# Using SIMULINK As a Design Tool

**John S. Klegka**
**Thomas E. O'Donovan**
**United States Military Academy**

Abstract

SIMULINK is a graphical user interface (GUI) to MATLAB for solving ordinary differential equations (ODE's). It is normally used as an analytical tool but may also be used as a design tool. This paper describes how to use SIMULINK from within the MATLAB environment to determine a target value for a design variable. This technique is applicable for use in a design course or in a numerical methods course, as well as for engineering practitioners.

Building a model in SIMULINK may be accomplished by following a process consisting of four distinct stages. These stages are model development, model definition, model analysis, and model verification. Model development consists of developing a mathematical equation that describes in some manner a physical phenomenon. Model definition consists of building the model in SIMULINK. Model analysis consists of running the program to get a solution. Model verification consists of making the assessment of whether the answer from the computer is right or wrong. This paper describes how to use this process to create a dynamic simulation of a falling skydiver.

Once the designer has a model describing the behavior of the system, it is often necessary to determine engineering targets for various parameters of the design. Designers frequently need to make trade-offs among competing design parameters affecting system performance. Thus, the designer must have a tool to predict how changing a particular design parameter will change system behavior. This paper concludes with a description of how to determine the optimum value of drag coefficient for the deployed parachute for the simple skydiver model in the case where we cannot solve for the design variable directly.

## Introduction

SIMULINK can be viewed as a graphical user interface (GUI) to MATLAB for solving ordinary differential equations (ODE's). In English, this means that SIMULINK is a way of getting solutions to ODE's from a computer using a programming language called MATLAB without having to write actual code. This situation is very similar to Windows. Windows is a GUI to DOS and when using Windows one normally does not have to type any actual DOS commands.

Building a model in SIMULINK is done in four distinct stages[1]. We will use this process to create a dynamic simulation of a falling skydiver. These stages are:

a. Model Development
b. Model Definition
c. Model Analysis
d. Verification

Analysis Problem

When a skydiver experiences a complete parachute malfunction, how much time does he or she have to take action before impact with the ground during a jump from 500 meters? Plot position versus time.

Model Development

Model development consists of developing a mathematical equation that describes in some manner a physical phenomenon. In many ways developing the model is the most challenging part of solving the problem. Often this is done once by some brilliant engineer and then modified to fit particular situations. The example we will use will be that of a falling body. Newton was the originator of the differential equation we will be solving. Specifically this ODE describes a falling body acting only under the forces of gravity and drag. If we assume the ground is the origin and up is the positive y direction, the equation looks like[2]:

$$\frac{d^2y}{dt^2} + \frac{c}{m}\frac{dy}{dt} + g = 0 \qquad (1)$$

where $y$ represents displacement, $g$ is the gravitational constant, $c$ is the constant drag coefficient (12.5 kg/s), and $m$ is the mass (70 kg or 154 lb for the skydiver).

Equation (1) is a simple one-degree-of-freedom model of a body acting under the influence of (constant) gravity. The power of SIMULINK lies in the fact that it can be used to model much more complex systems such as multiple-degree-of-freedom systems with variable parameters (like changing mass). A detailed discussion of the modeling process is beyond the scope of this paper. The interested reader is referred to references [3] and [6].

Model Definition

Model definition consists of building the model in SIMULINK. The steps required here are to:

a. Start SIMULINK.
b. Arrange the equation correctly.
c. Setup the equation in SIMULINK.
d. Establish initial values.
e. Document the model.
f. Verify the model definition.

We will not cover the details of building the SIMULINK model here. A brief description is included at Appendix A. Several good tutorials are available, including one at the Mathworks web site[3].

