Stay The Course: A Heading Correcting Algorithm for a Bottom-Crawling AUV

Drew Arlen Burrier, Moss Landing Marine Laboratories

Mentors: Richard Henthorn

Summer 2015

Keywords: Vehicle logs, Machine Learning, Bottom-Transecting AUV,

ABSTRACT

Abstract text (Body text 2, Times New Roman, 12 pt; 1.5 line spacing)

1. INTRODUCTION

1.1 BENTHIC ROVER

The Benthic Rover is a bottom-crawling Autonomous Underwater vehicle utilized and designed by the Dr. Ken Smith's team at the Monterey Bay Aquarium Research Institute. It is designed to enable long term monitoring of benthic communities with deployments of up to a year to depths up to 6000 m. It has recorded multiple deployments at station M at a depth of ~4000 m.

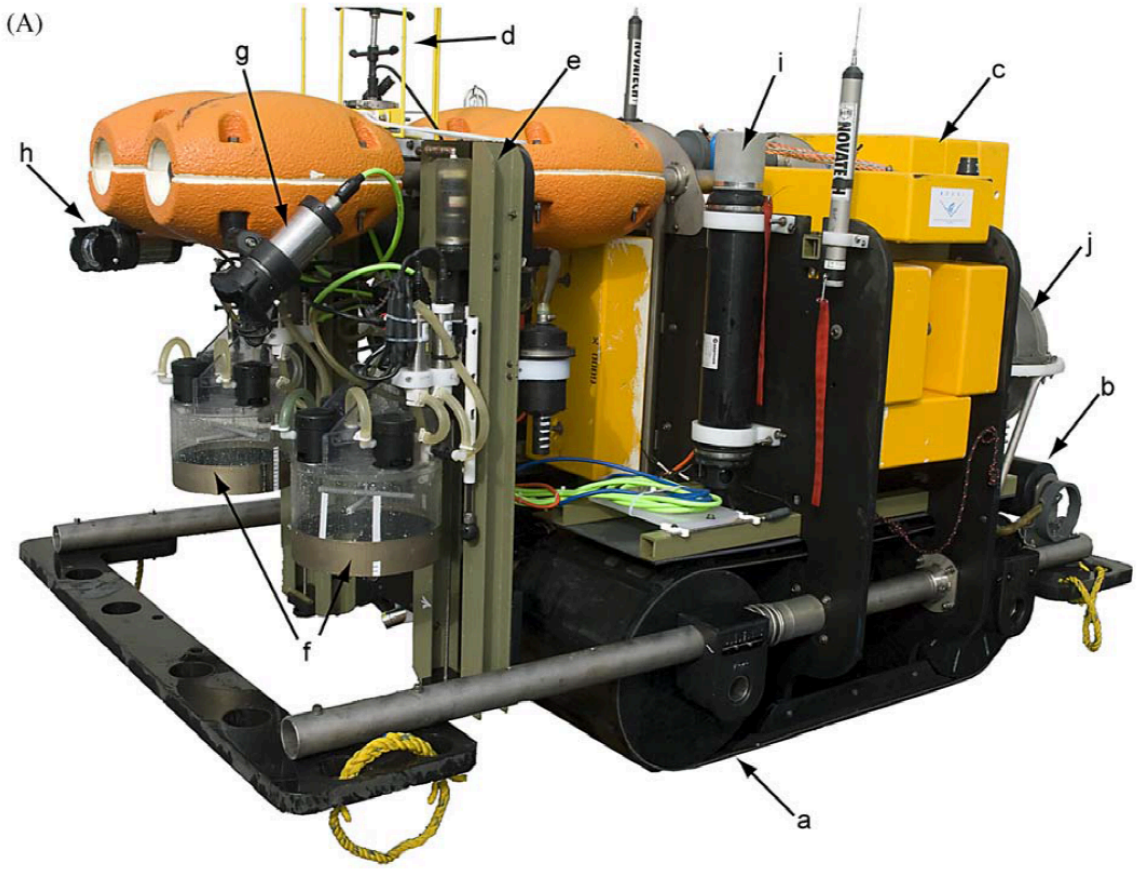

Figure 1 A side view of the Benthic Rover shows: (a) the treads that are part of the propulsion system, (b) one of the two propulsion motors housed in oil, (c) syntactic foam used for floatation, (d) the current meter used to ensure the Rover is driving up current, (e) the instrument rack, which lowers and inserts the respirometer chambers, (f) the respirometer chambers, which measure sediment community oxygen consumption, (g) the high-resolution digital camera aimed at the sea floor, (h) a 200-W-s strobe to illuminate the still images, (i) the acoustic modem that can release a drop weight, and (j) one of two titanium pressure housings used to carry batteries and electronics. Taken from Sherman and Smith (2009).

