Zhihan Wang
ID:30005124

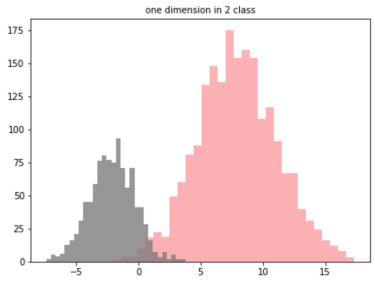
Report

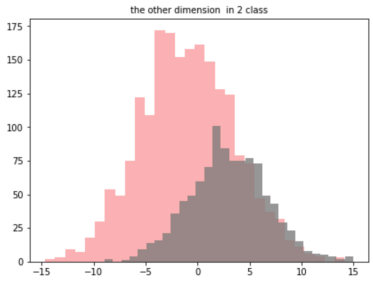
In this report, I am going to answer the questions in the coursework and illustrate my understandings by pointing to the results I obtained from the exercise.

Classification

I. 2 Gaussians

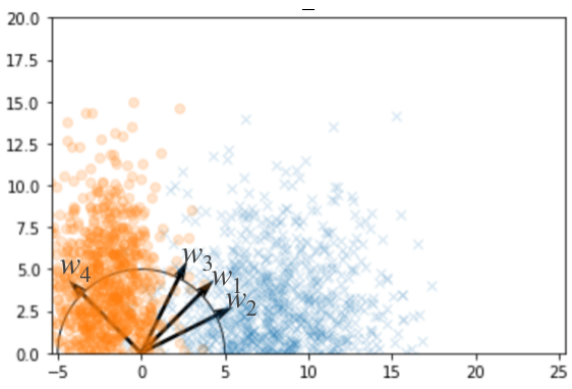
2-dimensional Gaussian distribution is used in this part. Firstly, generate two Gaussians, class a and class b, with mean vectors $\mu_a = (8, -1)$ and $\mu_b = (-2,3)$, and covariance matrices. To be more generally, the covariance matrices are set up not equal to each other, with $cov_a = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix}, cov_b = \begin{bmatrix} 3 & 0 \\ 0 & 15 \end{bmatrix}$; also, make the numbers of data points in two classes not equal, with 2000 points from class a, and 1000 points from class b. To check whether the classes chosen are appropriate, as I do not want the classes widely separated or to overlap significantly, I plotted histograms in 2 dimensions separately. As shown, the 2 classes are only well separated in one dimension, so they would have good performance in the 2-D circumstance.





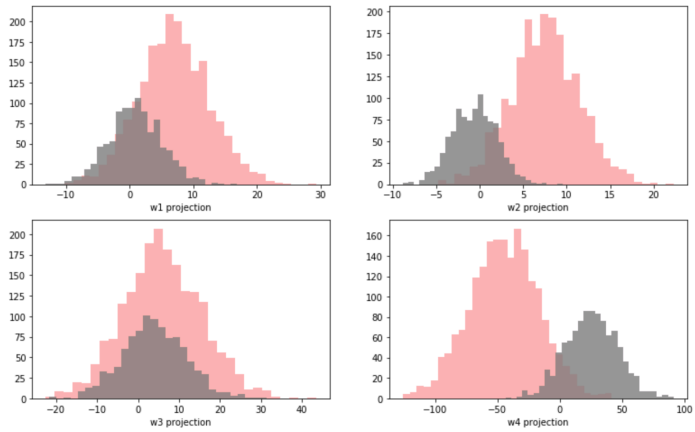
1. Data projection

a) 4 direction w were chosen to project data points



To be more clearly, I chose w in 4 distinct directions, as shown in the left graph. Then I calculated the values of data projected in each w and plot histograms respectively below. From charts, w_4 has the best performance as it separated 2 classes distinctly. In contrast, w_3 performs poorly because the points projected on w_3 overlap significantly. That's why the

mean values of projection from 2 classes points are almost equivalent.

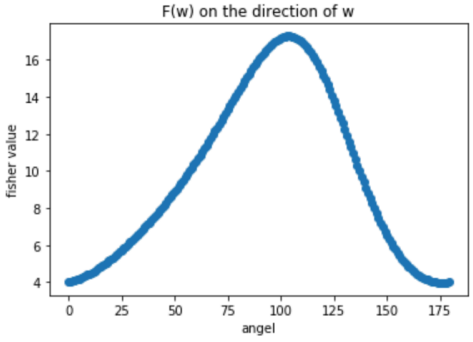


b) Fisher ratio

As shown, different w choice decides whether classification would be better. To find w^*, which has the best performance on classification, 'Fisher ratio' would be adopted. 'Fisher ratio' consists of two parts. One is the difference of expectation between 2 classes' projection. The other is the variance of projection in each class. If 'Fisher ratio' is large, it means the expectation of projection in 2 classes is quite different from each other and the data points in each class are concentrating at their central point. In this case, the classes would be distinguished from each other in projection. Therefore, obtaining the biggest 'Fisher ratio' makes sense. To make a simulation in computer, firstly, choose a random starting weight vector w_0, and then by rotating it with angle θ acquire a w set. Finally, calculate 'Fisher ratio' to find the maximum and corresponding $\hat{w}^*$. In theory, if angle θ is ergodic in the given angle space, then $\hat{w}^*$ in the simulation would be the true w^*. So in the computer simulation, I set up the angle space from 0° to 180°, and every time rotate 1°. Although the $\hat{w}^*$ computed here is not the true w^*, it would be very close to the truth.