To arrange the equation correctly, solve for the highest order derivative in the equation. For our example, the equation would look like:

$$\frac{d^2y}{dt^2} = -g - \frac{c}{m}\frac{dy}{dt} \tag{2}$$

or

$$\frac{dv}{dt} = -g - \frac{c}{m}v \tag{3}$$

These two equations are ODE's with constant coefficients and we can solve them in closed form. The solution for Equation (1) is made up of the complimentary and particular solutions of a second order ODE. The complimentary solution depends on the roots of the characteristic equation

$$r^2 + \frac{c}{m}r = 0 \tag{4}$$

and the particular solution depends of the forcing function, the constant $g$. The total solution to Equation (1) is

$$y(t) = \left(500 + \frac{gm^2}{c^2}\right) - \frac{gm}{c}t - \frac{gm^2}{c^2}\left(e^{-\left(c/m\right)t}\right) \tag{5}$$

and the solution[4] for Equation (3) is

$$v(t) = -\frac{gm}{c}(1 - e^{-(c/m)t}) \tag{6}$$

The solutions to Equations (1) and (3) will allow us to verify our SIMULINK model. In a more complex situation, we might be unable to obtain the closed form solution. In such a case, model verification is a more complex problem.

Notice that this problem can be represented either as a first or second order ODE. We could solve the first order ODE numerically using Euler's method or some other numerical technique. (In fact, this is exactly what SIMULINK does.) Setting up the second order equation in SIMULINK allows us to model both the position and velocity of the falling skydiver. We create the model by opening various categories of blocks available in the block library, dragging them

into the file window, and hooking them together. A SIMULINK model to solve Equation (1) is shown in Figure 1.

Model Analysis

Using a $4^{th}$ and $5^{th}$ order Runge-Kutta solver with adaptive step size, we ran the model for 20 seconds and obtained the results plotted below for velocity (Figure 2) and displacement (Figure 3). The results show that the skydiver would reach the ground in about 14 seconds with a speed of 50 m/s. The acceleration plot (not show) indicates that the skydiver has not reached terminal velocity but is still accelerating slightly. The key question now is whether we should believe the results of the SIMULINK simulation.

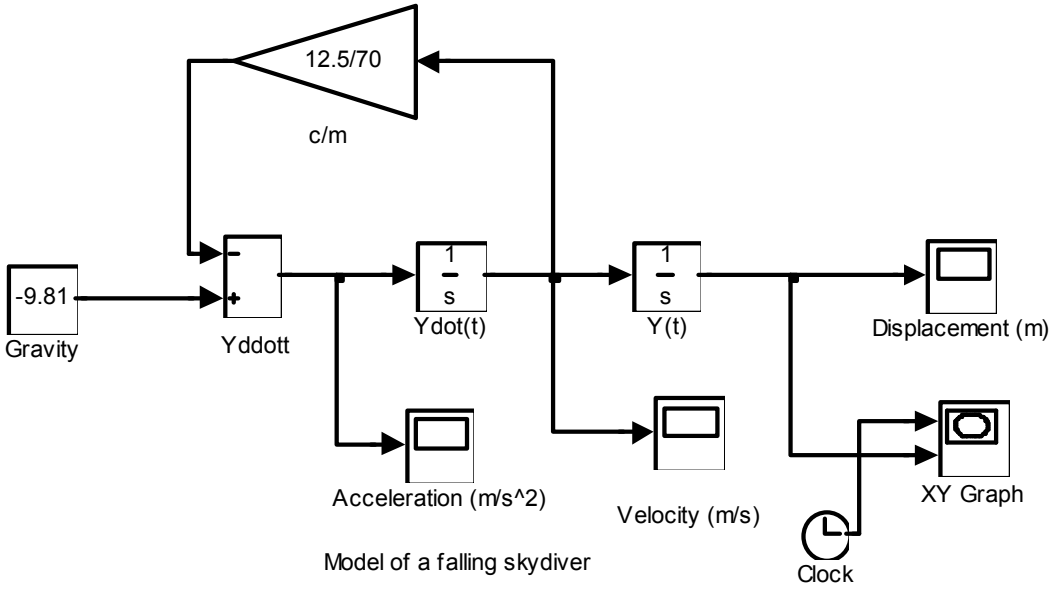


**Figure 1. Model of the Falling Skydiver (asee_jumper)**

Verification

Verification is where engineers earn their pay. It is the step that separates the real engineer from the computer operator. Basically it consists of making the decision of whether the answer from the computer is right or wrong. Many techniques exist for doing this ranging from rationalization (i.e., does it look right and, if not, why?) to running the problem on a different program, to making additional assumptions that simplify the model allowing approximate answers, to doing experiments. Verification is vital and without it all the work to this point is just pie in the sky.

