- For our W matrix, we populate the entries of W such that $W_{i,j} = 0$ when $i \neq j$ and $W_{i,j} = w^{(i)}$ when $i = j$. With this, we can analyze $(X\theta - y)^T W(X\theta - y)$.
  First, let $E = (X\theta - y)$. We see that E is an m x 1 vector, where $E_{i,1} = (x^{(i)})^T\theta - y^{(i)}$. Because $x^{(i)})^T\theta$ is a scalar, we can rearrange it to get that $E_{i,1} = \theta^T x^{(i)} - y^{(i)}$. This means that:
  $$(X\theta - y)^T W(X\theta - y) = E^T WE = B^T E$$
  where B is an m x 1 vector with entries $B_{i,1} = (\theta^T x^{(i)} - y^{(i)})w^i$. B^T has dimensions 1 x m and E has dimensions m x 1, so BE is a dot product, which gives us the scalar of $\sum_{i=1}^{m}((\theta^T x^{(i)} - y^{(i)})w(i))(\theta^T x^{(i)} - y^{(i)})$. This means that:
  $$(X\theta - y)^T W(X\theta - y) = \sum_{i=1}^{m} w^i(\theta^T x^{(i)} - y^{(i)})^2 = J(\theta)$$
- $$J(\theta) = (X\theta - y)^T W(X\theta - y)$$
  $$\nabla_\theta J(\theta) = \frac{\partial}{\partial\theta}(\theta^T X^T WX\theta + y^T Wy - \theta^T X^T Wy - y^T WX\theta)$$
  $$\nabla_\theta J(\theta) = 2(X^T WX)\theta - X^T Wy - X^T W^T y$$
  $$\nabla_\theta J(\theta) = 0$$
  $$2(X^T WX)\theta - X^T Wy - X^T W^T y = 2(X^T WX)\theta - X^T Wy - X^T Wy = 0$$
  $$2(X^T WX)\theta = 2X^T Wy$$
  $$\theta = (X^T WX)^{-1} X^T Wy$$

- Below is the algorithm for batch gradient descent

Algorithm 1: Locally Weighted Linear Regression by Batch Gradient Descent

Input: X, an m x d matrix of training examples; y, an m x 1 vector of the outputs for each example; x, input vector for weighting; n, the number of iterations; α, step size for gradient descent; τ, bandwidth defining sphere of influence around x

Output: θ, parameter vector that minimizes our cost function

for $i \leftarrow 1$ **to** d **do**
  $\theta_{i,1} \leftarrow 0$;
for $i \leftarrow 1$ **to** d **do**
  for $j \leftarrow 1$ **to** d **do**
    if $i = j$ **then**
      $W_{i,j} \leftarrow \frac{\exp((x-x^{(i)})^T(x-x^{(i)}))}{2\tau^2}$;
    else
      $W_{i,j} \leftarrow 0$;
for $i \leftarrow 1$ **to** n **do**
  $\theta \leftarrow \theta - \frac{\alpha}{m} X^T W(X\theta - y)$
return θ

- We note two things. First, we know that:

$$\theta = (X^T X)^{-1} X^T y$$

Second, we know that:

$$y = X\theta^* + \epsilon, \epsilon \ N(0, \sigma^2)$$

With this information, we can calculate $E[\theta]$ as follows:

$$E[\theta] = E[(X^T X)^{-1} X^T y]$$

$$E[\theta] = E[(X^T X)^{-1} X^T (X\theta^* + \epsilon)]$$
$$E[\theta] = E[(X^T X)^{-1} X^T X\theta^*] + E[(X^T X)^{-1} X^T \epsilon]$$
$$E[\theta] = (X^T X)^{-1} X^T X E[\theta^*] + (X^T X)^{-1} X^T E[\epsilon]$$
$$E[\theta] = E[\theta^*] + 0$$
$$E[\theta] = \theta^*$$

- We use our $E[\theta]$ to help calculate $Var(\theta)$:

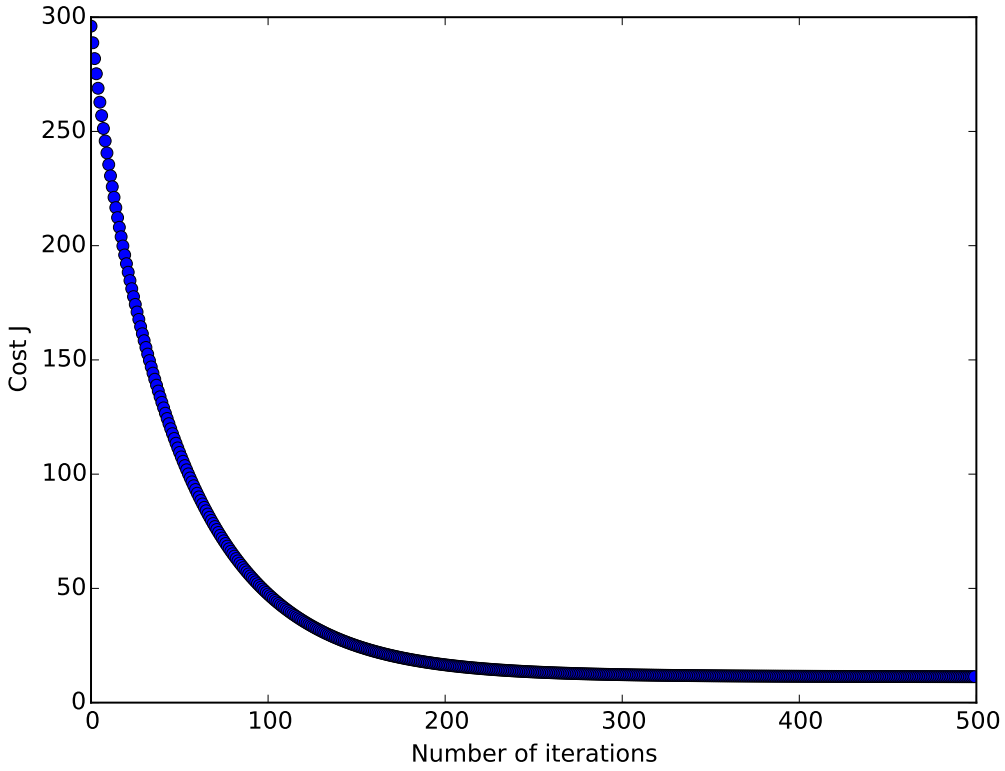
$$Var(\theta) = E[(\theta - E[\theta])^2] = E[(\theta - \theta^*)(\theta - \theta^*)^T]$$