The plot below illustrates the dependence of 'Fisher ratio' $F(w)$ on the direction of w. If w_0 is rotated with an angle $\theta = 104°$, 'Fisher ratio' would be the biggest, $F(w(\theta = 104°)=17.28$. To validate the result, the expressed method of 'Fisher ratio' is replaced. Instead of calculating projection, I obtain the discernibility matrix about mean and covariance in 2 classes from original data points. By calculating the eigenvectors and eigenvalues, the optimal w and the biggest $F(w)$ could be found. The result computed in this way is very

```
In this plot, the maximum fisher value is 17.28097054903637
The angle is 104
The corresponding direction w* is [-0.97604964  0.21754792]
```

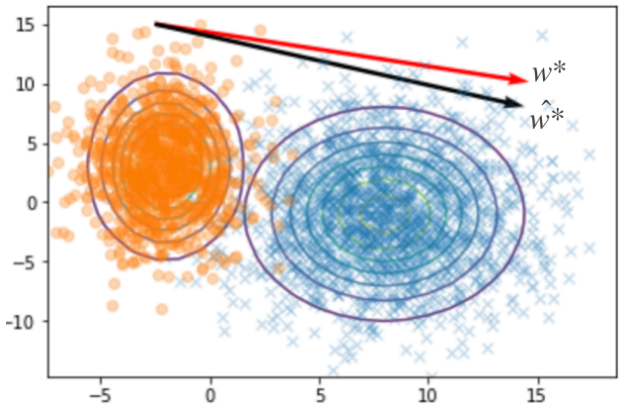


close to before. Then, I computed the rank of the between-class covariance matrix, which is the number of non-zero eigenvalue. It is 1, equal to 2-1. I will discuss something about rank later.

```
the eigenvalue is
 [0.00450629 0.        ]
the eighenvector,also the best weight is
 [[ 0.98587621  0.37139068]
 [-0.16747565  0.92847669]]
```

2. Contour plots

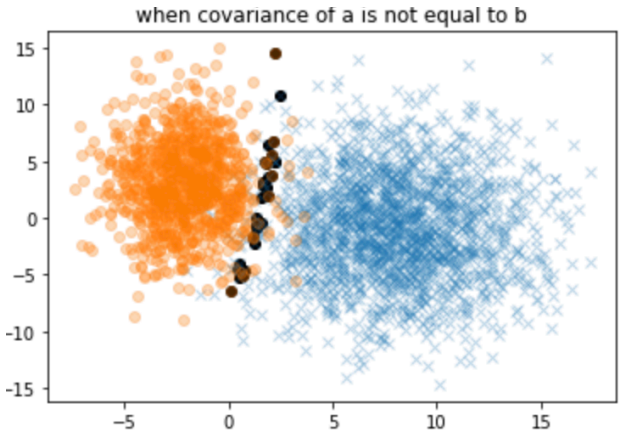
a) equi-probable contour lines and vector w^*



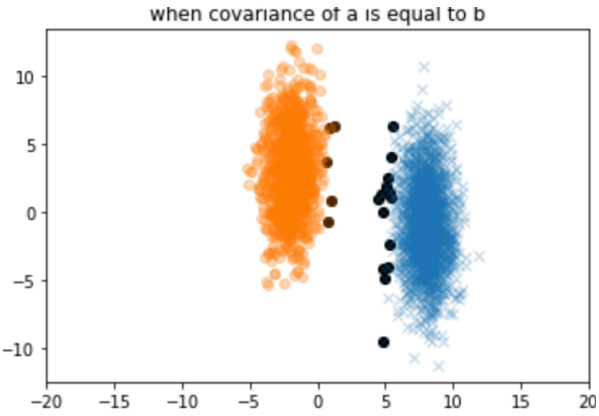
To acquire contour lines, the probability (probability density here) of each data point need to be calculated. Next, project the probability into a 2-D plain. 2 optimal direction w^*, $\hat{w}^*$ calculated before are also put on it.

b) decision boundary

In the decision boundary, the probability of the data point in class a is equal or nearly equal to that in class b. And if one data point is in class a, the probability of it in class a is bigger than that in class b. 'log-odds' is used to make sense. 'log-odds' is the logarithm of conditional probabilities of data points in class a divided by that in class b, written as $ln(\frac{P(c=a|x^n)}{P(c=b|x^n)})$. Using Bayes' theorem, the 'log-odds' could be written down as $ln(\frac{P(c=a|x^n)}{P(c=b|x^n)}) = ln(\frac{P(x^n|c=a)P(a)}{P(x^n|x=b)P(b)})$. By implementing 'log-odds', if the logarithm is positive, then we can say that the point should be in class a; if zero, it should be in decision boundary; if negative, it should be in class b. On the left graph, the set of black points, setting 'log-odds' very close or equal to zero, form the decision boundary, which is a curve. Notice that the covariance matrix in 2 classes here is not equal to each other.



