Lab 6: Bayesian Regression

Bayesian Treatment of Linear Regression

Based on Neil D. Lawrence MLAI2015 version

Modified by Haiping Lu on 27 October 2018

Note by Haiping

- The first person in this notebook refers to Neil D. Lawrence rather than Haiping Lu.
- Headings marked as (*optional*) are background info for your interest, which can be safely skipped if you would like to be more focused on the core and have less interest in the background.
- Related session: you are suggested to review the materials in Session 3.

Overdetermined Systems (*optional*)

In **Session 3**, we motivated the introduction of probability by considering systems where there were more observations than unknowns. In particular we thought about the simple fitting of the gradient and an offset of a line, i.e., linear regression

$$y = mx + c$$

and what happens if we have three pairs of observations of x and y, $\{x_i, y_i\}_{i=1}^3$. We solved this issue by introducing a type of slack variable (http://en.wikipedia.org/wiki/Slack_variable), ϵ_i, known as noise, such that for each observation we had the equation,

$$y_i = mx_i + c + \epsilon_i.$$

Underdetermined System (*optional*)

In contrast, today we'd like to consider the situation where you have more parameters than data in your simultaneous equation. So we have an *underdetermined* system. In fact this set up is in some sense *easier* to solve, because we don't need to think about introducing a slack variable (although it might make a lot of sense from a *modelling* perspective to do so).

In the overdetermined system, we resolved the problem by introducing slack variables, ϵ_i, which needed to be estimated for each point. The slack variable represented the difference between our actual prediction and the true observation. This is known as the *residual*. By introducing the slack variable we now have an additional n variables to estimate, one for each data point, $\{\epsilon_i\}$. This actually turns the overdetermined system into an underdetermined system. Introduction of n variables, plus the original m and c gives us $n + 2$ parameters to be estimated from n observations, which actually makes the system *underdetermined*. However, we then made a probabilistic assumption about the slack variables, we assumed that the slack variables were distributed according to a probability density. And for the moment we have been assuming that density was the Gaussian,

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2),$$

with zero mean and variance σ^2.

Sum of Squares and Probability

In the overdetermined system we introduced a new set of slack variables, $\{\epsilon_i\}_{i=1}^n$, on top of our parameters m and c. We dealt with the variables by placing a probability distribution over them. This gives rise to the likelihood and for the case of Gaussian distributed variables, it gives rise to the sum of squares error. It was Gauss who first made this connection in his volume on "Theoria Motus Corprum Coelestium" (written in Latin)

In [1]:
```
import pods
pods.notebook.display_google_book(id='ORUOAAAAQAAJ', page='213')
```

The relevant section roughly translates as

... It is clear, that for the product $\Omega = h^\mu \pi^{-frac12\mu} e^{-hh(vv+v'v'+v''v''+\dots)}$ to be maximised the sum $vv + v'v' + v''v'' + \text{etc.}$ ought to be minimized. *Therefore, the most probable values of the unknown quantities* $p, q, r, s\text{etc.}$*, should be that in which the sum of the squares of the differences between the functions* $V, V', V''\text{etc}$*, and the observed values is minimized*, for all observations of the same degree of precision is presumed.

It's on the strength of this paragraph that the density is known as the Gaussian, despite the fact that four pages later Gauss credits the necessary integral for the density to Laplace, and it was also Laplace that did a lot of the original work on dealing with these errors through probability. Stephen Stigler's book on the measurement of uncertainty before 1900 (http://www.hup.harvard.edu/catalog.php?isbn=9780674403413) has a nice chapter on this.

In [2]:
```
pods.notebook.display_google_book(id='ORUOAAAAQAAJ', page='217')
```

where the crediting to the Laplace is about halfway through the last paragraph. This book was published in 1809, four years after Legendre presented least squares (./week3.ipynb) in an appendix to one of his chapters on the orbit of comets. Gauss goes on to make a claim for priority on the method on page 221 (towards the end of the first paragraph ...).

In [3]:
```
pods.notebook.display_google_book(id='ORUOAAAAQAAJ', page='221')
```

A Philosophical Dispute: Probabilistic Treatment of Parameters? (*optional*)

The follow up question is whether we can do the same thing with the parameters. If we have two parameters and only one unknown can we place a probability distribution over the parameters, as we did with the slack variables? The answer is yes, and from a philosophical perspective placing a probability distribution over the *parameters* is known as the *Bayesian* approach. This is because Thomas Bayes, in a 1763 essay (http://en.wikipedia.org/wiki/An_Essay_towards_solving_a_Problem_in_the_Doctrine_of_Chances) published at the Royal Society introduced the Bernoulli distribution (http://en.wikipedia.org/wiki/Bernoulli_distribution) with a probabilistic interpretation for the *parameters*. Later statisticians such as Ronald Fisher (http://en.wikipedia.org/wiki/Ronald_Fisher) objected to the use of probability distributions for *parameters*, and so in an effort to discredit the approach the referred to it as Bayesian. However, the earliest practioners of modelling, such as Laplace applied the approach as the most natural thing to do for dealing with unknowns (whether they were parameters or variables). Unfortunately, this dispute led to a split in the modelling community that still has echoes today. It is known as the Bayesian vs Frequentist controversy. From my own perspective, I think that it is a false dichotomy, and that the two approaches are actually complementary. My own research focus is on *modelling* and in that context, the use of probability is vital. For frequenstist statisticians, such as Fisher, the emphasis was on the value of the **evidence** in the data for a particular **hypothesis**. This is known as **hypothesis testing**. The two approaches can be unified because one of the most important

approaches to hypothesis testing is to compute the ratio of the likelihoods (http://en.wikipedia.org/wiki/Likelihood-ratio_test), and the result of applying a probability distribution to the parameters is merely to arrive at a different form of the **likelihood**.

