Kin-Ho Lam
4/10/18
CS 434

Assignment 1

Instructions

Assignment has been tested and working on vm-cs434-1.engr.oregonstate.edu

1. In the same directory that `main.py` is located, create a folder named data
2. Place the following files inside data
   - housing_desc.txt
   - housing_train.txt
   - housing_test.txt
   - usps-4-9-test.csv
   - usps-4-9-train.csv
3. Execute `main.py` with the following python command
   - `python main.py`
4. Correct execution should produce the following .csv files
   - ase.csv
   - gradient_descent.csv
   - L2_regularization.csv

1.1 Dependencies

- python 2.7.5
- numpy 1.13.0

1 Linear Regression

1.

Weight vector with dummy column

```
Weight Vector with Dummy Column
[ 39.58432122  -0.10113705   0.04589353  -0.00273039   3.0720134
 -17.22540718   3.71125235   0.00715862  -1.5990021    0.37362337
  -0.01575642  -1.02417703   0.00969321  -0.58596927]
```

2.

Training and Testing ASE with dummy column

```
Training ASE:    22.081273187
Testing ASE:     22.6382562966
```

3.

Weight vector without dummy column:

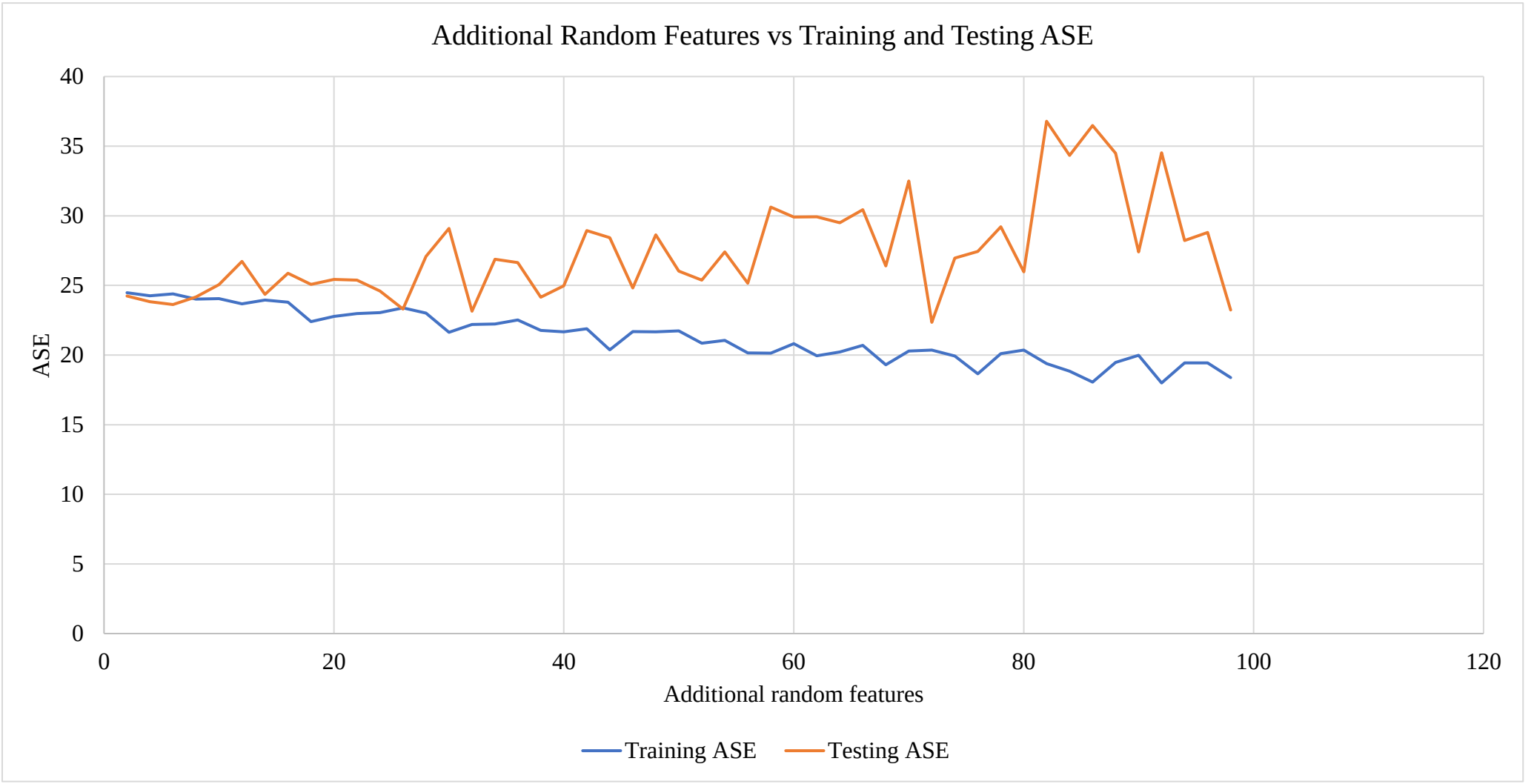
```
Weight Vector without Dummy Column
[-0.09793424  0.04895868 -0.02539285  3.45087927 -0.35545893  5.81653272
 -0.00331448 -1.02050134  0.22656321 -0.01224588 -0.38802988  0.0170215
 -0.48501296]
```

