Coursera - Free Online Courses From Top Universities

Here are the ex1 test cases by Paul T. Mielke and Tom Mosher:

computeCost:

>>computeCost([1 2; 1 3; 1 4; 1 5], [7;6;5;4], [0.1;0.2])

ans = 11.9450

>>computeCost([1 2 3; 1 3 4; 1 4 5; 1 5 6], [7;6;5;4], [0.1;0.2;0.3])

ans = 7.0175

==========

gradientDescent:

Test Case 1:

```
>>[theta J_hist] = gradientDescent([1 5; 1 2; 1 4; 1 5],[1 6 4 2]',[0
0]',0.01,1000);

% then type in these variable names, to display the final results
>>theta
theta =
    5.2148
   -0.5733
>>J_hist(1)
ans  =  5.9794
>>J_hist(1000)
ans = 0.85426
```

For debugging, here are the first few theta values computed in the gradientDescent() for-loop for this test case:

```
% first iteration
theta =
   0.032500
   0.107500
% second iteration
theta =
   0.060375
```

```
   0.194887
% third iteration
theta =
   0.084476
   0.265867
% fourth iteration
theta =
   0.10550
   0.32346
```

The values can be inspected by adding the "keyboard" command within your for-loop. This exits the code to the debugger, where you can inspect the values. Use the "return" command to resume execution.

Test Case 2:
This test case is similar, but uses a non-zero initial theta value.

```
>> [theta J_hist] = gradientDescent([1 5; 1 2],[1 6]',[.5 .5]',0.1,10);
>> theta
theta =
   1.70986
   0.19229

>> J_hist
J_hist =
   5.8853
   5.7139
   5.5475
   5.3861
   5.2294
   5.0773
   4.9295
   4.7861
   4.6469
   4.5117
```

======

featureNormalize():

```
[Xn mu sigma] = featureNormalize([1 2 3])

% result

Xn =
```

```
   NaN   NaN   NaN
mu =
   1   2   3
sigma =
   0   0   0

% ---------------
[Xn mu sigma] = featureNormalize([1 ; 2 ; 3])

% result

Xn =
  -1
   0
   1

mu =  2
sigma =  1

%----------------
[Xn mu sigma] = featureNormalize(magic(3))

% result

Xn =
   1.13389  -1.00000   0.37796
  -0.75593   0.00000   0.75593
  -0.37796   1.00000  -1.13389

mu =
   5   5   5
sigma =
   2.6458   4.0000   2.6458

%--------------
[Xn mu sigma] = featureNormalize([-ones(1,3); magic(3)])

% results

Xn =
  -1.21725  -1.01472  -1.21725
   1.21725  -0.56373   0.67625
  -0.13525   0.33824   0.94675
   0.13525   1.24022  -0.40575
```

```
mu =
   3.5000   3.5000   3.5000

sigma =
   3.6968   4.4347   3.6968
```

computeCostMulti and gradientDescentMulti:

```
>> X = [ 2 1 3; 7 1 9; 1 8 1; 3 7 4 ];
>> computeCostMulti( X, [ 2; 5; 5; 6 ], [ 0.4;  0.8;  0.8 ] )
ans =  7.5500

>>gradientDescentMulti([3 5 6; 1 2 3; 9 4 2],[1 6 4]',[0 0 0]',0.01,1000)
ans =
   1.2123
  -2.9458
   2.3219
```

46 Upvote · Follow 19 · Reply to Chirag Uttamsingh