The Bayesian Approach

The aim of this notebook is to study Bayesian approaches to regression. In the Bayesian approach we define a *prior* density over our parameters, m and c or more generally $\mathbf{w}$. This prior distribution gives us a range of expected values for our parameter *before* we have seen the data. The objective in Bayesian inference is to then compute the *posterior* density which is the effect on the density of having observed the data. In standard probability notation we write the prior distribution as,

$$p(\mathbf{w}),$$

so it is the *marginal* distribution for the parameters, i.e. the distribution we have for the parameters without any knowledge about the data. The posterior distribution is written as,

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}).$$

So the posterior distribution is the *conditional* distribution for the parameters given the data (which in this case consists of pairs of observations including response variables (or targets), y_i, and covariates (or inputs) $\mathbf{x}_i$, where we are allowing the inputs to be multivariate.

The posterior is recovered from the prior using *Bayes' rule*. Which is simply a rewriting of the product rule. We can recover Bayes rule as follows. The product rule of probability tells us that the joint distribution is given as the product of the conditional and the marginal. Dropping the inputs from our conditioning for the moment we have,

$$p(\mathbf{w}, \mathbf{y}) = p(\mathbf{y}|\mathbf{w})p(\mathbf{w}),$$

where we see we have related the joint density to the prior density and the *likelihood* from our previous investigation of regression,

$$p(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{n} \mathcal{N}(y_i|\mathbf{w}^\top \mathbf{x}_i, \sigma^2)$$

which arises from the assumption that our observation is given by

$$y_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i.$$

In other words this is the Gaussian likelihood we have been fitting by minimizing the sum of squares. Have a look at Session 3 notebook as a reminder.

We've introduce the likelihood, but we don't have relationship with the posterior, however, the product rule can also be written in the following way

$$p(\mathbf{w}, \mathbf{y}) = p(\mathbf{w}|\mathbf{y})p(\mathbf{y}),$$

where here we have simply used the opposite conditioning. We've already introduced the *posterior* density above. This is the density that represents our belief about the parameters *after* observing the data. This is combined with the *marginal likelihood*, sometimes also known as the evidence. It is the marginal likelihood, because it is the original likelihood of the data with the parameters marginalised, $p(\mathbf{y})$. Here it's conditioned on nothing, but in practice you should always remember that everything here is conditioned on things like model choice: which set of basis functions. Because it's a regression problem, its also conditioned on the inputs. Using the equality between the two different forms of the joint density we recover

$$p(\mathbf{w}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{y})}$$

where we divided both sides by $p(\mathbf{y})$ to recover this result. Let's re-introduce the conditioning on the input locations (or covariates), $\mathbf{X}$ to write the full form of Bayes' rule for the regression problem.

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{w}, \mathbf{X})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}$$

where the posterior density for the parameters given the data is $p(\mathbf{w}|\mathbf{y}, \mathbf{X})$, the marginal likelihood is $p(\mathbf{y}|\mathbf{X})$, the prior density is $p(\mathbf{w})$ and our original regression likelihood is given by $p(\mathbf{y}|\mathbf{w}, \mathbf{X})$. It turns out that to compute the posterior the only things we need to do are define the prior and the likelihood. The other term on the right hand side can be computed by *the sum rule*. It is one of the key equations of Bayesian inference, the expectation of the likelihood under the prior, this process is known as marginalisation,

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{w}, \mathbf{X})p(\mathbf{w})\text{d}\mathbf{w}$$

I like the term marginalisation, and the description of the probability as the *marginal likelihood*, because (for me) it somewhat has the implication that the variable name has been removed, and (perhaps) written in the margin. Marginalisation of a variable goes from a likelihood where the variable is in place, to a new likelihood where **all possible values** of that variable (under the prior) have been **considered and weighted** in the integral.

This implies that all we need for specifying our model is to define the likelihood and the prior. We already have our likelihood from our earlier discussion, so our focus now turns to the prior density.

The Bayesian Controversy: Philosophical Underpinnings (*optional*)

A segment from the lecture in 2012 on philsophical underpinnings.