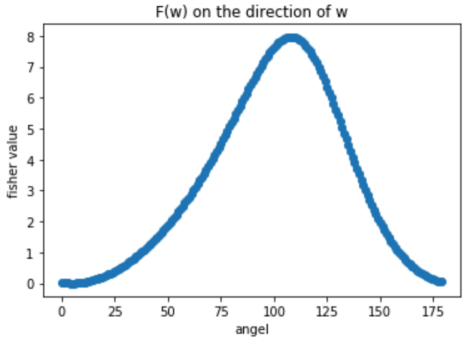
To see how the decision boundary would change if the 2 covariance matrices are equal, the matrices are both set as $\begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$, a new plot is on the right. In this plot, the decision boundary tends to be a straight line.



c) different fractions of data in each class

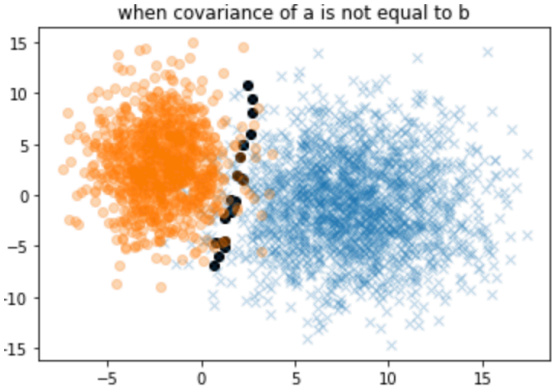
As said at the very beginning, to be more generally, the numbers of data points in 2 classes are not equivalent. So when the 'Fisher ratio' in 1.b) is computed, the parameters before variances in the fraction are $\frac{2}{3}$ and $\frac{1}{3}$. In this part, we do not account for the parameters anymore and set them equal to 1. Then comparing the result with before, as the plot showed on the right, the w^* has a slight change.

```
In this plot, the maximum fisher value is 7.971110922036502
The angle is 108
The corresponding direction w* is [-0.98884741  0.1489322 ]
```



Besides, in 2.b), when using Bayes' theorem to calculate 'log-odds' considering different scale. The probability of a and b is $\frac{2}{3}$ and $\frac{1}{3}$, with a small difference. Here, if we do not consider different parameters anymore. Then the 'log-odds' would be

$$ln(\frac{P(c=a\,|\,x^n)}{P(c=b\,|\,x^n)}) = ln(\frac{P(x^n\,|\,c=a)P(a)}{P(x^n\,|\,x=b)P(b)}) = ln(\frac{P(x^n\,|\,c=a)\times 1}{P(x^n\,|\,x=b)\times 1}) = ln(\frac{P(x^n\,|\,c=a)}{P(x^n\,|\,x=b)}).$$



The plot without accounting for different scales of data in each class is on the left. It is only a bit different from the plot in 2.b). However, if the difference between scales is significantly large, then not accounting for scales would have a worse effect.

II. Iris data

In this part, I am going to apply LDA on Iris dataset and analyse it using a more general way to obtain optimal w^* by calculating eigenvalues and eigenvectors as there are more features(four in this question) and classes(three in this question). Instead of calculating value of mean and variance in projection like previous part, between-class covariance matrix Σ_B and within-class variance matrix Σ_w would be computed here. Σ_B indicates the sum of the difference of mean in each class from the overall one. And Σ_w is the sum of covariance matrix in each class. In the Iris dataset, there are 50 points in each class, so the scale of each class is $\frac{1}{3}$.

