Assignment 6 — 2.086/2.090 Spring 2013

Released: *Friday, 3 May, at 5 PM.*

Due: *Thursday, 16 May, at 5 PM.*

Upload your solution as a zip file "`YOURNAME_ASSIGNMENT_6`" which includes for each question the `AxQy` script *as well as* any MATLAB functions of your own creation. Note the MATLAB functions of your own creation should include both specific MATLAB functions requested in the question statement as well as any other MATLAB functions (called directly or indirectly by your other functions) which you choose to develop as part of your answer. Both the scripts and (re-quested) functions must conform to the formats described in **Instructions** and **Questions** below. You should also include in your folder all the `grade_o_matic .p` files for Assignment 6.

Instructions

Before embarking on this assignment you should

(1) Complete the Textbook reading for Unit VII and review the Lecture Notes for Unit VII

(2) Execute ("cell-by-cell") the MATLAB Tutorial for Unit VII MAT- LAB `fsolve`. Note that in fact `fsolve` is only needed for the "Reverse" Challenge at the conclusion of assignment — which is (ungraded and hence) entirely elective on your part.

(3) Download the `Assignment_6_Materials` folder. This folder contains the script for each question (`A6Qy.p` for Question `y`), as well as a template for each function which we ask you to create (`func_Template` for a function `func`). The `Assignment_6_Materials` folder also contains the `grade_o_matic` codes needed for Assign-ment 6. (Please see Assignment 1 for a description of `grade_o_matic`.)

We indicate here several general format and performance requirements:

(a.) Your script for Question `y` of Assignment `x` *must* be the provided `A6Qy.p`: for Assign-ment 6, no modifications are needed to these scripts, and you should upload these scripts directly "as is" in your `YOURNAME_ASSIGNMENT_6` folder. (Note that although no modifi-cations are required, you must upload the `A6Qy.p` scripts or `grade_o_matic_A6` will not perform correctly.)

(b.) In this assignment, for each question `y`, we will specify inputs and outputs both for the script `A6Qy` and (as is more traditional) any requested MATLAB functions; we shall denote the former as script inputs and script outputs and the latter as function inputs and function outputs. For each question and hence each script, and also each function, we will identify *allowable instances* for the inputs — the parameter values or "parameter domains" for which the codes must work. *In fact, for Assignment 6, there are only function inputs and outputs — there are no script inputs or outputs.*

(c.) We ask that you not end any of your script names or function names with the suf-fix `_ref` in order to prevent conflicts with scripts or functions provided in the folder `Assignment_6_Materials` (and needed by `grade_o_matic_A6`).

(d.) We ask that in the submitted version of your scripts and functions you suppress all display by placing a "`;`" at the end of each line of code. (Of course, during debugging you will often choose to display many intermediate and final results.) We also require that **before** you upload your solution you should run `grade_o_matic_A6` (from your `YOURNAME_ASSIGNMENT_6` folder) for final confirmation that all is in order.

Preamble

The 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 consider only two degrees-of-freedom — "shoulder" and "elbow" — in this assignment.

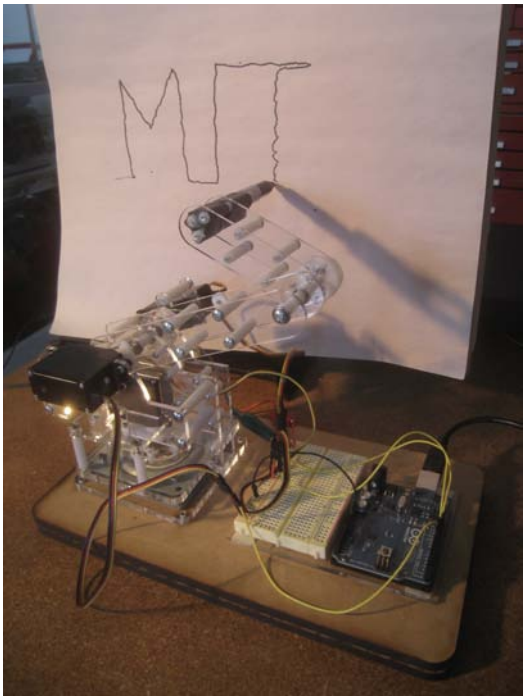

Figure 1: Robot arm.
(Robot and photograph courtesy of James Penn.)