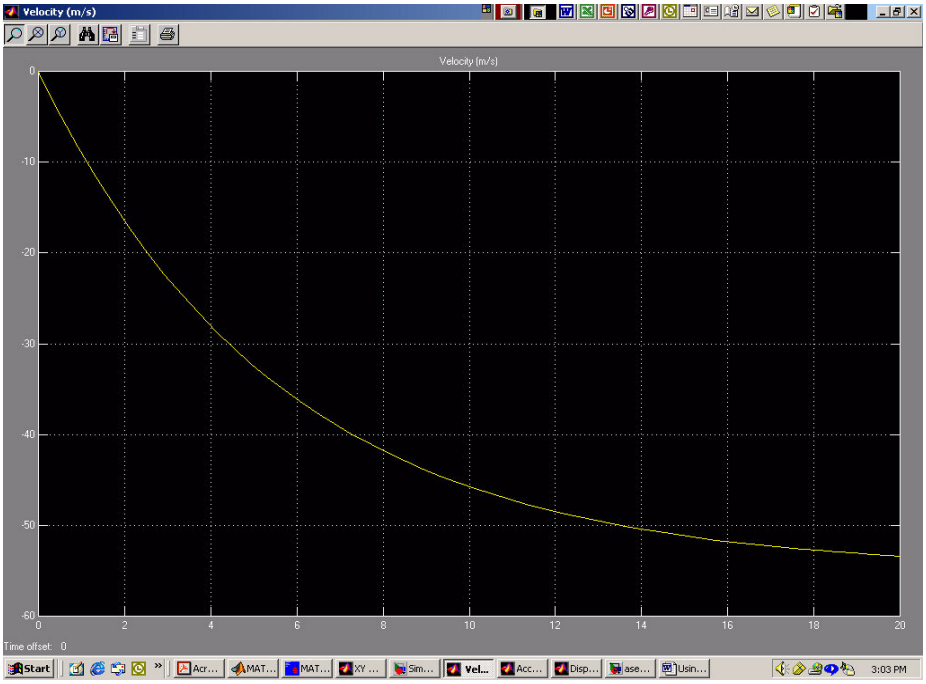


**Figure 2: Velocity Results**

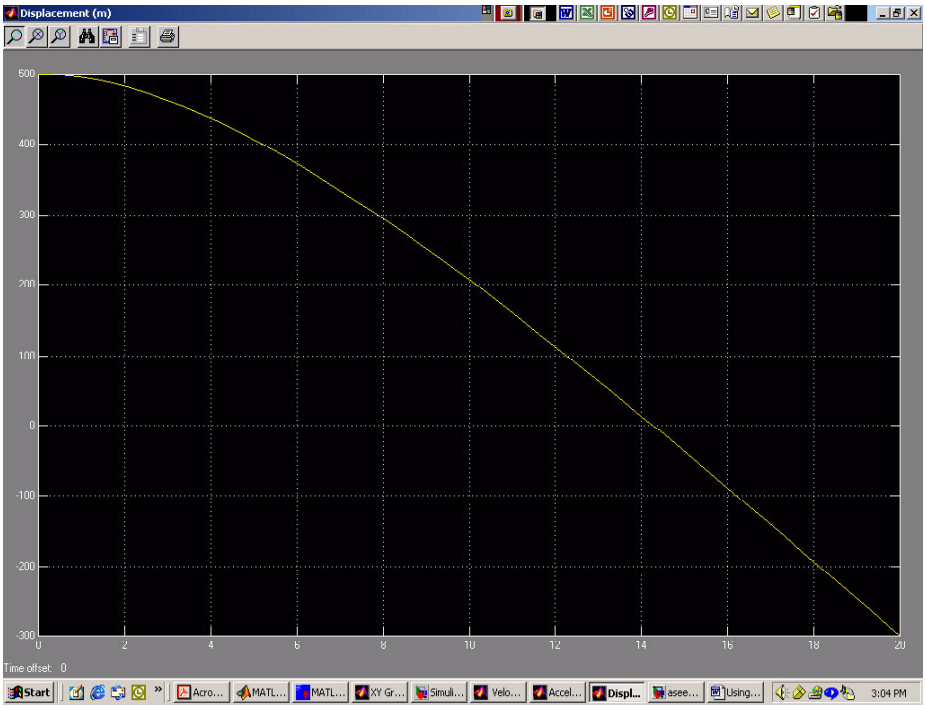


**Figure 3: Displacement Results**

In this case, the closed form solution for velocity is shown at Equation (6). Substituting in the given parameters for *c*, *m*, and *g* and looking at 10 seconds we see that the analytical solution yields a speed of 45.72 m/s. The SIMULINK model predicts a velocity of 45.72 m/s*. See Figure 4. The displacement is given by Equation (5). At 10 seconds it gives a value of 206.7 m. The SIMULINK solution predicts the same displacement. Thus, we can have confidence that the SIMULINK model is reasonably accurate.

Analysis Problem Solution

We are now in a position to answer the question posed earlier. We assume that the skydiver must be at least 100 m above the ground in order for the reserve chute to deploy successfully and slow the jumper to a new terminal velocity. By zooming in on the SIMULINK scope for displacement, we see that the skydiver will reach 100 m in about 12.25 s – not a lot of time. See Figure 5. (This helps explain the need for skydivers to be well trained in emergency procedures!)