$$Var(\theta) = E[((X^T X)^{-1} X^T \epsilon)((X^T X)^{-1} X^T \epsilon)^T]$$
$$Var(\theta) = E[(X^T X)^{-1} X^T \epsilon\epsilon X (X^T X)^{-1}]$$
$$Var(\theta) = \sigma^2 E[(X^T X)^{-1} X^T X (X^T X)^{-1}]$$
$$Var(\theta) = \sigma^2 (X^T X)^{-1}$$

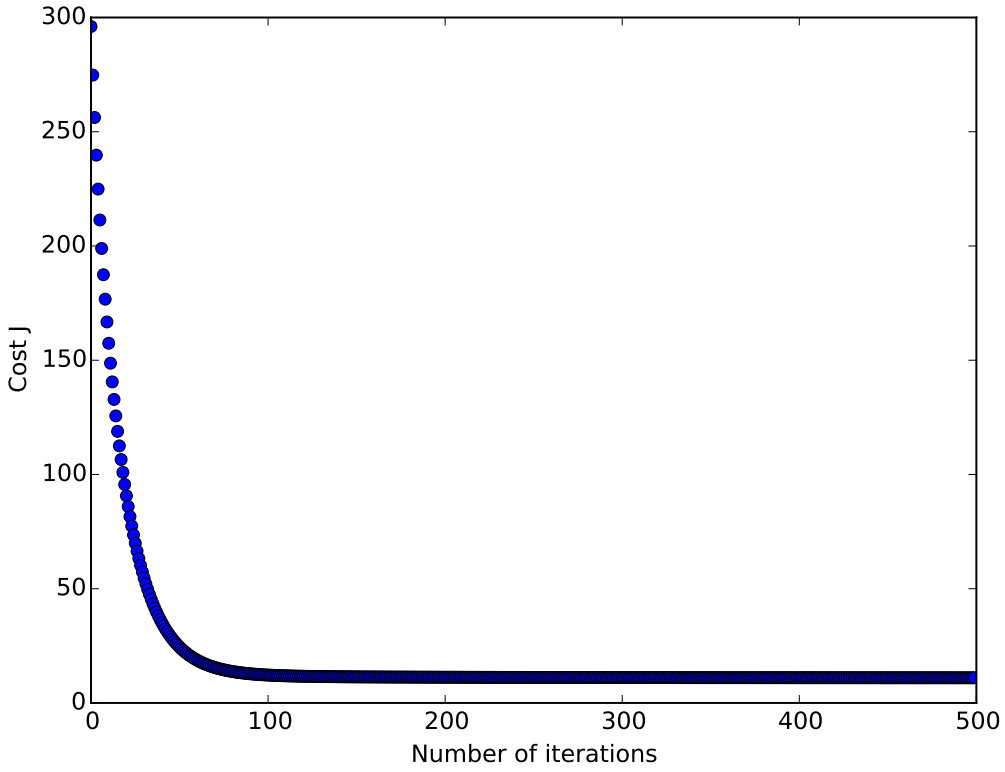
- 3.1.A3
  For lower status percentage = 5, we predict a median home value of 29.8034494122. For lower status percentage = 50, we predict a median home value of -12.9482128898

- 3.1.B5
  The learning curves for the step sizes of $\alpha = .01, .03, .1, .3, 1, 3$l are given below. From the plots, it looks like $\alpha = .3$ is the best stepsize, because the learning occurs quicker for $\alpha = .3$ than any value smaller than .3. Furthermore, step sizes of 1 and 3 cause the model to blow up, because the minimum cost is never truly reached. Furthermore, it looks like our 500 iterations is sufficient to reach a minimum, and we could even push it down to 400 or 300 iterations with most α values. However, with $\alpha = .3$, 100 or even 50 iterations is more than enough.

