ASEN 3801 Aerospace Vehicle Dynamics and Control Lab
Spring 2026

Lab 1: Simulating Dynamical Systems

Assigned Tuesday, January 13, 2026
Due Friday, January 23, 2026, at 11:59pm

OBJECTIVES

1. Introduce good programming habits.
2. Practice using MATLAB to numerically generate solutions to dynamical systems.

BACKGROUND

Foundational Program/Function Requirements

Good programming skills enable a student to create accurate codes or software that can potentially be reused across multiple simulations or scenarios. Furthermore, it is important to write code that other people, including your future self, can read and re-use. These practices also support effective collaboration and good scientific/engineering practices. There are many resources to learn how to program well, some of which you may have covered in your previous classes. One classic text is *Clean Code: A handbook of agile software craftsmanship* by Robert C. Martin.

In this course, we expect students to regularly use a few foundational practices when writing code. In particular, the following practices should be followed:

1. **File Headers:** All M-files developed for this course should have the same header information. The header should have the students' names, the course name and number, the file name, and the date the file was created. For example:

   ```
   % Contributors: Name of Student
   % Course number: ASEN 3801
   % File name: AirRelativeVelocityVectorToWindAngles
   % Created: 9/8/23
   ```

   Note that in a modern industrial software development context, file headers might not be recommended. Rather, a version control system would keep track of when every line of code is changed and by whom; file headers would thus contain redundant information.

2. **Function Descriptions and Comments:** Every software function will have a "docstring" that describes the inputs to the function, the outputs, and a basic description of the technical approach. For example:

   ```
   function wind_angles = AirRelativeVelocityVectorToWindAngles(velocity_body)
   %
   % Inputs: velocity_body = column vector of aircraft air-relative velocity in
   %                         body coordinates
   %                       = [u,v,w]'
   ```

```
%
% Outputs:  wind_angles = [speed beta alpha]'
%           speed = aircraft airspeed
%           beta = side slip angle
%           alpha = angle of attack
%
% Methodology: Use definitions to calculate wind angles and speed from velocity
% vector
```

MATLAB has the additional capability to display comments in the m-file immediately after the function declaration when the help function is used.

3. **Descriptive Variable and Function Names:** Variable and function names shall clearly and concisely describe what they are or the action they perform. Naming things well is a surprisingly difficult challenge – an entire chapter of *Clean Code* is devoted to it, and a common joke is "There are only two hard things in Computer Science: cache invalidation and naming things."

   For variable names, one approach is to use lowercase letters with the underscore to separate words. For example, the variable `velocity_body` would describe the aircraft velocity in body coordinates. One rule of thumb from *Clean Code* is that variable name length should be proportional to the size of the section of code the variable is used in. A variable named `vb` or `vel_b` might be appropriate in a 5 line for loop, but not in a 100-line script.

   For function names, one approach is to use capital letters for each word instead of the underscore. For example, use a name such as: `AirRelativeVelocityVectorToWindAngles`.

4. **Functional decomposition:** Modularity and code re-use are vital to any complex program. In our MATLAB programming for this course, we use functions to break up code into smaller, reusable segments. Any procedure that needs to be performed multiple times should be encapsulated into a function. Functions can be nested into other functions to build complex software. *Clean Code* has an entire chapter devoted to designing functions. Two key principles are that functions should be small and that they should perform one task. Modularizing code can become cumbersome in some programming environments, but some newer open-source languages are designed around modularity.

Simulating Dynamical Systems in MATLAB

Within your various aerospace level classes, you are learning that the equations of motion that govern aerospace vehicles are nonlinear and usually cannot be solved analytically. In that case, numerical methods can be used to solve a set of first-order ordinary differential equations (ODEs) given a specific initial condition and time interval of interest. This solution may represent the trajectory of a vehicle or the evolution of a variable of interest.

MATLAB supplies a variety of implementations of well-known numerical integration schemes that can be used to solve ordinary differential equations. The full array of methods and functions available in MATLAB are described in detail in the following help documentation: https://www.mathworks.com/help/matlab/ordinary-differential-equations.html. In this group of functions, it is important to note that each function for solving a set of ordinary differential equations uses a distinct numerical integration scheme. Each scheme, which is derived based on mathematical theory, uses a distinct approach to generate a numerical approximation of a true solution to the dynamical system.

Accordingly, every time you use a premade function in MATLAB or another programming environment, it is important to delve into the description and/or mathematics of the function. Always read the documentation carefully first. Then, consider whether the mathematical or technical approach is applicable and suitable for the problem you are trying to solve.

