6.867: HOMEWORK 1

1. Gradient Descent

1.1. **Simple Gradient Descent.** We implement gradient descent in Python and vary starting guess, step size, and convergence criterion for multidimensional negative Gaussian $g(x)$ and quadratic bowl $q(x)$ functions

$$g(x) = -\frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp\left[-\frac{1}{2}(x-u)^\top \Sigma^{-1}(x-u)\right], \quad q(x) = \frac{1}{2}x^\top Ax - x^\top b$$

with known parameters. We observe several interesting phenomena. The norm of the Gaussian gradient is small in two regimes: near the mean and also far from it in the "tail plateau" at initial steps (Figure 1a) which indicates initial guesses may converge without attaining the actual critical value if they are sufficiently far from the mean (Figure 1c). In particular, Figure 1d demonstrates the two convergence regimes, where the upper left blue group is actually converging to the mean but the upper right blue group converges in the outer plateau. There are convergence criteria sufficiently small for other given parameters such that convergence is not obtained for any guess distance, as indicated by the red regime near the bottom. None of these factors influence the quadratic bowl since it does not have any inflection points or plateau regions far from the optimum. We sweep through a variety of step sizes, guesses, and convergence criteria to exhibit general behavior for both functions (for example, Figures 2a, 2b, 2d). Note that it suffices to sample parameters to observe general trends since the functions are relatively simple.

1.2. **Central Difference Approximation.** To approximate the gradient of a function numerically, we implement the central difference approximation and test the closed form gradients for the previous functions. We plot a sweep of difference steps at various locations (Figure 3a) to demonstrate accuracy and hold location constant (Figure 3b) to enhance the visibility of the accuracy for different difference steps. We conclude that difference steps on the order of 10^{-12} to 10^{-10} are sufficient for the purposes of this paper.

1.3. **Batch Versus Stochastic Gradient Descent.** We implement batch gradient descent (BGD) and stochastic gradient descent (SGD) for future use, comparing the behavior of the two implementations (Figures 4a, 4b) to conclude stochastic gradient descent approaches batch gradient descent in accuracy when step size is below 10^{-7}. Because SGD frequently fails to converge with small step sizes, if we desire speed on large data sets we use SGD anyways. Batch gradient descent is useful for small step sizes, though it tends to exhibit oscillatory behavior in terms of iterations required for small step sizes. Note that the learning rate may be chosen to guarantee convergence for SGD[1] though we choose fixed step size for simplicity here, since it illustrates the results above.

2. Linear Basis Function Regression

2.1. **Polynomial Fitting.** We implement linear regression in Python and test by reproducing known results[2] (Figure 5).

2.2. **Sum of Squares Error.** We proceed to compute the sum of squares error (SSE) and its derivative for a data set and a hypothesis, verifying the gradient using the numerical derivative code. We find 0.001 tolerance for $M = 8$ at difference step 10^{-6} and initial guess at $\vec{0}$ (Table 0a).

2.3. **Gradient Descent.** We perform BGD and SGD on $M = 1, 3, 4, 6, 7, 10$. We find larger step size to improve accuracy of BGD in all cases (Figure 6a) save for $M = 10$ which overfitted the data. Stochastic descent failed to converge in most cases but outperformed batch in the $M = 1$ case, and we improve upon the error by increasing the max steps for longer running times (6b). We find that when the threshold for minimum gradient approaches values larger than 10^{-1}, both methods break down. We observe quick convergence when initial guess is close to the true values (Figure 6c).

[1] https://en.wikipedia.org/wiki/Stochastic_approximation#Robbins.E2.80.93Monro_algorithm

[2] HW1 Handout

2.4. **A Sinusoidal Basis.** Using a $\cos(M\pi x)$ basis to perform maximum likelihood weight vector calculation generates weight vectors which are not very representative of the true values $w_1 = 1, w_2 = 1.5$ (Table 0b). We examine regularization to include prior sparsity belief on regression coefficients in section 4.

3. Ridge Regression

A regularization term can be added to the error function in order to control overfitting. Regularization involves adding a penalty term to the error function in order to discourage coefficients from reaching large values. The coefficient λ controls the relative importance of the regularization term compared with the data-dependent error term.

