Module 09 – Assignment

Modern Navigation Systems – EN.525.645.81

Submitted: 11/6/2018

**Kyle Mercer**

1. Calculate the range, elevation angle, and azimuth to the XM satellites at **27W and 115W** from a presumed location of Latitude: 39.1427°N and Longitude: 76.8606°W, which corresponds to zip code 20723.

Look angles require both the azimuth and the elevation angles:

And the range/distance (ρ) of a satellite from a given point on Earth is given as

Where:

* R = radius of the Earth = 6,371,000m
* r = satellite distance from the Earth center = 42,240,234m
* θe = Earth station latitude = 39.1427°N
* ϕs1 = satellite 1’s longitudes = -27°
* ϕs2 = satellite 2’s longitudes = -115°
* ϕe = earth station longitude = -76.8606°
* ϕse1 = ϕs1 – ϕe = 49.8606°
* ϕse2 = ϕs2 – ϕe = -38.1394°
* δ = declination of satellites = 0
* and…

Thus:

And:

1. Check your results at <http://www.dishpointer.com>.

The resulting look angles for both satellites are very close to the suggested look angles on the dish pointer website. Satellite s2 is closer than satellite s1 to the suggested look angles for the location of zip code 20723. It is only off by about 1°.

1. Reproduce the navigation exercise shown herein, using the reference map approach shown in slide 6. This will allow you to use a ruler, protractor, and hand calculator to perform the necessary interpolations, to draw the LOPs on a printout of the map, and to estimate the improved position fix.

Using Google Earth’s ruler tool, I was able to plot LOPs with an accurate heading and distance. The Assumed Position (AP) is the zip code of the JHU APL (20723).

|  |  |
| --- | --- |
| Assumed position (AP) | 40°N, 76°W |
| Satellite 1’s longitude | 27.5°W |
| Satellite 2’s longitude | 115°W |

For the AP, according to <http://dishpointer.com>:

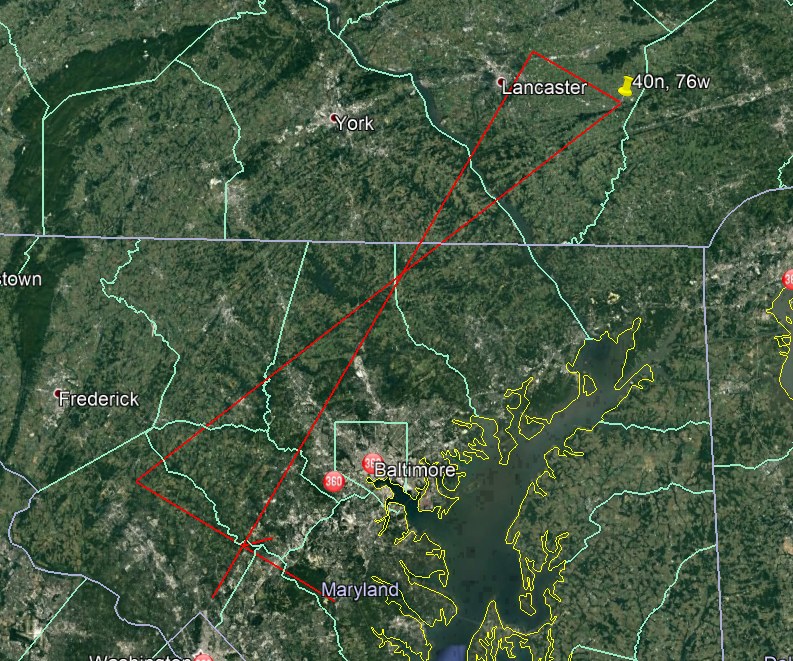
|  |  |
| --- | --- |
| Satellite 1’s Azimuth | 119.6° |
| Satellite 1’s Elevation (HC) | 22.5° |
| Satellite 2’s Azimuth | 231.5° |
| Satellite 2’s Elevation (HC) | 28.9° |

For the desired position, according to DishPointer, H0 is:

|  |  |
| --- | --- |
| Satellite 1’s Elevation (H0) | 22.3° |
| Satellite 2’s Elevation (H0) | 30.1° |

\*Note: “*nm*” units are nautical miles.