As an example, one well-known function that MATLAB uses is `ode45`. The MATLAB help documentation for `ode45` explains how the function can be called, the inputs and outputs, and any additional options/capabilities. The end of the page features the following description of the mathematical method that this function implements:

"ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a single-step solver – in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{n-1})$" (Quoted from https://www.mathworks.com/help/matlab/ref/ode45.html)

Sometimes additional mathematical expressions or discussion might appear in this location; in this case, the reader is referred to additional resources in the references list. Always take the time to read these mathematical descriptions. It is important to understand the technical approach used by a function before including it in your code. In addition, this attention to detail will save you from endless days of debugging or generating incorrect results when a function does not work as hoped as you encounter increasingly fun and challenging problems in aerospace engineering.

As described in the help documentation, one approach to calling `ode45` is:

```
[TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS,ADDVAR);
```

which takes as input the name of a function (`ODEFUN`) that returns the derivative of the state vector as calculated by evaluating a set of first-order differential equations, the time span of the simulation (`TSPAN`), and the initial state vector (`Y0`). This function returns a time vector (`TOUT`) and an array of states (`YOUT`) at each of these times within the specified time span. `OPTIONS` contains an important set of options that govern the numerical integration scheme including, for example, setting absolute and relative tolerances for a variable step-size method such as a Runge-Kutta (4,5) method. The `ADDVAR` variable is a sequence of additional parameters that may govern the dynamical system. Note, the actual function passed into `ODE45` can have any name, for example:

```
[tout,yout] = ode45(@MyDerivFunction, [0 20], 5,options).
```

The `ODEFUN` function is typically set up in the form $\dot{y} =$ `ODEFUN(`t`,`y`,ADDVAR)`, returning the derivative $\dot{y}$ of the state vector y at time t. For example, `ODEFUN` could be used to return the one-dimensional first-order derivative $\dot{y} = 5y + 20\sin 4t$.

You are expected to have studied the solution to ordinary differential equations and used `ode45` (or similar functions like `ode23`) in your prior classes and, accordingly, we will not review it further here. Please refer to the MATLAB help documentation and your prior coursework for more information.

Plotting Data

When creating a plot of data that you have generated from a numerical simulation, it is important to ensure that the plot is accurate, readable, and useful. A clear plot should include, at a minimum, the following information where applicable:

- Tick marks and values listed on each axis.
- Axis labels that indicate the variable plotted on each axis and the units, e.g., for a distance quantity in meters use "Distance (m)" or for a nondimensional quantity y use "y (n.d.)".
- Font should be sufficiently large and clear without overlapping other text or parts of the figure.
- The figure should be clear and readable when shown on letter size paper without needing to magnify/zoom in on the page.
- Curves reflecting continuous quantities as a function of another variable should be thick enough to be visible.
- Data points that do not correspond to a continuous quantity should not be connected by a curve.
- Colors used to differentiate curves or plot objects should be readable, e.g., do not use yellow on a white background. In addition, note that using some color combinations to differentiate data may produce a plot that is inaccessible to some color-blind readers.

Air-Relative Velocity Vector and Drag Force

Problem 2 of this assignment requires calculating the *air-relative velocity vector*, which is the velocity of an object relative to the background wind. The air relative velocity vector is often described as the "wind" experienced by an object and is the velocity vector that creates drag.

This assignment adopts the notation used in ASEN 3728 Aircraft Dynamics. The following three vectors are defined as follows:

- $\mathbf{v}_E^E$ is the inertial velocity vector of the object expressed in inertial coordinates.
- $\mathbf{w}_E^E$ is the inertial velocity vector of the background wind expressed in inertial coordinates.
- $\mathbf{v}_E$ is the air-relative velocity vector of the object expressed in inertial coordinates.

In these expressions, "E" refers to the inertial axes: the superscript identifies the observer axes and the subscript identifies the axes in which the vector is expressed. The bold notation indicates that these velocities are vectors, not scalar quantities.

These vectors are related by the *wind triangle* equation as:

$$\mathbf{v}_E = \mathbf{v}_E^E - \mathbf{w}_E^E$$

The drag force (in inertial coordinates) $\mathbf{f}_E^{drag}$ that acts on the object is computed from the drag coefficient and the airspeed V_a. The airspeed is the magnitude of the air relative velocity vector, calculated as:

$$V_a = \|\mathbf{v}_E\|$$

Let D be the magnitude of the drag force. The drag force acts opposite to the air-relative velocity vector of the object and is calculated as:

$$\mathbf{f}_E^{drag} = -D\frac{\mathbf{v}_E}{V_a}$$

Where $D = \frac{1}{2}\rho {V_a}^2 A C_D$. In this expression, ρ is the atmospheric density, A is the cross-sectional area of the object, and C_D is the drag coefficient.