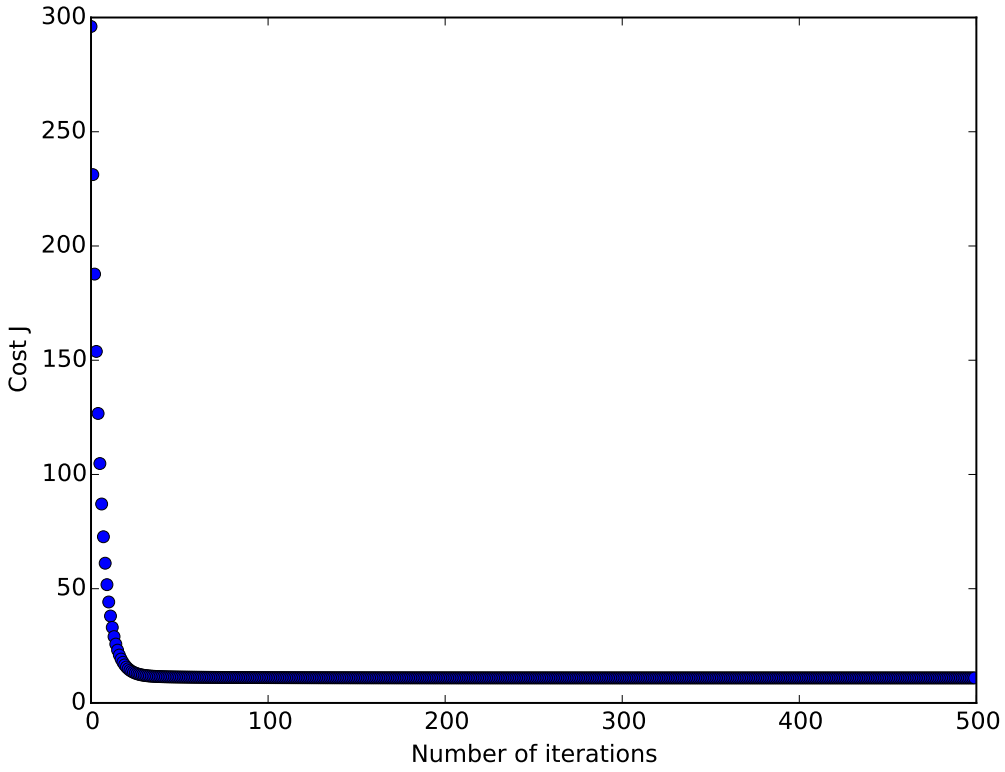
  Learning curve with $\alpha = .01$



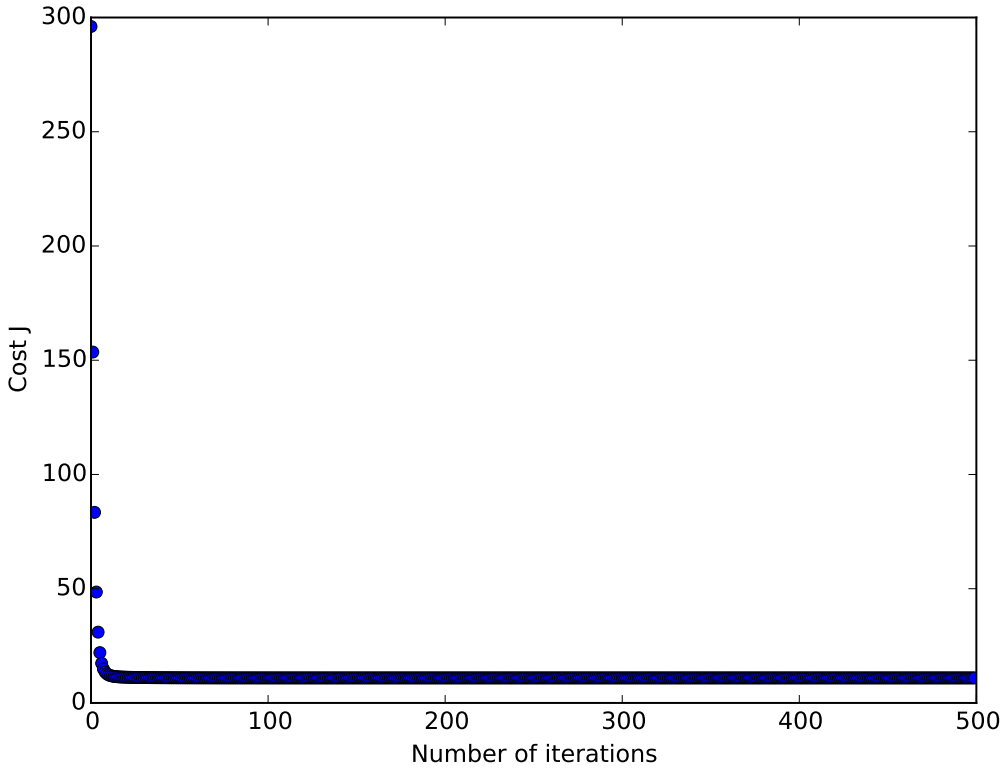
Learning curve with $\alpha = .03$



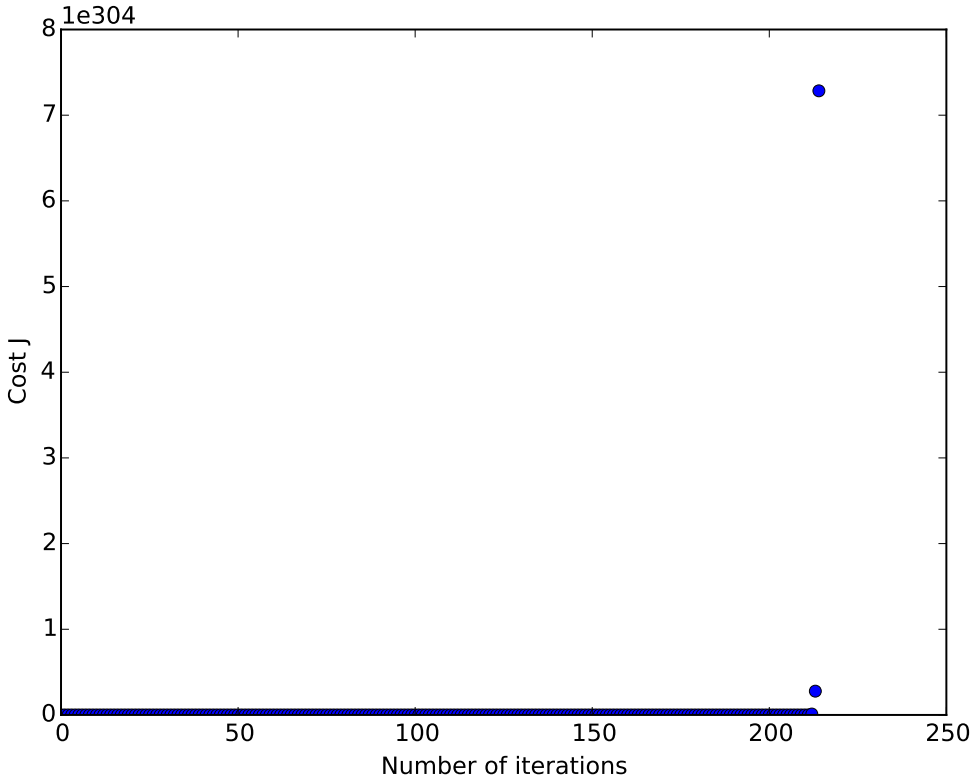
Learning curve with $\alpha = .1$



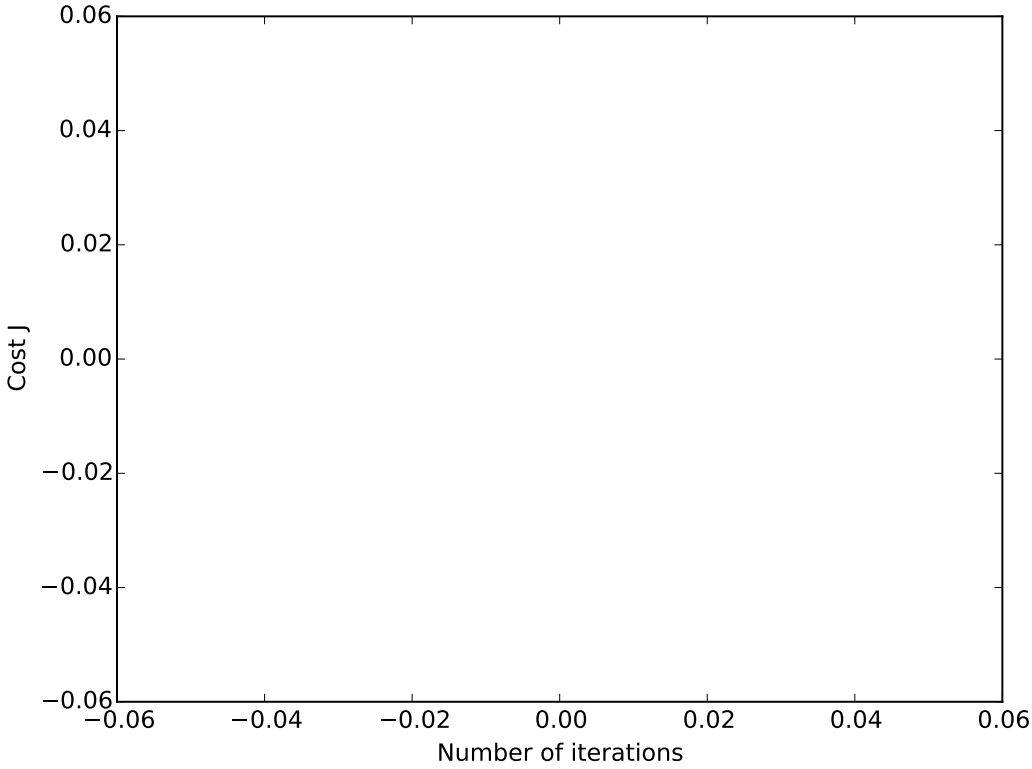
Learning curve with $\alpha = .3$



Learning curve with $\alpha = 1$