Following the procedure for LOP generation on a map yields:



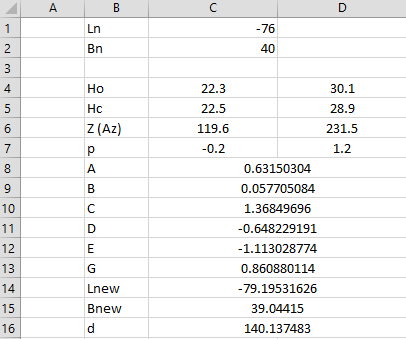
1. Compare your results with the known location from which the “measured” angles were derived.

The small red line extending from the intersection of the LOPs is the error line to the center of the 20723 zip code. This was measured to be ~3.6 nautical miles.

1. Explain why this method works, despite its use of azimuth angles that are computed relative to an assumed position that is known to be incorrect.

In the LOP algorithm your azimuth is not changing for small distances between your assumed and your calculated position. For short distances and flat earth models this is an acceptable approximation. By taking the difference between the observed H0 and the calculated HC and multiplying by 60, you effectively obtain the offset in nautical miles for which you would be off from an actual location in 2D Cartesian space. By doing this calculation for two satellites (at different azimuths) you form a basis for the 2D offset which will converge on your actual location by taking right angle lines (LOPs) and finding the intersection.

1. Use the algorithm from the Nautical Almanac (slide 28) to compute the updated position, and compare this value with that obtained from the plotting exercise.



After a single iteration of the sight reduction algorithm the distance remaining is approximately 140 nautical miles which is much greater than the Marc St. Hilaire algorithm.

1. Use Newton’s method for the geostationary satellite problem from 9A and 9B to compute a position fix iteratively using the elevation angle “measurements” to each satellite from the assumed position 40, -76.

|  |  |
| --- | --- |
| Assumed position (AP) | 40°N, 76°W |
| Satellite 1’s longitude | 27.5°W |
| Satellite 2’s longitude | 115°W |

For the AP, according to <http://dishpointer.com>:

|  |  |
| --- | --- |
| Satellite 1’s Azimuth | 119.6° |
| Satellite 1’s Elevation (HC) | 22.5° |
| Satellite 2’s Azimuth | 231.5° |
| Satellite 2’s Elevation (HC) | 28.9° |

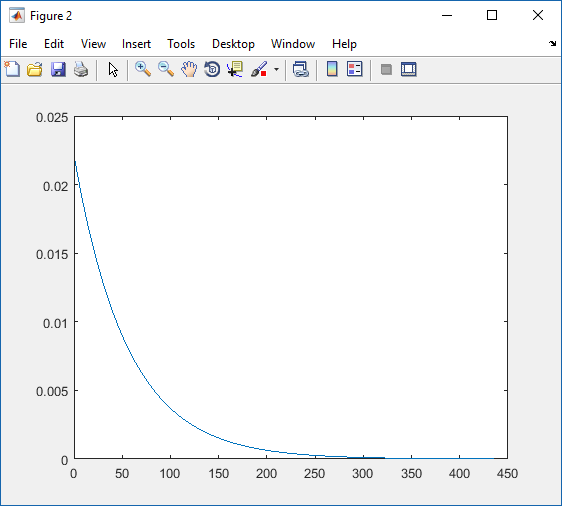
For the desired position, according to DishPointer, H0 is:

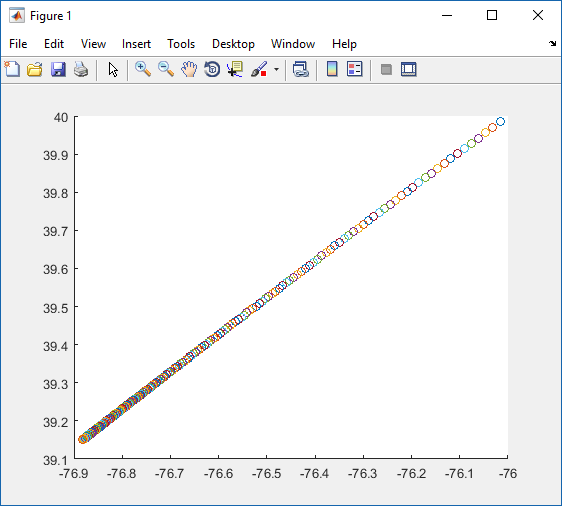
|  |  |
| --- | --- |
| Satellite 1’s Azimuth | 118.4° |
| Satellite 1’s Elevation (H0) | 22.3° |
| Satellite 2’s Azimuth | 231.2° |
| Satellite 2’s Elevation (H0) | 30.1° |