PROBLEMS

1. Use `ode45` in MATLAB to use a Runge-Kutta (4,5) numerical integration scheme to generate a solution to a nonlinear dynamical system:

   a. Consider the following dynamical system governing the evolution of the vector (w, x, y, z) over time t:

$$\dot{w} = -9w + y$$
$$\dot{x} = 4wxy - x^2$$
$$\dot{y} = 2w - x - 2z$$
$$\dot{z} = xy - y^2 - 3z^3$$

   Each of the variables w, x, y, z, t in this problem are nondimensional (you can identify this in your plots using the notation "n.d." in place of any units); nondimensionalization or normalization is a common approach when studying dynamical systems.

   Select a nonzero initial condition for the (w, x, y, z) vector; report the selected vector in your writeup. Then, numerically generate a solution to the dynamical system from the selected initial condition for a duration of 20 nondimensional time units. The `ODEFUN` function you create in this problem can return a vector so this problem should be solved as one single run of `ode45`. When implementing `ode45`, use relative and absolute tolerances equal to $1\text{x}10^{-8}$.

   Once you generate a solution to the dynamical system, display the evolution of each variable as a function of time by creating one figure with four subplots, stacked vertically. Each subplot should contain the plot of one variable component (w, x, y, or z) on the vertical axis and time on the horizontal axis.

   Note: As in this problem, students should expect to be given problems for which they must determine some of the inputs or parameters. In this case, you must select an initial condition vector.

   b. Examine the influence of the relative and absolute tolerances on the (w, x, y, z) vector after 20 nondimensional time units. Set both the relative and absolute tolerances to equal the same variable labeled '`tol`'. Regenerate the solution to the dynamical system using the same initial condition and duration as in Problem 1a but with each of the following values of the variable '`tol`': $1\text{x}10^{-2}$, $1\text{x}10^{-4}$, $1\text{x}10^{-6}$, $1\text{x}10^{-8}$, $1\text{x}10^{-10}$, $1\text{x}10^{-12}$. Consider the solution generated when `tol` = $1\text{x}10^{-12}$ as a reference, labeled (w_R,x_R,y_R,z_R). Then, create a table with the following structure:

	$1\text{x}10^{-2}$	$1\text{x}10^{-4}$	$1\text{x}10^{-6}$	$1\text{x}10^{-8}$	$1\text{x}10^{-10}$
$\lvert w(t=20) - w_R(t=20)\rvert$					
$\lvert x(t=20) - x_R(t=20)\rvert$					
$\lvert y(t=20) - y_R(t=20)\rvert$					
$\lvert z(t=20) - z_R(t=20)\rvert$					

   Discuss the influence of the absolute and relative tolerances on the deviation between the (w, x, y, z) vector and the reference values at 20 nondimensional time units. Also identify any additional

considerations that may be important when selecting suitable tolerance values when numerically generating a solution to a dynamical system.

2. Let's study the translational dynamics of a spherical object moving through the air in an approximate dynamical model. Because we are examining a spherical object, we can reasonably assume that the forces acting on the body are not a function of the body attitude. Rather, the forces acting on the body include aerodynamic drag (which acts opposite to the air-relative velocity vector) and gravity (which acts downward). For this problem, the right-handed and orthogonal axes of the inertial frame follow the convention North-East-Down. In this problem, let's model the object with a mass of 50 g, a diameter of 2.0 cm, and coefficient of drag of 0.6. Assume the object is here in Boulder to determine the air density ρ.

   a. In this problem, you will create a function in MATLAB of the following form:

      ```
      function xdot = objectEOM(t,x,rho,Cd,A,m,g,wind_vel)
      ```

      to calculate the derivative `xdot` of the state vector `x` with respect to time as a function of the time, the state, the physical parameters of the problem, and the three-dimensional wind velocity in inertial coordinates (i.e., `wind_vel` = $\mathbf{w}_E^E$).

      The 6-dimensional state vector describing the object should include its inertial position in inertial coordinates and its inertial velocity in inertial coordinates such that:

      $$\mathbf{x} = \begin{bmatrix} \mathbf{p}_E^E \\ \mathbf{v}_E^E \end{bmatrix}$$

      Then, the first-order derivative with respect to time is equal to:

      $$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_E^E \\ \dot{\mathbf{v}}_E^E \end{bmatrix} = \begin{bmatrix} \mathbf{v}_E^E \\ \mathbf{a}_E^E \end{bmatrix}$$

      First, derive the inertial acceleration using Newton's 2nd law so that you can write the equations of motion for the object. Use these second-order equations to write the 6 first-order differential equations describing this dynamical system, i.e., in the form $\dot{\mathbf{x}} = f(\mathbf{x})$. Show your mathematical derivation and use proper mathematical notation. Be sure to draw a diagram of the object, the axes, and the forces acting on the object.

