

Download the Matlab template for a three-dimensional Newton-Raphson solver from the course website and modify it to solve the following question.

- 1) You are estimating a receiver's position based upon known ranges from three different transmitters. Assume that all of the clocks (both satellite and receiver) are perfectly synchronized. The transmitter location and ranges (all in meters) are:

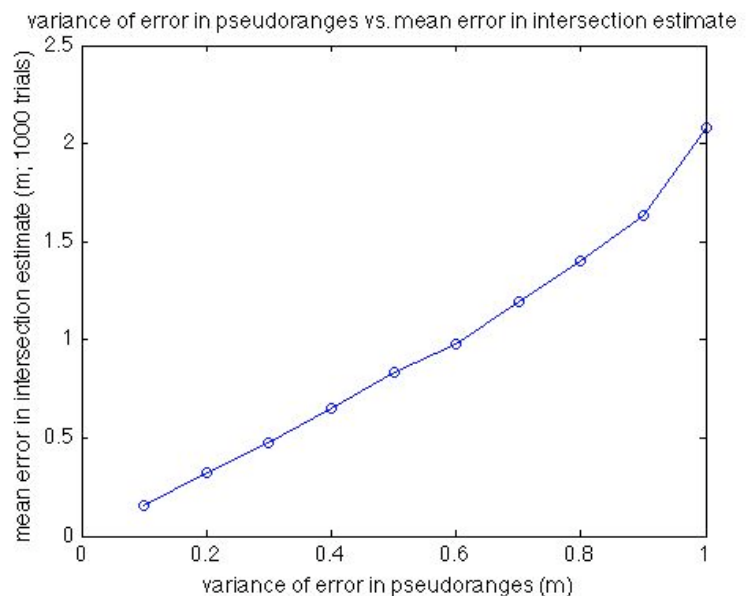
Location	Range
(5,2,3)	4
(1,8,3)	6
(1,2,-7)	10

- a) Use the Newton-Raphson method to find the two possible locations of the receiver. Hand in a copy of the code written to solve this problem.

*See attached code. The two possible solutions are [1.00, 2.00, 3.00] and [5.99, 5.32, 1.01].*

- b) Investigate the effects of uncertainty in the ranges on the estimated receiver location. Do this by adding random noise to the ranges given in the above table. Perhaps the easiest way to do this is to make use of the `randn` function in Matlab (use the built-in "doc" function in Matlab to learn more about this function by typing `doc randn` at the Matlab prompt). Create a plot that shows the error in the estimated receiver position versus the variance in the noise added to the ranges. Comment on the relationship (Is it linear? Do larger errors in the ranges lead to larger positional errors? etc). Consider variances for the noise in the range of 0 to 1 m. Hand in a copy of the code written to solve this problem. **NOTE:** Since `randn` is a statistical function, it might be a good idea to run your code multiple times for a given variance and report the mean error.

*See attached code. From the figure to the right, it appears that, at least for small errors, the relationship between the error in the range estimate is linearly related to the error in the estimated position. This relationship has a slope of  $>1$ , meaning that the error in the estimated position is larger than the error in the ranges. Note that the standard deviation of the error in the intersection estimate (not shown) increases rapidly with increasingly large variances in the range errors.*



---

```

% A script file to calculate the intersection between three spheres using
% the Newton-Raphson method.
%
% In the template file, replace the ??? with code to accomplish this.
%
% History:
%   01 Sept 2008: Template file created by Jonathan J. Makela
%   (jmakela@illinois.edu)

% The known centers (c1, c2, c3) of the circles. These can be 1x3 vectors
% (e.g., c1 = [x coordinate, y coordinate, z coordinate])
c1 = [5,2,3];
c2 = [1,8,3];
c3 = [1,2,-7];

% The known radii from each circle's center
r1 = 4;
r2 = 6;
r3 = 10;

% The initial guess at the intersection point. This must be a 3x1 vector
% (e.g., p_n = [x coordinate; y coordinate; z coordinate] (note the
% semicolon rather than comma separating coordinates))
p_n = [0;0;0];

% Specify the stopping requirement
tol = 1e-6;
dr = 999;

% Define a counting variable to keep track of iterations
i = 0;

% Loop until the stopping criterion is reached.
while(dr > tol)
    % Create the A matrix
    A = [2*(p_n(1)-c1(1)),2*(p_n(2)-c1(2)),2*(p_n(3)-c1(3));...
         2*(p_n(1)-c2(1)),2*(p_n(2)-c2(2)),2*(p_n(3)-c2(3));...
         2*(p_n(1)-c3(1)),2*(p_n(2)-c3(2)),2*(p_n(3)-c3(3))];

    % Create the F matrix
    F = [(p_n(1)-c1(1))^2+(p_n(2)-c1(2))^2+(p_n(3)-c1(3))^2-r1^2;...
         (p_n(1)-c2(1))^2+(p_n(2)-c2(2))^2+(p_n(3)-c2(3))^2-r2^2;...
         (p_n(1)-c3(1))^2+(p_n(2)-c3(2))^2+(p_n(3)-c3(3))^2-r3^2];

    % Solve for the guess at the position for the next iteration
    p_n1 = p_n - inv(A)*F;

    % Calculate the change in position from the current and next guess
    dr = sqrt((p_n1(1)-p_n(1))^2+(p_n1(2)-p_n(2))^2+(p_n1(3)-p_n(3))^2);

    % Save the new guess at the position
    p_n = p_n1;

    % Increase the counter variable
    i = i+1
end

sprintf('Estimated position: [%5.2f %5.2f %5.2f] reached in %d iterations',...
        p_n(1), p_n(2), p_n(3), i)