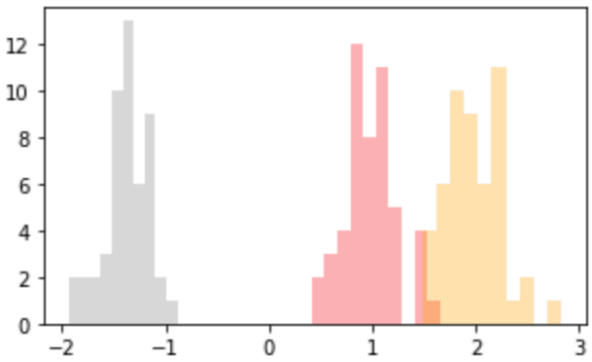
1. optimal direction

After importing dataset into the workspace, Σ_B and Σ_w could be obtained. The theorem used before to compute 'Fisher ratio' is same as the theorem used here. The difference is using matrix notation here to write down 'Fisher ratio' as $\Sigma_B w = \lambda \Sigma_w w$. By computing the eigenvectors and eigenvalues of matrix $\Sigma_w^{-1}\Sigma_B$, optimal direction w could be obtained. The table below shows the result. The first eigenvalue is the biggest and corresponds an eigenvector in the first column which is the optimal direction w^*. Then check the results in the table whether also meet the condition $(\Sigma_B - \lambda \Sigma_w)w = 0$. The left part in the equation is calculated for 4 times for each λ and w in Python. They are all very close to 0. So there is no problem on the eigenvectors and eigenvalues. The optimal w^* is [-0.2,-0.39,0.55,0.71]. At the same time, compute the rank of Σ_B and the rank is 2, equal to 3-1. Compared with the rank calculated at the final part in I.1.b, the rank is alway equal to $c-1$ where c is the number of classes. So rank is limited by the number of classes. I think the reason is that when computing Σ_B which is the sum of c classes deviation, the density of freedom is c; however, when we use the mean vector of each class to estimate the overall mean vector, it would lose 1 degree of freedom. So the rank should be at most $c-1$.

```
the the eigenvalue is
 [1.07573193e+01 9.25222879e-02 6.37489996e-16 2.29298245e-15]
the eighenvector,also the weight is
 [[-0.20490976 -0.00898234 -0.65243798  0.06674523]
 [-0.38714331 -0.58899857 -0.01145214 -0.35848605]
 [ 0.54648218  0.25428655 -0.0773472  -0.42574773]
 [ 0.71378517 -0.76703217  0.75379768  0.82811333]]
```

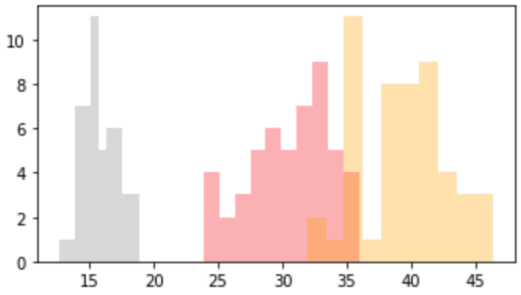
2. histograms

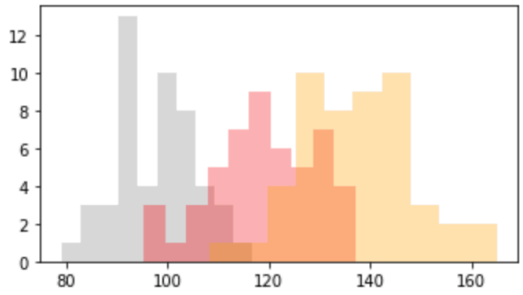
Project the data on w^* to reduce dimension from 4 to 1, and then plot histograms of 3 classes. As shown on the right plot, in the optimal direction, 3 classes are separated widely.



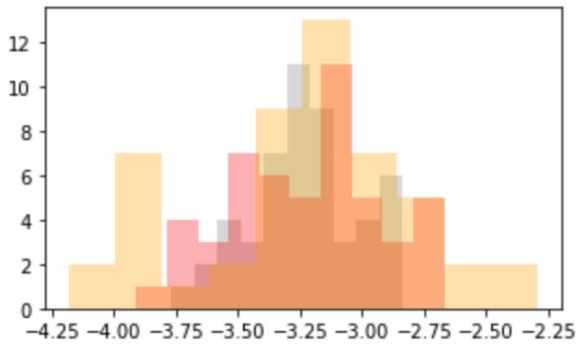
3. a different vector w

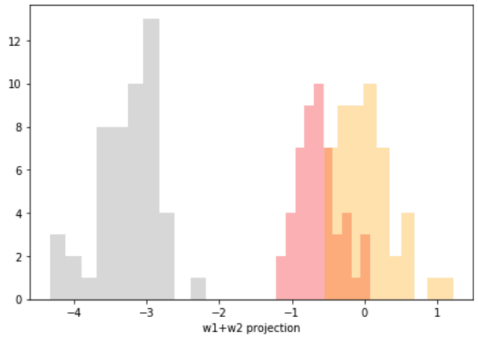
Add a vector a to optimal direction w^* to acquire another direction w to see what would happen if project the data to this new w. Firstly, choose two random vectors $a_1 = [1,2,3,4]$, $a_2 = [10,12,5,3]$, and then plot projection histograms. The vector a_1 on the left performs better.

