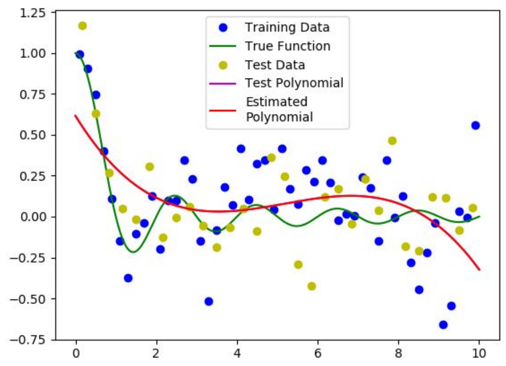
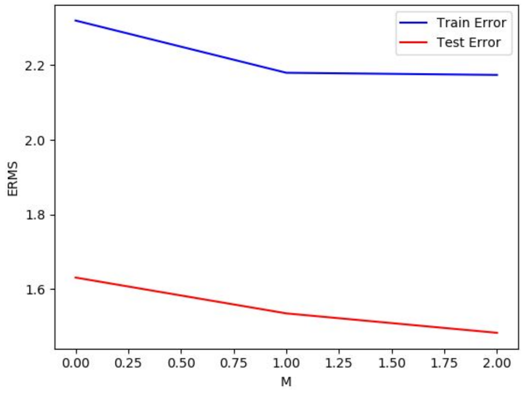
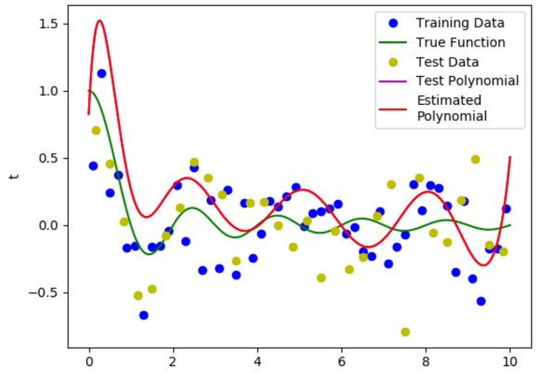
Joshua Siy
EEL5840
Warning: due to instances of small values plotted in the graphs. The assignment doer's perspective considers small values such as 0.25 or less to be large. And on instances of graphs associated with big values, a huge number such as 1000 may be seen small

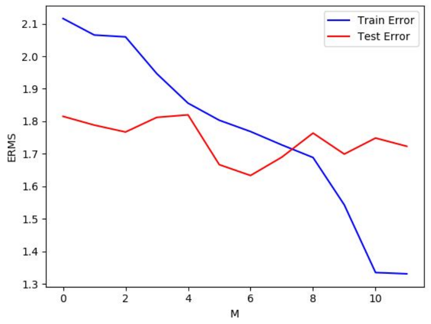
1) Discussing the results of the first programming task done for hw01. We are asked of which the model order M should be used to avoid underfitting or overfitting, from the understanding of the assignment taker, depending on the levels of error, the model order that is just right would produces, as much as possible, low error error difference or the test error would be bigger than the training error. Given the default values for the assignment where model order is 3 and sample size of 50 generating weights, we generated an unstable test to be used with the weights with a test size of 30 data and observe the following graphs as output.



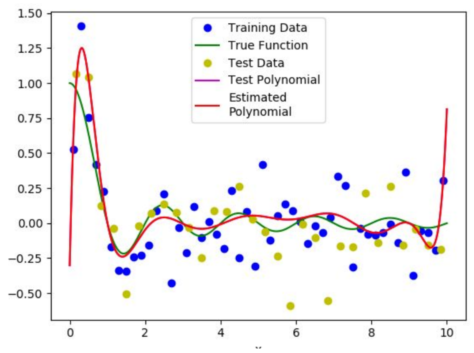


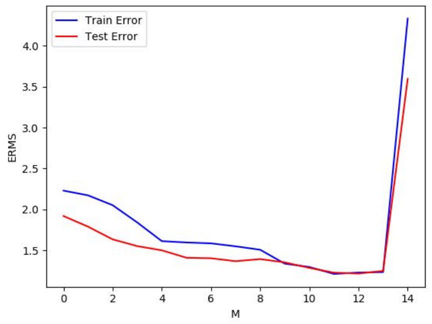
By the assignment doer's perspective, the model isn't exactly overfit but underfit due to the high amount of errors, however the test error are too unstable to define so up until that point the assignment taker believes model order 3 is still under-fitted due to the error isn't being consistent and the error difference is too big. What if we try to increase the model size to 12.