The vehicle propulsion system is comprised of two 46-cm-wide lugged tracks driven independently by two brushless motors mounted in pressure-compensated oil-filled housings. The Rover is also equipped with a cylindrical ballast weight suspended between the treads and is made detachable by means of a burn-wire (to which a current is applied upon receiving an acoustic signal, causing it to corrode and release) and two lever arms. The vehicle when deployed has an air weight of 1136 kg and an in water weight of 68 kg and upon release of ballast has a positive buoyancy of 45 kg. The Rover is deployed with a two-dimensional current meter, which is mounted on top of the Rover, two respirometer chambers mounted on instrument racks move up and down on a carriage and are depressed into the

sediments in between transits. The respirometer chambers house stirrers to homogenize the chamber, and optodes that measure oxygen. The Rover also is equipped with a reference optode outside of the chambers to compare ambient seawater to the chambers. Two video cameras capture the area enclosed by the chambers, and a high-resolution camera is mounted on the front of the vehicle and captures images of the sea floor during transits with the assistance of a strobe. The transit camera is coupled with a flouremetric imaging system in order to measure the size of detrital aggregates as well as provide estimates of the detrital quality.

1.2 THE MISSION

The Benthic Rover is deployed via free fall from a surface ship. Once it awakens from its deployment sleep mode, a typical mission consists of transits of a pre-programmed distance of 10 m away from the drop site due East. The Rover is programmed to move into a "favorable current" (or one that is within a 45° window of the front of the vehicle) so that any sediment resuspended by the treads will be carried away from the new study site. The Rover will check the current meter every 15 minutes until a favorable current is found up to a and elapsed time of twelve hours, at which time it will move regardless of the direction of the current. Upon the completion of the transit, the Rover will enter a pause behavior in order for the ambient environmental conditions to stabilize. It will then lower the respirometer chambers into the sediments and take optode measurements for a period of two days.

1.3 MOTIVATION

The project grew out of the initial task of developing a log grinder for rapid vehicle performance assessments between deployments. The Rover's syslogs consist of a time stamped record of every behavior it executes, which is tremendously useful in evaluating the Rover's operation. The downside is that as deployments increase in duration these syslogs can be millions of lines long making it very difficult for the engineering team to see performance trends. As such it was extremely useful to have a program that would sift through the logs

and pull out lines of specific interest. One of the first behaviors that we decided to look at was when the Rover was moving. This would tell us right away how many moves the Rover made over the course of the deployment, and when they took place To do that we needed a record that marked the beginning of the move in the syslog. Operationally the Rover is programmed to check it's current heading, and compare that to its pre-programmed, desired heading, which it reports as a degree difference. It then corrects for this difference, turning back to its desired heading by engaging one motor a certain number of rotations relative to the magnitude of the difference. Once we got here, it was clear that on a lot of the deployments that the Rover was timing out frequently. We then wanted to find out, of those times that the Rover timed out, which direction was the current going. The ruby code that was developed for this process can be found in Appendix 1. This revealed some interesting things about the Rover's performance that became extremely valuable to the team.

However given the early success we had with the log grinder applications, we now had performance data from the Rover that in addition to rapid assessment could be used to develop improved control methods to address certain aspects of the Rovers programming. I had previously had an interest in machine learning algorithms and with the support of the team decided to try to write a learning algorithm that the Rover could use to correct errors that develop in its propulsion system.