In [4]:
```
from datetime import timedelta
start=int(timedelta(hours=0, minutes=20, seconds=15).total_seconds())
from IPython.display import YouTubeVideo
YouTubeVideo('AvlnFnvFw_0',start=start)
```

Out[4]:

The Prior Density

Let's assume that the prior density is given by a zero mean Gaussian, which is independent across each of the parameters,

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \alpha\mathbf{I})$$

In other words, we are assuming, for the prior, that each element of the parameters vector, w_i, was drawn from a Gaussian density as follows

$$w_i \sim \mathcal{N}(0, \alpha)$$

Let's start by assigning the parameter of the prior distribution, which is the variance of the prior distribution, α.

In [5]:
```
# set prior variance on w
alpha = 4.
# set the order of the polynomial basis set
order = 5
# set the noise variance
sigma2 = 0.01
```

Generating from the Model

A very important aspect of probabilistic modelling is to *sample* from your model to see what type of assumptions you are making about your data. In this case that involves a two stage process.

1. Sample a candiate parameter vector from the prior.
2. Place the candidate parameter vector in the likelihood and sample functions conditiond on that candidate vector.
3. Repeat to try and characterise the type of functions you are generating.

Given a prior variance (as defined above) we can now sample from the prior distribution and combine with a basis set to see what assumptions we are making about the functions *a priori* (i.e. before we've seen the data).

Firstly we compute the basis function matrix. We will do it both for our training data, and for a range of prediction locations (`x_pred`).

In [6]:
```
import numpy as np
data = pods.datasets.olympic_marathon_men()
x = data['X']
y = data['Y']
num_data = x.shape[0]
num_pred_data = 100 # how many points to use for plotting predictions
x_pred = np.linspace(1890, 2016, num_pred_data)[:, None] # input locations for predi
```

now let's build the basis matrices. We define the polynomial basis as follows.

In [7]:
```
def polynomial(x, degree, loc, scale):
    degrees = np.arange(degree+1)
    return ((x-loc)/scale)**degrees
```

In [8]:
```
loc = 1950.
scale = 1.
degree = 5.
Phi_pred = polynomial(x_pred, degree=degree, loc=loc, scale=scale)
Phi = polynomial(x, degree=degree, loc=loc, scale=scale)
```

Sampling from the Prior

Now we will sample from the prior to produce a vector $\mathbf{w}$ and use it to plot a function which is representative of our belief *before* we fit the data. To do this we are going to use the properties of the Gaussian density and obtain a sample from a *standard normal* using the function `np.random.normal` .

Scaling Gaussian-distributed Variables

First, let's consider the case where we have one data point and one feature in our basis set. In otherwords $\mathbf{f}$ would be a scalar, $\mathbf{w}$ would be a scalar and $\mathbf{\Phi}$ would be a scalar. In this case we have

$$f = \phi w$$

If w is drawn from a normal density,

$$w \sim \mathcal{N}(\mu_w, c_w)$$

and ϕ is a scalar value which we are given, then properties of the Gaussian density tell us that

$$\phi w \sim \mathcal{N}\left(\phi\mu_w, \phi^2 c_w\right)$$

Let's test this out numerically. First we will draw 200 samples from a standard normal,

In [9]:
```
w_vec = np.random.normal(size=200)
```

We can compute the mean of these samples and their variance

In [10]:
```
print('w sample mean is ', w_vec.mean())
print('w sample variance is ', w_vec.var())
```

```
w sample mean is  0.014463071350089894
w sample variance is  0.8444294124271058
```

These are close to zero (the mean) and one (the variance) as you'd expect. Now compute the mean and variance of the scaled version,

In [11]:
```
phi = 7
f_vec = phi*w_vec
print('True mean should be phi*0 = 0.')
print('True variance should be phi*phi*1 = ', phi*phi)
print('f sample mean is ', f_vec.mean())
print('f sample variance is ', f_vec.var())
```

```
True mean should be phi*0 = 0.
True variance should be phi*phi*1 =  49
f sample mean is  0.10124149945062913
f sample variance is  41.37704120892818
```

If you increase the number of samples then you will see that the sample mean and the sample variance begin to converge towards the true mean and the true variance. Obviously adding an offset to a sample from `np.random.normal` will change the mean. So if you want to sample from a Gaussian with mean `mu` and standard deviation `sigma` one way of doing it is to sample from the standard normal and scale and shift the result, so to sample a set of w from a Gaussian with mean μ and variance α,

$$w \sim \mathcal{N}(\mu, \alpha)$$

We can simply scale and offset samples from the *standard normal*.