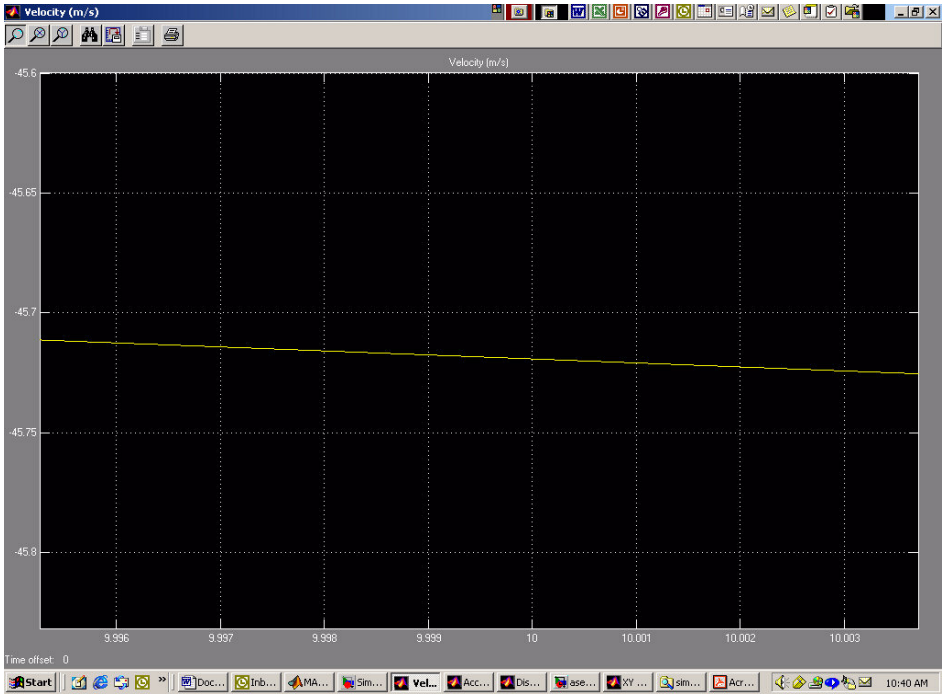


**Figure 4: Close-up of Velocity Results**

* There is no discrepancy between the two solutions out to seven significant figures.

Design Problem

Analysis is an important part of the design process. Analysis allows the designer to predict the performance of a system based on the values of various operating parameters. Often, however, the designer needs to set engineering targets for design variables. For example, the engineer may try to achieve specific values of speed, acceleration, and fuel economy by varying weight, drag coefficient, and engine power. In this case, the designer needs a way to determine the optimum values for these design variables.

We will now look at the effect of changing the drag coefficient on the terminal speed of the skydiver. Clearly, there is an upper limit on the safe landing velocity of the skydiver. What drag coefficient must the deployed parachute have to achieve this speed?