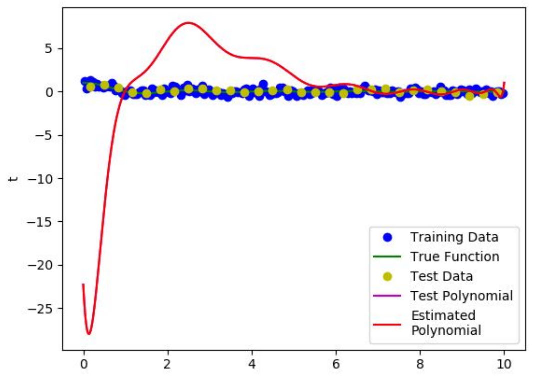
In this case, the error is too high when the model number is too low and the errors of the following models follow a normal pattern of error until reaching 8- 10 where the difference between the errors become slightly large, but still acceptable. Considering this model, model number 8-9 might be a very good order to use. Let us take the model order to 15.

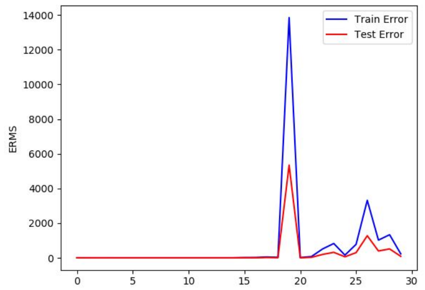




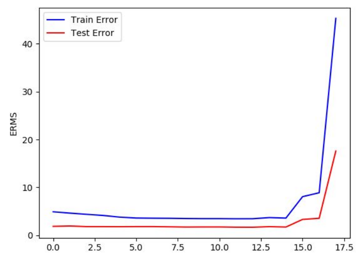
In this observation, we may see that alot of the test data and the train data tends to follow the curve where it may propose a sign of overfitting and looking at the ERMS data starting from model order number 0-8 produces alot of error while model order from 8-13 seems to produce less error compared to other orders. While looking for what comes after 13 maybe 13.5 and beyond the error spiked too high representing overfitted values. It convinces the assignment taker that model orders 8-13 seems like a good fit for the data.

However things seems to change depending on the number of sample sizes.the assignment takers observation is that to fit the data of a very big sample, one must increase the model size further to a fair amount. Assuming i just simply changed the sample size of the training data to 200

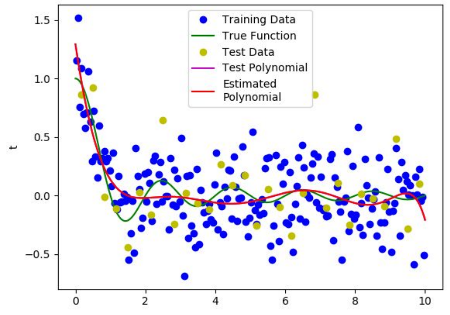




Since the amount of data is too miniscule to see and knowing that at 0-7 still gives more error, sometimes model order 8 to 13 seems to be a good model order to fit the data . But still, it still depends and needs to be experimented on regarding which model order is best. Since jsut basing it off the error graph will not exactly help us determine if it is the best model but can only tell which region is good. By some observation of generating random data it falls around 7-10



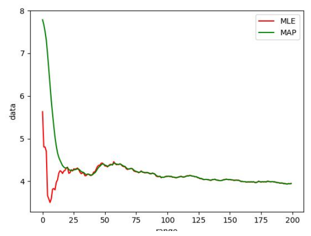
Assume we try to use 8 as a model number would it overfit or underfit?