Training and Testing SSE without dummy column:

```
Training ASE:   24.4758827846
Testing ASE:    24.2922381757
```

The training and testing SSE grew slightly larger when removing the dummy variable. This means that the resulting model is less accurate.

4.

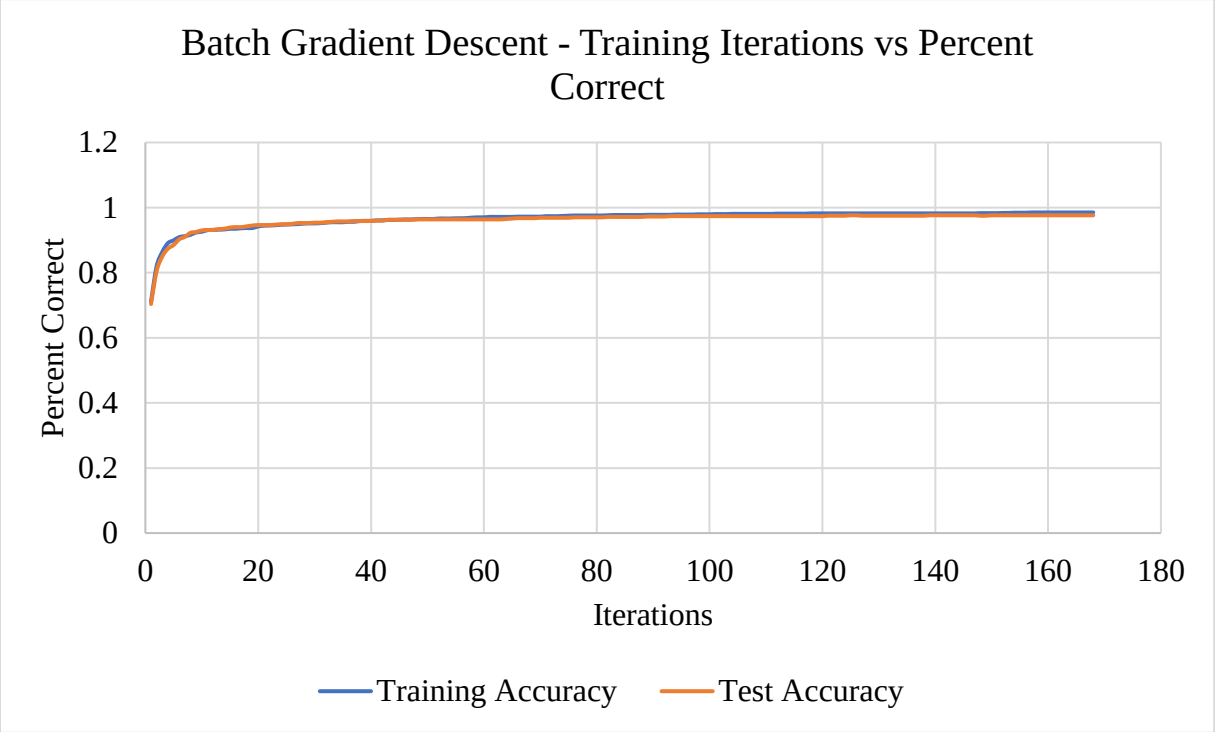