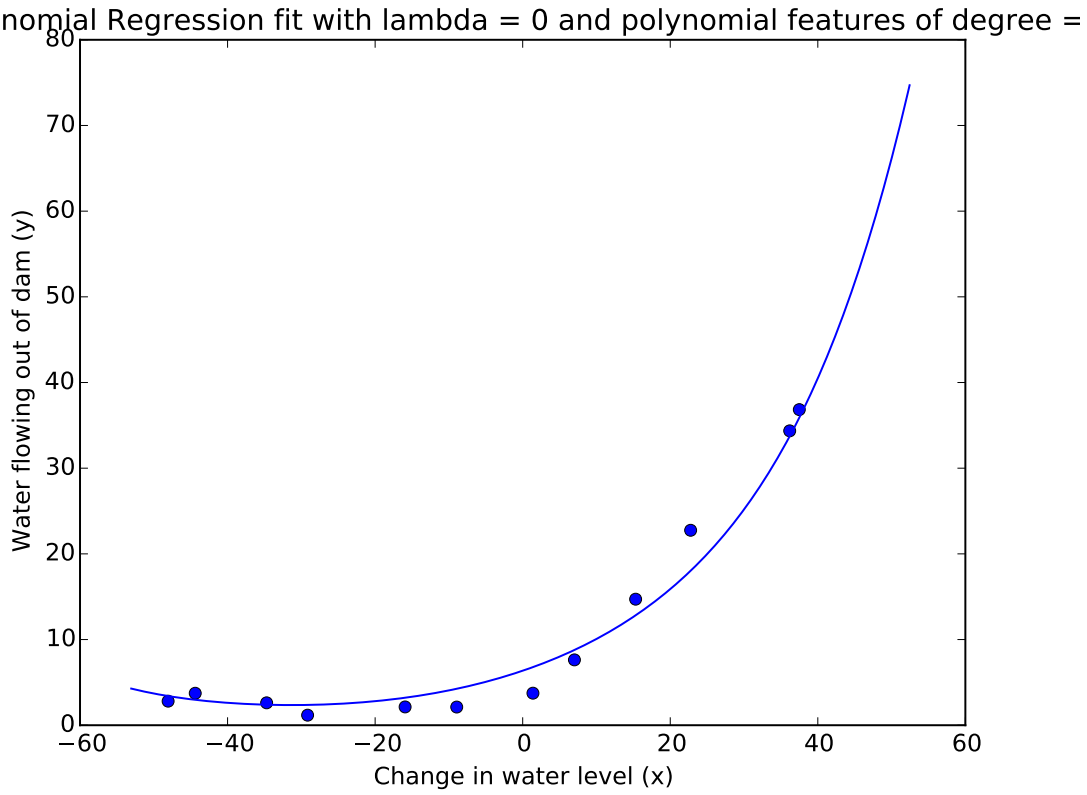


$\alpha = .3$

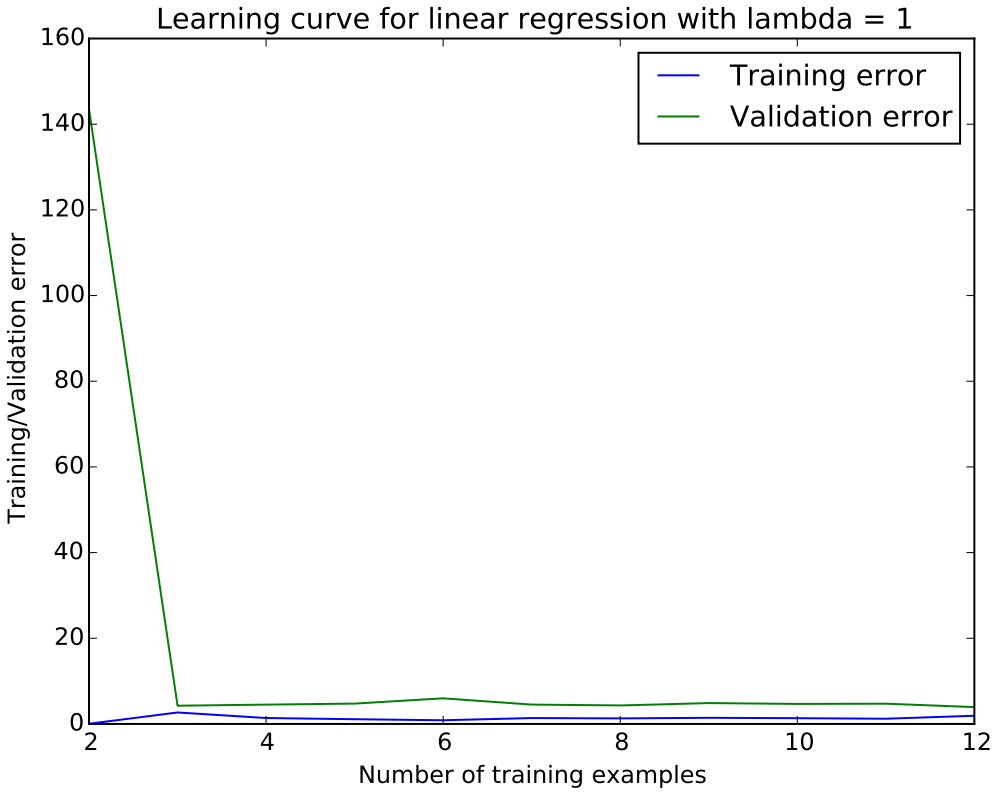

0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
Cost J
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
Number of iterations

- 3.2.A4
  The fit and learning curves for $\lambda = 1, 10, 100$ are posted below. The fit and learning