1.4 MACHINE LEARNING

While machine learning has received a great deal of press of late, and is notably associated with complex scenarios like self driving cars, and advanced robotics, it is at it's core a field of computer science that was developed as a branch of statistics as it relates to pattern recognition. Machine learning relies on the constructions and development of algorithms that can learn from, and make predictions based on data as opposed to trying to determine causality in a data set. This can be applied to many complex problems, but it is as a field not inherently complex. This is stated because while the Rover is an extremely sophisticated

instrument, it is intentionally "stupid" in terms of its navigational operation. The reason the Rover was designed this way is that the benthic environments in which it has been deployed are flat, and uncomplicated and there are not obstacles it needs to avoid. Additionally the locations of the samples do not need to be known with any amount of certainty under the current mission objectives. All that matters is that the Rover moves, does not disturb future study sites, and is recoverable. Given that the primary objective of the Rover's operating system is to conserve battery power, which the engineering team accomplished by limiting the amount of information it collects, and minimizing the amount of computing it has to do, it was therefore critical to similarly minimize the amount, and complexity of the computing necessary for the Rover to solve for its own error. As such, since "machine learning" is a large umbrella, covering statistical analyses with a range of complexities, this project elected to utilize some of the more simple algorithms.

Linear regression

2. MATERIALS AND METHODS

The Vehicle and instrument control applications aboard the Rover are executed in the Ruby programming language. Ruby is and object-oriented scripting language and interpreter implemented in C, allowing it to be readily ported to embedded computing platforms without the need for a supporting tool-chain. Ruby is a bit unusual as a programming language for a vehicle of this nature, however it does offer a fairly user-friendly means to writing concise and readable applications, as well strong object-oriented support and high-level programming features. These attributes allow the engineers on the Rover team to Isolate failure logic and apply methods across the code-base in a consistent manner, improving fault detection (Mcgill et al 2007). As such the log-grinder applications were all developed in the Ruby programming language utilizing Sublime text as an interface. Matlab was then utilized to create visualizations with the log-grinder results. Also the linear regression algorithm was first written in matlab, as it is a language that I was

more familiar with using. The algorithm was then translated into Ruby, however, the internship ended before I was able to complete and test the Ruby version.

3. RESULTS

3.1 LOG GRINDER

The log grinder functioned exactly as intended by filtering out the information of interest from the syslogs, and placing them in new files containing all of the particular behavior. From there this information could be easily scanned, but perhaps more importantly it was able to be utilized by matlab. One of the things that we were able to notice as a result of these new individual behavior logs, was that the Rover was timing-out a lot more frequently than previously thought. Which meant that the Rover was sitting idle a lot longer than expected. Over the course of a longer deployment means missed opportunities to collect data. For pulse 64, the most recent deployment, the rover timed-out on 60% of its moves. This means that out of the 103 moves the Rover made, 61 of them were made without a favorable current, and therefore at risk of having sediment kicked up during the move land at the next sample site (figure 2).

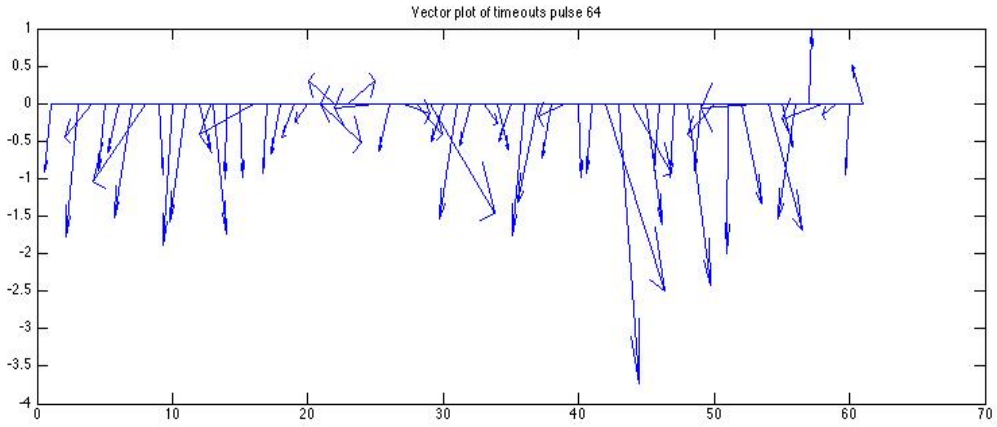

Figure 2 is a vector plot for Pulse 64, and shows the direction that the current was heading when the Rover timed-out and made a move without a favorable current. The length of the arrow indicates the magnitude of the current, while the angle indicates the direction of the current.

3.2 LINEAR REGRESSION

The linear regression worked surprisingly well for Pulse 62 and 64, hitting the global minimum for each deployment presented to the algorithm. The thetas found for Pulse 62 was -0.307 for the slope, and 5.192 for the intercept (figure 3 and 4). As this was a short deployment, I projected what the heading difference would have been after 50 moves, and found that the Rover would have been off of its desired heading by -10.1718°.

