please also upload to Stellar any MATLAB *function/script files you are required to supply by hardcopy in your problem set document.*

The demonstration robot arm of Figure 1 is represented schematically in Figure 2. Note that although the robot of Figure 1 has three degrees-of-freedom ("shoulder," "elbow," and "waist"), we will be dealing with only two degrees-of-freedom — "shoulder" and "elbow" — in this assignment.



Robot and photograph courtesy of James Penn.

Figure 1: Demonstration robot arm.

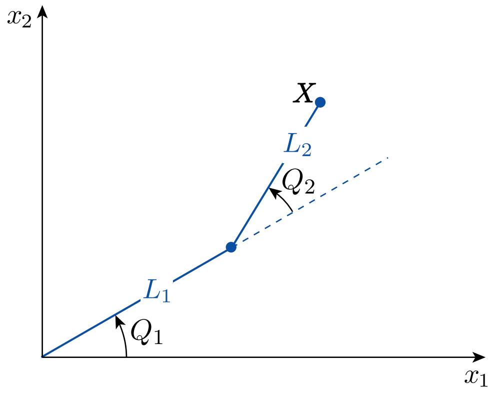


Figure 2: Schematic of robot arm.

The forward kinematics of the robot arm determine the coordinates of the end effector $\boldsymbol{X} = [X_1, X_2]^{\mathrm{T}}$ for given joint angles $\boldsymbol{Q} = [Q_1, Q_2]^{\mathrm{T}}$ as

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} (\boldsymbol{Q}) = \begin{bmatrix} L_1 \cos(Q_1) + L_2 \cos(Q_1 + Q_2) \\ L_1 \sin(Q_1) + L_2 \sin(Q_1 + Q_2) \end{bmatrix} , \qquad (1)$$

where L_1 and L_2 are the lengths of the first and second arm links, respectively. For our robot, $L_1 = 4$ inches and $L_2 = 3.025$ inches.

The inverse kinematics of the robot arm — the joint angles $\boldsymbol{Q}$ needed to realize a particular end effector position $\boldsymbol{X}$ — are not so straightforward and, for many more complex robot arms, a closed-form solution does not exist. In this assignment, we will solve the inverse kinematic problem for a two degree-of-freedom, planar robot arm by solving numerically for Q_1 and Q_2 from the set of nonlinear Equations (1) .

Given a trajectory of data vectors $\boldsymbol{X}_{(i)}$, $1 \le i \le p$ — a sequence of p desired end effector positions — the corresponding joint angles satisfy

$$\boldsymbol{F}\big(\boldsymbol{Q}_{(i)}, \boldsymbol{X}_{(i)}\big) = 0, \quad 1 \le i \le p \;, \tag{2}$$

where

$$\boldsymbol{F}(\boldsymbol{q}, \boldsymbol{X}) = \begin{matrix} F_1 \\ F_2 \end{matrix} = \begin{matrix} L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) - X_1 \\ L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) - X_2 \end{matrix}$$

For the robot "home" position, $\boldsymbol{X}^{\text{home}} \approx [-0.7154, 6.9635]^{\text{T}}$, the joint angles are known: $\boldsymbol{Q}^{\text{home}} = [1.6, 0.17]^{\text{T}}$ (in radians). We shall assume that $\boldsymbol{X}_{(1)} = \boldsymbol{X}^{\text{home}}$ and hence $\boldsymbol{Q}_{(1)} = \boldsymbol{Q}^{\text{home}}$ in all cases; it will remain to find $\boldsymbol{Q}_{(2)}, \ldots, \boldsymbol{Q}_{(p)}$.

Based on the design of our robot, we impose the following physical constraints on Q_1 and Q_2:

$$\sin(Q_1) \ge 0\,; \qquad \sin(Q_2) \ge 0 \;. \tag{3}$$

Note that a mathematically valid solution of Equation (2) might not satisfy the constraints of Equation (3) and, therefore, will need to be checked for physical consistency.

Note in the nomenclature of the general formulation of Chapter 23 of the notes, $n = 2$, $\boldsymbol{F}$ here corresponds to $\boldsymbol{f}$ of the notes, $\boldsymbol{q}$ here corresponds to $\boldsymbol{z}$ of the notes, $\boldsymbol{Q}$ here corresponds to $\boldsymbol{Z}$ of the notes, Equation (3) here corresponds to (more precisely, can be posed as) the constraint condition $C(\boldsymbol{Z}) = 1$ of the notes, $\ell = 2$, and $\boldsymbol{X}$ here corresponds to $\boldsymbol{\mu}$ of the notes.