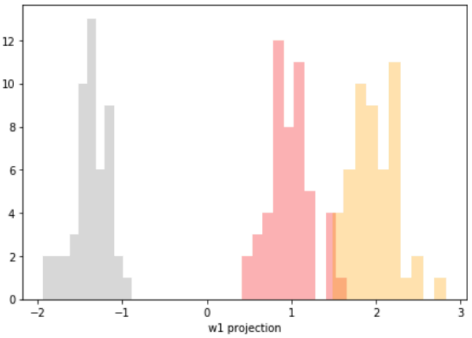


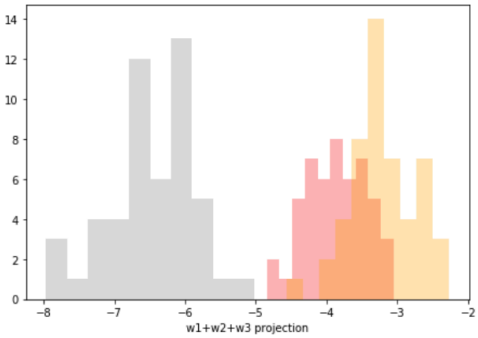


Then instead of making random choice, acquire a by constructing it out from the other eigenvectors. Firstly, replace w^* using the eigenvector given by the third big eigenvalue to see what would happen. As shown on the right chart, the performance is extremely poor. Then I added one or two eigenvectors to w^*. At the first time, only the second eigenvector is added and then both the second and the third eigenvectors. The histograms below compare different circumstances. From the charts, the more vectors added









to optimal direction, the more poorly the separation performs. The reason why this happened is that every time when an eigenvector added to w^* to make a new w, w would contain more information than before; therefore, it could explain more features. The table about how much each eigenvalue explains is shown on the right. However, what we want to know is how a problem is decomposed and what is the most principle feature of it rather than how to obtain the most information. So in the direction of eigenvector corresponding the biggest eigenvalue, the eigenvector is stretched the most. So in this direction, the corresponding feature is the most principal, explains the most, without so much noise. That's why only one optimal w^* would perform the best.

```
eigenvalues explain:
99.14724756595075 %
0.8527524340492321 %
2.1133787399885347e-14 %
5.875569634709319e-15 %
```

4. compare with method in I

As discussed in I.1.b), the results of two methods to find optimal w are similar because the fundamental theorem in both are totally same. In I, 'Fisher ratio' is to calculate projection. By rotating initial w_0 to acquire 'Fisher ratio' corresponding to each w, then the maximum of 'Fisher ratio' could be found. This is a direct searching method. One disadvantage is how to make rotating angle θ continuous in the computer simulation. Of course, the rotation step could be set extremely small, but also could not be infinitesimal. An alternative way to do this is to substitute the function of 'Fisher ratio' with derivative of the function on w, and then to find w^* by making derivative equal to 0. However, this would make the computation more complex because we have already computed projection on w at first. On the other hand, there is no additional calculation and no need to consider step like direct searching in the method used here. And only need to obtain the between-class and within-class variance matrices from original data, and then calculate eigenvectors and eigenvalues. In Iris data and 2 Gaussian problem, we only select the most principle eigenvector. Another important difference between 2 methods is that, instead of only select one eigenvector, if we select more than 2 eigenvectors to form a matrix w in a higher dimension problem, using

general method used in Iris data is better. So the general method is more simple, accurate and easily applied.

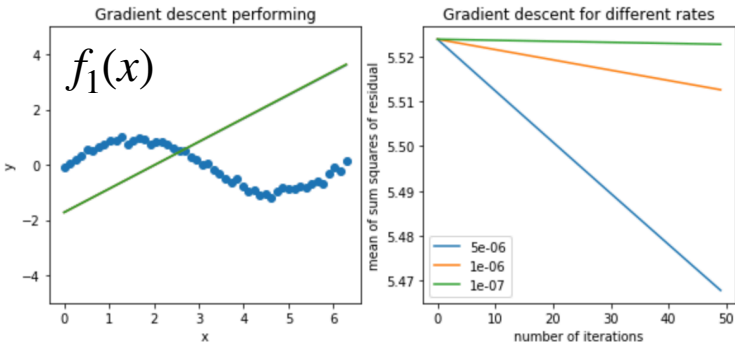
Regression

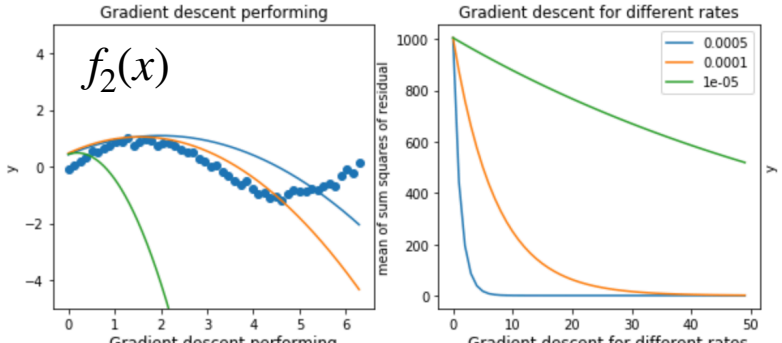
I. Linear regression with polynomial function