      Using the result of your derivation, create the MATLAB function `objectEOM` to capture the first-order differential equations governing the evolution of the state vector. Be sure to follow the good programming practices covered at the beginning of this lab document. Include your commented code in your response to this problem.

   b. Download the `stdatmo.m` file available on Canvas. This file calculates various properties, including the density, of the Earth's atmosphere at various altitudes and temperatures using the 1976 Standard Atmosphere model (Note: accurately modeling the properties of an atmosphere can become a very complex and time-dependent problem). Carefully read through the function description comments to understand the possible inputs and outputs.

Use this function to report the density of the atmosphere in Boulder at a geopotential altitude of 1655 m on a standard day; don't forget the units. Also list the line of code you used to generate this quantity.

c. Define the initial value of the state vector of the object as follows: the object is initially located at the origin of the inertial frame, with an initial velocity component of 20 m/s upward and 20 m/s East (inertial y direction). In this problem only, assume the wind velocity is zero.

   Use `ode45` to numerically generate a solution to the dynamical system you derived in Problem 2a from the specified initial state vector until the object lands on the ground, i.e., when it possesses a height of zero. To generate the trajectory until this stopping condition, construct an event function to terminate `ode45` when the z-component of the position vector equals zero. When implementing `ode45`, use relative and absolute tolerances equal to 1×10^{-8}.

   Plot the trajectory of the object in three-dimensional space using a three-dimensional figure in MATLAB. To create an easily interpretable figure, flip the third axis so that negative z values appear above the xy-plane, consistent with the definition of the reference axes used in this problem.

   Discuss whether the results make sense, using the intuition you have gained from your prior physics and dynamics classes.

d. Using the same initial condition as in Problem 2c, let's explore the impact of the magnitude of the wind velocity on the trajectory of the object. In this problem, you will vary the wind speed and regenerate the solution to the dynamical system at each instance. Hint: consider how you can implement this procedure efficiently and cleanly in your code. After this process has been repeated for a variety of wind speeds, you will need to devise a way to visually represent the evolution of each of the requested characteristics of the trajectory on a single plot.

   Answer the following two questions:

   1) How does the horizontal displacement of the landing location (i.e., the inertial x coordinate of the landing position) depend on the wind speed in the North (inertial x) direction, in meters of deflection per m/s of wind speed?

   2) How does the total distance from the origin to the landing location depend on the wind speed in the North (inertial x) direction, in meters of deflection per m/s of wind?

   Devise and generate a single plot for each of the two questions to summarize this information. Analyze these two plots and discuss your responses to each of the two questions posed.

e. Let's now investigate the impact of the geopotential altitude of the object on the distance between the origin and the landing location of the object (for various values of the wind speed). Note that you will use the same initial state vector with an initial position at the origin of the inertial frame for all simulations; however, the atmospheric density will vary based on the geopotential altitude.

   Create a single figure with multiple curves: each curve corresponds to a single value of the geopotential altitude and displays the distance between the origin and landing location as a function of the wind speed. On a second figure, plot the minimum distance between the origin and landing

location as a function of the geopotential altitude. Describe the impact of the geopotential altitude on the landing location.

f. Let's now consider a scenario where the initial speed of the object is constrained by a limited kinetic energy (i.e., due to the initial launch mechanism); assume the kinetic energy is equal to that of the scenario examined in Problem 2c. Vary the mass of the object and, for each mass value, calculate a new velocity vector with the same direction as in Problem 2c but a magnitude that maintains the specified kinetic energy. Use this analysis to determine whether a heavier or lighter object would provide a larger distance between the origin and landing location. Repeat this analysis for various values of the wind speed. Provide plots to support your answer and discuss your findings.

SUBMISSION REQUIREMENTS & FORMAT

This assignment does not require the same extensive writeup as a full lab report. Rather, focus on directly answering the questions and providing the discussion, quantities, and/or plots requested in each problem. Use precise, clear, and complete sentences in your writeup.

Submit a PDF copy of your assignment through Gradescope. Your submission should answer all questions and include a readout of all code created in this lab.

Specific functions requested in the assignment should be included in the main text of your submission for each problem. All files should also be included at the end of the submission and assigned to the "Code" section in Gradescope. Be sure to add titles to all figures that include both the problem number and a description of the plot, e.g. Problem 2c: Time evolution of x.

Be sure to select the correct pages that are associated with each problem in Gradescope and double-check that your submission is readable.

The assignment will be evaluated based on i) correct numerical answers; ii) prose for comments and answers to questions; iii) proper commenting and documenting of code; and iv) the quality of the figures submitted (e.g. labeling, axis, etc).

All lab assignments should include the Team Participation table and should be completed and acknowledged by all team members. Description of the Team Participation table is provided in a separate document.