Equation (6) describes how the velocity changes as a function of time and drag coefficient. However, we cannot solve Equation (6) for $c^5$. We cannot use Equation (6) directly to find $c$ given $v$. We need another way to find the optimum value for the drag coefficient.

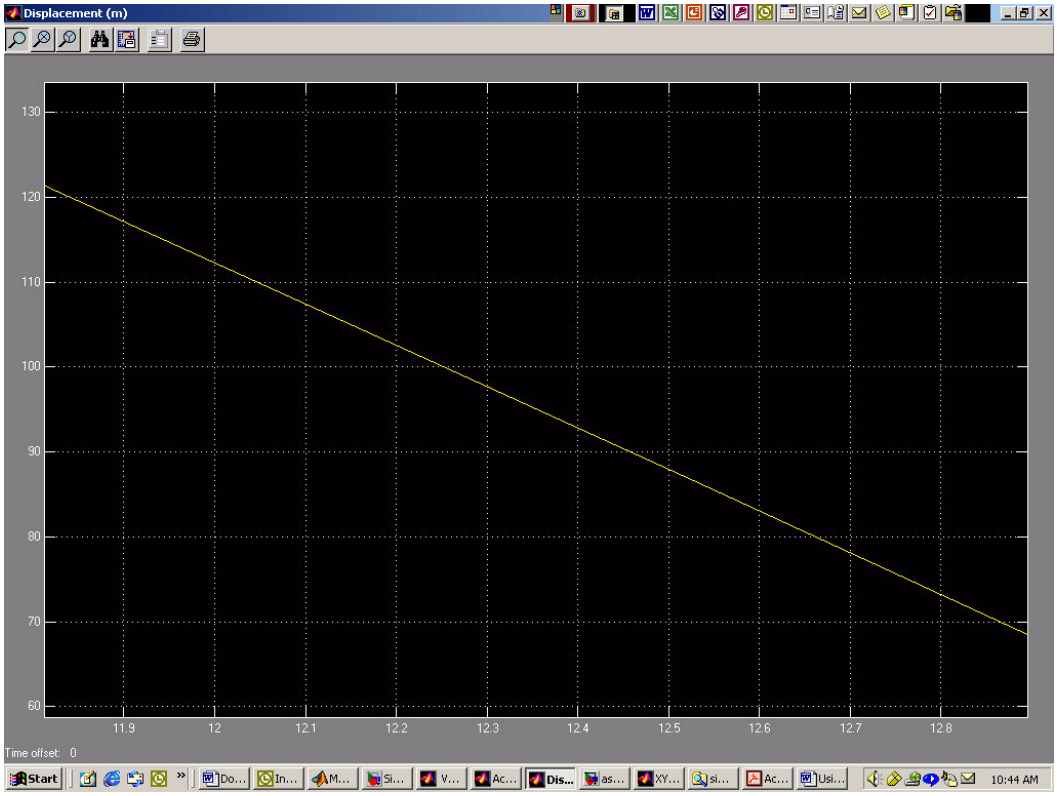


**Figure 5: Close-up of Displacement Results**

The most straightforward way to answer this question is simply to assign values to the drag coefficient, $c$, in the SIMULINK model, run the model, and observe the predicted final velocity.

If that velocity is too high, increase *c* and try again. In a more complex situation, it may not be obvious what the effect of design variables will be on system performance. Alternatively, there may be several design variables to be considered. In either case, it would be helpful to be able to automate the process.

Since SIMULINK works inside MATLAB (and, in fact, writes MATLAB code) we can use MATLAB to control our SIMULINK model and search for an optimum value based on whatever criteria are appropriate.

There are several ways to do this optimization procedure. One way is to use the Optimization Toolbox [6] found in MATLAB. This procedure is discussed in reference [6]. Another method is to write our own simple optimization routine. We will look at a way to do the latter. (An instructor in a numerical methods course might have their students solve this problem as an exercise, for example.)

Suppose the maximum safe landing speed for the skydiver in this problem is 3 m/s. To achieve this terminal velocity, we must increase the drag coefficient of the parachute (presumably by making the parachute larger.) However, larger parachutes cost more and are heavier, so we want to find the drag coefficient that gets us as close to 3 m/s as possible. We first need to bound the problem. Using our SIMULINK model, we set c to a large value – say 500 kg/s – and find the terminal velocity is about 1.5 m/s. So the optimal value lies somewhere between 12.5 and 500.