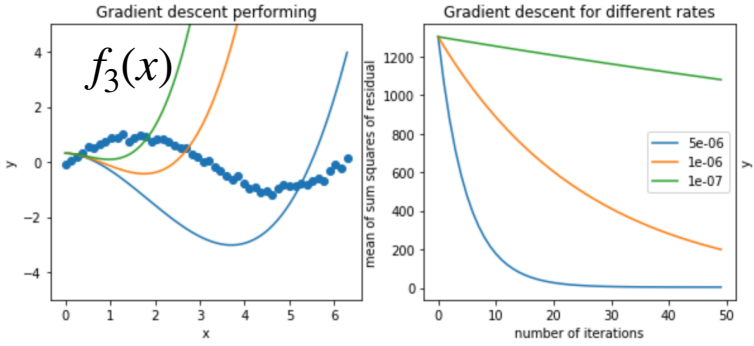
The aim function is $y(x) = sin(x) + \epsilon$ where ϵ is a noise term from normal distribution, $\epsilon \sim N(\mu = 0, \sigma = 0.1)$. Polynomial functions would be used to fit this function.

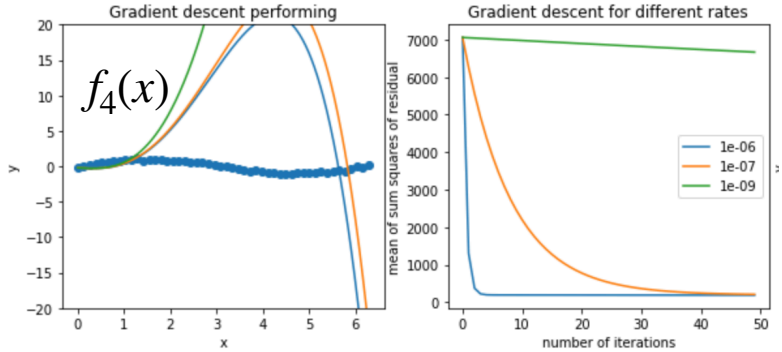
1. gradient decent

In this part, I am going to minimise the loss function by gradient descent to learn weights. An initial weight w_0 is selected randomly, and then find the gradient direction of w_0 in the loss function. After this, obtain one new w in the opposite direction of gradient descent by function $w_i = w_{i-1} - \eta \frac{\partial L}{\partial w}$ where η stands for learning rate, controlling each step length. Iterate this process until gradient is equal to zero. Firstly, fit aim function using $f_1(x) = w_0 + w_1 x$ in different learning rates. It is shown on the right. Then substitute $f_1(x)$ with $f_2(x) = w_0 + w_1 x + w_2 x^2$, $f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ and $f_4(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$







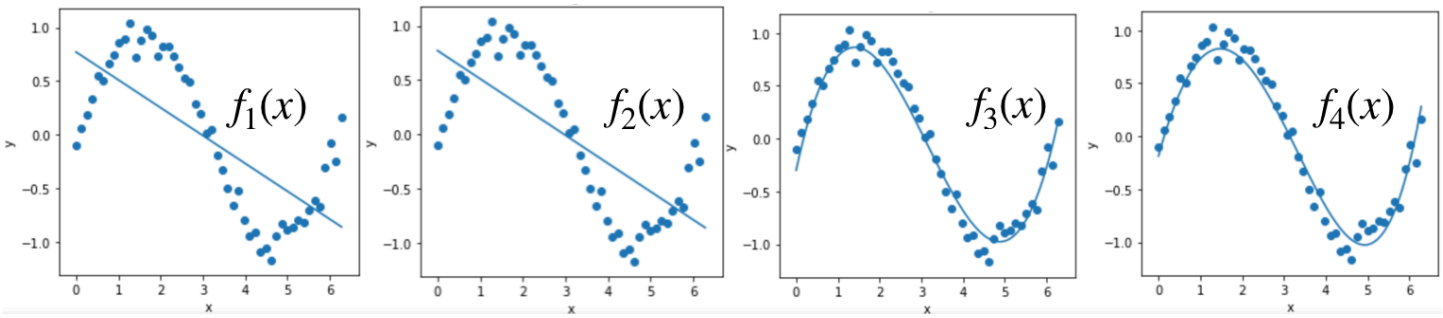


From these graphs, it is $f_2(x)$ that performs the best, not $f_4(x)$. So a higher degree does not stand for a better fitting performance.

Also, a suitable learning rate is required. If it is too small, then w would not converge even with a large number of iterations. On the other hand, if it is too big, it might also take a long time or lose the ability to find the optimal w because it might miss the minimum point with a long step.

2. obtain w in another way

Instead of using gradient descent, obtain weights by analytical expression $w = (X^T X + \lambda I_{(p+1)\times(p+1)})^{-1} X^T y$. The fitting results are as below.