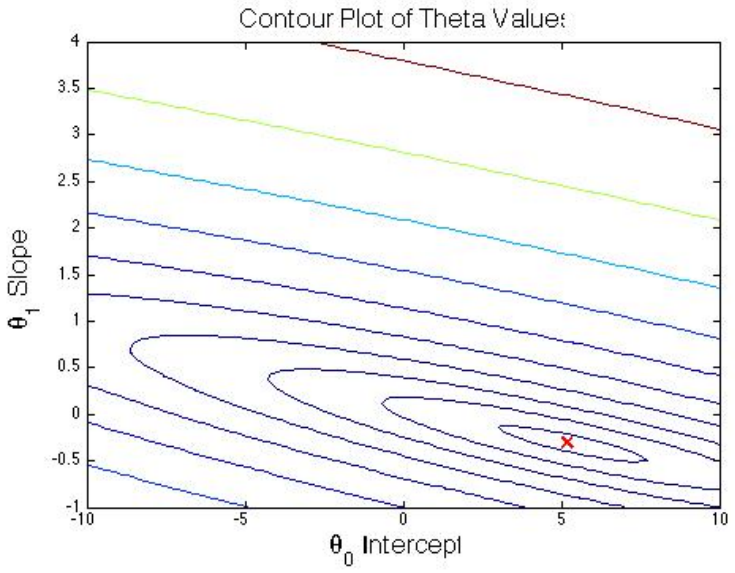

Figure (3) shows a contour plot of the theta values found by gradient descent for Pulse 62

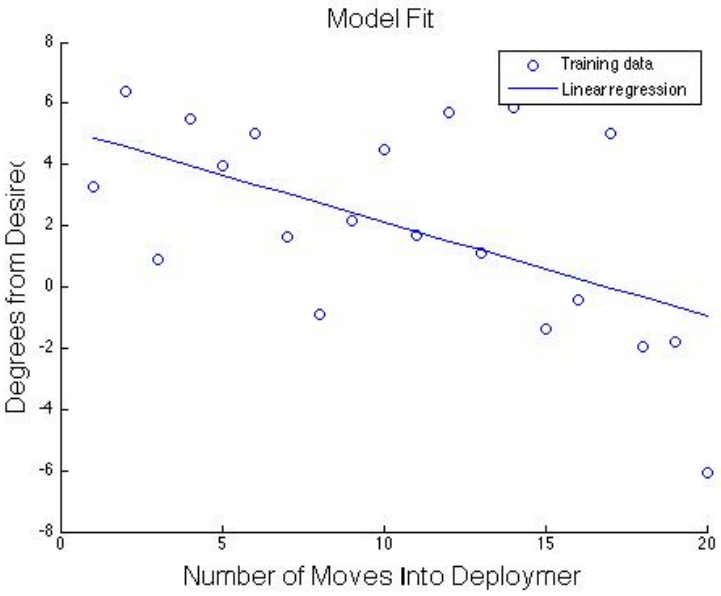

Figure (4) shows the model fit for Pulse 62

4. DISCUSSION

4.1 LOG GRINDER

Discussion of heading change.

4.2 LINEAR REGRESSION

If the heading difference is 6 in either direction, which was the average for Pulse 64, then the Rover is deviating from its course by a meter. If this difference was in one direction for the entirety of the deployment then the rover would have deviated from its course by 100 meters. Given the topography of station M, which is relatively flat, this is not a huge problem. Also given the nature of the current scientific mission, it is not critical that the rover stay on its course. However with this simple heading correcting algorithm, would allow the Rover to follow it's course more reliably, which would help the engineering team and recovery crew have a better idea of where the Rover will end up, particularly over the course of longer deployments. Secondly, the Rover was built with the intent to be

5. CONCLUSIONS/RECOMMENDATIONS

This project has shown that with low computing cost a simple machine learning algorithm could be utilized by the Rover to improve the predictability of the path the Benthic AUV negotiates across the sea floor. While this is not a necessity of the Rover's current scientific mission, a better idea of where specific samples come from spatial could lead to a more clear picture of the spatial flux of carbon into the deep ocean, and the ecology of the benthic organisms utilizing it. While this is a small step forward in the development of a more complex navigation programming for the rover, similar techniques could be utilized by the engineering team to address other types of errors and impeded performance during deployments. Fault detection algorithms are currently being used with the pelagic AUVs at MBARI, and there are avenues for their successful implementation aboard the Rover.