Using Newton’s method in MATLAB, the following script was calculates the inverse of the Jacobian and iteratively solves for DeltaX. The objective equations were derived from the Richharia equation for elevation as:

|  |
| --- |
| %% User defined inputs  % Initial inputs to satelite locations  el1 = 22.3; % Elevation in deg  phis1 = -27.5; % Satallite longitude in degrees    el2 = 30.1; % Elevation in deg  phis2 = -115; % Satallite longitude in degrees    % Initial guess for location on Earth  thetae0 = 40; % Person's latitude  phie0 = -76; % Person's longitude    %% Algorithm starts here  % Iterative input  X = [thetae0; phie0];  DeltaX = [Inf; Inf];  r\_ratio = 6.371e6 / 42240000;    %% Iterative portion starts here  loop = 0;  dx\_arr = ones(1000,1);  figure; hold on;  while (norm(DeltaX) > 1e-5 && ~isnan(norm(DeltaX)))    % Equations for our two circles are:  F1 = (cosd(X(1)) \* cosd(phis1-X(2)) - r\_ratio) /...  cos(asin(cosd(X(1)) \* cosd(phis1-X(2)))) - tand(el1);    F2 = (cosd(X(1)) \* cosd(phis2-X(2)) - r\_ratio) /...  cos(asin(cosd(X(1)) \* cosd(phis2-X(2)))) - tand(el2);    F = [F1; F2];    % Jacobian, J, for F(x0,y0) is:  j11 = -sind(X(1))\*cosd(phis1-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis1-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis1-X(2))^2)^(3/2);  j12 = cosd(X(1))\*sind(phis1-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis1-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis1-X(2))^2)^(3/2);  j21 = -sind(X(1))\*cosd(phis2-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis2-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis2-X(2))^2)^(3/2);  j22 = cosd(X(1))\*sind(phis2-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis2-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis2-X(2))^2)^(3/2);  J = [j11, j12; j21, j22];    % Newton's equation for a vector is defined as:  % J\*(x2 - x1) = -F  % Or Delta X = inv(J)\*-F    disp('==================');  loop = loop+1  % Calculate Delta X  DeltaX = J\-F % Equivalent to inv(J)\*-F;    % Calculate new X to feed into next iteration  X = X + DeltaX    scatter(X(2),X(1));  %drawnow;    dx\_arr(loop) = norm(DeltaX);  disp('==================');  end  figure;  plot(dx\_arr(1:loop)); |

The algorithm was able to converge onto the correct location for the observed measurements, the final calculated position on Earth was marked as 39.1506°N and 76.8822°W after 436 iterations. The following plots show the trend of the normal of DeltaX as it approaches 0 as well as a longitude vs latitude plot which show the convergence onto the desired position.





1. Compare your results with the known location from which the “measured” angles were derived.

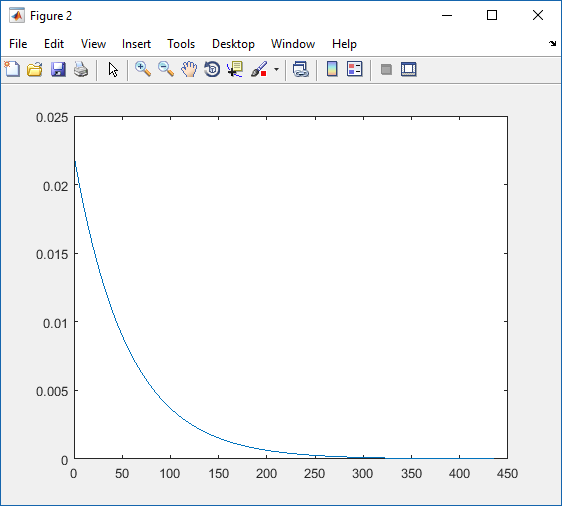
The final position calculated was 39.1506°N and 76.8822°W after 436 iterations. This is almost identical to the actual positioned.