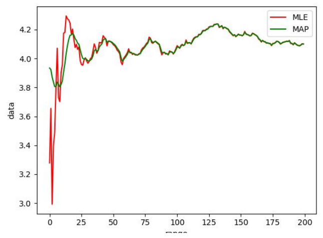
From the assignment taker's perspective, the polynomial doesn't entirely try to skewer every result so the viewer sees this model to have errors from time to time or could probably mistake some values, but considers this to be a good model. The assignment taker assumes that model 8-12 would be a good model order for this randomly generated data. However the decision of the assignment taker is that the model order can depend on the number of sample sizes provided and depending on how random the data is.

2) Provided are the sample datas of how the MLE and MAP, [that we know whatever we do at some point of the iteration converges to the true mean given], react given to the number of draws and changes in supplied data:

This plot has a prior mean of 8

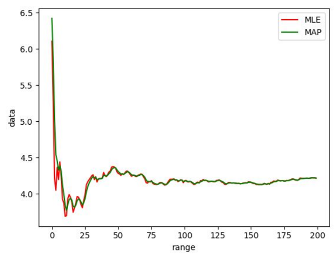


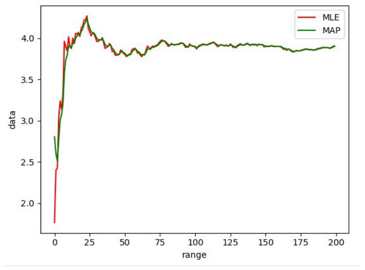
This has a prior mean of 4 which is true mean



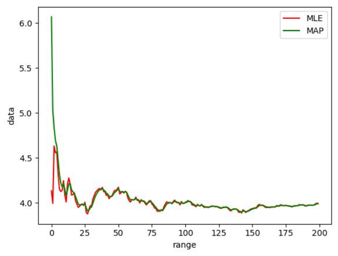
The prior mean affects the value of the MAP heavily. So from the initial steps of calculating the MAP, the starting point of it would be too offset maybe higher or lower from the real mean and slowly descends to the true mean. It would just probably take alot of draws/flips or any kind of repetition just for the MAP to settle to the proper mean. Putting the prior mean to be equal to the true mean makes MAP turns more stable and following the true mean faster compared to putting it in the wrong values.

The plots are associated with a high prior variance around 10 instead of 0.2

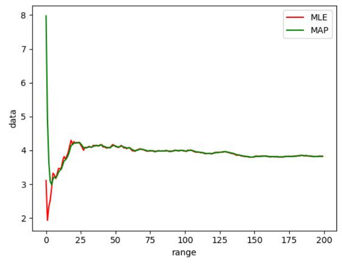




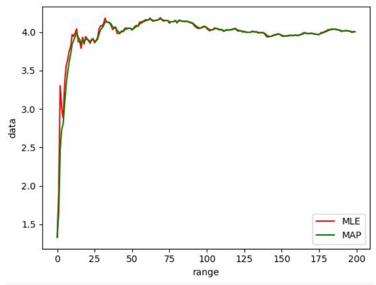
This plot is associated with putting the prior variance to be equal to the variance used to make the MLE with is 2



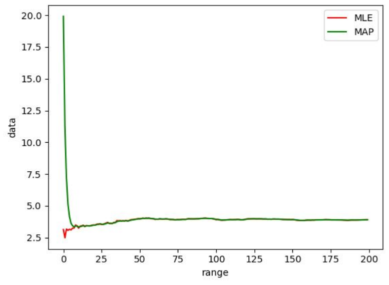
This plot has really low prior variance around 0.01 while the true variance is just 2