ACKNOWLEDGEMENTS (Normal, Times New Roman, 12 pt, bold)

First I would like to take the opportunity to thank the family of Drew Gashler for establishing this scholarship. It has made all of this possible for me, and all of the interns that have come before me. It is a fitting tribute to the memory of Drew, allowing Moss Landing students to broaden their horizons, and strengthens the ties between MLML and MBARI. Secondly I'd like to thank George and Linda for all of their wonderful work making this internship the crown jewel that it is. None of this work would have been possible without their support and encouragement. It is also important for me to thank Richard, Paul, Ken and the rest of the Pelagic-Benthic lab for allowing me to indulge my curiosity with this project, and their help putting it all together. Richard gave me the latitude to pursue my own idea, and he and Paul were there for me the whole way with advice and support. Finally I would like to thank my fellow interns for making this summer unforgettable.

References: (Heading 1, Times New Roman, 12 pt, bold)

References (Normal, Times New Roman, 12 pt). Format should match the format that can be found online at http://www.mbari.org/news/publications/pr-pubs.html

Mcgill et al 2007

Appendix A Log Grinder

```
=begin
 ********************************************************************************
 * Copyright 1990-2015 MBARI
 * MBARI Proprietary Information. All rights reserved.
 ********************************************************************************
 * Summary  : What is contained within the file
 * Filename :
 * Author   :
 * Project  : Benthic Rover
 * Version  :
 * Created  :
 * Modified :
 ********************************************************************************
=end

class LogGrinder

  # Constants
  #
  TRANSIT_FILENAME   = "transits.csv"
  FAVORABLE_FILENAME = "favorables.csv"
  START_OF_TRANSIT   = " propulsion motors on"
  ACTUAL_TRANSIT     = " resuming from hibernating"
  START_OF_FAVORABLE = " FavorableCurrent behavior "
  HEADING_FILENAME   = "heading.csv"
  START_OF_HEADING   = " difference between current and desired heading"

  def initialize(syslog)
    @src_file = File.open(syslog)
    @src_line = nil
    @line_num = 0

    @move_file = nil
    @fc_file = nil
    @course_file = nil
    @start_time = nil
  end

  ##
  # Read each line of the syslog.
  # When a line contains the start of a section, pass control to a handler
  #
  def grind

    # Find the mission start time by looping over the first
```

```
  # lines in the log file until one with a timestamp is
  # found (probably the first one)
  #
  while (!@start_time)
    @start_time = timestamp(next_line())
  end

  if (@start_time)
    puts("Start time value is #{@start_time}")
  else
    puts("Hmm. You sure this is a barbo syslog?")
    return   # If no timestamps are found, we're done
  end

  while(next_line())

    if (@src_line.index(START_OF_TRANSIT))

      handle_transit()

    elsif (@src_line.index(START_OF_FAVORABLE))

      handle_favorable_current()

    elsif (@src_line.index(START_OF_HEADING))

      handle_heading()

    end
  end
end

def next_line()
  if (@src_line = @src_file.gets())
    @line_num += 1
  end
  @src_line
end

def handle_transit()
  puts("Got a move, lets get the time it started...")

  unless (@move_file)
    @move_file = File.new(TRANSIT_FILENAME, "w")
  end

  t_time = timestamp(@src_line) - @start_time
  @move_file.puts("#{t_time}: #{@src_line}")

end

def handle_favorable_current()
  @src_line = @src_file.gets()
  if (@src_line.index(ACTUAL_TRANSIT))
    puts("Nope, nothing to see here...")
    return
```

```
    else
      puts("Yup, we got a live one here...")
    end

    unless (@fc_file)
      @fc_file = File.new(FAVORABLE_FILENAME, "w")
    end

    f_time = timestamp(@src_line) - @start_time
    @fc_file.puts("#{f_time}: #{@src_line}")

  end


  def handle_heading()
    puts("found course correction")
    unless (@course_file)
      @course_file = File.new(HEADING_FILENAME, "w")
    end

    h_time = timestamp(@src_line) - @start_time
    @course_file.puts("#{h_time}: #{@src_line}")
  end


  # Given a line from a syslog, return a Float epoch seconds value
  # calculated from the ISO time stamp.
  # Returns 0.0 if no ISO timestamp is found in the line
  #
  def timestamp(line)
    value = 0.0

    # Match the ISO time stamp 2012-05-13T08:30:12
    #
    if (line.match(/[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}.[0-9]{2}/))
      dt = line.split("T")       # split the date segment from time segment
      dates = dt[0].split("-")   # split the date segment
      yr  = dates[0].to_i
      mon = dates[1].to_i
      day = dates[2].to_i
      times = dt[1].split(":")   # split the time segment
      hr  = times[0].to_i
      min = times[1].to_i
      sec = times[2].to_f
      t = Time.new(yr, mon, day, hr, min, sec)
      value = t.to_f             # + sec
#     puts(t)
#     puts(value)
    end
    value
  end

end

# THE ACTUAL START OF THE SCRIPT
#
```

```
# Default to using the syslog.csv in the local folder.
# Otherwise, use the first argument to the script
# E.g.,
#  Uses  local  syslog.csv ===> $ ruby log_grinder.rb
#  Uses another syslog.csv ===> $ ruby log_grinder.rb /rover/pulse_x/syslog.csv
#
syslog_file = "syslog.csv"
syslog_file = ARGV[0] if (ARGV[0])
puts("Preparing to grind #{syslog_file}...")
lg = LogGrinder.new(syslog_file)
lg.grind
```