In [12]:
```
mu = 4 # mean of the distribution
alpha = 2 # variance of the distribution
w_vec = np.random.normal(size=200)*np.sqrt(alpha) + mu
print('w sample mean is ', w_vec.mean())
print('w sample variance is ', w_vec.var())
```

```
w sample mean is  4.182484487992662
w sample variance is  2.1613809173233527
```

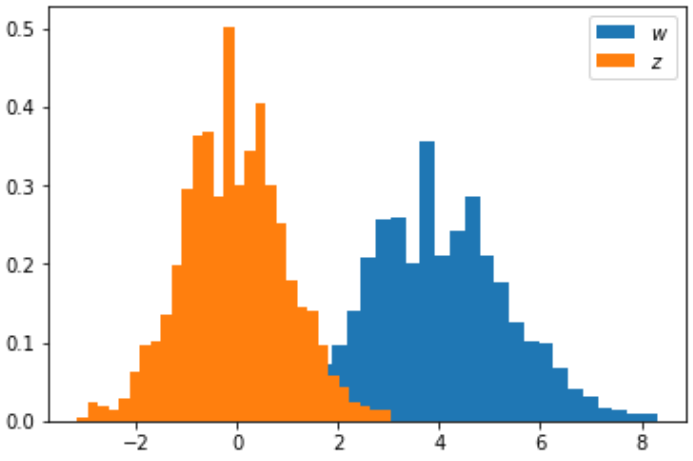
Here the `np.sqrt` is necesssary because we need to multiply by the standard deviation and we specified the variance as `alpha`. So scaling and offsetting a Gaussian distributed variable keeps the variable Gaussian, but it effects the mean and variance of the resulting variable.

To get an idea of the overal shape of the resulting distribution, let's do the same thing with a histogram of the results.

In [13]:
```
# First the standard normal
import matplotlib.pyplot as plt
%matplotlib inline
z_vec = np.random.normal(size=1000) # by convention, in statistics, z is often used
w_vec = z_vec*np.sqrt(alpha) + mu
# plot normalized histogram of w, and then normalized histogram of z on top
plt.hist(w_vec, bins=30, normed=True)
plt.hist(z_vec, bins=30, normed=True)
plt.legend(('$w$', '$z$'))
```

```
C:\Users\eehpl\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWa
rning: The 'normed' kwarg is deprecated, and has been replaced by the 'density'
kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[13]:
```
<matplotlib.legend.Legend at 0x254e981e208>
```



Now **re-run** this histogram with 100,000 samples and check that the both histograms look qualitatively Gaussian.

Sampling from the Prior

Let's use this way of constructing samples from a Gaussian to check what functions look like *a priori*. The process will be as follows. First, we sample a random vector of K dimensional from `np.random.normal`. Then we scale it by $\sqrt{\alpha}$ to obtain a prior sample of $\mathbf{w}$.

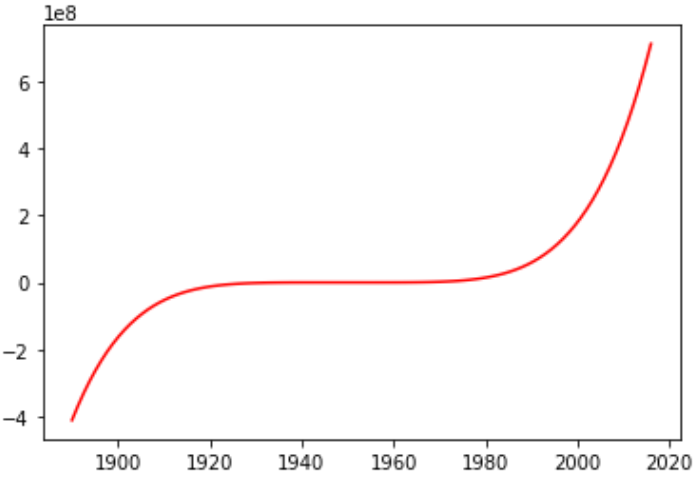
In [14]:
```
K = int(degree) + 1
z_vec = np.random.normal(size=K)
w_sample = z_vec*np.sqrt(alpha)
print(w_sample)
```