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}). \tag{3.1}$$

A common form of the regularizer is the sum-of-squares of the weight vector elements, also known as the L_2 norm.

$$E_W(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{3.2}$$

Using the sum-of-squares error function and this regularization term, the total error function becomes:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\Phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{3.3}$$

3.1. **Experimenting With Lambda and M.** Let's see how ridge regression performs on our previous data, curvefittingp2.txt. The closed form solution is easily calculated to be:

$$\mathbf{w} = (\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{t}. \tag{3.4}$$

The hyperparameters that aren't dependent on the parameters in the data are our choice of M (which we use to calculate the polynomial basis Φ from $\mathbf{x}$) and λ. We experiment with different values of M and λ in Figure 7 to see how they affect the solution.

We observed that increasing λ will eventually drive regression coefficients (the weight vector) to zero. This is why the regularizer here is known in machine learning literature as "weight decay". When M increases, the order of the polynomial is higher, which means that the line is likely to overfit to the training data.

Figure 7 shows a plot of the final fit on the training data.

3.2. **Model Selection.** When implementing grid search, we try M values from 0 to 15, because the order of the polynomial used to fit the data should not be greater than the number of data points. If we had n data points and M = n, then the line of best fit will curve to fit the training data exactly. This is undesirable, because it is unlikely that an arbitrary test set will fit to this same line.

For values of λ, we try values from 0 to 1, because we found that having a higher λ increases the error in some cases. In the regression objective function, as the importance of the λ term increases, the data-dependent term vanishes. We want to find the optimal balance between these terms.

When A is used for training, and B is used as test set, we observe the results in Figure 8a. By using a grid search to sweep through combinations of M and λ values, we find the optimal values to minimize the regression error.

We observe the effect of varying the M and lambda values respectively in Figure 8b, which shows the effect of varying M values on regression squared error as a function of $\lambda = 0.0$, which we found to be our best λ value, and the effect of varying λ values on regression squared error, as a function of M=2, which we found to be our best M value.

The resulting regression lines on the validation and test sets are displayed in Figure 8c.

When B is used for training, and A is used as test set, we have Figure 9 with 9. Because one point in dataset B is an outlier, the weight vector resulting from ridge regression is skewed in such a way to produce the graphs we see here.

4. Sparsity and LASSO

4.1. **Comparison With Ridge Regression.** We used a quadratic regularizer in ridge regression. A more general regularizer can be used, in which the regularized error takes the form

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\Phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q\,. \tag{4.1}$$

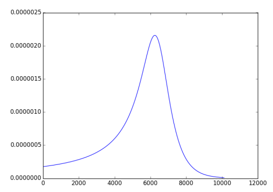
The lasso estimator uses this regularizer when $q = 1$. When λ is sufficiently large, some of the coefficients w_j will go to zero. The zeroed elements of the weight vector play no role. We call this a sparse model.

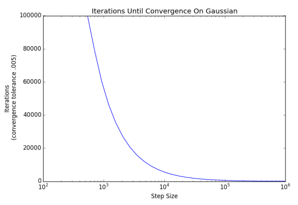
We want to estimate sparse weights for the LASSO estimator, using the dataset: lasso_train.txt, lasso_validate.txt, and lasso_test.txt. The dataset was generated according to $y = w_{true}^T\phi(x) + \epsilon$, where $x, y \subset \mathbb{R}$, $w \subset \mathbb{R}^{13}$, and ϵ is a small noise. The function used to generate the basis vector is

$$\phi(x) = (x, \sin(0.4 * \pi * x \times 1), sin(0.4 * \pi * x \times 2), ..., \sin(0.4 * \pi * x \times 12)) \subset \mathbb{R}^{13}. \tag{4.2}$$

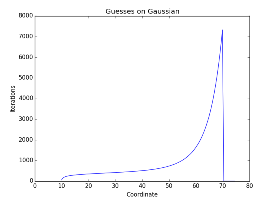
Since the closed form solution for LASSO doesn't exist, we use the Python Scikit-learn implementation of Lasso, which uses coordinate descent. We compare the result of LASSO the estimated weights from ridge regression and the true values in Figure 10. We do get sparse estimation from LASSO.

5. Figures

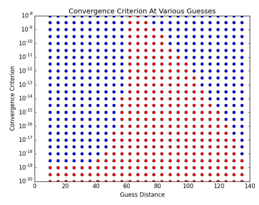

(A) Converging to (10.045, 10.045): step = 10^4, guess = (55,55), threshold = 10^{-8}. x: iteration step, y: norm value.



(B) Step Size versus Iterations: guess = (30,30), threshold = 10^{-9}.