Appendix B. Machine Learning

```
%% ROVER MACHINE LEARNING
r= -50 + (50+50)*rand(100,1);
E=ones(20,1)*5;
y = Magnitude(4:end);
m = length(y);% number of training examples
d=(1:1:m);
D=d';
X = [ones(m,1), D(:,1)];
 % Add a column of ones to x

theta = zeros(2, 1); % initialize fitting parameters
% Some gradient descent settings
iterations = 15000;
alpha = 0.001;

% compute and display initial cost
computeCost(X, y, theta)

theta = gradientDescent(X, y, theta, alpha, iterations);


% print theta to screen
fprintf('Theta found by gradient descent: ');
fprintf('%f %f \n', theta(1), theta(2));

predict1 = [1, 50] *theta;
fprintf('predicted heading difference after 50 moves %f\n',predict1);
% Plot the linear fit
scatter(D,y)
hold on; % keep previous plot visible
plot(X(:,2), X*theta, '-')
legend('Training data', 'Linear regression')
hold off % don't overlay any more plots on this figure


% Grid over which we will calculate J
theta0_vals = linspace(-10, 10, 100);
theta1_vals = linspace(-1, 4, 100);

% initialize J_vals to a matrix of 0's
J_vals = zeros(length(theta0_vals), length(theta1_vals));
```

```
% Fill out J_vals
for i = 1:length(theta0_vals)
    for j = 1:length(theta1_vals)
      t = [theta0_vals(i); theta1_vals(j)];
      J_vals(i,j) = computeCost(X, y, t);
    end
end


% Because of the way meshgrids work in the surf command, we need to
% transpose J_vals before calling surf, or else the axes will be
flipped
J_vals = J_vals';
% Surface plot
figure;
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0'); ylabel('\theta_1');

% Contour plot
figure;
% Plot J_vals as 15 contours spaced logarithmically between 0.01 and
100
contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 3, 20))
xlabel('\theta_0'); ylabel('\theta_1');
hold on;
plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);

figure;
plot(J_vals)
```

Appendix C. Cost Function

```
function J = computeCost(X, y, theta)
%COMPUTECOST Compute cost for linear regression
%   J = COMPUTECOST(X, y, theta) computes the cost of using theta as
the
%   parameter for linear regression to fit the data points in X and y

% Initialize some useful values
m = length(y); % number of training examples
J =0;
h = X * theta;
squaredErrors = (h - y) .^ 2;
J = (1 / (2 * m)) * sum(squaredErrors);

end
```

Appendix D. Gradient Descent

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha,
num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
%   theta = GRADIENTDESENT(X, y, theta, alpha, num_iters) updates theta
by
%   taking num_iters gradient steps with learning rate alpha
```

```
% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters

temp0 = theta(1) - alpha / m * sum(theta(1) + theta(2) * X(:, 2) - y);
temp1 = theta(2) - alpha / m * sum((theta(1) + theta(2) * X(:, 2) - y)
.* X(:, 2));
theta(1) = temp0;
theta(2) = temp1;


    % ============================================================

    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);

end

end
```