**Homework 2 – Orbit Estimators**

1. Create a directory called “orbit\_model” to save all your inputs and results for the orbit model.

2. Write a function to generate and save inputs for your orbit model.

Function Name: generate\_orbit\_inputs()

Steps:

1. (if needed) Clear all variables
2. Create a variable called params with the following values for measurement and state covariance
   1. Po = diag([100, 100, 100, 1e-4, 1e-4, 1e-4])
   2. Xo\_true = [757.700, 5222.607, 4851.500, 2.21321, 4.67834, -5.37130].T (column vector)
   3. Rk = diag([1e-6, (1/206265)^2, (1/206265)^2])
   4. theta0 = 0
   5. dtheta = 7.2921158553e-5
   6. GM = 3.986004e5
   7. Re = 6378.1363
   8. stat\_ecef = [-5466.071, -2403.990, 2242.473]
   9. Q = diag([1e-8, 1e-8, 1e-8])
   10. (Note) theta0 and dtheta can be used for a simple z-axis only rotation between ECEF and ECI. If using a higher fidelity transform, replace with appropriate Earth Orientation Parameters (EOPs) as needed.
3. Save output in a file

Note: all distances in km, angles in radians, mass in kg, time in seconds

Run the script and save a file called “orbit\_model\_inputs\_rgradec.xxx” in your orbit model directory.

Change the measurement noise covariance to the following:

Rk = diag([(1/206265)^2, (1/206265)^2])

Run the script again and save a file called “orbit\_model\_inputs\_radec.xxx” in your orbit model directory

3. Write a function to work with an ODE solver to integrate the orbit motion model.

Function Name: int\_twobody()

Inputs:

t – time

X – input state vector

params – input parameters

Outputs:

dX – output derivative vector

Steps:

1. Retrieve values from input state vector
2. Retrieve values from params (e.g. GM)
3. Compute accelerations in X,Y,Z using 2-body equations
4. Form output derivative vector

4. Write a function to generate and save truth data for your orbit model.

Function Name: generate\_orbit\_truth()

Steps:

1. (if needed) Clear all variables
2. Load the input data file
3. Retrieve the true input state vector Xo
4. Setup ode options, set RelTol and AbsTol to 1e-12 (if using variable step)
5. Setup a time vector from 0 to 4\*3600 in 60 second increments
6. Initialize the output truth matrix Xt\_mat as zeros (nxL matrix)
7. Integrate to get truth values for Xt\_mat
   1. Run ode solver with the linear derivative function and (tvec, Xo, options, params)
   2. Retrieve the values of position and velocity at each time in tvec
   3. Store these values as a column of Xt\_mat
8. Save a file called “orbit\_model\_truth.xxx” with the variables tvec and Xt\_mat in your linear model directory

5. Write a function to compute measurements for your orbit model.

Function Name: generate\_orbit\_meas()

Steps:

1. (if needed) Clear all variables
2. Load input values
3. Load truth values
4. Set the random seed to 1
5. Compute the value of range, ra, and dec standard deviation from Rk using (sqrt(diag(Rk))
6. At each time,
   1. Retrieve the true position in ECI
   2. Rotate ground station coordinates from ECEF to ECI
   3. Compute the measurements (use formulas for topocentric RA/DEC)
   4. Add Gaussian noise to true measurement
   5. (Note) this doesn’t account for visibility constraints (e.g. above the horizon) and you can choose what level of fidelity to use for ECEF -> ECI transformation
7. Store in a variable called obs\_data.
8. Save a file called “orbit\_model\_meas\_rgradec.xxx” with the variables tvec and obs\_data.

Repeat the above steps, but this time, only save the angles measurements (ra, dec) in obs\_data. Name this second file, “orbit\_model\_meas\_radec.xxx”

1. Compute the partial derivatives required for the orbital model. Use the Cartesian position and velocity in ECI for the state vector. For the dynamics, use the 2-body equations of motion. For the measurements, use range and topocentric right ascension and declination. Compute all the terms needed to fill in the *A* matrix (6x6) and matrix (3x6). In total, this will be 36 + 18 = 54 terms, but many will be zero. Note that for the second measurement scenario you will use matrix (2x6) just for the angles.
2. Write a function to work with ode solver to integrate the two-body orbit motion model with an STM. The partials used to create the *A* matrix come from Q6.

Function Name: int\_twobody\_stm()

Inputs:

t – time

X – input state vector

params – input parameters

Outputs:

dX – output derivative vector

Steps:

1. Retrieve values from input state vector
2. Retrieve GM from params
3. Compute orbit radius
4. Compute derivatives for translation states (pos/vel) using full nonlinear equations (just like Q3)
5. Compute partials and fill in A matrix
6. Compute derivatives for STM (use A matrix and reshape)
7. Form output derivative vector
8. Write a function to compute the expected measurement *G* and the observation mapping matrix for the orbit model. This function will use all 3 measurements, range, right ascension, and declination. The steps to compute *G* are similar to Q5. The partials for come from Q6.