Prior variance experimentally set to 10000

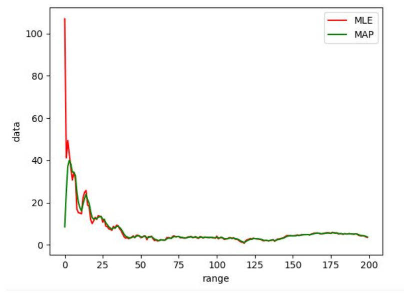


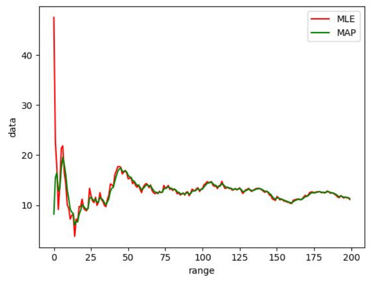
Prior variance and prior mu set= 20 to be big



The results of setting the variance to a high value bigger or equal to the true variance is that the initial MAP values attempts to mimic the value of the MLE's starting point but not necessairly be equal. Depending on how low the prior variance is from the true variance. The results of MAP's initial computation may be offset heavily or slightly depending on the variance given. However it was observed , no matter how high the prior variance is as long as it is equal to the true variance, it will always be almost close to the MLE's starting point and both of the MAP and MLE values will always converge to the true mean over time or iteration .

These plots react to the likelihood variance value of sqrt(2000) which i assume can be big enough





The results of setting the likelihood variance to a very big value sets the initial value of the MLE to be very very high. Making the graphs look like above and the next iterations of MLE makes the value to spike up high and low making its graph look unstable however still it converges to the true mean set.

So if the assignment taker was asked: how is the likelihood variance, prior mean and prior variance affect the final estimate of the mean? This assignment taker says that these values, ,regardless of how high or how low, will always end up converge to the true mean in an unstable way example the ending could be around 4.3 or a 3.7 instead of a perfect 4 however those decimals decrease over time till it becomes too negligible and considers it as a 4 such as 4+0.000001*10-34. However explaining what does these value do to the plot, is that it affects how the plot behaves , reacts or begins such as having a high prior mean makes the MAP's starting point to be higher or lower than the true mean, having a high prior variance or equal prior variance to the true variance as discussed from above makes the MAP's starting point either decrease or increase as it tries to follow the true mean closer. So the more the higher the prior variance is, the higher it tries to amplify or decrease the MAP's starting to be closer to MLE starting point. And having a very big likelihood variance makes the initial likelihood estimation's value to be smaller than the true mean or even overshoot the MLE's starting point to a negative value first then the value spikes up and down normalizing itself to the true mean.

3)f=3x^T*x+4y^t*x-1
  f=3x^2 +4y^t*x-1
  df/dx=6x+4y^T

4)
f=-10x^T*Q*x+4y^T*x+2
 =-10x^2*Q+4y^T*x+2
 =-20x*Q+4y^T

5)
f=8x^T*Q*x-2y^T*Q^T*x+6
f= 8x^2*Q -2y^t*Q^T*x+6
a)df/dx=16xQ-2y^t*Q^T
b)df/dQ=8x^2-2y^t*x

6)