Questions

1. (10 pts) Derive and provide the expression for the Jacobian of the two degree-of-freedom, planar robot arm, defined as

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial F_1}{\partial q_1} & \dfrac{\partial F_1}{\partial q_2} \\ \dfrac{\partial F_2}{\partial q_1} & \dfrac{\partial F_2}{\partial q_2} \end{bmatrix} , \tag{4}$$

   in terms of L_1, L_2, and trigonometric functions of q_1 and q_2. Make sure to explicitly indicate all four entries of $\boldsymbol{J}$.

2. (30 pts) We provide a function `Newton` (with simple continuation/homotopy) which can be called as in the script `run_Newton` to solve (2). (These codes are included in the Appendix and are also available, along with all the requisite data for the assignment, on Stellar.)

   Write and provide the MATLAB functions for `robot_f`, `robot_Jac`, and `robot_constraint` which must be supplied to complete the package. The function `robot_f` takes as its input arguments column vectors `q` and `X` (in that order) and returns the column vector `F(q,X)` as defined above in relation to Equation (2). The function `robot_Jac` takes as its input arguments column vectors `q` and `X` (even though `X` will not affect the return value) and returns the Jacobian matrix `J` as defined in Equation (4). The function `robot_constraint` takes as its input argument a column vector `Q` and returns `1` if the constraint of Equation (3) is satisfied and `0` if it is not.

Deliverable: Your three functions `robot_f`, `robot_Jac`, and `robot_constraint` (which you should copy and paste into your problem set document and also upload to Stellar).

3. (10 pts) Create and provide two test trajectories $(\boldsymbol{X}^{\text{home}}, \boldsymbol{X}^{\text{near}})$ and $(\boldsymbol{X}^{\text{home}}, \boldsymbol{X}^{\text{far}})$ with which you can test your Newton algorithm; note that in both cases, $p = 2$ — the robot starts at $\boldsymbol{X}^{\text{home}}$ and moves to either $\boldsymbol{X}_{(2)} = \boldsymbol{X}^{\text{near}}$ for the "near" trajectory or $\boldsymbol{X}_{(2)} = \boldsymbol{X}^{\text{far}}$ for the "far" trajectory. The points $\boldsymbol{X}^{\text{near}}$ and $\boldsymbol{X}^{\text{far}}$ should be located "near" and "far," respectively, from $\boldsymbol{X}^{\text{home}}$, where "near" corresponds to a small change in $\boldsymbol{Q}$ and "far" corresponds to a large change in $\boldsymbol{Q}$. Provide exact solutions to (2), $(\boldsymbol{Q}^{\text{home}}, \boldsymbol{Q}^{\text{near}})$ and $(\boldsymbol{Q}^{\text{home}}, \boldsymbol{Q}^{\text{far}})$, for your two trajectories.

   Hint: First choose joint angles $\boldsymbol{Q}^{\text{near}}$ and $\boldsymbol{Q}^{\text{far}}$ (with respect to $\boldsymbol{Q}^{\text{home}} = [1.6, 0.17]^{\text{T}}$) and *then* do the forward kinematics of (1) to determine your trajectory. To avoid convergence issues, be careful not to choose joint angles too close (for example, within less than 0.1 radians) to 0 or π. Also, be sure to choose Q_1 and Q_2 that satisfy the constraint of Equation 3.

   Deliverable: Your test trajectories $(\boldsymbol{X}^{\text{home}}, \boldsymbol{X}^{\text{near}})$ and $(\boldsymbol{X}^{\text{home}}, \boldsymbol{X}^{\text{far}})$ and the corresponding exact solutions $(\boldsymbol{Q}^{\text{home}}, \boldsymbol{Q}^{\text{near}})$ and $(\boldsymbol{Q}^{\text{home}}, \boldsymbol{Q}^{\text{far}})$.

Note in all cases below you should set the Newton tolerance `tol` to `1e-6` (as done for you already in the provided `run_Newton.m`).

4. (10 pts) Implement your code of Question 2 in MATLAB and run the provided `run_Newton` code for the "near" and "far" trajectories of Question 3. You will need to modify the single (incomplete) line in `run_Newton` in which the variable `Xadd` is defined, but you should leave the other lines as they are.

