```
figsize( 9, 3.5 )
```

Chapter 2: Modeling with PyMC

This chapter introduces more PyMC design patterns, examples + prob distributions to compliment the examples, MCMC (I'm not sure how to present this part yet. I might leave the more technical arguments for the appendix). This is still flexible

---

A little more on PyMC

Parent and Child relationships

To help with terminology, and to be consistent with PyMC, we introduce *parent and children* variables.

- *parent variables* are variables that influence another variable.
- *children variable* are variables that are affected by other variables, i.e. are the subject of parent variables.

Variables can be both parent and children variables. For example, condsider the PyMC code below

```
import pymc as mc

poi_parameter = mc.Exponential( "Poisson_param", 1 )

data_generator = mc.Poisson("data_generator", poi_parameter )
```

`poi_parameter` controls the parameter of `data_generator`, hence influence its value. The former are parents of the latter. By symmetry, `data_generator` is a child of `poi_parameter`.

This nomenclature is introduced to help us describe relationships in PyMC modeling. You can access a variables children and parent variables using the `children` and `parents` methods attached to variables.

```
print poi_parameter.children
print
print data_generator.parents
```

```
set([<pymc.distributions.Poisson 'data_generator' at
0x0000000015A54E48>])

{'mu': <pymc.distributions.Exponential 'Poisson_param' at
0x0000000015A54D68>}
```

Of course a child can have more than one parent, and parent can have many children.

PyMC Variables

PyMC is concerned with two types of programming variables: `stochastic` and `deterministic`.

- *deterministic variables* are variables that not random if the variables' parents are not random. This might be confusing at first: a quick mental test to check is *if I knew all of variable x's parent variables, I could determine completely what x is.* This is relevant as we are dealing with random variables.
- *stochastic variables* are variables that are not deterministic, i.e., even if you knew all the values of the variables' parents (if it even had any parents), it would still be random. Included in this catagory are instances of classes `Poisson`, `DiscreteUniform`, and `Exponential`.

Stochastic variables

Initializing a stochastic variable requires a `name` argument, plus any additional parameters. For example:

```
some_variable = mc.DiscreteUniform( "discrete_uni_var", 0, 4 )
```

where 0,4 are the `DiscreteUniform`-specific bounds on the random variable. The PyMC docs contain the specific parameters for stochastic variables. (Or use `??` if you are using IPython!)

Rather than creating a Python array of stochastic variables, addressing the `size` keyword in the call to `Stochastic` creates multivariate (indepedent) array. The array behaves like a Numpy array when used like one, and references to its `value` attribute and `random()` method return Numpy arrays.

Though often redundant, we can also specify the initial value of stochastic variables by addressing the `value` keyword when calling the class. This does not *fix* the variable to always be `value`. Hence I say it is redundant because the `value` attribute changes as soon as `random` is called (and it is called often in the backend.) There are two main use cases. The first is it isused to include data in the model, something we will see shortly. The second case is for initial numerical stability, though this rarely comes up (we will see example of this occuring below).

Determinstic variables

Since most variables you will be modeling are are stochastic, we distinguish deterministic variables with the `pymc.deterministic` wrapper. This is the easist way, but not the only way, to create deterministic variables. This is not completely true: elementary operations, like addition, exponentials etc. implicity create determinsitic functions.

The use of the `deterministic` wrapper was seen in the previous text-message example. Recall the model for λ looked like:

$$\lambda = \begin{cases} \lambda_1 & \text{if } t < \tau \\ \lambda_2 & \text{if } t \geq \tau \end{cases}$$

And in PyMC code:

```
lambda_1 = mc.Exponential( "lambda_1",  1 )
lambda_2 = mc.Exponential( "lambda_2", 1 )
tau = mc.DiscreteUniform( "tau", lower = 0, upper = 10 )


@mc.deterministic
def lambda_( tau = tau, lambda_1 = lambda_1, lambda_2 = lambda_2 ):
    out = np.zeros( 10 )
    out[:tau] = lambda_1 #lambda before tau is lambda1
    out[tau:] = lambda_2 #lambda after tau is lambda1
    return out
```

Clearly, if τ, λ_1 and λ_2 are known, then λ is known completely, hence it is deterministic.

