Posting Date: Tuesday Nov. 12th.
Due Date: Tuesday Nov. 19th.

1. Complete the MATLAB template file `script_simaircraft05_temp.m` by completing the parts of the code where ???? appears. The result will be the MATLAB script `script_simaircraft05.m`. This script re-does the simulation in the Assignment 5 script `script_simaircraft04.m` that uses `ffunctaircraft03.m` in order to simulate aircraft 3-dimensional motion with the Coriolis and centrifugal forces that are caused by the Earth's rotation. It performs the simulation in two ways. One uses `ode45.m`. The other method uses Euler numerical integration. It performs Euler integration using 3 different numbers of integration steps, N = 1,000, 10,000, and 100,000. It compares the results to those produced by `ode45.m` when using a very tight error tolerance in '`RelTol`'.

   Run your code and hand in your code and the first two plots that it makes, the one that plots the ground track and the one that plots the altitude, airspeed, flight-path-angle, and heading-angle time histories. Also hand in the values of the final state errors of the Euler integration method, `errorxfinal_10000` and `errorxfinal_100000`, given to 15 significant digits, i.e., using MATLAB'S format long command. As a way of checking your code, the correct value for the final state error of the coarsest Euler integration is:

   ```
   errorxfinal_1000 =
     1.0e+03 *
    -2.249895893562921
     0.251280598994137
     0.152175509991668
     0.003678205103694
     0.000056219544104
    -0.000066881571850
   ```

   Also hand in your computation times for the numerical integrations. They are contained in `timetoode45` and in `timetoeuler_vec`. How does Euler integration compare to `ode45.m` in terms of execution speed?

   Compute the vector of error ratios `errorxfinal_10000./errorxfinal_100000` and hand it in. The theory of Euler's method predicts that these ratios should be about 10. Is that true?

2. Complete the MATLAB template file `script_simaircraft06_temp.m` by completing the parts of the code where ???? appears. The result will be the MATLAB script `script_simaircraft06.m`. This script re-does the simulation in the Assignment 5 script `script_simaircraft04.m` that uses `ffunctaircraft03.m` in order to simulate aircraft 3-dimensional motion with the Coriolis and centrifugal forces that are caused by the Earth's rotation. It performs the simulation in two ways. One uses `ode45.m`. The other method uses trapezoidal numerical integration. It performs trapezoidal integration using 3 different numbers of integration steps, N = 500, 2,000, and 8,000. It compares the results to those produced by `ode45.m` when using a very tight error tolerance in '`RelTol`'.

Run your code and hand in your code and the first two plots that it makes, the one that plots the ground track and the one that plots the altitude, airspeed, flight-path-angle, and heading-angle time histories. Also hand in the values of the final state errors of the trapezoidal integration method, `errorxfinal_2000` and `errorxfinal_8000`, given to 15 significant digits, i.e., using MATLAB's format long command. As a way of checking your code, the correct value for the final state error of the coarsest trapezoidal integration is:

```
errorxfinal_500 =
   1.0e+02 *
   2.470874422823008
   0.312781372675090
  -0.020091213827387
  -0.001686689529057
   0.000002200901484
   0.000076213593823
```

How do `errorxfinal_2000` and `errorxfinal_8000` for this run compare `errorxfinal_10000` and `errorxfinal_100000` for the Euler integration run?

Also hand in your computation times for the numerical integrations. They are contained in `timetoode45` and in `timetotrapez_vec`. How does trapezoidal integration compare to `ode45.m` in terms of execution speed?

Compute the vector of error ratios `errorxfinal_2000./errorxfinal_8000` and hand it in. The theory of Euler's method predicts that these ratios should be about 16. Is that true?

3. Complete the MATLAB template file `script_simaircraft07_temp.m` by completing the parts of the code where ???? appears. The result will be the MATLAB script `script_simaircraft07.m`. This script re-does the simulation in the Assignment 5 script `script_simaircraft04.m` that uses `ffunctaircraft03.m` in order to simulate aircraft 3-dimensional motion with the Coriolis and centrifugal forces that are caused by the Earth's rotation. It performs the simulation in two ways. One uses `ode45.m`. The other approach uses the following 4th-order Runge-Kutta numerical integration method:

$$t_{ak} = t_k, \qquad \boldsymbol{x}_{ak} = \boldsymbol{x}_k, \qquad \boldsymbol{f}_{ak} = \boldsymbol{f}(t_{ak}, \boldsymbol{x}_{ak})$$