```
[ 0.28730628  0.69318767 -1.47790903 -0.47947297  1.26594254  0.55132007]
```

Now we can combine our sample from the prior with the basis functions to create a function,

In [15]:
```
f_sample = np.dot(Phi_pred,w_sample)
plt.plot(x_pred.flatten(), f_sample.flatten(), 'r-')
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x254e9afc390>]
```



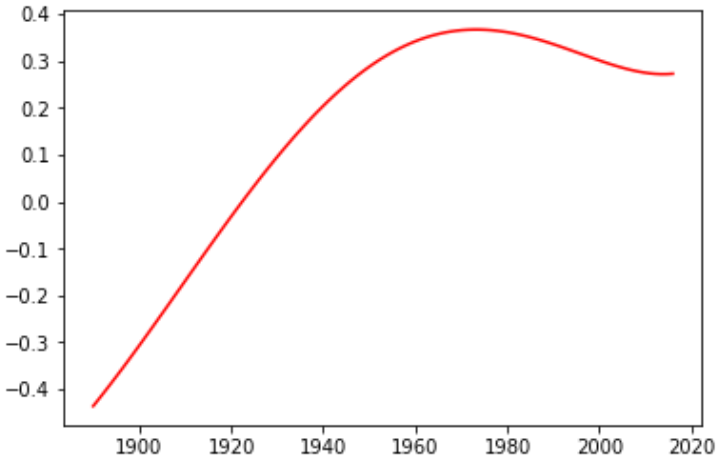
This shows the recurring problem with the polynomial basis. Our prior allows relatively large coefficients for the basis associated with high polynomial degrees. Because we are operating with input values of around 2000, this leads to output functions of very high values. The fix we have used for this before is to rescale our data before we apply the polynomial basis to it. Above, we set the scale of the basis to 1. Here let's set it to 100 and try again.

In [16]:
```
scale = 100.
Phi_pred = polynomial(x_pred, degree=degree, loc=loc, scale=scale)
Phi = polynomial(x, degree=degree, loc=loc, scale=scale)
```

Now we need to recompute the basis functions from above,

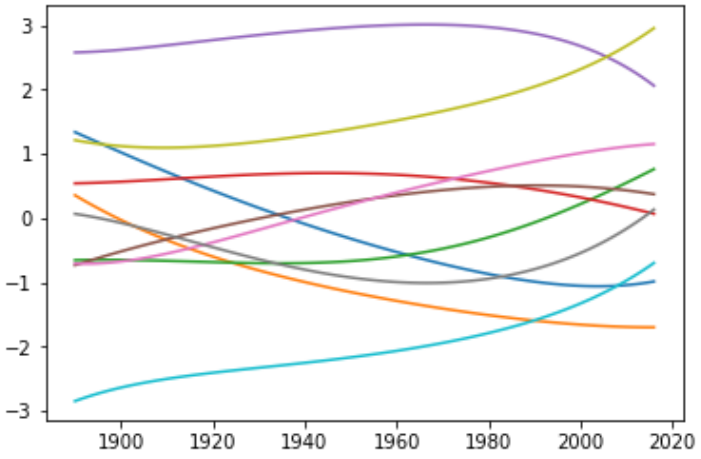
In [17]:
```
f_sample = np.dot(Phi_pred,w_sample)
plt.plot(x_pred.flatten(), f_sample.flatten(), 'r-')
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x254e9b05780>]
```



Now let's loop through some samples and plot various functions as samples from this system,

In [18]:
```
num_samples = 10
K = int(degree)+1
for i in range(num_samples):
    z_vec = np.random.normal(size=K)
    w_sample = z_vec*np.sqrt(alpha)
    f_sample = np.dot(Phi_pred,w_sample)
    plt.plot(x_pred.flatten(), f_sample.flatten())
```



The predictions for the mean output can now be computed. We want the expected value of the predictions under the posterior distribution. In matrix form, the predictions can be computed as

$$\mathbf{f} = \mathbf{\Phi w}.$$

This involves a matrix multiplication between a fixed matrix $\mathbf{\Phi}$ and a vector $\mathbf{w}$ that is drawn from a distribution. Because $\mathbf{w}$ is drawn from a distribution, this imples that $\mathbf{f}$ should also be drawn from a distribution. There are two distributions we are interested in though. We have just been sampling from the *prior* distribution to see what sort of functions we get *before* looking at the data. In Bayesian inference, we need to compute the *posterior* distribution and sample from that density.

Bayesian Inference

The process of Bayesian inference involves combining the prior, $p(\mathbf{w})$ with the likelihood, $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ to form the posterior, $p(\mathbf{w}|\mathbf{y}, \mathbf{x})$ through Bayes' rule,

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y})}$$

We've looked at the samples for our function $\mathbf{f} = \mathbf{\Phi w}$, which forms the mean of the Gaussian likelihood, under the prior distribution. I.e. we've sampled from $p(\mathbf{w})$ and multiplied the result by the basis matrix. With the Baye's rule above, we can sample from the posterior density, $p(\mathbf{w}|\mathbf{y}, \mathbf{x})$, and check that the new samples fit do correspond to the data, i.e. we can that the updated distribution includes information from the data set.

Computing the Posterior - Univariate Case

We will now attempt to compute the *posterior distribution*. In the lecture we went through the maths that allows us to compute the posterior distribution for the univariate case, i.e., the posterior mean and *variance*.

Bayesian Inference in the Univariate Case

This video below is Prof Neil Lawrence's lecture about Bayesian inference across the single parameter, the offset c, illustrating how the prior and the likelihood combine in one dimension to form a posterior.

In [19]:
```
from datetime import timedelta
start=int(timedelta(hours=0, minutes=0, seconds=15).total_seconds())
YouTubeVideo('AvlnFnvFw_0',start=start)
```

Out[19]:

Multivariate Bayesian Inference (*derivations are optional*)

This section of the lecture talks about how we extend the idea of Bayesian inference for the multivariate case. It goes through the multivariate Gaussian and how to complete the square in the linear algebra as we managed below.

In [20]:
```
start=int(timedelta(hours=0, minutes=22, seconds=42).total_seconds())
YouTubeVideo('Os1iqgpelPw', start=start)
```

Out[20]:

Compute the posterior distribution for multivariate $\mathbf{w}$, i.e., the posterior mean and *covariance*. This distribution is also Gaussian,

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}, \sigma^2) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_w, \mathbf{C}_w)$$

with covariance, $\mathbf{C}_w$, given by

$$\mathbf{C}_w = \left(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \alpha^{-1}\mathbf{I}\right)^{-1}$$

whilst the mean is given by

$$\boldsymbol{\mu}_w = \mathbf{C}_w\sigma^{-2}\boldsymbol{\Phi}^\top\mathbf{y}$$

Let's compute the posterior covariance and mean, then we'll sample from these densities to have a look at the posterior belief about $\mathbf{w}$ once the data has been accounted for.

Assignment Question 1

Compute the covariance for $\mathbf{w}$ given the training data (using the formulae above) and call the resulting variable `w_cov`. Compute the mean for $\mathbf{w}$ given the training data (using the formulae above) and call the resulting variable `w_mean`. Assume that $\sigma^2 = 0.01$

10 marks

In [21]:
```
# Question 1 Answer Code
# Write code for you answer to this question in this box
# Do not delete these comments, otherwise you will get zero for this answer.
# Make sure your code has run and the answer is correct *before* submitting your not
sigma2 =
w_cov =
w_mean =
```