One method to follow is to search the design space. We will implement a bisection search method[7] to do this. The code below represents a MATLAB m-file which implements the bisection search, calls the SIMULINK model (asee_jumper), and finds the optimum value for the drag coefficient:

```
% This M-file calculates the optimum value of the parachute
% drag coefficient required to achieve a specified landing speed.
% It uses a bisection search technique between previously
% established limits for the drag coefficient.

% Establish initial values
cl=12.5; % Lower limit for drag coefficient
cu=500;  % Upper limit for drag coefficient
test=1.0;% Initial value for stopping criterion
cr=(cl+cu)/2; % Calculate estimate for optimum c

%Use the bisection method to search for the optimum c
while abs(test) > 0.001 % Are we close enough?
   [t,vel,y] = sim('asee_jumper') %Run the SIMULINK model
   test = vel(30) + 3; %Update the test variable
   if test > 0
      cu = cr
   else
      cl = cr
   end
   cr = (cl + cu) / 2 %Update the estimate for optimum c
end

[t,vel,y] = sim('asee_jumper') % Run final simulation with optimum c

copt = cr %Return the optimum value
```

We have to make one change to the SIMULINK model shown in Figure 1. We replace 12.5/70 in the gain block (*c/m*) with the value cr/70. Now, each time MATLAB calls the SIMULINK model 'asee_jumper', the current estimate of the drag coefficient (cr) will be used in the simulation.

Design Solution

The results of using the m-file are shown below:
copt = 228.9948
» test
test =-3.1809e-004
» vel(30)
ans = - 2.9988

The velocity as a function of time is shown in Figure 6. We see that velocity stabilizes at about 3 m/s (actually 2.9988 m/s, given our stopping criterion). Thus, the designer can set an engineering target for the drag coefficient of the parachute (229 kg/s). This engineering target can be used with others in a design tool such as Quality Function Deployment to trade off among competing design parameters.