$$t_{bk} = t_k + \tfrac{1}{2}\Delta t, \qquad \boldsymbol{x}_{bk} = \boldsymbol{x}_k + \tfrac{1}{2}\Delta t\, \boldsymbol{f}_{ak}, \qquad \boldsymbol{f}_{bk} = \boldsymbol{f}(t_{bk}, \boldsymbol{x}_{bk})$$

$$t_{ck} = t_k + \tfrac{1}{2}\Delta t, \qquad \boldsymbol{x}_{ck} = \boldsymbol{x}_k + \tfrac{1}{2}\Delta t\, \boldsymbol{f}_{bk}, \qquad \boldsymbol{f}_{ck} = \boldsymbol{f}(t_{ck}, \boldsymbol{x}_{ck})$$

$$t_{dk} = t_k + \Delta t, \qquad \boldsymbol{x}_{dk} = \boldsymbol{x}_k + \Delta t\, \boldsymbol{f}_{ck}, \qquad \boldsymbol{f}_{dk} = \boldsymbol{f}(t_{dk}, \boldsymbol{x}_{dk})$$

$$t_{k+1} = t_k + \Delta t$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{\Delta t}{6}(\boldsymbol{f}_{ak} + 2\boldsymbol{f}_{bk} + 2\boldsymbol{f}_{ck} + \boldsymbol{f}_{dk})$$

The script performs 4th-order Runge-Kutta numerical integration using 3 different numbers of integration steps, N = 100, 400, and 1,600. It compares the results to those produced by `ode45.m` when using a very tight error tolerance in '`RelTol`'.

Run your code and hand in your code and the first two plots that it makes, the one that plots the ground track and the one that plots the altitude, airspeed, flight-path-angle, and heading-angle time histories. Also hand in the values of the final state errors of the 4th-order Runge-Kutta numerical integration method, `errorxfinal_400` and `errorxfinal_1600`, given to 15 significant digits, i.e., using MATLAB'S format long command. As a way of checking your code, the correct value for the final state error of the coarsest 4th-order Runge-Kutta integration is:

```
errorxfinal_100 =
   1.0e+02 *
  -7.127606248392213
  -0.264484908926461
  -0.035371532136971
  -0.002812622432203
   0.000014204895534
  -0.000218345320909
```

How do `errorxfinal_400` and `errorxfinal_1600` for this run compare `errorxfinal_2000` and `errorxfinal_8000` for the trapezoidal integration run?

Also hand in your computation times for the numerical integrations. They are contained in `timetoode45` and in `timeto4thOrdRK_vec`. How does the 4th-order Runge-Kutta integration method compare to `ode45.m` in terms of execution speed?

Compute the vector of error ratios `errorxfinal_400./errorxfinal_1600` and hand it in. The theory of Euler's method predicts that these ratios should be about 256. Is that true?

4. Someone has proposed the following Runge-Kutta numerical integration scheme as an alternative to trapezoidal integration:

$$t_{ak} = t_k\,, \qquad \boldsymbol{x}_{ak} = \boldsymbol{x}_k\,, \qquad \boldsymbol{f}_{ak} = \boldsymbol{f}(t_{ak}, \boldsymbol{x}_{ak})$$

$$t_{bk} = t_k + \tfrac{1}{2}\Delta t\,, \qquad \boldsymbol{x}_{bk} = \boldsymbol{x}_k + \tfrac{1}{2}\Delta t\,\boldsymbol{f}_{ak}\,, \qquad \boldsymbol{f}_{bk} = \boldsymbol{f}(t_{bk}, \boldsymbol{x}_{bk})$$

$$t_{k+1} = t_k + \Delta t$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta t(b_1 \boldsymbol{f}_{ak} + b_2 \boldsymbol{f}_{bk})$$

Determine the values of b_1 and b_2 that will make this method 2nd-order and prove that the resulting method is 2nd-order.

<u>Hints</u>: Develop an analysis of the order of this method that is similar to the analysis that was developed in lecture for the trapezoidal numerical integration method. This analysis will

involve developing a Taylor series approximation of $\boldsymbol{f}_{bk}$. Find the b_1 and b_2 values which cause the resulting series to have Δt and Δt^2 terms that match the true Taylor series for $\boldsymbol{x}(t)$ expanded about t_k.