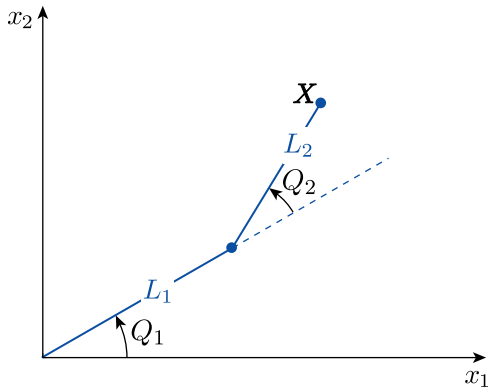

Figure 2: Schematic of robot arm: end effector $\boldsymbol{X} = (X_1\ X_2)^{\mathrm{T}}$ and joint angles $\boldsymbol{Q} = (Q_1\ Q_2)^{\mathrm{T}}$.

The geometry of the robot arm imposes a relationship between the end effector location $\boldsymbol{X} = (X_1\ X_2)^{\mathrm{T}}$ and the joint angles $\boldsymbol{Q} = (Q_1\ Q_2)^{\mathrm{T}}$:

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} L_1\cos(Q_1) + L_2\cos(Q_1+Q_2) \\ L_1\sin(Q_1) + L_2\sin(Q_1+Q_2) \end{pmatrix}, \tag{1}$$

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. (Our robot is British.) Note that throughout this assignment we shall assume (and impose), without any loss of generality, that $0 \le Q_1 < 2\pi$ and $0 \le Q_2 < 2\pi$.

In this assignment we are interested in the *inverse kinematics* of the robot arm: for given end effector position $\boldsymbol{X}$ (in units of inches), we wish to find the robot joint angles $\boldsymbol{Q}_{\boldsymbol{X}}$ (in units of

radians) which will realize $\boldsymbol{X}$. Towards that end, we introduce the function

$$\boldsymbol{f}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X}) \equiv \begin{pmatrix} f_1^{\text{robot}} \\ f_2^{\text{robot}} \end{pmatrix} = \begin{pmatrix} L_1\cos(q_1) + L_2\cos(q_1+q_2) - X_1 \\ L_1\sin(q_1) + L_2\sin(q_1+q_2) - X_2 \end{pmatrix} . \tag{2}$$

We note that $\boldsymbol{f}^{\text{robot}}$ is a 2×1 vector which depends on the joint angles $\boldsymbol{q} = (q_1 \ q_2)^{\text{T}}$ — for $0 \leq q_1 < 2\pi, 0 \leq q_2 < 2\pi$ — and the desired end effector location $\boldsymbol{X}$. It follows from Equations (1) and (2) that, for given end effector position $\boldsymbol{X}$, the joint angles $\boldsymbol{Q}_{\boldsymbol{X}}$ must satisfy

$$\boldsymbol{f}^{\text{robot}}(\boldsymbol{Q}_{\boldsymbol{X}};\boldsymbol{X}) = \boldsymbol{0} . \tag{3}$$

(In Equation (2) $\boldsymbol{q}$ is a dummy variable which refers to any joint angles we may wish to consider; in contrast, $\boldsymbol{Q}_{\boldsymbol{X}}$ refers to a solution, for given $\boldsymbol{X}$, of $\boldsymbol{f}^{\text{robot}}(\boldsymbol{Q}_{\boldsymbol{X}};\boldsymbol{X}) = \boldsymbol{0}$.)

We should note that $\boldsymbol{f}^{\text{robot}}$ in fact has two interpretations, both of which should prove useful in this assignment. First, $\boldsymbol{f}^{\text{robot}}(\boldsymbol{Q};(0\ 0)^{\text{T}})$ is, from Equations (1) and (2), the end effector location $\boldsymbol{X}_{\boldsymbol{Q}}$ for joint angles $\boldsymbol{Q}$; this ("forward kinematics") interpretation will be important in the development of test cases and first guesses for the inverse kinematics problem, Equation (3). Second, for given $\boldsymbol{q}$ and $\boldsymbol{X}$, $\boldsymbol{f}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X})$ is the extent to which the joint angle–end effector relation (Equation (1)) is *not* satisfied for some proposed joint angle $\boldsymbol{q}$ and end effector position $\boldsymbol{X}$; this "residual" interpretation will be important in the iterative solution of the inverse kinematics problem, Equation (3).