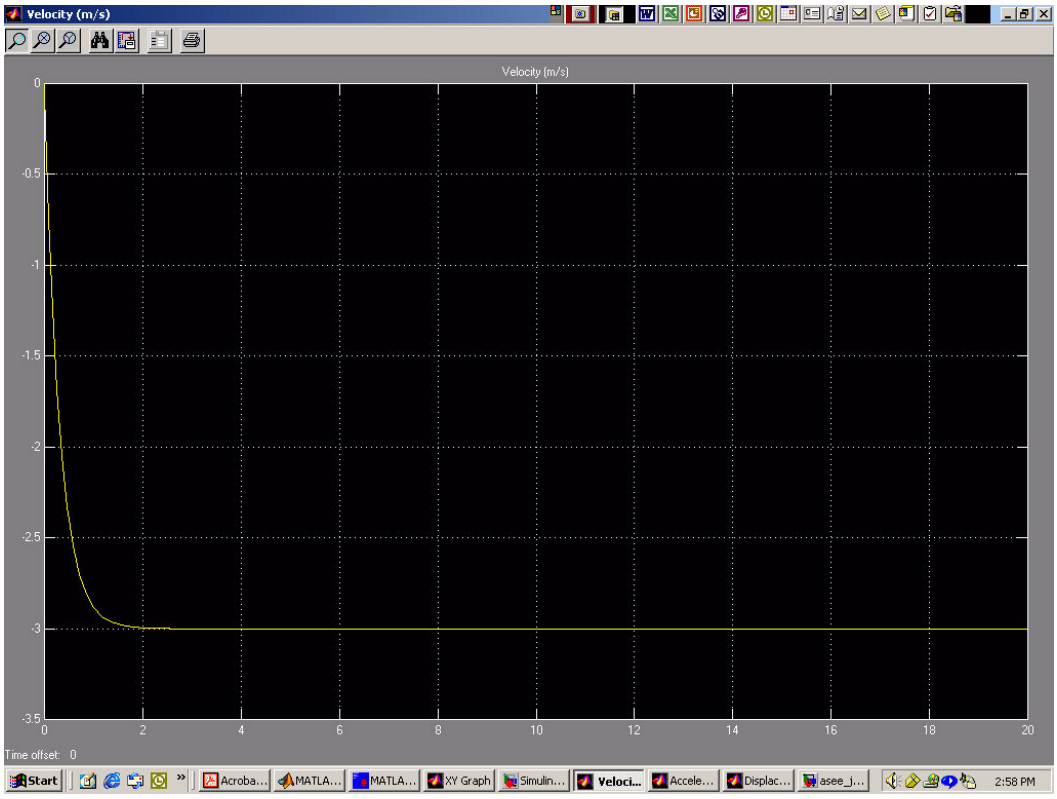


**Figure 6: Velocity Results for Optimum Drag Coefficient**

## Conclusion

In this paper, we have seen how to use SIMULINK and MATLAB together as both an analytical tool and as a design tool. The ease of programming in MATLAB and of passing information back and forth between MATLAB and SIMULINK facilitate the use of this software in undergraduate engineering education. The power and flexibility of MATLAB as a programming environment make it useful in courses ranging from basic computer programming through numerical methods to mechanical design.

We have solved a well-known single degree of freedom dynamics problem to validate the proposed methodology. A more interesting problem is to apply this method to more complex problems such as multi-degree of freedom problems with variable coefficients. This is beyond the scope of this paper and will be addressed in a future paper.

---

[1] This procedure is adapted from a tutorial written by CPT Thomas O'Donovan.
[2] Chapre, Steven C. and Raymond P. Canale, *Numerical Methods for Engineers, 4th Ed.*, (New York, McGraw Hill, 2002) p. 14.
[3] http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/simulink.shtml
[4] Chapre and Canale, p. 14.
[5] Ibid.
[6] Dabney, James B. and Thomas L. Harman, *Mastering SIMULINK 2*, (Upper Saddle River, NJ, Prentice Hall, 1997), pp. 213-216.
[7] Chapre and Canale, p. 122.

JOHN S. KLEGKA
John Klegka is Professor of Mechanical Engineering and the ME Division Director in the Department of Civil and Mechanical Engineering at the United States Military Academy, West Point. He received a B.S. from USMA in 1973, a M.S. degree from the University of Michigan in 1982, and a PhD from Texas A&M University in 1989. He has enjoyed many assignments within the U.S. Army during his more than 29 years of active duty service.

THOMAS E. O'DONOVAN
Tom O'Donovan is a former Assistant Professor in the Mechanical Engineering Division of the Department of Civil and Mechanical Engineering at the United States Military Academy. He received a B.S. degree from the University of Vermont and an M.S. degree from the University of Washington. He is currently serving as a Corps of Engineers officer in the United States Army.

Appendix A: Model Development

Open SIMULINK by starting MATLAB and clicking the SIMULINK icon. We will start by clicking on the MATH Library Block. This opens the Library and shows us what blocks are available. Position the mouse on the SUM block and push down with the left-hand mouse button, then move the pointer into the file window and release the mouse button. This is called click and drag and moves the SUM block into the file window. Click and drag the GAIN block into the file window. Open the CONTINUOUS Library and drag two INTEGRATOR blocks into the file window. You can close the libraries by clicking the + box next to the Library name. Open the SOURCES Library and drag the CONSTANT block and open the SINKS Library and bring the SCOPE block into your file window.

We will now begin hooking these blocks together as dictated by our equation. First, note that our highest order derivative is equal to the sum of two items. So we start by positioning the SUM block in the middle of an open area within our file window. The first term coming into the SUM block is a constant, so we position the CONSTANT block just to the left of the SUM block and connect them with a line by positioning the cursor near the output point on the CONSTANT block, holding down the left mouse button, moving over to an input point on the SUM block, and releasing the mouse button. The result will look like the blocks in the left side of the figure below.