```

---

```

% A script file to calculate the intersection between three spheres using
% the Newton-Raphson method.
%
% In the template file, replace the ??? with code to accomplish this.
%
% History:
%   01 Sept 2008: Template file created by Jonathan J. Makela
%   (jmakela@illinois.edu)

% The known centers (c1, c2, c3) of the circles. These can be 1x3 vectors
% (e.g., c1 = [x coordinate, y coordinate, z coordinate])
c1 = [5,2,3];
c2 = [1,8,3];
c3 = [1,2,-7];

for j = 1:10
    for k = 1:1000
        % The known radii from each circle's center
        r1 = 4+j/10*randn(1,1)
        r2 = 6+j/10*randn(1,1)
        r3 = 10+j/10*randn(1,1)

        % The initial guess at the intersection point. This must be a 3x1 vector
        % (e.g., p_n = [x coordinate; y coordinate; z coordinate] (note the
        % semicolon rather than comma separating coordinates))
        p_n = [2;5;4];

        % Specify the stopping requirement
        tol = 1e-6;
        dr = 999;

        % Define a counting variable to keep track of iterations
        i = 0;

        % Loop until the stopping criterion is reached. Limit to 20 iterations.
        while((dr > tol) & (i < 20))
            % Create the A matrix
            A = [2*(p_n(1)-c1(1)),2*(p_n(2)-c1(2)),2*(p_n(3)-c1(3));...
                2*(p_n(1)-c2(1)),2*(p_n(2)-c2(2)),2*(p_n(3)-c2(3));...
                2*(p_n(1)-c3(1)),2*(p_n(2)-c3(2)),2*(p_n(3)-c3(3))];

            % Create the F matrix
            F = [(p_n(1)-c1(1))^2+(p_n(2)-c1(2))^2+(p_n(3)-c1(3))^2-r1^2;...
                (p_n(1)-c2(1))^2+(p_n(2)-c2(2))^2+(p_n(3)-c2(3))^2-r2^2;...
                (p_n(1)-c3(1))^2+(p_n(2)-c3(2))^2+(p_n(3)-c3(3))^2-r3^2];

            % Solve for the guess at the position for the next iteration
            p_n1 = p_n - inv(A)*F;

            % Calculate the change in position from the current and next guess
            dr = sqrt((p_n1(1)-p_n(1))^2+(p_n1(2)-p_n(2))^2+(p_n1(3)-p_n(3))^2);

            % Save the new guess at the position
            p_n = p_n1;

            % Increase the counter variable
            i = i+1;
        end

        de(j,k) = sqrt(sum((p_n-[1;2;3]).^2));
    end
end
end

```

- 2) Two different ways to reference time via celestial objects are to use the sun or the stars as references. Will these two time standards provide the same time? To investigate, imagine you could simultaneously observe the sun and the stars. You make note of the time when the sun passes your meridian (the line that runs north-south and passes directly overhead) and find a reference star that is also passing your meridian. You define this time as local noon. At around the same time on the following day you go to make the same observation. Will the sun and the reference star cross the meridian at the same time? If not, which will pass first and why? Is using the sun or the reference star a more accurate measure of the rotational period of the Earth? **Hint:** What else is happening to the Earth as it rotates on its axis?

*No, the two will not provide the same time and this is the basis for the difference between solar (referenced to the sun) and sidereal (referenced to the stars) time. A solar day is the time between consecutive transits of the sun across the local meridian. A sidereal day is the time between consecutive transits of a star across the local meridian. Because the Earth is orbiting the sun at the same time it is rotating, these two reference frames will be slightly different. The sidereal day is a better measure of the rotational period of the Earth, as the stars are "fixed" in our reference frame (for all intents and purposes) while the sun does have some relative motion.*

*Essentially, as the Earth orbits (in a right-hand sense) it is also rotating (in a right-hand sense). Thus, our reference star (which stays fixed in space and is not affected by the movement of the Earth in its orbit) will cross the local meridian before the sun (which is affected by the Earth's movement in its orbit). This is illustrated in the figure below. The time difference adds up to one extra revolution in the sidereal reference frame each solar year. One solar year is:*

$$365 \text{ days/solar year} \times 24 \text{ hours/day} \times 60 \text{ min/hour} = 525,600 \text{ min/solar year}$$

*This is equal to 366 sidereal days, so we divide the above quantity by 366 to get the number of minutes in a sidereal day:*

$$\frac{365 \text{ days/solar year} \times 24 \text{ hours/day} \times 60 \text{ min/hour}}{366 \text{ sidereal days/solar year}}$$

*Thus, one sidereal day has approximately 1436.07 minutes or 23 hours, 56 minutes, 4 seconds.*