```
  File "<ipython-input-21-46cf73a4b9d8>", line 5
    sigma2 =
             ^
SyntaxError: invalid syntax
```

Sampling from the Posterior

Before we were able to sample the prior values for the mean *independently* from a Gaussian using `np.random.normal` and scaling the result. However, observing the data *correlates* the parameters. Recall this from the first lab where we had a correlation between the offset, c and the slope m which caused such problems with the coordinate ascent algorithm. We need to sample from a *correlated* Gaussian. For this we can use `np.random.multivariate_normal` .

In [ ]:
```
w_sample = np.random.multivariate_normal(w_mean.flatten(), w_cov)
f_sample = np.dot(Phi_pred,w_sample)
plt.plot(x_pred.flatten(), f_sample.flatten(), 'r-')
plt.plot(x, y, 'rx') # plot data to show fit.
```

Now let's sample several functions and plot them all to see how the predictions fluctuate.

In [ ]:
```
for i in range(num_samples):
    w_sample = np.random.multivariate_normal(w_mean.flatten(), w_cov)
    f_sample = np.dot(Phi_pred,w_sample)
    plt.plot(x_pred.flatten(), f_sample.flatten())
plt.plot(x, y, 'rx') # plot data to show fit.
```

This gives us an idea of what our predictions are. These are the predictions that are consistent with data and our prior. Try plotting different numbers of predictions. You can also try plotting beyond the range of where the data is and see what the functions do there.

Rather than sampling from the posterior each time to compute our predictions, it might be better if we just summarised the predictions by the expected value of the output funciton, $f(x)$, for any particular input. If we can get formulae for this we don't need to sample the values of $f(x)$ we might be able to compute the distribution directly. Fortunately, in the Gaussian case, we can use properties of multivariate Gaussians to compute both the mean and the variance of these samples.

Properties of Gaussian Variables

Gaussian variables have very particular properties, that many other densities don't exhibit. Perhaps foremost amoungst them is that the sum of any Gaussian distributed set of random variables also turns out to be Gaussian distributed. This property is much rarer than you might expect.

Sum of Gaussian-distributed Variables

The sum of Gaussian random variables is also Gaussian, so if we have a random variable y_i drawn from a Gaussian density with mean μ_i and variance σ_i^2,

$$y_i \sim \mathcal{N}\left(\mu_i, \sigma_i^2\right)$$

Then the sum of K independently sampled values of y_i will be drawn from a Gaussian with mean $\sum_{i=1}^K \mu_i$ and variance $\sum_{i=1}^K \sigma_i^2$,

$$\sum_{i=1}^K y_i \sim \mathcal{N}\left(\sum_{i=1}^K \mu_i, \sum_{i=1}^K \sigma_i^2\right).$$

Let's try that experimentally. First let's generate a vector of samples from a standard normal distribution, $z \sim \mathcal{N}(0, 1)$, then we will scale and offset them, then keep adding them into a vector `y_vec`.

Sampling from Gaussians and Summing Up

In [ ]:
```
K = 10 # how many Gaussians to add.
num_samples = 1000 # how many samples to have in y_vec
mus = np.linspace(0, 5, K) # mean values generated linearly spaced between 0 and 5
sigmas = np.linspace(0.5, 2, K) # sigmas generated linearly spaced between 0.5 and 2
y_vec = np.zeros(num_samples)
for mu, sigma in zip(mus, sigmas):
    z_vec = np.random.normal(size=num_samples) # z is from standard normal
    y_vec += z_vec*sigma + mu # add to y z*sigma + mu