All PyMC variables also expose a `value` attribute. This method produces the *current* (random) value of the variable, given the variable's parents. For example:

```
np.set_printoptions(precision=3)
print "lambda_1.value = %.3f"%lambda_1.value
print "lambda_2.value = %.3f"%lambda_2.value
print "tau.value = %.3f"%tau.value
print
print "lambda_.value = ",lambda_.value
```

```
lambda_1.value = 0.820
lambda_2.value = 0.801
tau.value = 5.000

lambda_.value =  [ 0.82   0.82   0.82   0.82   0.82   0.801  0.801
0.801  0.801  0.801]
```

Notice the switch in `lambda_.value`? (10% of you won't, but try it after the next exercise.)

We can also call on a stochastic variables `random()` method, which (given the parent values) will generate a new (random) value.

```
lambda_1.random(), lambda_2.random(), tau.random()
print "lambda_1.value = %.3f"%lambda_1.value
print "lambda_2.value = %.3f"%lambda_2.value
print "tau.value = %.3f"%tau.value
print
print "lambda_.value = ",lambda_.value
```

```
lambda_1.value = 0.334
lambda_2.value = 1.991
tau.value = 10.000

lambda_.value =  [ 0.334  0.334  0.334  0.334  0.334  0.334  0.334
0.334  0.334  0.334]
```

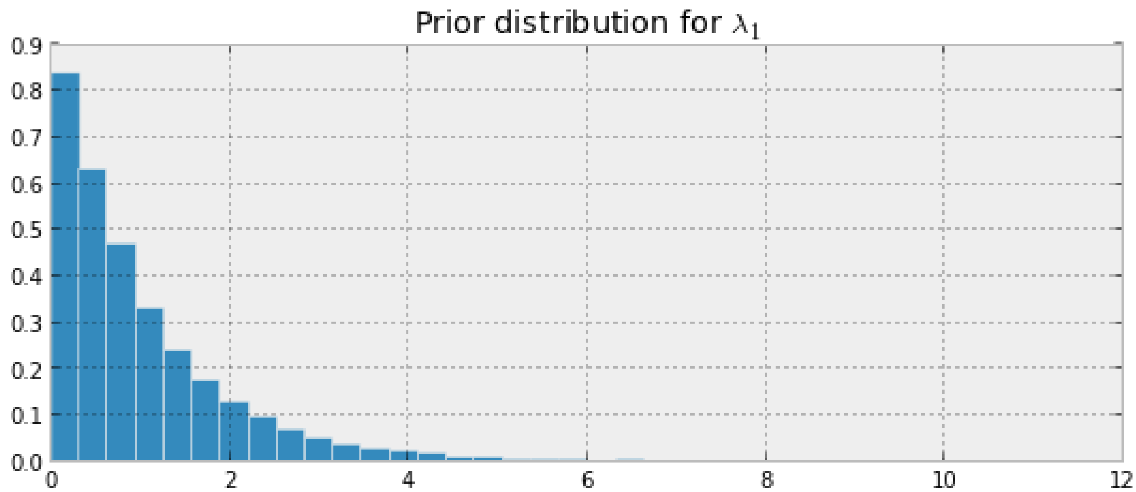
The call to `random` stores a new value into the variables `value` attribute. In fact, this new value is stored in the computers cache for faster recall and efficiency.

Including observations and data

At this point, it may not look like it, but we have fully specified our priors. For example, we can ask and answer questions like "What does my prior distribution of λ_1 look like?"

```
samples = [ lambda_1.random() for i in range( 20000) ]
hist( samples, bins = 35, normed=True )
plt.title( "Prior distribution for $\lambda_1$")
```

```
<matplotlib.text.Text at 0x158382b0>
```



To frame this in the notation of the first chapter, though this is a slight abuse of notation, we have specified $P(A)$. Our next goal is to include data/evidence/observations X into our model.

PyMC stochastic variables constructors have a keyword argument `observed` which accepts a boolean (`False` by default). `observed` has a very simple role: fix the variables current value, i.e. make it immutable. For this to make sense, we have to specify an intial `value` in the class call too. For example:

```
fixed_variable = mc.Poisson( "fxd", 1, value = 10, observed = True )
print "value: ",fixed_variable.value
print "calling .random()"
fixed_variable.random()
print "value: ",fixed_variable.value
```

```
value:  10
calling .random()
value:  10
```

This is how we include data into our models: initializing the stochastic variable to have a *fixed value*.

> ...the stochastic variable to have a *fixed value*.

That might seem strange to hear at first. In fact, Bayesian analysis sees observed data as simply fixed parameters. Taking this to its logical conclusion, any predictions made by Bayesian analysis is seen as fitting another parameter in the model.