curves show us that $\lambda = 1$ is the best λ value for regularization, because λ values of 10 and 100 cause the model to underfit. This makes sense because too much regularization causes us to underfit and increase bias in our model.

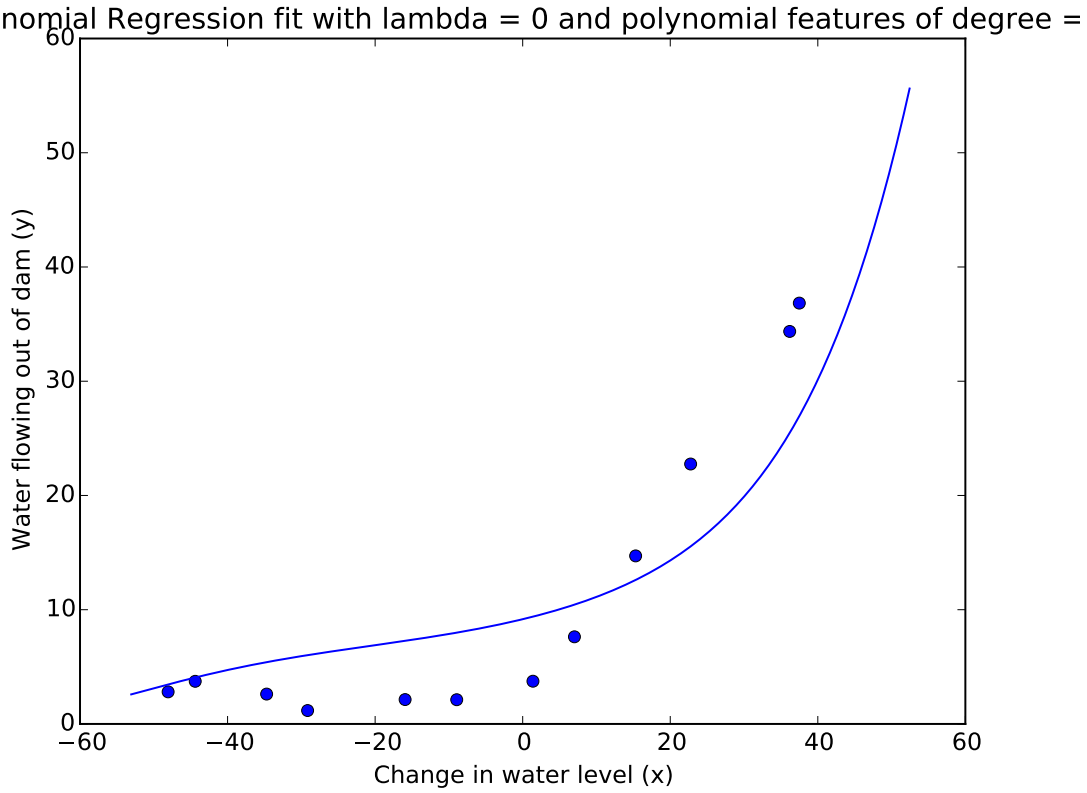
  Polynomial Regression fit with Regularization $\lambda = 1$