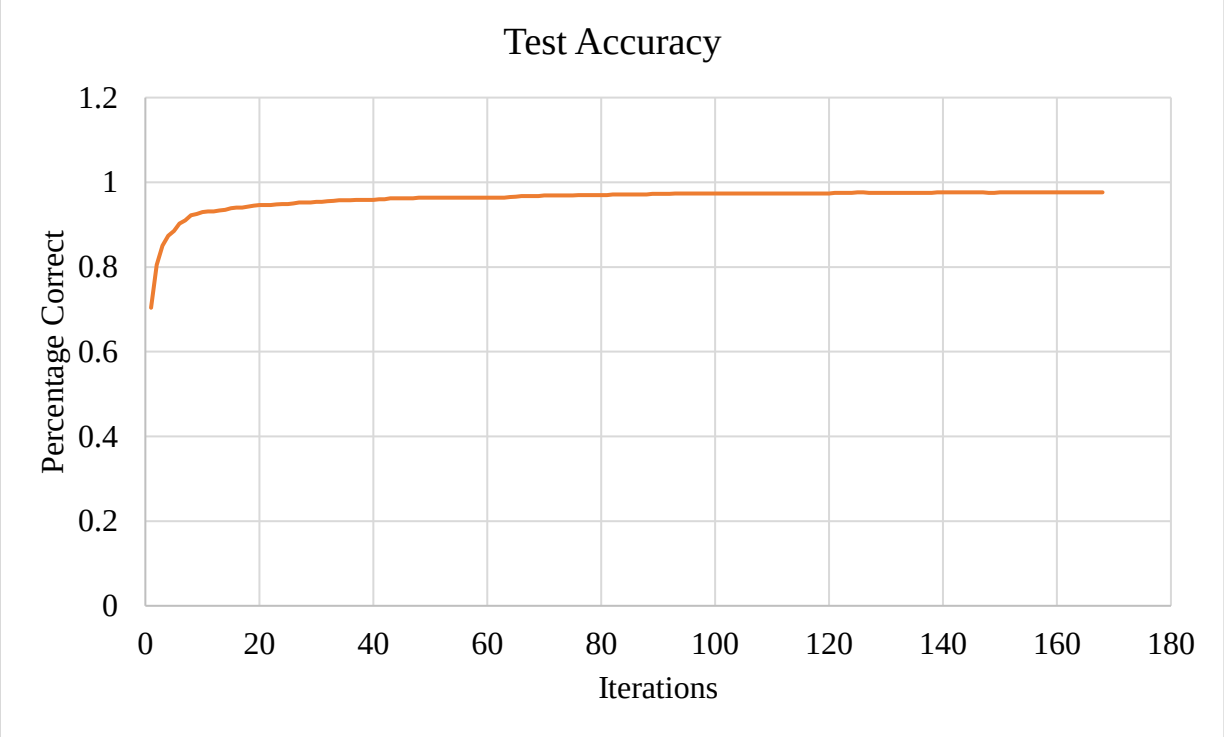
Increasing the number of random features marginally decreased training ASE and marginally increased testing ASE. From these results, one can conclude that adding more random uncorrelated features does not improve the performance.