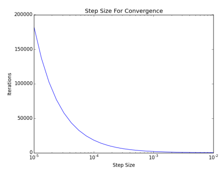


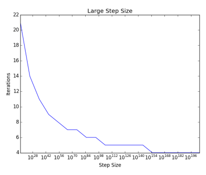
(C) Coordinate x,x versus Iterations: step = 10^4, threshold = 10^{-8}.



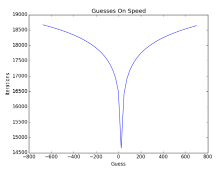
(D) Coordinate x,x versus Threshold: step = 10^4, max steps = 2×10^4. Red: max steps reached, Blue: converged.

Figure 1. Gaussian Analysis

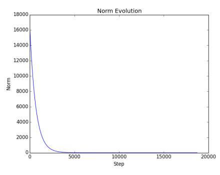




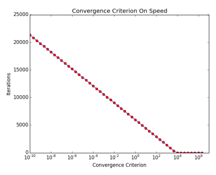
(A) Small, Large Step Size Convergence: guess = (400,400), threshold = 10^{-8}.



(B) Coordinate x,x versus Iterations: step = 10^{-4}, threshold = 10^{-8}.

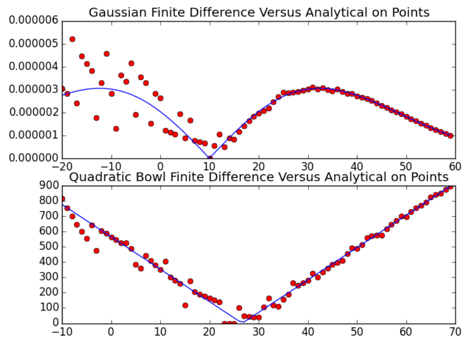


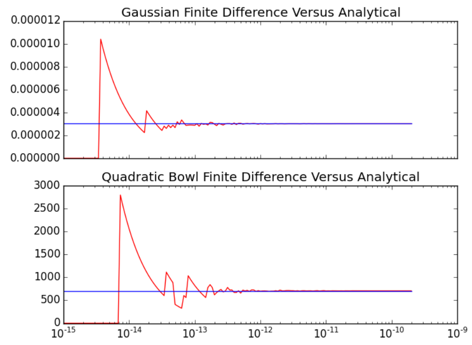
(C) Norm Converging to (26.6667, 26.6667): step = 10^{-4}, guess = (800,800), threshold = 10^{-8}.



(D) Threshold versus Iterations: step = 10^{-2}, guess = (400,400).

Figure 2. Quadratic Bowl Analysis

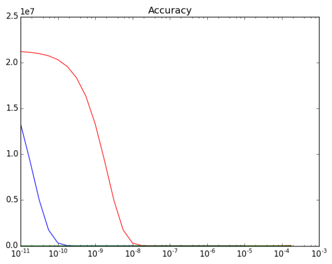


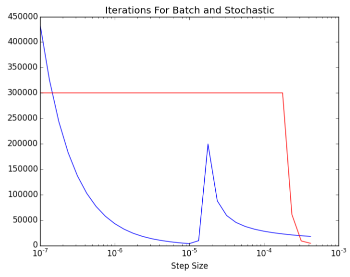


(A) On x,x: $\delta = 10^{-14.2}$ to $\delta = 10^{-11.5}$ left to right.

(B) Gaussian @ (30,30), quad bowl @ (60,60).

FIGURE 3. Difference Step Sweeps versus Analytical Solution





(A) max steps = 3×10^5, min grad = 10^{-3}.

(B) max steps = 3×10^5, min grad = 10^{-3}.

FIGURE 4. BGD versus SGD Accuracy and Iterations. Blue = batch, Red = stochastic.