Modeling approaches

A good starting place to modeling is to think about *how your data might have been generated*. Think from a god-like position, and try to think about how *you* would recreate the data. For example, in the last chapter, we investigated text message data:

I. We started by thinking "what is the best random variable to describe this data?" A Poisson random variable is a good candidate because it assigns probabilities to count data.
II. Next, we think, "Ok, assuming texts are Poisson-distributed, what do I need for the Poisson distribution?" Well, the Poisson distribution has a parameters λ.
III. Do we know λ? No. In fact, we have a suspicion that there are *two* λ values, one for the earlier behaviour and one for the latter behaviour. We don't know when the behaviour switches though.
IV. What is a good distribution for the two λs? The exponential is good, as it assigns probabilites to positive real numbers.
V. Do we know what the parameter α might be? No. We could continue and assign a disitribution to α, but it's better to stop once we reach a set level of ignorrance: whereas we have a prior belief about λ, ("it probably changes over time", "it's likely between 1 and 8", etc.), we don't really have any strong beliefs about α. So it's best to stop here.
VI. What is a good value for α then? We think that the λs are between 1-8, so if we set α really low (which corresponds to larger probabilties on high values) we are not reflecting our prior well. The same is true if α is too high. A good idea for α as to reflect our belief is to set the value so that the mean of λ, given α, is equal to our observed mean.

In this way, we are formalizing how the data might have been created.

Example: Poisson Regression

Perhaps the most important result from medical research was the *now obvious* link between *smoking and cancer*. We'll try to establish a link using Bayesian methods. We have a decision here: should we include a prior that biases us towards there existing a significant link between smoking and cancer? I think we should act like scientists at the turn of the century, and assume there's is no *a priori* reason to assume a link.

Our dataset contains 36 cohorts, each cohort has variables:

- age: in five-year age groups coded 1 to 9 for 40-44, 45-49, 50-54, 55-59, 60-64, 65-69, 70-74, 75-79, 80+.
- cigar/pipe smoking status: 1 if the only smoke cigars/pipes, 0 else.
- cigar & cigarette smoking status: 1 if the smoke *both* cigars/pipes and cigarettes, 0 else.
- cigarette smoking status: 1 if the only smoke cigarettes, 0 else.
- population: the population, in hundreds of thousands, of the age group *and* smoking status cohort. Denote this P_i.
- deaths: number of lung cancer deaths in a year. Denote this D_i.

Now, this data was not segmented along *what it means to be a smoker*, that is, there are many types of smokers: something to remember.

We will try to fit the *rate of deaths* of lung cancer to age and smoking statuses. We are interested in the quantity:

$$\frac{D_i}{P_i}$$

and how it is effected by the other variables x_i. We could use linear regression, but there are two reasons why this is not the best choice.

I. The variables deaths and population are naturally integers. By only considering their (no-integer) ratio, we are ignoring this peice of information.
II. Linear regression can predict negative values for the rate, which is impossible for our rates to be.

Similarly, how to we connect the rate with the variables anyways? A common approach is to connect them using a *link function*:

$$\frac{D_i}{P_i} = \exp\left(\beta^T x_i + \text{noise}\right)$$

We require the $\exp$ link function because the linear combination of variables may be negative, but we require the rate to be positive.

To make the left-hand a count data, we multiple both sides by P_i, and bring the P_i into the exponential:

$$D_i = \exp\left(\beta^T x_i + \log P_i + \text{noise}\right)$$

This is a common *trick*, called including an *offset* term in the regression. Now we have D_i, integer data, as a function of the other variables. We can model D_i as a Poisson variable:

$$D_i \sim \text{Poi}(\lambda_i)$$

where λ_i is to be determined. As a Poisson variable is fully defined by its parameter λ, we only need to focus on this.

This example is quite different from our last example on text-messaging rates, though the two look similar. We are not trying to estimate a *global* λ, that is a single parameter λ that determines the distributions of all the observations, but we are actual trying to model a unique λ for each data point using the observed variables, i.e. $\lambda_i = f(\mathbf{x}_i)$.

Our conclusions are determined by the posterior distributions of β_1, β_2 and β_3. If the distributions are shifted to be positive, then a 1 in cigarette smoking or cigar smoking status will shift the λ_i forward, resulting in an increase in the rate of deaths. Our task in now to find the posteriors of β_i. We first need a prior for the βs: the most natural being the Normal distrubition, described below.