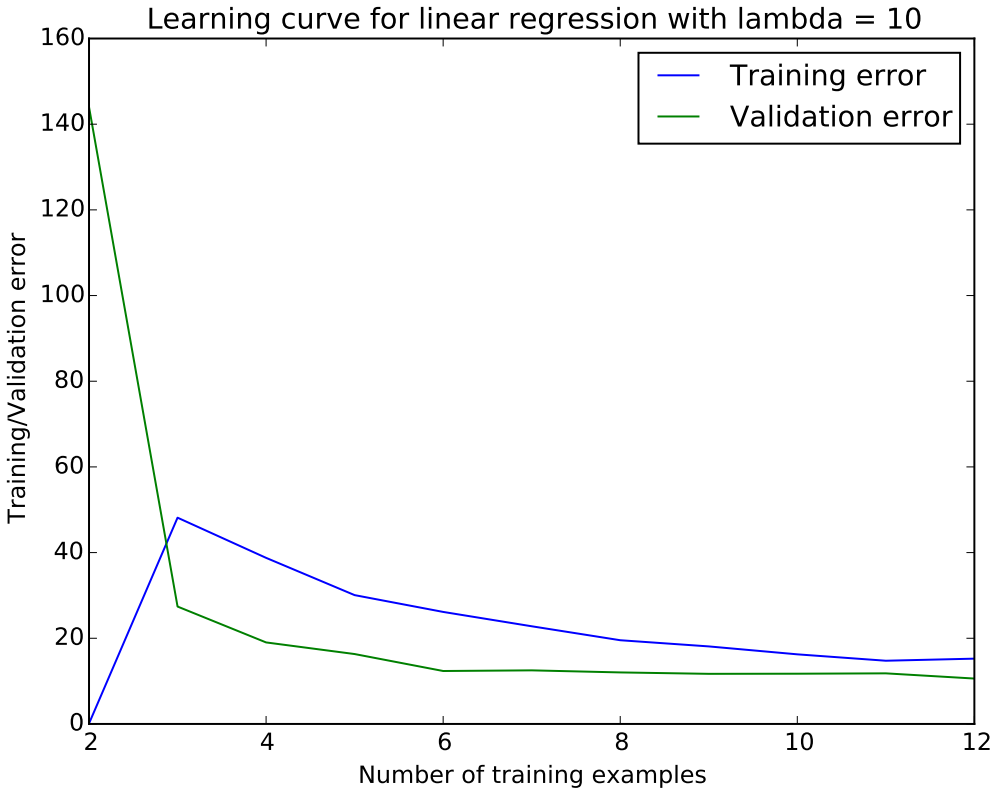
Learning Curve for Polynomial Regression fit with Regularization $\lambda = 1$



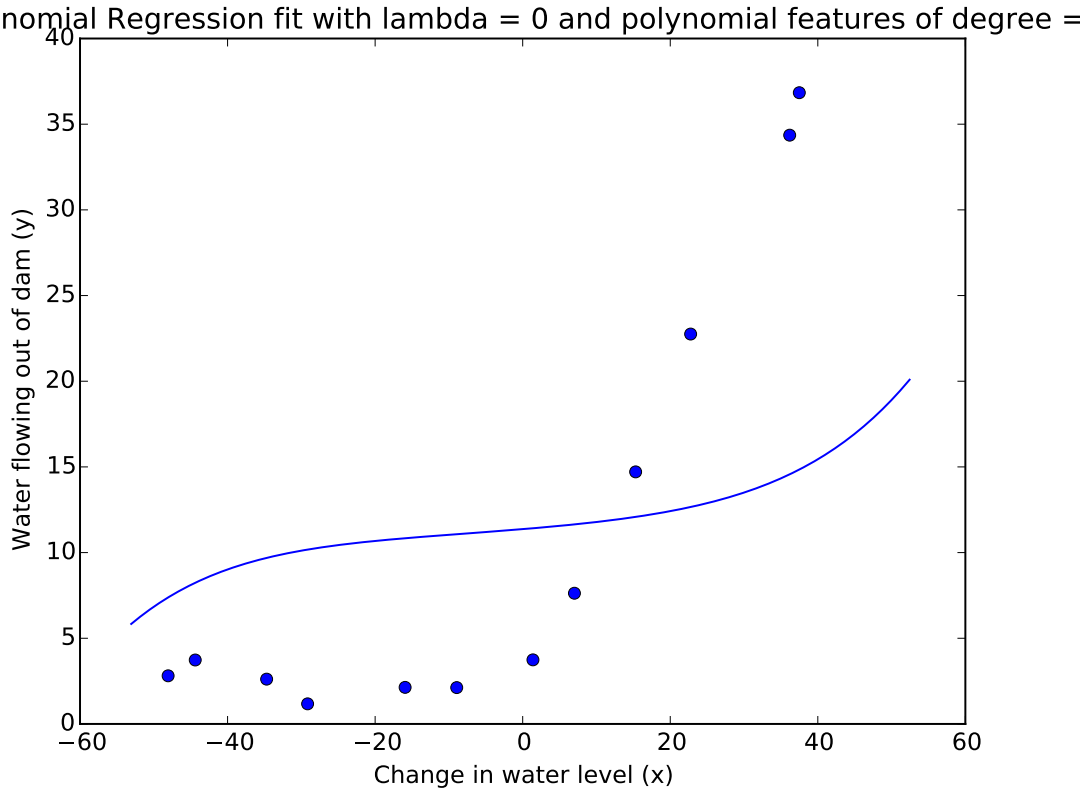
Polynomial Regression fit with Regularization $\lambda = 10$