# now y_vec is the sum of each scaled and off set z.
print('Sample mean is ', y_vec.mean(), ' and sample variance is ', y_vec.var())
print('True mean should be ', mus.sum())
print('True variance should be ', (sigmas**2).sum(), ' standard deviation ', np.sqrt
```

Of course, we can histogram `y_vec` as well.

In [ ]:
```
plt.hist(y_vec, bins=30, normed=True)
plt.legend('$y$')
```

Matrix Multiplication of Gaussian Variables

We are interested in what our model is saying about the sort of functions we are observing. The fact that summing of Gaussian variables leads to new Gaussian variables, and scaling of Gaussian variables *also* leads to Gaussian variables means that matrix multiplication (which is just a series of sums and scales) also leads to Gaussian densities. Matrix multiplication is just adding and scaling together, in the formula, $\mathbf{f} = \mathbf{\Phi}\mathbf{w}$ we can extract the first element from $\mathbf{f}$ as

$$f_i = \boldsymbol{\phi}_i^\top \mathbf{w}$$

where $\boldsymbol{\phi}$ is a column vector from the ith row of $\mathbf{\Phi}$ and f_i is the ith element of $\mathbf{f}$. This vector inner product itself merely implies that

$$f_i = \sum_{j=1}^K w_j \phi_{i,j}$$

and if we now say that w_i is Gaussian distributed, then because a scaled Gaussian is also Gaussian, and because a sum of Gaussians is also Gaussian, we know that f_i is also Gaussian distributed. It merely remains to work out its mean and covariance. The results are below and the derviations follow. You need the results to work out the assignment while the derivations are *optional* and only for your interest and information.

The expectation under the prior is given by

$$\langle \mathbf{f} \rangle_{\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha\mathbf{I})} = \mathbf{0}$$

and the covariance is

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha\mathbf{I})} = \alpha\mathbf{\Phi}\mathbf{\Phi}^\top$$

Derivations of the mean and covariance of $\mathbf{f}$ (*optional*)

We can do this by looking at the expectation under a Gaussian distribution. The expectation of the mean vector is given by

$$\langle \mathbf{f} \rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \int \mathbf{f}\mathcal{N}\left(\mathbf{w}|\boldsymbol{\mu},\mathbf{C}\right)\text{d}\mathbf{w} = \int \mathbf{\Phi}\mathbf{w}\mathcal{N}\left(\mathbf{w}|\boldsymbol{\mu},\mathbf{C}\right)\text{d}\mathbf{w} = \mathbf{\Phi}\int \mathbf{w}\mathcal{N}\left(\mathbf{w}|\boldsymbol{\mu},\mathbf{C}\right)\text{d}\mathbf{w} = \mathbf{\Phi}\boldsymbol{\mu}$$

Which is straightforward. The expectation of $\mathbf{f} = \mathbf{\Phi}\mathbf{w}$ under the Gaussian distribution for $\mathbf{f}$ is simply $\mathbf{f} = \mathbf{\Phi}\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is the *mean* of the Gaussian density for $\mathbf{w}$. Because our prior distribution was Gaussian with zero mean, the expectation under the prior is given by

$$\langle \mathbf{f} \rangle_{\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha\mathbf{I})} = \mathbf{0}$$

The covariance is a little more complicated. A covariance matrix is defined as

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \left\langle \mathbf{f}\mathbf{f}^\top \right\rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} - \langle \mathbf{f} \rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} \langle \mathbf{f} \rangle^\top_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})}$$

we've already computed $\langle \mathbf{f} \rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \mathbf{\Phi}\boldsymbol{\mu}$ so we can substitute that in to recover

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \left\langle \mathbf{f}\mathbf{f}^\top \right\rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} - \mathbf{\Phi}\boldsymbol{\mu}\boldsymbol{\mu}^\top\mathbf{\Phi}^\top$$

So we need the expectation of $\mathbf{f}\mathbf{f}^\top$. Substituting in $\mathbf{f} = \mathbf{\Phi}\mathbf{w}$ we have

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \left\langle \mathbf{\Phi}\mathbf{w}\mathbf{w}^\top\mathbf{\Phi}^\top \right\rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} - \mathbf{\Phi}\boldsymbol{\mu}\boldsymbol{\mu}^\top\mathbf{\Phi}^\top$$

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \mathbf{\Phi}\left\langle \mathbf{w}\mathbf{w}^\top \right\rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})}\mathbf{\Phi}^\top - \mathbf{\Phi}\boldsymbol{\mu}\boldsymbol{\mu}^\top\mathbf{\Phi}^\top$$

Which is dependent on the second moment of the Gaussian,

$$\left\langle \mathbf{w}\mathbf{w}^\top \right\rangle_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \mathbf{C} + \boldsymbol{\mu}\boldsymbol{\mu}^\top$$

that can be substituted in to recover,

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \mathbf{\Phi}\mathbf{C}\mathbf{\Phi}^\top$$

so in the case of the prior distribution, where we have $\mathbf{C} = \alpha\mathbf{I}$ we can write

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha\mathbf{I})} = \alpha\mathbf{\Phi}\mathbf{\Phi}^\top$$

This implies that the prior we have suggested for $\mathbf{w}$, which is Gaussian with a mean of zero and covariance of $\alpha\mathbf{I}$ suggests that the distribution for $\mathbf{f}$ is also Gaussian with a mean of zero and covariance of $\alpha\mathbf{\Phi}\mathbf{\Phi}^\top$.

Compute the Marginal Likelihood

Since our observed output, $\mathbf{y}$, is given by a noise corrupted variation of $\mathbf{f}$, the final distribution for $\mathbf{y}$ is given as

$$\mathbf{y} = \mathbf{f} + \epsilon$$

where the noise, ϵ, is sampled from a Gaussian density $\epsilon \sim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$. So, in other words, we are taking a Gaussian distributed random value $\mathbf{f}$,

$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \alpha\mathbf{\Phi}\mathbf{\Phi}^\top\right)$$

and adding to it another Gaussian distributed value, $\epsilon \sim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$, to form our data observations, $\mathbf{y}$. Once again the sum of two (multivariate) Gaussian distributed variables is also Gaussian, with a mean given by the sum of the means (both zero in this case) and the covariance given by the sum of the covariances. So we now have that the marginal likelihood for the data, $p(\mathbf{y})$ is given by

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y}|\mathbf{0}, \alpha\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma^2\mathbf{I}\right)$$

This is our *implicit* assumption for $\mathbf{y}$ given our prior assumption for $\mathbf{w}$.

Computing the Mean and Error Bars of the Functions

These ideas together, now allow us to compute the mean and error bars of the predictions. The mean prediction, before corrupting by noise is given by,

$$\mathbf{f} = \mathbf{\Phi}\mathbf{w}$$

in matrix form. This gives you enough information to compute the predictive mean.

Assignment Question 2

Compute the predictive mean for the function at all the values of the basis function given by `Phi_pred` . Call the vector of predictions `f_pred_mean` . Plot the predictions alongside the data. We can also compute what the training error was. Use the output from your model to compute the predictive mean, and then compute the sum of squares error of that predictive mean.

$$E = \sum_{i=1}^{n}(y_i - \langle f_i \rangle)^2$$

where $\langle f_i \rangle$ is the expected output of the model at point x_i.

15 marks

In [ ]:
```
# Question 2 Answer Code
# Write code for you answer to this question in this box
# Do not delete these comments, otherwise you will get zero for this answer.
# Make sure your code has run and the answer is correct *before* submitting your not