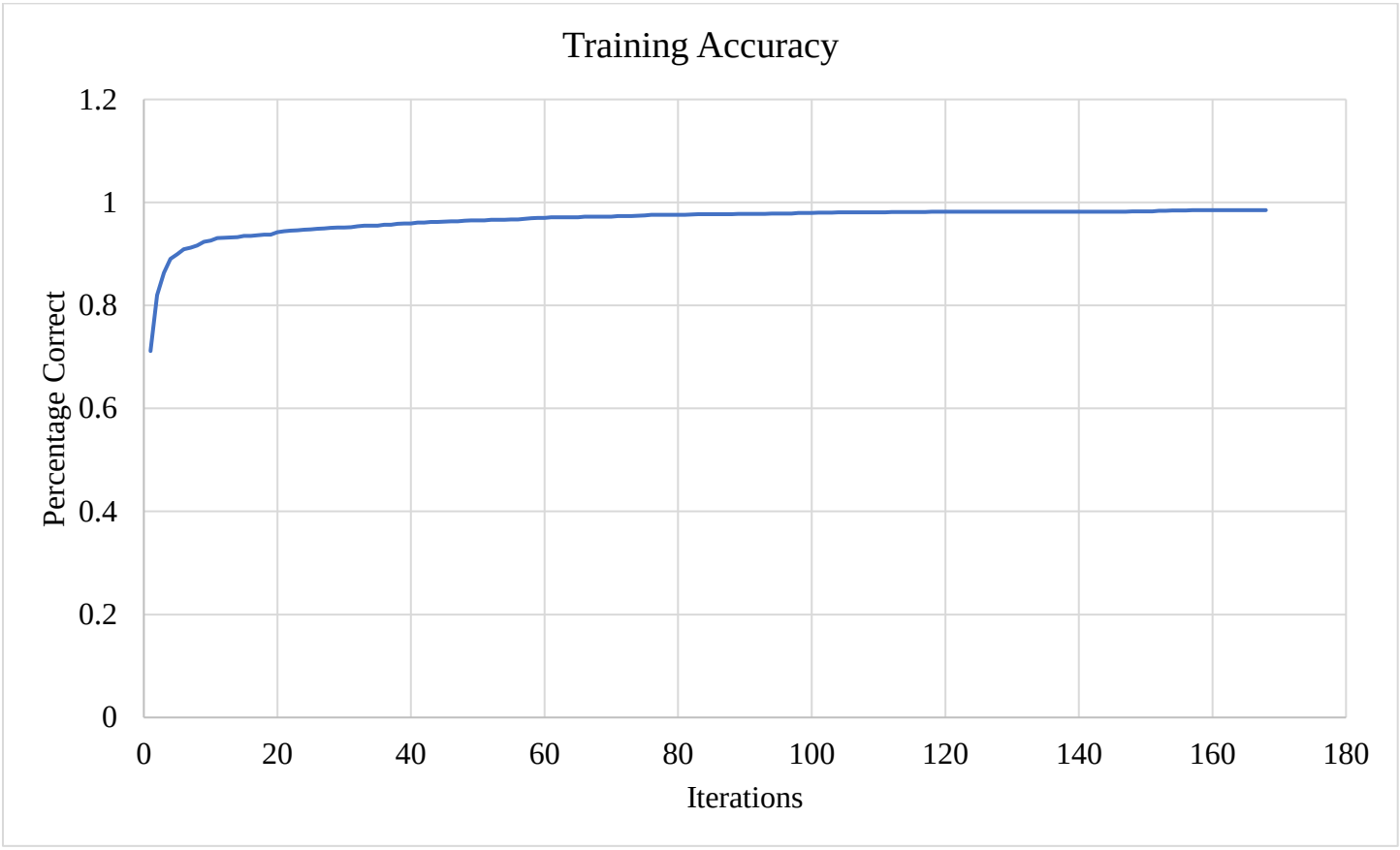
2 Logistic regression with regularization

1.

The following graphs depict my batch gradient descent implementation, plotting training iterations vs percentage of predictions correct. These values were measured over 169 total training iterations with a learing rate of 0.0001. Initially, both training and testing initlize to a low of about 70% correct. The accuracy of both reach over 90% by the 6th iteration. The model peaks at 98.5% accuracy in training data at about 157 iterations. Testing accuracy peaks at 97.6% at about 147 iterations.






Training Accuracy
Percentage Correct
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
160
180
Iterations

2.