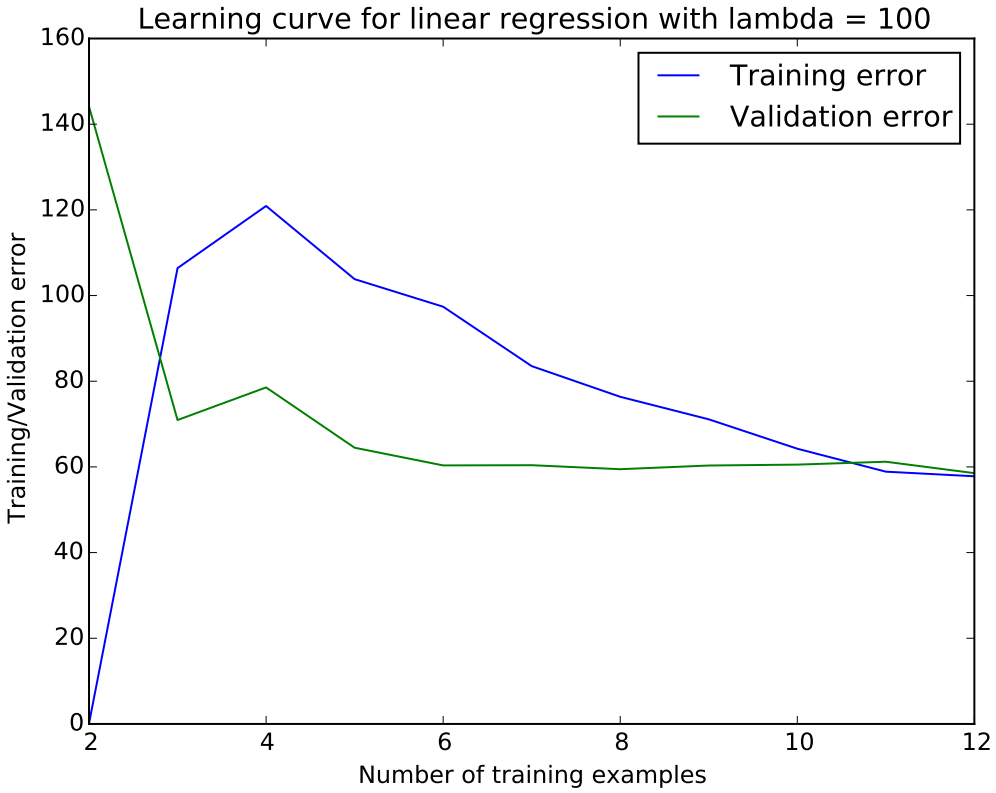
Learning Curve for Polynomial Regression fit with Regularization $\lambda = 10$



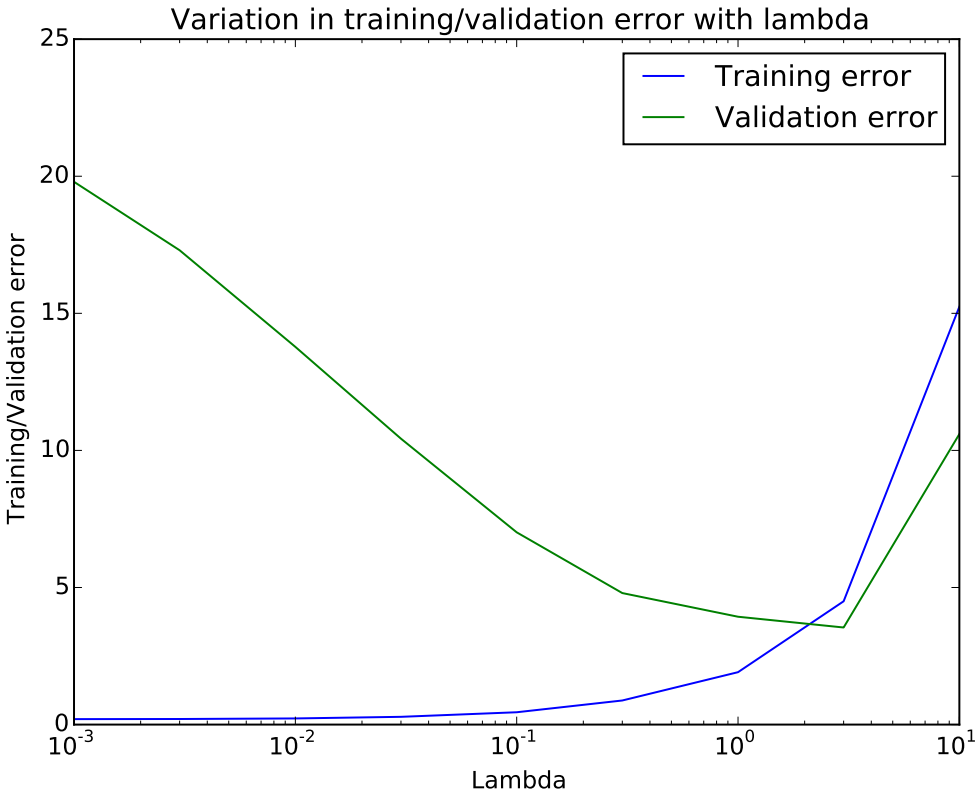
Polynomial Regression fit with Regularization $\lambda = 100$



Learning Curve for Polynomial Regression fit with Regularization $\lambda = 100$



- 3.2.A5
  Below is our Validation Curve.
  Note: While our solution gives us a λ of 3 for minimizing the validation error, we recognize the $\lambda_{min} = .3$ in the validation curve sanity check posted on the homework. However, we went to Devika's office, and we found out that our measurements are exactly the same except for, peculiarly enough, $\lambda = .3, 1, 3, 10$. We think the strange error may be due to differences between scipy 1.16 and 1.17, and for that reason, we're reporting our test set errors for $\lambda = .3, 1, 3$ ($\lambda = 10$ is clearly too large of a regularization parameter for our model).