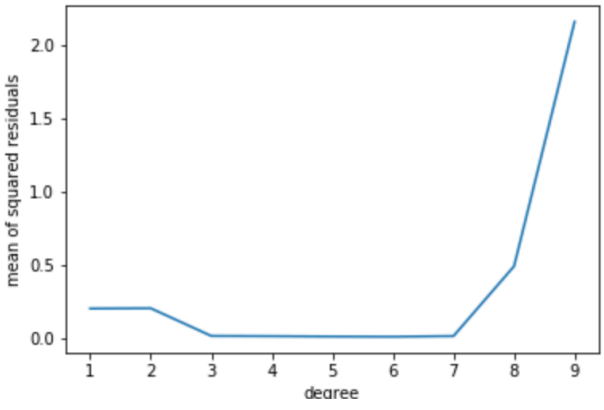
Comparing 2 methods to obtain optimal weights to fit the given data points, the second method performs better than gradient descent.

3. measure function

The data is divided into training set and test set. Then use training set to obtain weight w and apply it to the test set to measure the mean of squared residuals, which is same as computing loss function.

a) <u>complexity of the model</u>

As referred before, a higher degree does not mean a better performance. To be more precisely, the 'measure' would be plotted as a function of the number of components ϕ. From the line graph on the right, when degree is 3 and 7, the model would perform the best. But when degree is 7, the model would be extremely complex and therefore be added too much noise. So it could be inferred that when degree is 3, that is to say $f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$, the model has the best performance.

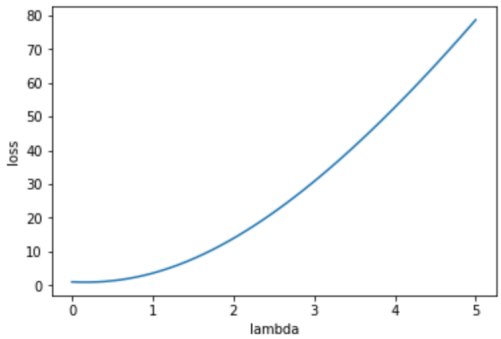


b) <u>the strength of λ</u>

In the simulation before, I set up $\lambda = 1$. In this part, λ would be a variable in the 'measure' function, $\lambda \in D[0,5]$. The model applied here would be

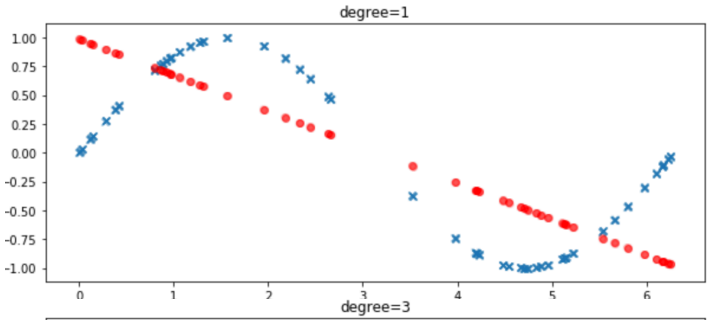
```
the minimum loss is 0.9502972544610475
the lambda is 0.20408163265306123
```

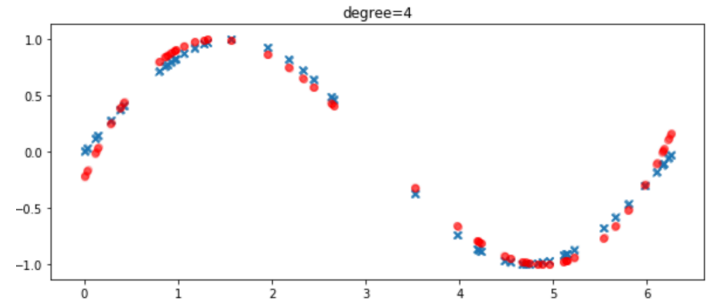
the optimal one obtained in I.2.a), where the degree of best model is 3. The plot about the relationship between λ and 'measure' is on the right.