In fact, our problem statement is not yet complete: we must include one more feature of the actual robot arm. In particular, the joint actuation is limited: the joint angles $\boldsymbol{Q}$ (and hence also $\boldsymbol{Q}_{\boldsymbol{X}}$) must satisfy the condition

$$0 \leq Q_1 \leq \pi \, ; 0 \leq Q_2 \leq \pi \ . \tag{4}$$

In many cases Equation (3) will have two solutions — corresponding to two "branches" — and we must make sure to only accept a root which satisfies the constraint Equation (4). Note that, unlike our reference requirement $0 \leq Q_1 < 2\pi, 0 \leq Q_2 < 2\pi$, the condition of Equation (4) is a real restriction on the locations which may be reached by the end effector. The latter is referred to as the *robot workspace*, which we can define more precisely (from our first interpretation of $\boldsymbol{f}^{\text{robot}}$) as the set of all $\boldsymbol{X}$ such that $\boldsymbol{X} = \boldsymbol{f}^{\text{robot}}(\boldsymbol{Q};(0\ 0)^{\text{T}})$ for some joint angle $\boldsymbol{Q}$ which satisfies Equation (4).

We shall consider Newton iteration for solution of Equation (3). We will thus need the Jacobian of $\boldsymbol{f}^{\text{robot}}$, which we recall is the 2×2 matrix

$$\boldsymbol{J}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X}) = \begin{pmatrix} \dfrac{\partial f_1^{\text{robot}}}{\partial q_1}(\boldsymbol{q};\boldsymbol{X}) & \dfrac{\partial f_1^{\text{robot}}}{\partial q_2}(\boldsymbol{q};\boldsymbol{X}) \\ \dfrac{\partial f_2^{\text{robot}}}{\partial q_1}(\boldsymbol{q};\boldsymbol{X}) & \dfrac{\partial f_2^{\text{robot}}}{\partial q_2}(\boldsymbol{q};\boldsymbol{X}) \end{pmatrix} . \tag{5}$$

You will need to derive explicit expressions for the entries of $\boldsymbol{J}^{\text{robot}}$ as a prerequisite to Question 2 below.

Questions

1. (10 points: Note both components of `f_robot` must be correct or no credit is earned.) In this question we would like you to write a MATLAB function with signature

```
function [f] = f_robot(q,X)
```

which implements the function $\boldsymbol{f}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X})$ of (2). Note that `f_robot` is defined for our particular robot arm and inasmuch the upper arm and lower arm lengths, L_1 and L_2, respectively, should be "hardwired" inside `f_robot`.

The function must be named `f_robot` and furthermore must be stored in a file named `f_robot.m`. The function takes two function inputs. The first input is the 2×1 vector `q` which corresponds to the joint angle $\boldsymbol{q} = (q_1 \ q_2)^{\text{T}}$ of Equation (2); the set of allowable instances, or parameter domain, is $0 \leq q_1 < 2\pi, 0 \leq q_2 < 2\pi$. The second input is the 2×1 vector `X` which corresponds to the end effector position $\boldsymbol{X}$ of Equation (2); the set of allowable instances, or parameter domain, is not restricted. The function yields a single function output: the output is the 2×1 vector `residual` which corresponds to $\boldsymbol{f}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X})$ of Equation (2).

The script for this question is provided in `A6Q1.p`; you should not modify this script in any way. We also provide a template for your `f_robot` function in `f_robot_Template`. You should include both `A6Q1.p` and `f_robot.m` in the `YOURNAME_ASSIGNMENT_6` folder you upload Note that `grade_o_matic_A6` shall choose various "student" and "grader" instances of the function inputs to determine if your code is correct; make sure that you also test your code yourself — a few "by hand" calculations and limiting cases should suffice.

2. (20 points: Note all four entries of `J_robot` must be correct or no credit is earned.) In this question we would like you to write a MATLAB function with signature

```
function [J] = J_robot(q,X)
```

which implements the function $\boldsymbol{J}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X})$ of (5). Note that `J_robot` is defined for our particular robot arm and inasmuch the upper arm and lower arm lengths, L_1 and L_2, respectively, should be "hardwired" inside `J_robot`.

The function must be named `J_robot` and furthermore must be stored in a file named `J_robot.m`. The function takes two function inputs. The first input is the 2×1 vector `q` which corresponds to the joint angle $\boldsymbol{q} = (q_1 \ q_2)^{\text{T}}$ of Equation (5); the set of allowable instances, or parameter domain, is $0 \leq q_1 < 2\pi, 0 \leq q_2 < 2\pi$. The second input is the 2×1 vector `X` which corresponds to the end effector position $\boldsymbol{X}$ of Equation (5); the set of allowable instances, or parameter domain, is not restricted. The function yields a single function output: the output is the 2×2 matrix `J` which corresponds to $\boldsymbol{J}^{\text{robot}}(\boldsymbol{q};\boldsymbol{X})$ of Equation (5).