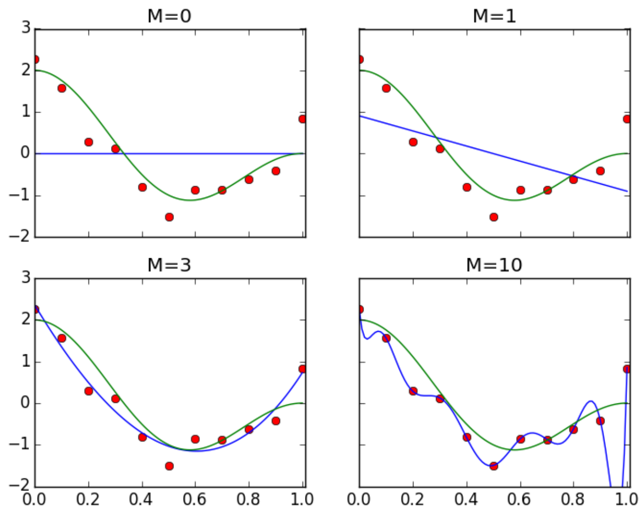
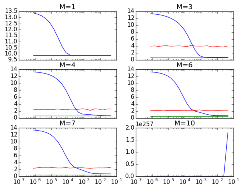
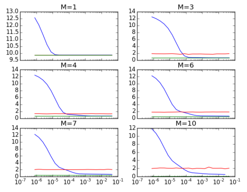
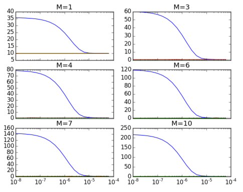


FIGURE 5. Linear Regression on Polynomial Basis: Blue = regression, Red = data, Green = underlying generative sinusoid


(A) max steps = 10^4, guess = $\vec{0}$, threshold = 10^{-4}.


(B) max steps = 10^5, guess = $\vec{0}$, threshold = 10^{-4}.


(C) max steps = 10^4, guess = true values + $\vec{1}$, threshold = 10^{-4}.

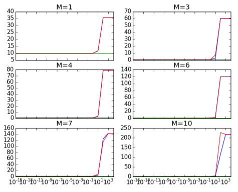

(D) step = 10^{-4}, max steps = 10^4, guess = true values + $\vec{1}$.

FIGURE 6. Batch Error versus Step Size, Threshold of BGD and SGD at Various Guesses and M. Blue = BGD, Red = SGD, Green = Actual.

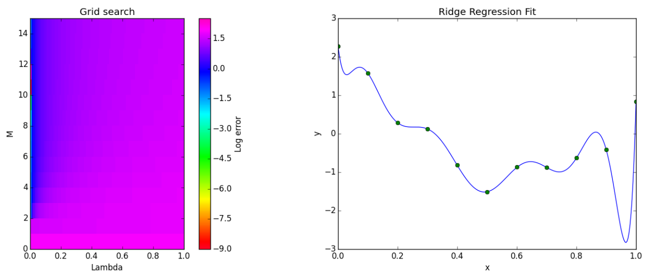
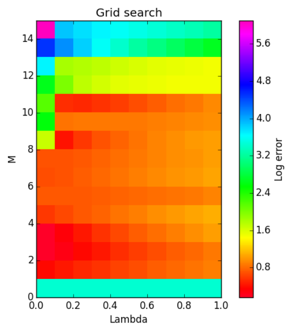
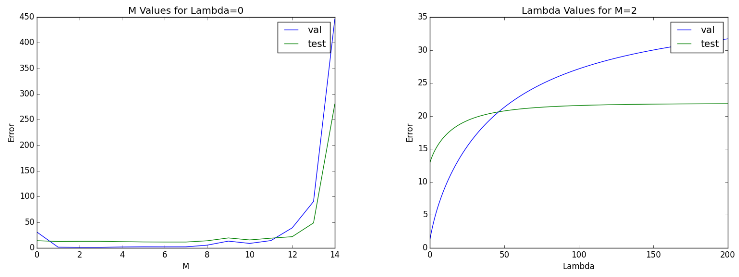


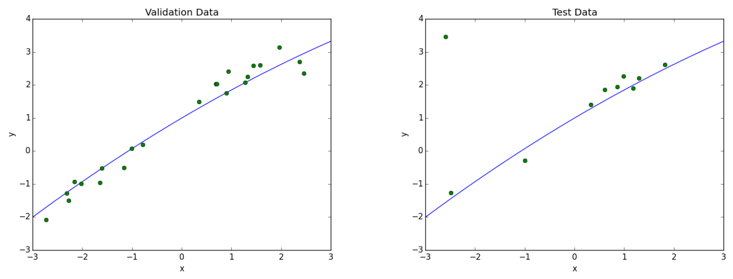
FIGURE 7. Grid Search for Best M = 10, Best $\lambda = 0$



(A) Grid Search



(B) Effect of Varying M and *lambda* on Regression Squared Error: λ=0.0 and M=2, respectively