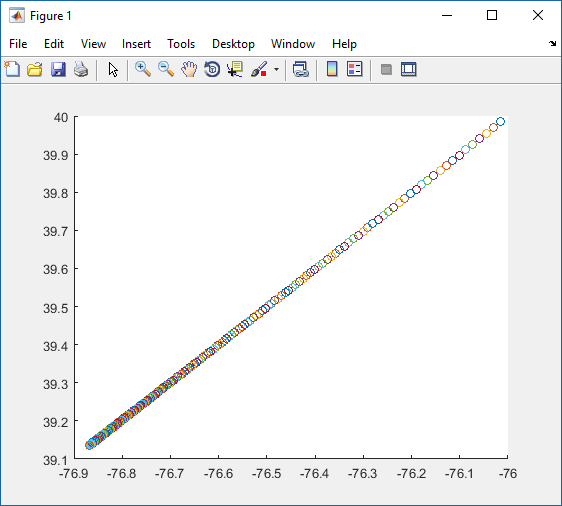
1. Use Newton’s method again for the geostationary satellite problem from 9A and 9B to compute a position fix iteratively using the elevation and azimuth angle “measured” for the XM satellite at 115 West.

Using Newton’s method in MATLAB, the following script was calculates the inverse of the Jacobian and iteratively solves for DeltaX. The objective equations were derived from the Richharia equation for elevation as:

|  |
| --- |
| %% User defined inputs  % Initial inputs to satelite locations  el = 30.1; % Elevation in deg  Ameas = 231.2; % Azimuth in degrees  phis = -115; % Satallite longitude in degrees    % Initial guess for location on Earth  thetae0 = 40; % Person's latitude  phie0 = -76; % Person's longitude    %% Algorithm starts here  % Iterative input  X = [thetae0; phie0];  DeltaX = [Inf; Inf];  r\_ratio = 6.371e6 / 42240000;    %% Iterative portion starts here  loop = 0;  dx\_arr = ones(1000, 1);  figure; hold on;  while (norm(DeltaX) > 1e-5 && ~isnan(norm(DeltaX)))    % Equations for our two circles are:  F1 = (cosd(X(1)) \* cosd(phis-X(2)) - r\_ratio) /...  cos(asin(cosd(X(1)) \* cosd(phis-X(2)))) - tand(el);    F2 = sind(phis-X(2)) / (-sind(X(1))\*cosd(phis-X(2))) - tand(Ameas);    F = [F1; F2];    % Jacobian, J, for F(x0,y0) is:  j11 = -sind(X(1))\*cosd(phis-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis-X(2))^2)^(3/2);  j12 = cosd(X(1))\*sind(phis-X(2)) \* (1-r\_ratio\*cosd(X(1))\*cosd(phis-X(2))) /...  (1-cosd(X(1))^2\*cosd(phis-X(2))^2)^(3/2);  j21 = cotd(X(1)) \* cscd(X(1)) \* tand(phis-X(2));  j22 = cscd(X(1)) \* secd(phis-X(2))^2;  J = [j11, j12; j21, j22];    % Newton's equation for a vector is defined as:  % J\*(x2 - x1) = -F  % Or Delta X = inv(J)\*-F    disp('==================');  loop = loop+1  % Calculate Delta X  DeltaX = J\-F % Equivalent to inv(J)\*-F;    % Calculate new X to feed into next iteration  X = X + DeltaX    scatter(X(2),X(1));  drawnow;    dx\_arr(loop) = norm(DeltaX);  disp('==================');  end  figure;  plot(dx\_arr(1:loop)); |

The algorithm was able to converge onto the correct location for the observed measurements, the final calculated position on Earth was marked as 39.1365°N and 76.8673°W after 436 iterations. The following plots show the trend of the normal of DeltaX as it approaches 0 as well as a longitude vs latitude plot which show the convergence onto the desired position.





1. Compare your results with the known location from which the “measured” angle and azimuth to this satellite were derived.

The final position calculated was 39.1365°N and 76.8673°W after 436 iterations. This is almost identical to the actual positioned.

1. Provide your thoughts on why Newton’s method requires so many iterations as opposed to the Marc St. Hilaire algorithm and its graphical counterpart.

In the Marc St. Hilaire method your delta between iterations is much larger and based linearly upon the difference of the measured and calculated elevation angles. Thus you converge much quicker on the actual position. In Newton’s method the deltas are weighted to a degree because of the Jacobian that is calculated. Thus is take many more iterations to calculate a final position.