Armstrong State University
Engineering Studies
MATLAB Marina – Algorithm Development I Primer

Prerequisites
The Algorithm Development I Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, and scripts. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module.

Learning Objectives
1. Be able to write algorithms to solve sequential problems.
2. Be able to implement sequential algorithms using MATLAB.

Terms
algorithm, stepwise refinement

MATLAB Functions, Keywords, and Operators
none

Algorithms
Algorithms are the logical recipes that underlie computer programs. Virtually all disciplines now rely on some form of algorithmic model: medicine, economic theory, social theory, etc. Computer programs express algorithms in a particular programming language. The goal is to take something that can be done in a human mind and use a computer to perform the mind's work. Algorithms must be precise and unambiguous.

Consider generating an algorithm to brew coffee. Assumptions: will make cowboy coffee (in a pot on the stove). Figure 1a shows an imprecise sequence of steps to make coffee and Figure 1b shows a more precise sequence of steps to make coffee.

Get a pot
Fill with water
Place on stove
Heat water
Add coffee
Turn off heat
Let sit
Pour into cup

Figure 1a, Algorithm (not precise)

Get a two quart saucepan
Fill with one quart of tap water
Place on stove
Heat water to 210 degrees F
Add 8 Tablespoons coffee (2 Tbsp per cup of water)
Turn off heat
Let sit for 4 minutes
Pour coffee through a strainer into cup

Figure 1b, Algorithm (more precise)

There may be a need for additional algorithms that precisely describe how to get the pot (where it is, how to recognize a saucepan), how the coffee should be ground (is it pre-ground or does it need to be ground, where the coffee grinder is located and how to grind if not pre-ground), how to heat things on a stove, etc.

Algorithms for computer programs need to be precise, more like the sequence of steps of Figure 1b.

Stepwise Refinement

Larger more complex problems are typically solved by breaking the problem up into smaller problems each of which is relatively easily solved or has been solved before. This is called stepwise refinement. The solutions to the smaller problems are integrated into a solution for the larger problem.

A general procedure for solving large complex problems is:

1. Decompose the larger problem into smaller subproblems.
2. Continue decomposing the subproblems until you obtain subproblems that are manageable (easily solvable or already solved).
3. Solve each subproblem and use these solutions to solve the more complex subproblems and finally the original problem.

Keep your initial problems general and add details as necessary. Do not worry about program syntax until you are ready to solve the subproblems.

Some additional tips:

- Solve the sample problem using a set of data by hand. If you cannot solve the problem, you will not be able to write an algorithm to solve the problem.
- It is often helpful to describe the problem from an input – output perspective, i.e. what information does the program need to solve the problem and what must be calculated.
- Logic errors (errors in the algorithm) will generally not be caught until the program is tested. Just because a program compiles and links successfully does not mean it works. Compilers/linkers catch syntax errors, not logic errors. Test your solution with some fairly simple data that you can verify by hand and then with a reasonable number of sets of additional data (both valid and invalid). It is here where you will catch any logic errors.

Problem Solving Example

Write a MATLAB program to determine if a thrown snowball will hit a target.

Problem analysis:

Have we been given enough information to solve the problem? If not, what do we need to be able to solve the problem? Determining if a thrown object hits a target involves projectile motion (physics). The equations of motion in two dimensions are:

$s_x = s_{x0} + v_{x0} \cdot t + \frac{1}{2} \cdot g_x \cdot t^2$

$s_y = s_{y0} + v_{y0} \cdot t + \frac{1}{2} \cdot g_y \cdot t^2$

Where s is position, v is velocity (speed plus direction), g is the acceleration due to gravity, and t is time. To determine if a projectile hits a target we need to know the location of the target relative to the initial location of the projectile, i.e. how far away (horizontal and vertical distance) is the target from the thrower. We will also need to know the initial velocity and initial direction of the projectile (how hard and in what direction was the snowball thrown) and the size of the target.

Assuming that gravity is constant, that the target does not move and is at position $\left(s_x, s_y\right)$, that the thrower is stationary with initial position $\left(s_{x0} = 0, s_{y0} = \text{height}\right)$, and that the snowball does not lose any energy due to air resistance:

- The inputs to the program will be the initial velocity v_0 (how hard thrown), initial height (how tall is thrower), and release angle of the snowball (direction thrown).
- The program will determine and output if the target was hit. The program must determine if the target is in the snowball's path of motion.
- The program will need to solve for the time it takes for the snowball to reach the target and the height of the snowball at that time, i.e. what is the height of the snowball when it covers the horizontal distance to the target. The program can then check if this height is within the area occupied by the target.