$$f(\mathbf{x}) = \|\mathbf{4x}\|_2^2$$

f=4(sqrt(x^2))^2
=4x^2
df/dx=8x

The code:

```
# -*- coding: utf-8 -*-
"""
File:   hw01.py
Author: Joshua Siy
Date:   started 6/9/2018
Desc:   I tried my best, Dr. Zare or whoever is grading this.

"""

""" =====================  Import dependencies ========================== """

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import math

plt.close('all') #close any open plots

"""
==========================================================================
=======
==========================================================================
=======
============================ Question 1
==========================================
==========================================================================
=======
==========================================================================
=======
"""
""" ======================  Function definitions ========================== """

def generateUniformData(N, l, u, gVar):
        '''generateUniformData(N, l, u, gVar): Generate N uniformly spaced data points
  in the range [l,u) with zero-mean Gaussian random noise with variance gVar'''
        # x = np.random.uniform(l,u,N)
        step = (u-l)/(N);
        x = np.arange(l+step/2,u+step/2,step)
        e = np.random.normal(0,gVar,N)
        t = np.sinc(x) + e
        return x,t

def plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]):
  '''plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]): Generate a plot of the
    training data, the true function, and the estimated function'''
  p1 = plt.plot(x1, t1, 'bo') #plot training data
  p2 = plt.plot(x2, t2, 'g') #plot true value
  if(x3 is not None):
    p3 = plt.plot(x3, t3, 'r') #plot training data

  #add title, legend and axes labels
  plt.ylabel('t') #label x and y axes
  plt.xlabel('x')

  if(x3 is None):
    plt.legend((p1[0],p2[0]),legend)
  else:
    plt.legend((p1[0],p2[0],p3[0]),legend)

  plt.show()
def plotDataTest(x1,t1,x2,t2,x_test,t_test,x4,t4,x3=None,t3=None,legend=[]):
        '''plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]): Generate a plot of the
        training data, the true function, and the estimated function'''
        p1 = plt.plot(x1, t1, 'bo') #plot training data
        p2 = plt.plot(x2, t2, 'g') #plot true value
        p_test = plt.plot(x_test, t_test,'yo') #plot test data
        p_testcurve= plt.plot(x4, t4,'m')#plot test curve
        if(x3 is not None):
                p3 = plt.plot(x3, t3, 'r') #plot training data
    #add title, legend and axes labels
                plt.ylabel('t') #label x and y axes
                plt.xlabel('x')
        if(x3 is None):
                plt.legend((p1[0],p2[0],p_test[0]),legend)
        else:
                plt.legend((p1[0],p2[0],p_test[0],p_testcurve[0],p3[0]),legend)

        plt.show()
def plotDataError(x1,e1,x2,e2,legend=[]):
        '''plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]): Generate a plot of the
        training data, the true function, and the estimated function'''
        p1 = plt.plot(x1, e1, 'b') #plot training data
        p2 = plt.plot(x2, e2, 'r') #plot true value
        plt.ylabel('ERMS') #label x and y axes
        plt.xlabel('M')
        plt.legend((p1[0],p2[0],),legend)
        plt.show()

def plotconverge(like,post,range,legend=[]):
        p1 = plt.plot(range, like, 'r') #plot training data
        p2 = plt.plot(range, post, 'g') #plot true value
        plt.legend((p1[0],p2[0],),legend)
    #add title, legend and axes label
        plt.ylabel('data') #label x and y axes
        plt.xlabel('range')
        plt.legend((p1[0],p2[0]),legend)
        plt.show()


"""
This seems like a good place to write a function to learn your regression
weights!

"""
def fitdata(x,t,M):
        X = np.array([x**m for m in range(M+1)]).T
        w = np.linalg.inv(X.T@X)@X.T@t
        return w


""" =====================  Variable Declaration ========================= """

l = 0 #lower bound on x
u = 10 #upper bound on x
N = 50 #number of samples to generate
gVar = .25 #variance of error distribution
M = 9 #regression model order
""" ======================  Generate Training Data ======================= """
data_uniform  = np.array(generateUniformData(N, l, u, gVar)).T

x1 = data_uniform[:,0]
t1 = data_uniform[:,1]

x2 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
t2 = np.sinc(x2) #compute the true function value

""" =======================  Train the Model ============================ """

w = fitdata(x1,t1,M)
x3 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
X = np.array([x3**m for m in range(w.size)]).T
print(X)
t3= X@w #compute the predicted value