The script for this question is provided in `A6Q2.p`; you should not modify this script in any way. We also provide a template for your `J_robot` function in `J_robot_Template`. You should include both `A6Q2.p` and `J_robot.m` in the `YOURNAME_ASSIGNMENT_6` folder you upload Note that `grade_o_matic_A6` shall choose various "student" and "grader" instances of the function inputs to determine if your code is correct; make sure that you also test your code yourself.

3. (30 points: 7.5 points for each of `Q_ws(1,:)`, `Q_ws(2,:)`, `X_ws(1,:)`, and `X_ws(2,:)`.) In

this question we ask you to prepare a set of M joint angle–end effector positions,

$$(\boldsymbol{X}_k, \boldsymbol{Q}_k), 1 \leq k \leq M \ , \tag{6}$$

such that

Requirement I. the $\boldsymbol{Q}_k, 1 \leq k \leq M$, are admissible — satisfy the constraint Equation (4) — and are distinct — $\boldsymbol{Q}_k \neq \boldsymbol{Q}_{k'}$ for $k \neq k'$;

Requirement II. $\boldsymbol{f}^{\text{robot}}(\boldsymbol{Q}_k; \boldsymbol{X}_k) = \boldsymbol{0}, 1 \leq k \leq M$ — the joint angle–end effector position pairs satisfy the relationship Equation (1).

Note the set of joint angle-end effector pairs can serve several useful purposes: first, to visualize the robot workspace — which we recall is the set of end effector positions which can be "reached" by admissible joint angles; second, to improve the Newton iteration robustness and convergence — a set from which to choose a good initial guess. (The latter is discussed in greater depth in Question 4.)

We would thus like you to write a MATLAB function with signature

```
function [Q_ws,X_ws] = workspace_robot(M)
```

which creates the "discrete" robot workspace as defined in Equation (6). The function must be named `workspace_robot` and furthermore must be stored in a file named `workspace_robot.m`. The function takes one function input: this function input is `M` which corresponds to M of Equation (6) — the number of joint-angle–end effector position pairs generated; the set of allowable instances, or parameter domain, is $1 \leq M \leq 10000$. The function yields two function outputs. The first output is the $2 \times M$ array `Q_ws` which corresponds to $\boldsymbol{Q}_k, 1 \leq k \leq M$ of Equation (6); note that `Q_ws(:,k)` $= \boldsymbol{Q}_{\texttt{k}}$. The second output is the $2 \times M$ array `X_ws` which corresponds to $\boldsymbol{X}_k, 1 \leq k \leq M$ of Equation (6); note that `X_ws(:,k)` $= \boldsymbol{X}_{\texttt{k}}$.

The script for this question is provided in `A6Q3.p`; you should not modify this script in any way. We also provide you with a template for your `workspace_robot` function in `workspace_robot_Template`. You should include both `A6Q3.p` and `workspace_robot` in the `YOURNAME_ASSIGNMENT_6` folder you upload Note that `grade_o_matic_A6` shall choose various "student" and "grader" instances of the function inputs to determine if your code is correct and in particular to confirm that you honor Requirement I. and Requirement II. in your construction; make sure that you also test your code yourself.

Hints and Guidelines. You will note that the Requirement I. above does not completely specify the $\boldsymbol{Q}_k, 1 \leq k \leq M$. To choose the M joint angles we ask that you follow a particular approach: you should generate the $\boldsymbol{Q}_k, 1 \leq k \leq M$, as independent random darts from the bivariate uniform distribution over the rectangle $0 \leq Q_1 \leq \pi, 0 \leq Q_2 \leq \pi$.[1] We make two further observations: first, in the random approach, the "distinct" aspect of Requirement I. is satisfied probabilistically — but with probability extremely close to unity, which is sufficient for our purposes; second, in the random approach, we naturally avoid (with probability very close to unity) darts exactly on the boundary of the constraint set — which is advantageous, in particular to guide the Newton iteration towards solutions which satisfy Equation (4). Note that, once the $\boldsymbol{Q}_k, 1 \leq k \leq M$, are specified, the $\boldsymbol{X}_k, 1 \leq k \leq M$, are uniquely

[1] In higher dimensions, random darts are a reasonably good idea; in just two dimensions, a uniform mesh might be slightly better — but a bit more difficult to implement, and for that reason we opt for the random approach.

determined by Requirement II; Requirement II invokes forward and not inverse kinematics since $\boldsymbol{X}_k = \boldsymbol{f}^{\text{robot}}(\boldsymbol{Q}_k; (0\ 0)^T)$ — thus Requirement II is computationally very simple, and of course you should take good advantage of your `f_robot` MATLAB function of Question 1.