   (*i*) Do you converge to the known values for $\boldsymbol{Q}^{\text{near}}$ and $\boldsymbol{Q}^{\text{far}}$?

   (*ii*) Do you require the insertion of intermediate data vectors in the Newton continuation process of `Newton` for either the "near" or "far" trajectories? Note that the variable `orig_inds` returned by `Newton` can be used to check for inserted data vectors.

Download the data file `trajectories.mat` containing the two trajectories `X_A` and `X_B`.

5. (20 pts) Apply the Newton code to trajectory `X_A` provided to you.

   (*i*) Show on a single plot the data vectors $\boldsymbol{X}_{(i)}$, the (inserted by `Newton.m`) intermediate data vectors, and the positions of both arm links for each point (and inserted point) in the trajectory. Figure 3 is an example of such a plot. Make sure to include axis labels, a title, and a legend.

   (*ii*) How many Newton iterations in total — in other words, how many "`J \`" — are required to generate the plot? (You should modify `Newton.m` to include a "counter variable" immediately following the statement in which the backslash occurs (i.e., the Jacobian system is solved).)

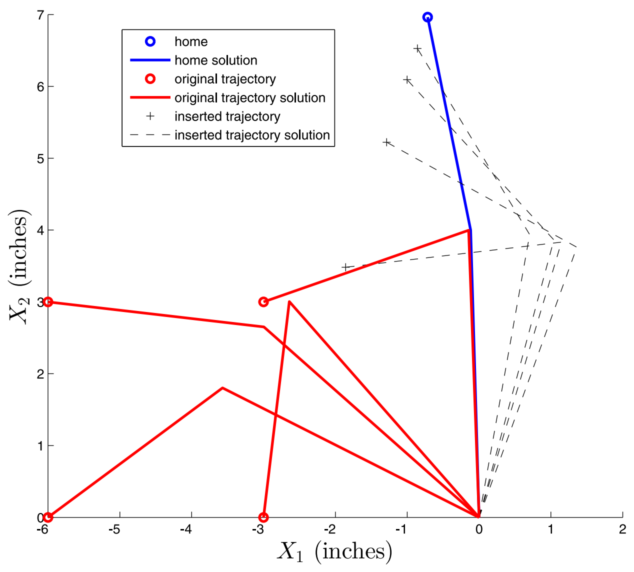


Figure 3: Example plot of trajectory points and the positions of both arm links at each point.

(*iii*) If you change `numNewtonitersmax_per` in `Newton` (from its default value of 6) can you reduce this total iteration count and why or why not? As part of your answer you should consider a few values for `numNewtonitersmax_per` less than and greater than 6 and report the total iteration count for each case.

6. (20 pts) Apply the Newton code to trajectory `X_B` provided to you.

(*i*) Show on a single plot the data vectors $\boldsymbol{X}_{(i)}$, the (inserted by `Newton.m`) intermediate data vectors, and the positions of both arm links for each point (and inserted point) in the trajectory. Make sure to include axis labels, a title, and a legend.

(*ii*) How many Newton iterations in total — in other words, how many "`J \`" — are required to generate the plot?

(*iii*) If you change `numNewtonitersmax_per` in `Newton` can you reduce this total iteration count and why or why not? As part of your answer you should consider a few values for `numNewtonitersmax_per` less than and greater than 6 and report the total iteration count for each case.