plotData(x1,t1,x2,t2,x3,t3,['Training Data', 'True Function', 'Estimated\nPolynomial'])
""" ======================= Generate Test Data =========================== """


"""This is where you should generate a validation testing data set.  This
should be generated with different parameters than the training data!   """


data_uniform_test= np.array(generateUniformData(30, l, u,gVar )).T

x_test= data_uniform_test[:,0]
t_test= data_uniform_test[:,1]


x4 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
XT = np.array([x4**m for m in range(w.size)]).T
```

```
t4= XT@w #compute the predicted value




plotDataTest(x1,t1,x2,t2,x_test,t_test,x4,t4,x3,t3,['Training Data', 'True Function','Test
Data','Test Polynomial','Estimated\nPolynomial'])


Mdivided=np.arange(0,M,1)
print(Mdivided)
testerms=[]
trainerms=[]
for x in range(M):# this block of code tries to determine the error of the values as the model rises
from 0 to M-1.
        w = fitdata(x1,t1,x)
        X = np.array([x1**m for m in range(w.size)]).T
        XT = np.array([x_test**m for m in range(w.size)]).T
        ttrainpred= X@w
        ttestpred=XT@w
        trainpredandobs=(ttrainpred-t1)@(ttrainpred-t1).T#@(ttrainpred-t1).T
        testpredandobs=(ttestpred-t_test)@(ttestpred-t_test).T#@(ttestpred-t_test).T
        trainerms.append(np.sqrt(trainpredandobs/trainpredandobs.size))
        testerms.append(np.sqrt(testpredandobs/testpredandobs.size))
plotDataError(Mdivided,trainerms,Mdivided,testerms,['Train Error','Test Error'])

"""
Discussing which Model value to use for the first problem. I believe due to the error generated, it
is sort of hard to tell which M since the errors tend to be flunctuating

So given whatever Error diference there is between the training and the testing data. We must
select the models that doesn't produce so much error against each other.

So i believe Model sizes of 0-6 is still acceptable for the methods because the errors aren't very
big compared to going beyond. However some tests are made observing errors get bigger and
bigger depending on the sample size.

Amount of data also affects the data that it can overfit or underfit
"""
trueMu = 4
trueVar = 2
```

```
#Initial prior distribution mean and variance (You should change these parameters to see how
they affect the ML and MAP solutions)
priorMu = 20
priorVar = 1

numDraws = 200 #Number of draws from the true distribution
data_uniform_test= np.array(generateUniformData(30, l, u,gVar )).T

x_test= data_uniform_test[:,0]
"""
================================================================================
=======
================================================================================
=======
============================ Question 2
=======================================
================================================================================
=======
================================================================================
=======
"""
""" ======================  Variable Declaration ========================== """
"""========================= Plot the true distribution =================="""
#plot true Gaussian function
step = 0.01
l = -20
u = 20
x = np.arange(l+step/2,u+step/2,step)
plt.figure(0)
p1 = plt.plot(x, norm(trueMu,trueVar).pdf(x), color='b')
plt.title('Known "True" Distribution')
plt.show()
#x=x_test
mu=np.sum(x)/x.size
priori=1/math.sqrt(2*math.pi*priorVar)*math.e**(-1/(2*priorVar)*(trueMu-priorMu)**2)
#print(priori)
sigma=0.01
MLEcoll=[]
MAPcoll=[]
flipResult = []

"""======================== Perform ML and MAP Estimates =================="""
#Calculate posterior and update prior for the given number of draws
```

```
for flip in range(numDraws):
        flipResult.append(np.random.normal(trueMu,math.sqrt(trueVar),1))
   #print(flipResult)
        MLE=sum(flipResult)/len(flipResult)
        #MAP=MLE*priori
        MLEcoll.append(MLE)
        MAP=(trueVar*priorMu+len(flipResult)*priorVar*MLE)/(len(flipResult)*priorVar+trueVar)
        MAPcoll.append(MAP)
        print('Frequentist/Maximum Likelihood Probability of Heads:' + str(MLE))
        print('Bayesian/MAP Probability of Heads:' + str(MAP))
        priorMu=MAP
        priorVar=trueVar/(1+N)
        #plotgauss(MLE,MAP,flip,['MLE','MAP'])
print (np.asarray(MLEcoll).size)
numdrawsinnp=np.arange(0,numDraws,1)


"""
You should add some code to visualize how the ML and MAP estimates change
with varying parameters.  Maybe over time?  There are many differnt things you could do!
"""

plotconverge(np.asarray(MLEcoll),np.asarray(MAPcoll),numdrawsinnp,['MLE','MAP'])
```

References:
Daniel
Mr. McCurley
Dr. Zare
Nick
Akash
(sorry, i forgot their last names and some of them, i just randomly talked to most of them to gather information)