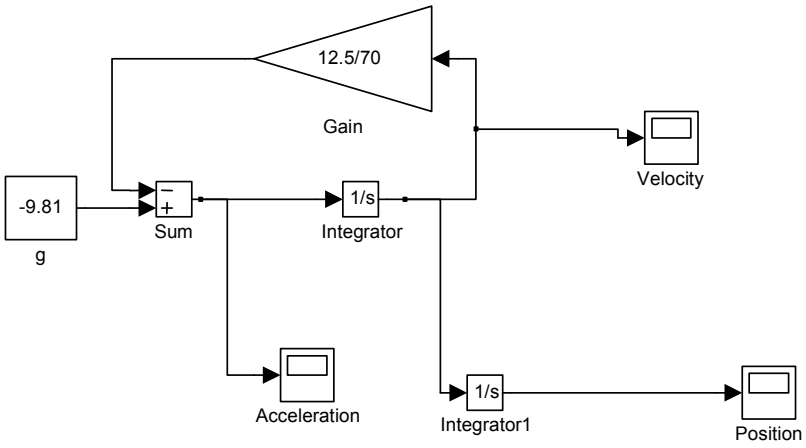


Figure A-1: Model of a falling skydiver

If we double click on the CONSTANT block we can set what the constant is in the window that opens. Do this and set the constant to -9.81. This is our gravitational constant in $m/s^2$. Now we create the second term. This one is a little more complex as it involves a constant, *c/m*, times the velocity. To use this term, we must calculate the velocity. Recalling that velocity is the integral

of acceleration, we move the INTEGRATOR block into position to the right of the SUM and connect them with a line. The output from the INTEGRATOR block will be velocity, which we can now use after first multiplying it by the *c/m* factor. To do this we need to use the GAIN block. We position the blocks and connect them as shown in Figure A-1. Position the INTEGRATOR1 block and position scope as in the figure above.

We set the magnitude of the gain by double clicking the gain box and filling in the value with the *c/m* value for the jumper, 12.5/70. We have now completed modeling the equation; however what we are interested in is a plot of the jumper's velocity with respect to time. To get this plot we need to use the SCOPE block from the SINKS library. We position it as shown above.

Using the right mouse button will allow you to branch a line off of an existing line as shown above. We have now completed our setup of the equation in SIMULINK and are ready to establish the initial conditions.

To set the initial conditions for the problem we simply double click on the INTEGRATOR block and set the initial value to the initial velocity of the jumper. If he just stepped out of the plane this would be zero, the default value. In the INTEGRATOR1 block, set the jump altitude to 500m. The output of the second integrator is position. This demonstrates a powerful SIMULINK capability: SIMULINK can quickly model and solve a second order ODE.

Now we need to document our model. This consists of editing the labels under each block and adding labels or titles so that they show very clearly to the user what is actually going on. To change a block label, click on it once and then edit it. To add text somewhere in the file window just click there and start typing. You can also click and drag text. Also it might be a good idea to save your file at this point by clicking on FILE>SAVE and filling in the appropriate name. (NOTE: Do NOT use the name SIMULINK.)

We now must verify our setup by working through it in a logical manner. Starting at Acceleration in the figure above we can trace back by noting that our SIMULINK model says: "Acceleration is the sum of g and c/m times Velocity." This in fact is wrong, as we actually want to subtract the c/m times Velocity term. To correct this we double click on the SUM block and change the signs appropriately. We now have completed model definition (compare with Figure 1).

Model analysis consists of running the program to get a solution. To do this we first double click the three SCOPE blocks and set their parameters for viewing. First, we move them to convenient locations out of the way. Then we set the horizontal and vertical ranges. We will run this model for 20 seconds so we set the horizontal range to 20. We don't know how fast the skydiver is going to go but we can estimate it by noting that actual terminal velocity for free fall parachutists is about 60 m/s. This brings out an important point: YOU BETTER HAVE A GOOD IDEA ABOUT THE SOLUTION BEFORE YOU RUN YOUR MODEL or you're headed for trouble. Now with the ranges set, we need to establish the parameters for the solution process. To do this, we click on SIMULATION at the top of your file window and then PARAMETERS. First, we select the appropriate solution method. We will use a fixed step solver for the Euler Method

(ode1). Change the stop time to 20 seconds. Close the parameters window and click START to run the model. Watch your SCOPE for a plot of velocity over time. Note how the jumper picks up speed very quickly until terminal velocity is reached. Save the model.