Appendix

Newton.m

```
function [ zvec_expanded, muvec_expanded, orig_inds ] = Newton( f, muvec, Jac, constraint, tol, z_start )

% Newton is a code to find zvec(:,k) such that f(zvec(:,k),muvec(:,k))=0,
% k = 1,...,p: here muvec(:,k) shall be denoted
% the k^th "(continuation) data vector."

% Here f and Jac are handles to functions which evaluate f and the Jacobian of
% f, 'constraint' is a handle to a function which checks the constraints
% (true = satisfied), tol is the Newton residual tolerance, and z_start
% must satisfy f(z_start,muvec(:,1)) = 0 --- a good starting point.

numparvals = size(muvec,2);
% Here numparvals = p is the number of data vectors in our trajectory.

orig_inds = logical(ones(1,numparvals));
% We will introduce intermediate data vectors to form an
% expanded trajectory (or homotopy) muvec_expanded in order to ensure/improve
% convergence of the Newton iteration.
% We denote by zvec_expanded the Newton solutions for the data vectors
% of this expanded trajectory: f(zvec_expanded(:,k),muvec_expanded(:,k)) = 0
% for k = 1,...,p_expanded (the number of points in the expanded trajectory).
% The logical array orig_inds allows us to extract the p solutions zvec
% from the p_expanded solutions zvec_expanded: zvec = zvec_expanded(:,orig_inds).

numparvals_left = numparvals;
% Here numparvals_left is the number of data vectors in our expanded
% trajectory which we must still visit.

ind_current = 1;
% Here ind_current is the current index of the data vector in our expanded trajectory.

zvec_expanded = [];
muvec_expanded = muvec; % We begin with data vectors of interest as input to Newton.

z_iter = z_start; % Here z_iter is the current Newton iterate.


numNewtonitersmax_per = 6;
% If Newton has not converged for a particular data vector
% by numNewtonitersmax_per iterations, we insert a intermediate
% data vector --- in our expanded trajectory --- to ensure/improve convergence.
% (Recall the algorithm can not treat turning points or bifurcation points.)

while (numparvals_left > 0)
    % We visit each trajectory point and perform Newton iteration.

    mu_current = muvec_expanded(:,ind_current);
    % Here mu_current is the current data vector from our expanded trajectory.

    numNewtoniters = 0;
    while ( norm(f(z_iter,mu_current)) >= tol ...
            & numNewtoniters < numNewtonitersmax_per )
        dz = -Jac(z_iter,mu_current)\f(z_iter,mu_current);
        z_iter = z_iter + dz;
        numNewtoniters = numNewtoniters + 1;
    end



    if ( norm(f(z_iter,mu_current)) <= tol & constraint(z_iter) )
        % If we have converged and our constraints on the solution are
        % satisfied then we store the solution, we decrease the number
        % of data vectors still to visit, we increment the data vector
        % counter, and we set the initial guess for the next data vector in
        % the trajectory to be the converged solution for the current data
        % vector.
        zvec_expanded = [zvec_expanded,z_iter];
        numparvals_left = numparvals_left - 1;
        ind_current = ind_current + 1;
        z_iter = zvec_expanded(:,end);
    else
        % If we have not converged, or our constraints are not satisfied,
        % we insert a new data vector "halfway" between our last data
        % vector in the expanded trajectory (perforce converged) and the
        % current data vector in the expanded trajectory.
        % We also insert a false in orig_inds to indicate that
        % this new data vector is an intermediate data vector.
        % Note since we have inserted a new point in muvec_expanded,
        % we need to increment numparvals_left. In addition,
        % we need to keep ind_current unchanged.
        % Finally, the initial guess for the new intermediate data
        % vector remains the last converged data vector.
        muvec_expanded = [muvec_expanded(:,1:ind_current-1),...
            0.5*(muvec_expanded(:,ind_current-1)+muvec_expanded(:,ind_current)),muvec_expanded(:,ind_current:end)];
        orig_inds = [orig_inds(1:ind_current-1),false,orig_inds(ind_current:end)];
        numparvals_left = numparvals_left+1;
        ind_current = ind_current;
        z_iter = zvec_expanded(:,end);
    end
end


end
```

run_Newton.m

```
%run_newton defines a starting point and trajectory and
%calls Newton() with the appropriate arguments. You
%will need to specify the trajectory where asked.

Qhome = [1.6; 0.17]; %joint angles for robot home position
                   %(and starting point for Newton)
Xhome = robot_f(Qhome,[0;0]); %X coordinates for robot home position

%SPECIFY TRAJECTORY ON FOLLOWING LINE (don't include Xhome
%because that will be added in the Xvec definition below).
Xadd = %X trajectory vectors (not including Xhome)

Xvec = [Xhome, Xadd]; %full trajectory is the concatenation
                     %of Xhome and Xadd

%call Newton (see Newton.m code for additional documentation)
%you should not need to alter this line, but you will
%need to write the robot_f, robot_Jac, and robot_constraint
%functions that are passed as arguments
[Qvec_expanded, Xvec_expanded, orig_inds] = Newton( @robot_f, Xvec, @robot_Jac, @robot_constraint, 1e-6, Qhome );
```