Gradient of $\frac{d}{dw}$ $L(w) = \lambda * ||w||$

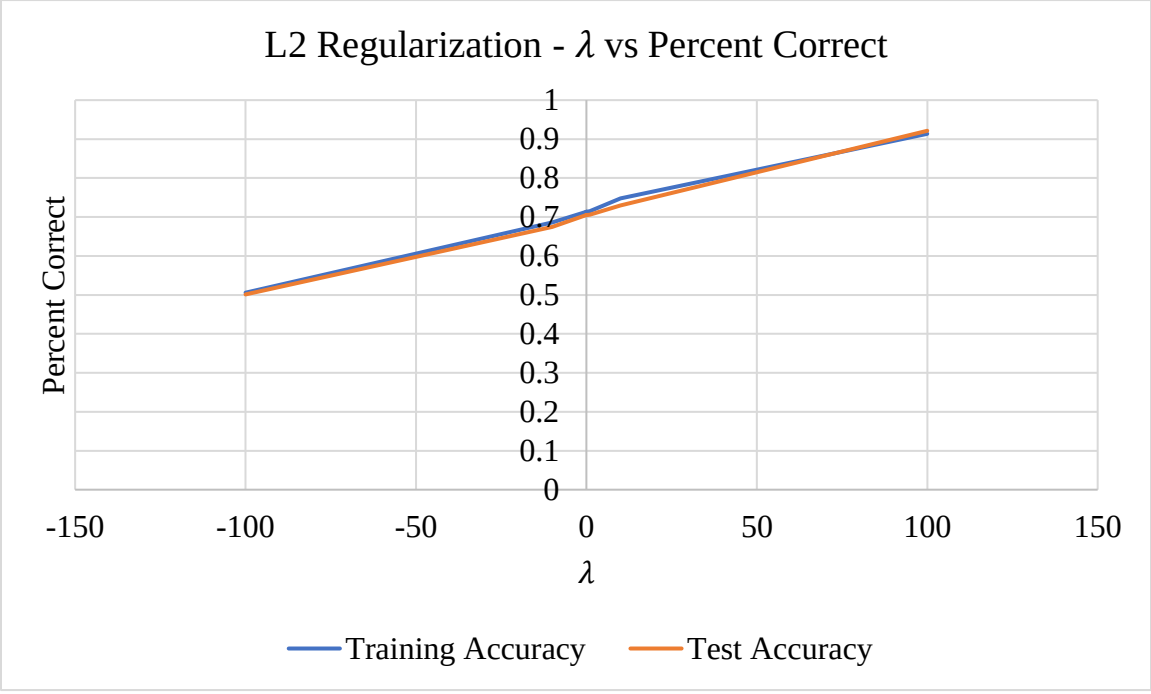
Pseudocode for L2 Regularization

```
Training Features  x_n = [x_1, x_2, ... x_n]
Training Outputs  y_n = [y_1, y_2 ... y_n]
Let  w ← [0, ... ,0]
Repeat:
    ∇ ← [0, ... ,0]
    for  i = 1, ... , n :
        ŷ = 1/(1+e^(−w^T x_i)) + λ||w_i||^2
        error = y_i − ŷ
        ∇ = ∇ + (error * x_i)
w = w + (η * ∇)
```

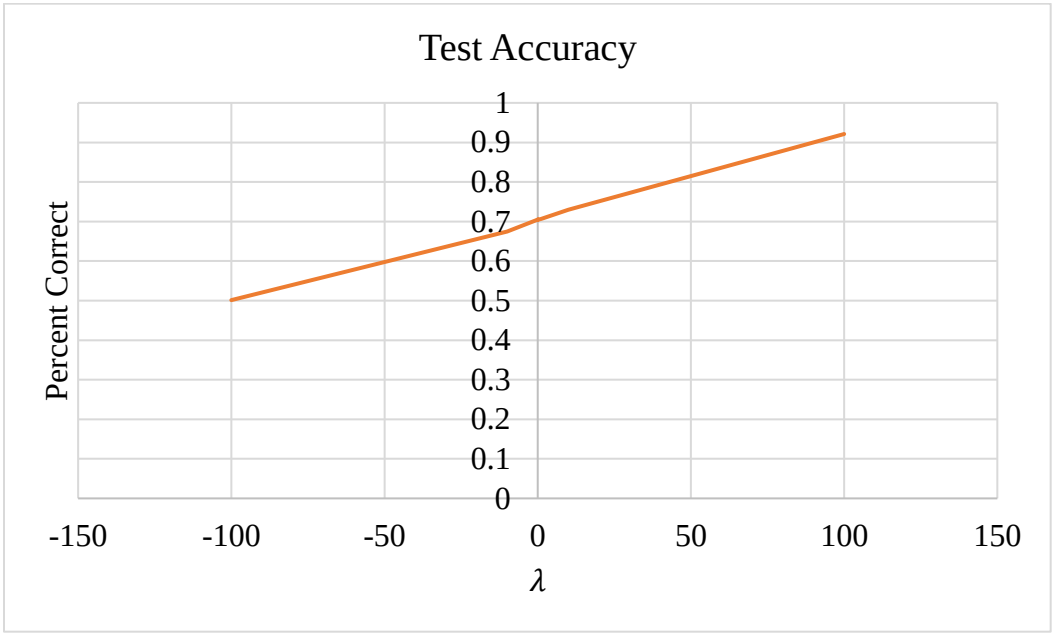
3.

The following graphs depict my batch gradient descent L2 regression implementation, λ values vs percentage of predictions correct. Each lamda value took 2 total training iterations with a learing rate of 0.000001. Python's floating-point limitation prevented my trials from increasing total training iterations or the learning rate because the numpy exp() function would overflow.

This data shows that as λ increases to a larger positive value, testing and training prediction accuracy increases.




Training Accuracy
Percent Correct
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-150
-100
-50
0
50
100
150
λ


Test Accuracy
Percent Correct
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-150
-100
-50
0
50
100
150
λ