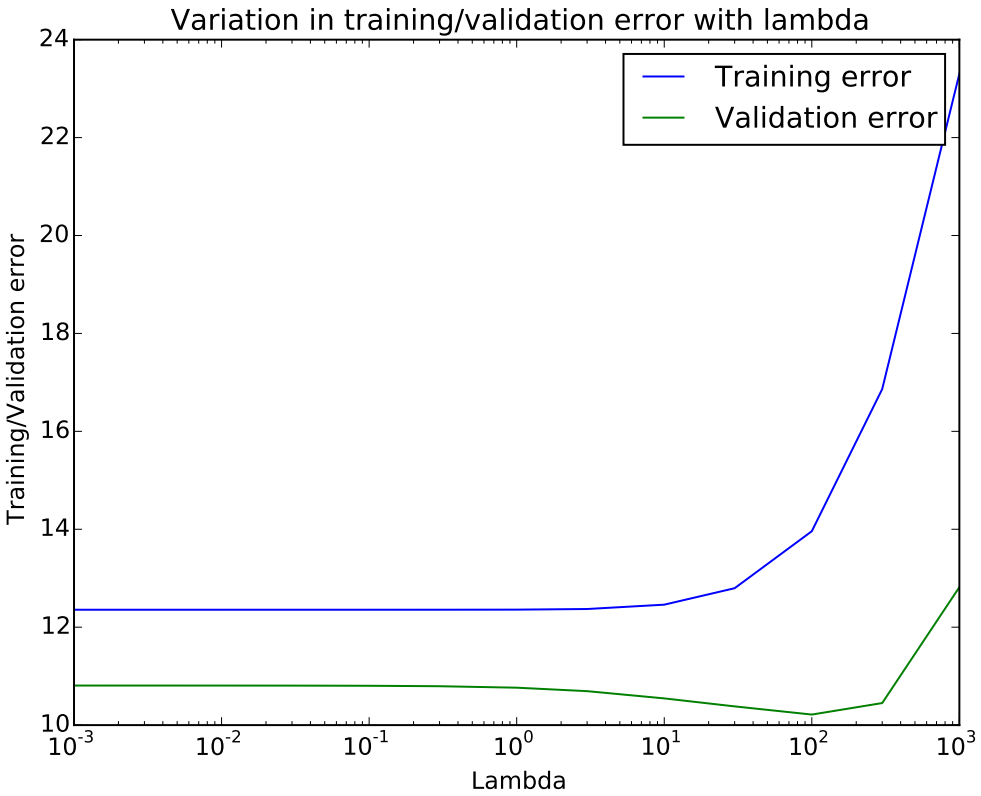


- 3.2.A6
  Here is the test error for $\lambda = .3, 1, 3$
  Test Error: 3.09877918082, Lambda = 1
  Test Error: 4.39761235717, Lambda = 3.0
  Test Error: 4.68411270916, Lambda = 0.3
  Interestingly enough, the errors reported when $\lambda = 1, 3$ beat the final solution!

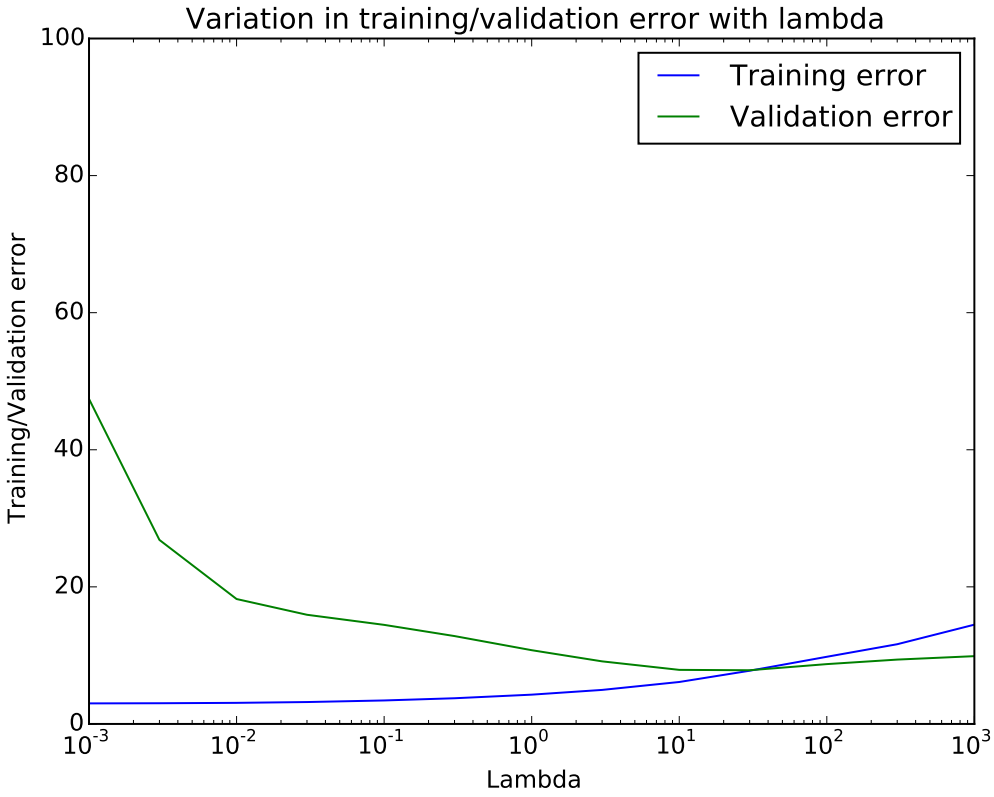
- 3.3
  Below, we have the best achievable error when $\lambda = 0$, and then the best λ values for models linear, quadratic and cubic features. Below these numbers, we have the validation curves for the models with linear, quadratic, and cubic features. We can see that the model that does the best has quadratic features and a regularization parameter of $\lambda = 30$. More generally, we can see that regularization is pretty critical for building better models, especially when we consider that we almost halved our error with our best model.

  Degree = 1, lambda = 0, error = 12.3565290896 (This isn't a best lambda, it's a baseline lambda that's asked for)
  Degree = 1, lambda = 100, error = 10.2153815124
  Degree = 2, lambda = 30, error = 7.83507429739
  Degree = 3, lambda = 1000, error = 8.14447624869

  Validation Curve for Model with Linear features



Validation Curve for Model with Quadratic features



Validation Curve for Model with Cubic features