(C) Validation Testing

FIGURE 8. Best M = 2, best $\lambda = 0.0$

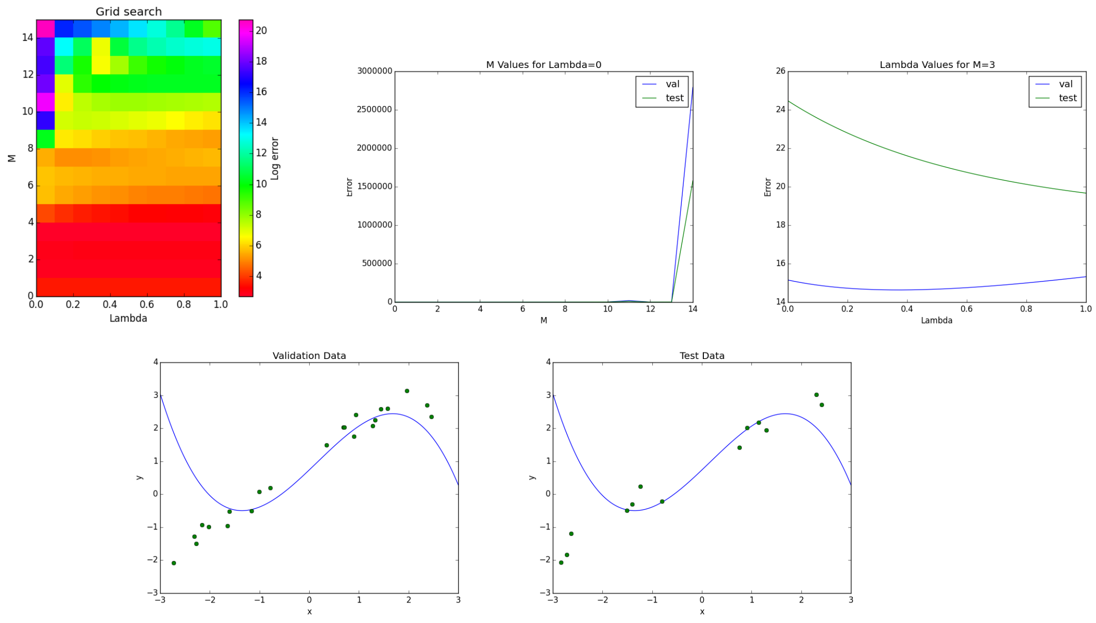


FIGURE 9. Grid Search and Best M = 3, Best $\lambda = 0.37037037037$

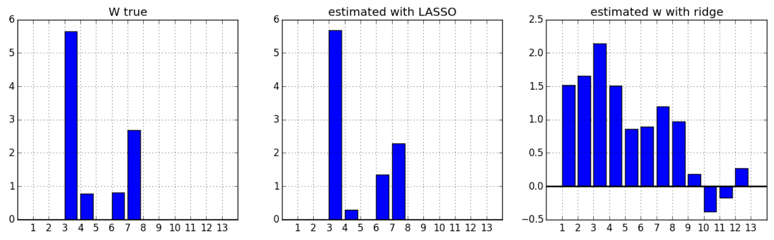


FIGURE 10

θ	SSE	Finite Difference
θ_1	0.0192	0.0191999980359
θ_2	3.99782	3.9978200
θ_3	2.221006	2.22100599
θ_4	1.014797	1.014796998
θ_5	0.25565878	0.255658779
θ_6	-0.2347044	-0.23470439
θ_7	-0.56491086	-0.5649108
θ_8	-0.79653944	-0.796539437
θ_9	-0.96501544	-0.9650154382

(A) SSE versus Finite Difference Estimation

w	M=1	M=2	M=3	M=4
w_1	-0.00087273	-0.00087273	-0.10926758	-0.10926758
w_2	0	0.7789928	0.7789928	0.76335952
w_3	0	0	1.19234339	1.19234339
w_4	0	0	0	0.09379968
w	M=5	M=6	M=7	M=8
w_1	-0.12581095	-0.12581095	-0.15130151	-0.15130151
w_2	0.76335952	0.770386	0.770386	0.76885361
w_3	1.15925665	1.15925665	1.10827553	1.10827553
w_4	0.09379968	0.10082616	0.10082616	0.09929377
w_5	0.21506381	0.21506381	0.16408268	0.16408268
w_6	0	-0.04918536	-0.04918536	-0.05071775
w_7	0	0	0.38235842	0.38235842
w_8	0	0	0	0.01225909

(B) Cosine Basis Weights

TABLE 1