You can readily visualize the workspace once you have created `Q_ws,X_ws`. We might suggest `plot(Q_ws(1,:),Q_ws(2,:),'o')` to display the joint angles, and then (a second figure) `plot(X_ws(1,:),X_ws(2,:),'o')` to display the corresponding end effectors; the second figure "fills out" the robot workspace for M chosen sufficiently large. Note these plots are entirely elective and not part of the deliverable.

4. (40 points) We now consider Newton's method to solve the inverse kinematics problem: for given $\boldsymbol{X}$, we look for $\boldsymbol{Q}_{\boldsymbol{X}}$ which satisfies Equation (3) as well as the condition Equation (4). Our approach will comprise two stages. In the "offline" stage, we create with `workspace_robot(M)` a discrete workspace $(\boldsymbol{Q}_k, \boldsymbol{X}_k), 1 \le k \le M$. In the "online" stage we perform the Newton procedure.

   The online stage also comprises two steps. *In the first step*, we find k^* such that

   $$\|\boldsymbol{X} - \boldsymbol{X}_{k^*}\| \le \|\boldsymbol{X} - \boldsymbol{X}_k\| \text{ for all } k \text{ such that } k = k^* \ , \tag{7}$$

   and then choose $\boldsymbol{Q}_{k^*}$ as the initial guess (for $\boldsymbol{Q}_{\boldsymbol{X}}$) in the subsequent Newton algorithm. In words, $\boldsymbol{X}_{k^*}$ is the end effector in our discrete workspace which is closest to the desired end effector $\boldsymbol{X}$; our initial guess for the Newton algorithm, $\boldsymbol{Q}_{k^*}$, is the the joint angle which realizes $\boldsymbol{X}_{k^*}$. By continuity arguments we can expect that if M is large enough then our first guess $\boldsymbol{Q}_{k^*}$ should be close enough to $\boldsymbol{Q}_{\boldsymbol{X}}$ such that the Newton algorithm will converge and converge quickly (and furthermore to a solution which satisfies Equation (4)). *In the second step*, we perform the Newton algorithm — compute a sequence of iterates — until we satisfy a tolerance or exceed a prescribed maximum number of iterations; we must also confirm that Equation (4) is satisfied — and if not, alert the "user" that the calculated root is not admissible.

   We have already addressed the offline stage in Question 3: in the offline stage, we invoke `[Q_ws,X_ws] = workspace_robot(M)` for some given `M`. In the current question we address the online stage: we would like you to write a MATLAB function with signature

```
function [Q_X_hat,exitflag] = Newton_robot(X,Q_ws,X_ws,Newton_tol,num_iter_max)
```

   which implements the two steps of the online stage described above. Note that `Q_ws` and `X_ws` are inputs to `Newton_robot`, as described in more detail below.

   The function must be named `Newton_robot` and furthermore must be stored in a file named `Newton_robot.m`. The function takes five function inputs. The first input is the 2×1 vector `X` and corresponds to the end effector position $\boldsymbol{X}$ (of Equation (3)) for which we wish to find the corresponding joint angle $\boldsymbol{Q}_{\boldsymbol{X}}$; the set of allowable instances is not restricted. The second and third inputs are respectively the $2 \times M$ arrays `Q_ws` and `X_ws` which are simply the outputs of `workspace_robot(M)` for some given `M` (see Question 3); note that `M` is not an input to `Newton_robot`, but of course can be deduced in `Newton_robot` from (say) `Q_ws`. The fourth input is the scalar `Newton_tol` which is our residual tolerance: we terminate the Newton algorithm as soon as $\|\boldsymbol{f}^{\text{robot}}(\hat{\boldsymbol{Q}}_{\boldsymbol{X}}; \boldsymbol{X})\| \le$ `Newton_tol`, where $\hat{\boldsymbol{Q}}_{\boldsymbol{X}}$ is the current Newton iterate (and hence current approximation to $\boldsymbol{Q}_{\boldsymbol{X}}$); the set of allowable instances for `Newton_tol` is $1 \times 10^{-6} \le$ `Newton_tol` ≤ 0.01. The fifth and final input is `num_iter_max` which is the