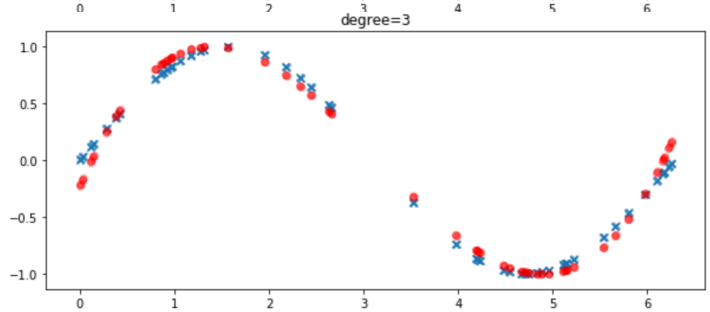


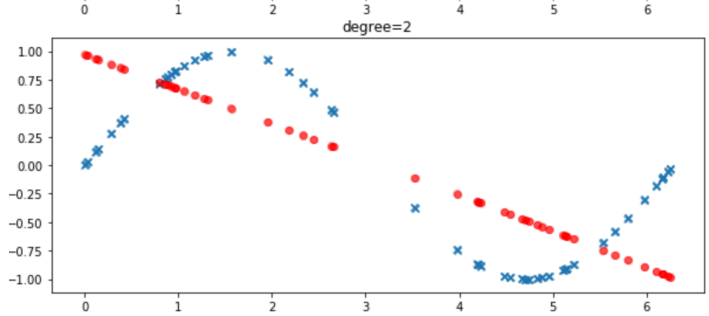
II. 10-fold

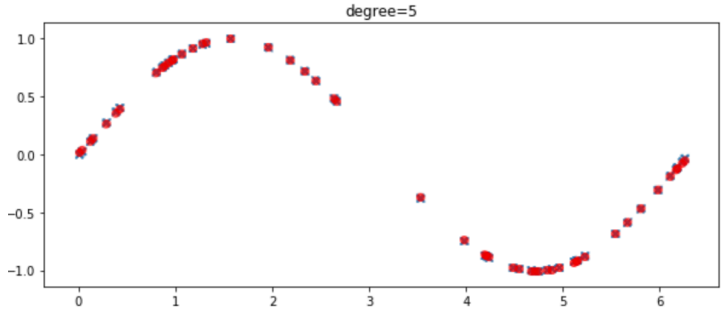
Firstly, generate 5000 data points. Divide data into 10 parts, and then in each part, split up it into sub-training set and sub-test set in a proportion 9:1. Models should be learnt in each sub-training set at first. Then apply the model in the overall-test set, consisting of 10 sub-test sets to evaluate the model. The results of model with different degrees evaluated in test set are shown as below. The blue points are the real points in the test set, and the red points are generated by the regression model with the mean vector of weights learned from 10 sub-training set. As shown, when degree is











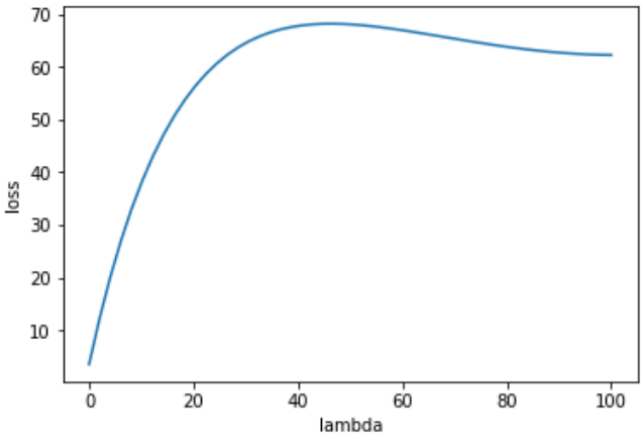
bigger than 3, the model becomes to fit better and better. When degree is equal to 5, the difference between the model and real value is extremely small. And if we do not want our model to be overfit, then $f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ would be a good choice.

```
number of points in test set is 500,
value of loss function is  4.53192868645047
the Loss using model learned from sub-training set 1 is
  4.53243072955292
the difference of this model from the optimal model is
 0.015415603727555904
the Loss using model learned from sub-training set 2 is
  4.5263169666423995
the difference of this model from the optimal model is
 0.009301840817035334
the Loss using model learned from sub-training set 3 is
  4.521717119581146
the difference of this model from the optimal model is
 0.004701993755781508
the Loss using model learned from sub-training set 4 is
  4.518631416544831
the difference of this model from the optimal model is
 0.0016162907194665621
the Loss using model learned from sub-training set 5 is
  4.517060099138007
the difference of this model from the optimal model is
 4.497331264285975e-05
the Loss using model learned from sub-training set 6 is
  4.517003422222851
the difference of this model from the optimal model is
 -1.1703602512902478e-05
the Loss using model learned from sub-training set 7 is
  4.518461654154047
the difference of this model from the optimal model is
 0.0014465283286826391
the Loss using model learned from sub-training set 8 is
  4.52143507658473
the difference of this model from the optimal model is
 0.004419950759365854
the Loss using model learned from sub-training set 9 is
  4.525923984554496
the difference of this model from the optimal model is
 0.008908858729132163
the Loss using model learned from sub-training set 10 is
  4.53192868645047
the difference of this model from the optimal model is
 0.014913560625106292
```

To look at the dependence of the optimal weights on datasets, the values of loss function in different models learned from 10 sub-training sets are calculated separately. The difference about loss function between each model and the expectation model used to fit is also printed on the left. In the most circumstances, the model from sub-training set is worse than the expectation model from all 10 models, only except the model obtained from set 6. Therefore, the expectation model would perform better in general.

In I.3.b), one suitable λ is equal to 0.2, so in this fitting process, $\lambda = 0.2$ is set up in advance.

Then to see the dependence of w^* on the scale of regularisation, adopt the variable $\lambda \in D[0,100]$. The results are as below. So in this method, a smaller λ would be better.



I use Jupiter notebook to write code. And the code files are all in an another file folder.