Initial Algorithm:

- Read in the horizontal and vertical distance to the target.
- Read in the initial velocity and release angle of the snowball.
- Determine if snowball hit target.
- Output whether snowball hit or missed target.

Further refining the determination if the snowball hit the target:

- Calculate the x and y components of the snowball's velocity.
- Calculate the time it takes the snowball to reach the target.
- Calculate the height of the snowball when it reaches target.
- Determine if snowball height is less than the target height. If yes, the snowball hit the target.

Final Algorithm:

- Read in the horizontal and vertical distance to the target.
- Read in the initial velocity and release angle of the snowball.
- Determine if snowball hit target.
  a) Calculate the x and y components of the snowball's velocity.
  b) Calculate the time it takes the snowball to reach the target.
  c) Calculate the height of the snowball when it reaches target.

d) Determine if snowball height is less than the target height. If yes, the snowball hit the target.

- Output whether snowball hit or missed target.

A comment skeleton corresponding to the algorithm is shown in Figure 2a. The coded program is shown in Figure 2b.

```
% Horizontal and vertical distance to the target

% Initial velocity and release angle of the snowball

% Determine if snowball hit target
% Calculate the x and y components of the snowball's velocity

% Calculate the time it takes the snowball to reach the target

% Calculate the height of the snowball when it reaches target

% Determine if snowball height is less than the target height

% Output whether snowball hit target.
```

Figure 2a, Comment Skeleton for thrownSnowball Program

To determine if the program operates correctly, one must test the program for a representative set of test cases. Test for a fairly simple case that can be verified in your head first then test for the other cases. For the thrownSnowball program, one representative set of test cases is: a hit where the distance is small (< 10 feet), a hit for a distance of around 20 – 50 feet, a miss where snowball comes up short, and a miss where the snowball flies over the target's head.

Some sample results for the thrownSnowball program are:

- Height thrower 6 feet, height target 5.5 feet, distance to target 10 feet, snowball velocity 50 fps, and snowball angle 0 degrees. This is a hit, height of snowball at the target is 5.34 feet.
- Height thrower 6 feet, height target 5.5 feet, distance to target 50 feet, snowball velocity 50 fps, and snowball angle 20 degrees. This is a miss, height of snowball at the target is 5.6 feet. The snowball flew just over the target.
- Height thrower 6 feet, height target 5.5 feet, distance to target 50 feet, snowball velocity 50 fps, and snowball angle 10 degrees. This is a miss, height of snowball at the target is -2.1 feet. The snowball fell short.
- Height thrower 6 feet, height target 5.5 feet, distance to target 50 feet, snowball velocity 50 fps, and snowball angle 15 degrees. This is a hit, height of snowball at the target is 1.8 feet.
- Height thrower 6 feet, height target 5.5 feet, distance to target 50 feet, snowball velocity 100 fps, and snowball angle 10 degrees. This is a miss, height of snowball at the target is 10.6 feet. The snowball flew well over the target.

Note: 50 fps is 34.1 mph and 100 fps is 68.2 mph.

```
clear all;
clc;
close all;

% gravity
gx = 0.0;       % ft/s^2
gy = -32.81;

% Initial position of thrower
sx0 = 0;    % ft
sy0 = input('Enter height of snowball thrower in ft: ');
% Horizontal and vertical distance to the target
sx = input('Enter horizontal distance to the target in ft: ');
height = input('Enter height of target in ft: ');
% Initial velocity and release angle of the snowball
v0 = input('Enter initial velocity of snowball in ft/s: ');
theta = input('Enter release angle of snowball in degrees: ');

% Determine if snowball hit target
% Calculate the x and y components of the snowball's velocity
vx0 = v0*cosd(theta);
vy0 = v0*sind(theta);
% Calculate the time it takes the snowball to reach the target
ttarget = sx/vx0;
% Calculate the height of the snowball when it reaches target
sy = sy0 + vy0*tTarget + 0.5*gy*ttarget^2;
% Determine if snowball height is less than the target height
if (sy > 0.0 && sy < height)
    message = 'Snowball hit target';
else
    message = 'Snowball missed target';
end
% Output whether snowball hit target.
disp(message);
```

Figure 2b, thrownSnowball Program (Omitting Program Comment Header)

Last modified Tuesday, September 09, 2014