COMMON PITFALLS: HW 1, PART 6

1. About this document

As part of an ongoing research project to provide high quality autonomous feedback in online courses, we are making this list of common errors from Homework 1 available to all current students. This list was generated by automatically mining "exemplar submissions" from hundreds of thousands of submissions in the previous iteration of this course. If you have any questions, complaints or general feedback, please email codewebresearch@gmail.com. And stay tuned for the release of our interactive feedback tool in a future homework!

2. Common Errors

Error 1. (The extraneous sum)

This was similar to a common error from Part 3, in which the expression $(X \cdot \theta - y)' \cdot X$ should have been transposed with no sum.

```
function [theta, J_history] = gradientDescentMulti (X, y, theta, alpha, num_iters)
  m = length (y);
  J_history = zeros (num_iters, 1);
  for iter = 1:num_iters
    theta = theta - alpha / m * sum ((X * theta - y)' * X);
    J_history (iter) = computeCostMulti (X, y, theta);
  endfor;
```

Error 2. (Forgetting to divide by m)

```
function [theta, J_history] = gradientDescentMulti (X, y, theta, alpha, num_iters)
  m = length (y);
  J_history = zeros (num_iters, 1);
  for iter = 1:num_iters
    theta = theta - alpha * X' * (X * theta - y)
    J_history (iter) = computeCostMulti (X, y, theta);
  endfor;
```

Error 3. (Asynchronous updates for θ)

This is again very much like a common error from part 3, where, if the gradient terms for each i had been placed inside a temporary variable then were used to update θ simultaneously, then the implementation would have been correct.

```
function [theta, J_history] = gradientDescentMulti (X, y, theta, alpha, num_iters)
  m = length (y);
  J_history = zeros (num_iters, 1);
  for iter = 1:num_iters
    for i = 1:size (X, 2)
      theta (i) = theta (i) - alpha / m * sum ((X * theta - y) .* X (:, i))
    endfor;
    J_history (iter) = computeCostMulti (X, y, theta);
  endfor;
```

Error 4. (Hardcoding the dimension)

As with the previous two parts, a common mistake was to try to implicitly assume that X had a specific number of dimensions:

```
function [theta, J_history] = gradientDescentMulti (X, y, theta, alpha, num_iters)
  m = length (y);
  J_history = zeros (num_iters, 1);
  for iter = 1:num_iters
    temp1 = theta (1) - (alpha / m) * sum ((X * theta - y) .* X (:, 1));
    temp2 = theta (2) - (alpha / m) * sum ((X * theta - y) .* X (:, 2));
    temp3 = theta (3) - (alpha / m) * sum ((X * theta - y) .* X (:, 3));
    theta (1) = temp1;
    theta (2) = temp2;
    theta (3) = temp3;
    J_history (iter) = computeCostMulti (X, y, theta);
  endfor;
```