maximum number of iterations we permit: we terminate the Newton algorithm once we have performed `num_iter_max` "Jacobian solves"; the set of allowable instances for `num_iter_max` is $4 \leq$ `num_iter_max` ≤ 10. The function yields two function outputs. The first output is the 2×1 vector `Q_X_hat` which is the Newton iterate $\hat{\boldsymbol{Q}}_{\boldsymbol{X}}$ upon termination of the Newton algorithm — and hence our approximation to $\boldsymbol{Q}_{\boldsymbol{X}}$ of Equation (3); recall we terminate as soon as either our tolerance is satisfied or we reach the maximum number of iterations permitted. The second output is the scalar `exitflag`: `exitflag = 1` if $\|\boldsymbol{f}^{\text{robot}}(\hat{\boldsymbol{Q}}_{\boldsymbol{X}}; \boldsymbol{X})\| \leq$ `Newton_tol` and $\hat{\boldsymbol{Q}}_{\boldsymbol{X}}$ satisfies Equation (4); `exitflag = 2` if $\|\boldsymbol{f}^{\text{robot}}(\hat{\boldsymbol{Q}}_{\boldsymbol{X}}; \boldsymbol{X})\| \leq$ `Newton_tol` but $\hat{\boldsymbol{Q}}_{\boldsymbol{X}}$ does not satisfy Equation (4); `exitflag = 3` if $\|\boldsymbol{f}^{\text{robot}}(\hat{\boldsymbol{Q}}_{\boldsymbol{X}}; \boldsymbol{X})\| >$ `Newton_tol`.

Hints and Guidelines. You should at the end of each Newton iteration — after each Newton update — enforce our condition $0 \leq Q_1 < 2\pi, 0 \leq Q_2 < 2\pi$ — easily implemented with the MATLAB built–in function `mod`; without this adjustment, `grade_o_matic_A6` will not award you credit. Note, however, that you should not enforce the condition Equation (4). (There are methods in which we can incorporate constraints (on which branch should be considered) directly into the iterative procedure, however these techniques require considerable additional background and are beyond our 2.086 introduction.)

You should make sure to invoke your MATLAB functions `f_robot` and `J_robot`. Functions permit encapsulation and re-use. You have debugged these functions in Questions 1 and 2 and now you reap the benefits. Note that you do not need to pass handles for these functions to `Newton_robot`, but rather you may directly hardwire (in `Newton_robot`) the calls to `f_robot` and `J_robot`: your `Newton_robot` function need only treat our particular robot arm.

You may find that the MATLAB built–in function `sort` with *two* outputs is very convenient to perform the first step, Equation (7), of the online procedure.

In order to test `Newton_robot` yourself, you will need to first call your `workspace_robot` function to generate `Q_ws` and `X_ws`. Note, however, that `grade_o_matic_A6` will provide its own `Q_ws` and `X_ws` when checking the student and grader instances. In fact, for this question, the `grade_o_matic` student and grader instances *coincide* — so you will know your points for this question before submission. But `grade_o_matic_A6` will test instances which exercise both the first step and the second step of your online procedure, as well as the different possible exit (`exitflag`) conditions, and all instances must be correct to earn the 20 points each for `Q_X_hat` and `exitflag`. For debugging purposes you might consider very small discrete workspaces ($M = 1$ or $M = 2$ joint angle-end effector pairs) which will permit you to better isolate the two steps — first guess, and Newton algorithm, respectively — of the online procedure.

(Reverse) Challenge. Reconsider your procedure of Question 4 but replace the Newton algorithm with an appropriate call to `fsolve`. Note you may retain the offline stage as before, and also the first step of the online procedure — the first guess based on Equation (7); you need only replace the second step of the online stage. You should also bear in mind that `fsolve` may find minima of the residual squared which do not correspond to roots (of Equation (3)), and hence you should reflect this new "failure" mode in your `exitflag` (based on the corresponding `exitflag` provided by `fsolve` itself). As always, these challenges bear no credit and hence are entirely elective.

2.086 Numerical Computation for Mechanical Engineers

Spring 2013