# compute mean under posterior density
f_pred_mean =

# plot the predictions

# compute mean at the training data and sum of squares error
f_mean =
sum_squares =
print('The error is: ', sum_squares)
```

Computing Error Bars

Finally, we can compute error bars for the predictions. The error bars are the standard deviations of the predictions for $\mathbf{f} = \mathbf{\Phi}\mathbf{w}$ under the posterior density for $\mathbf{w}$. The standard deviations of these predictions can be found from the variance of the prediction at each point. Those variances are the diagonal entries of the covariance matrix. We've already computed the form of the covariance under Gaussian expectations,

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu},\mathbf{C})} = \mathbf{\Phi}\mathbf{C}\mathbf{\Phi}^\top$$

which under the posterior density is given by

$$\text{cov}(\mathbf{f})_{\mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_w,\mathbf{C}_w)} = \mathbf{\Phi}\mathbf{C}_w\mathbf{\Phi}^\top$$

Assignment Question 3

The error bars are given by computing the standard deviation of the predictions, f. For a given prediction f_i the variance is $\text{var}(f_i) = \langle f_i^2 \rangle - \langle f_i \rangle^2$. This is given by the diagonal element of the covariance of $\mathbf{f}$,

$$\text{var}(f_i) = \boldsymbol{\phi}_{i,:}^\top \mathbf{C}_w \boldsymbol{\phi}_{i,:}$$

where $\boldsymbol{\phi}_{i,:}$ is the basis vector associated with the input location, $\mathbf{x}_i$.

Plot the mean function and the error bars for your basis.

20 marks

In [ ]:
```
# Question 3 Answer Code
# Write code for you answer to this question in this box
# Do not delete these comments, otherwise you will get zero for this answer.
# Make sure your code has run and the answer is correct *before* submitting your not

# Compute variance at function values
f_pred_var =
f_pred_std =

# plot the mean and error bars at 2 standard deviations above and below the mean
```

Validation

Now we will test the generalisation ability of these models. Firstly we are going to use hold out validation to attempt to see which model is best for extrapolating.

Assignment Question 4

Now split the data into training and *hold out* validation sets. Hold out the data for years after 1980. Compute the predictions for different model orders between 0 and 8. Find the model order which fits best according to *hold out* validation. Is it the same as the maximum likelihood result fom last week?

25 marks

In [ ]:
```
# Question 4 Answer Code
# Write code for you answer to this question in this box
# Do not delete these comments, otherwise you will get zero for this answer.
# Make sure your code has run and the answer is correct *before* submitting your not
```

Assignment Question 5

Now we will use leave one out cross validation to attempt to see which model is best at interpolating. Do you get the same result as for hold out validation? Compare plots of the hold out validation area for different degrees and the cross validation error for different degrees. Why are they so different? Select a suitable polynomial for characterising the differences in the predictions. Plot the mean function and the error bars for the full data set (to represent the leave one out solution) and the training data from the hold out experiment. Discuss your answer.

30 marks

In [ ]:
```
# Question 5 Answer Code
# Write code for you answer to this question in this box
# Do not delete these comments, otherwise you will get zero for this answer.
# Make sure your code has run and the answer is correct *before* submitting your not
```

Question 5 Answer

In [ ]:
```

```