Function Name: gen\_H\_rgradec()

Inputs:

t – time

X – input state vector

params – input parameters

Outputs:

Hk\_til – observation mapping matrix

Gk – computed measurement (aka expected measurement)

Steps:

1. Retrieve position value from X
2. Retrieve values from inputs structure (stat\_ecef, etc)
3. Rotate ground station coordinates from ECEF to ECI
4. Compute the measurements (range, right ascension, declination)
5. Compute the partials for Hk\_til
6. Form output Gk and Hk\_til matrices
7. Write a function to compute the expected measurement *G* and the observation mapping matrix for the orbit model. This function will use only the 2 angle measurements, right ascension, and declination. The steps to compute *G* are similar to Q5. The partials for come from Q6.

Function Name: gen\_H\_radec()

Inputs:

t – time

X – input state vector

params – input parameters

Outputs:

Hk\_til – observation mapping matrix

Gk – computed measurement (aka expected measurement)

Steps:

1. Retrieve position value from X
2. Retrieve values from inputs structure (stat\_ecef, etc)
3. Rotate ground station coordinates from ECEF to ECI
4. Compute the measurements (range, right ascension, declination)
5. Compute the partials for Hk\_til
6. Form output Gk and Hk\_til matrices
7. Use your run\_filter script to run the batch processor using the orbit model and full 3D measurements. The batch function itself should not require any updates from its use in HW1. This is the point of breaking out all of these smaller functions to create modular code. In the run\_filter script, make sure you are using the correct input and measurement files, and the correct function handles for your orbit model.

Per my naming conventions, these are:

1. orbit\_model\_inputs\_rgradec.xxx
2. orbit\_model\_meas\_rgradec.xxx
3. int\_twobody\_stm
4. gen\_H\_rgradec

You will need to set the initial state Xo for the filter. The first time you run it, use Xo\_true as a debugging step to see if everything is working right. For your final filter run, generate a small offset from the true value to make sure your filter can converge on the right solution. I do this using the initial covariance matrix and the following lines of code.

# Set the filter initial state

Po = params[‘Po’]

pert\_vect = sqrt(diag(Po)) \*(element-wise) randn(6,1)

Xo = Xo\_true + pert\_vect

The first part of computing pert\_vect gets the standard deviations from the covariance matrix by taking the square root of the diagonal of the matrix. This is a 6x1 vector. Then use element-wise multiply with the randn function to get a 6x1 vector of Gaussian noise where each term uses the standard deviation from Po. Add this to the true state to get an initial Xo that is consistent with your initial uncertainty.

Save your estimated state Xk\_mat, P\_mat, and resids. Call the output file “orbit\_model\_batch\_output\_rgradec.xxx” (For the batch, Xk\_mat is the same as the reference trajectory Xref\_mat).

11. (Optional) Write a function to rotate a 3x1 vector or a 3x3 matrix from ECI to RIC.

Function Name: eci2ric()

Inputs:

rc\_vect – 3x1 position vector in EC1

vc\_vect - 3x1 velocity vector in ECI

Qin – 3x1 vector or 3x3 matrix in ECI

Outputs:

Qout – 3x1 vector or 3x3 matrix in RIC

Steps:

1. Compute the radial direction unit vector in ECI
2. Compute the cross-track (angular momentum) unit vector in ECI
3. Compute the in-track unit vector in ECI
4. Form the rotation matrix to RIC
5. Check the size of Qin
   1. If 3x1, multiply input to get output
   2. If 3x3, pre- and post-multiply (transpose) to get output
6. Run your function to compute\_errors for the batch output. You should be able to load the output and truth files, and compute errors just like before,

Xerr\_mat = Xk\_mat – Xt\_mat;

Additional Steps:

1. Compute the standard deviations for all the positions and velocities in ECI
2. Compute the position errors and standard deviations in the Radial-Intrack-Crosstrack (RIC) frame. At each time, take the true position and velocity vectors and use them for the “chief” orbit rc\_vect and vc\_vect. Then use the position error vector as Qin. My code looks like this:

Qin = Xerr\_mat(1:3,i)

RIC\_err = eci2ric(rc\_vect, vc\_vect, Qin)

Qin = P\_mat(1:3,1:3,i)

RIC\_cov = eci2ric(rc\_vect, vc\_vect, Qin)

Save the RIC position errors and standard deviations in vectors that you can plot later.

1. Generate Plots
   1. Make a 3x1 plot of position errors in ECI (X,Y,Z) with 3-sigma bounds
   2. Make a 3x1 plot of velocity errors in ECI with 3-sigma bounds
   3. Make a 3x1 plot of position errors in RIC with 3-sigma bounds
   4. Make a 3x1 plot of residuals (range, right ascension, declination)

Use hours for the time axes. Use meters for position errors and range residuals and arcseconds for angle residuals. You can convert from radians to arcseconds by multiplying by 206265.

1. Repeat questions 10&12 for the batch processor using the 2D angles-only measurements. You will need to use the correct inputs and measurements files, as well as the function handle gen\_H\_radec. You can add a column to your previous table for the standard deviations for the new entries, note that there will be no range residuals. Your code to compute errors and generate plots should also use a 2x1 plot for the angle residuals.