Normal distributions

A Normal random variable, denoted $X \sim N(\mu, 1/\tau)$, has a distribution with two parameters: the mean, μ, and the *precision*, τ. Those familar with the Normal distribution already have probably seen σ^2 instead of τ. They are infact recipricals of each other. The change was motivated by easier mathematics and is an artifact of Bayesian methods. Just remember: The smaller τ, the larger the spread of the distribution (we are more uncertain); the larger τ, the tighter the distribution (we are more certain). Regardless, τ is always positive.

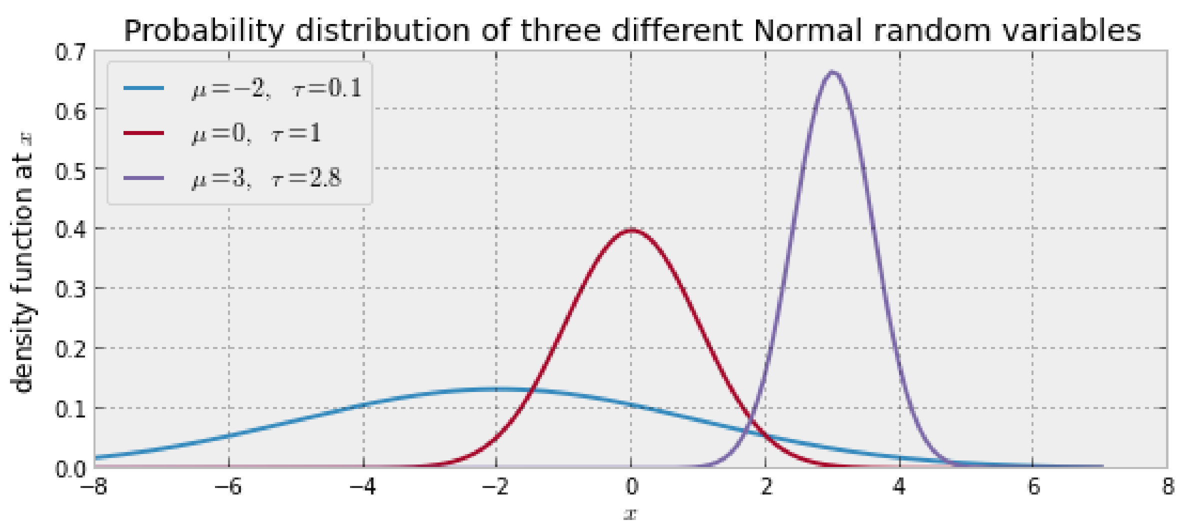
The probability density function of a $N(\mu, 1/\tau)$ random variable is:

$$f(x|\mu,\tau) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2} (x-\mu)^2 \right)$$

We plot some different density functions below.

```
import scipy.stats as stats
nor = stats.norm
x = np.linspace( -8, 7, 150 )

plot( x, nor.pdf( x, -2 , scale = 3 ), label ="$\mu = -2,\;\\tau = %.1f$"%(1./3**2))
plot( x, nor.pdf( x, 0 , 1 ), label = r"$\mu = 0,\;\tau = 1$" )
plot( x, nor.pdf( x, 3, scale = 0.6 ), label ="$\mu = 3,\;\\tau = %.1f$"%(1/0.6**2))
plt.legend(loc = "upper left")
plt.xlabel("$x$")
plt.ylabel("density function at $x$")
plt.title( "Probability distribution of three different Normal random variables" )
```

```
<matplotlib.text.Text at 0x1480e198>
```



A Normal random variable can be any real number, but the variable is very likely to be relatively close to μ. In fact, the expected value of a Normal is equal to its μ parameter:

$$E[X|\mu,\tau] = \mu$$

Smoking and Cancer

We will model the β_i as Normal distrbutions. Let's start the PyMC code:

```
import pymc as mc

data = np.genfromtxt( "chp2data/smoking_death.csv", skip_header = 1,
             delimiter=",", dtype = float )

population = data[:,-2].copy()
deaths = data[:,-1].copy()
data[:,-2] = 1 #replace the last column with a constant to represent beta_0
data = data[:,:-1]

#Instead of creating variable beta_1, beta_2, etc.,
beta = mc.MvNormal( "beta_coefs", mu = np.zeros(5) , \
    tau = 0.0001*np.identity(5), value = np.zeros(5))
print "initial beta.value = ", beta.value

#we'll create a deterministic function that represents the exponential
#of a linear combination
@mc.deterministic
def exp_lin_comb( beta = beta ):
    return np.exp( np.dot( data, beta ) + np.log(population)  )

observations = mc.Poisson( "obs", exp_lin_comb, value = deaths, observed = True )
```

```
initial beta.value =  [ 0.  0.  0.  0.  0.]
```

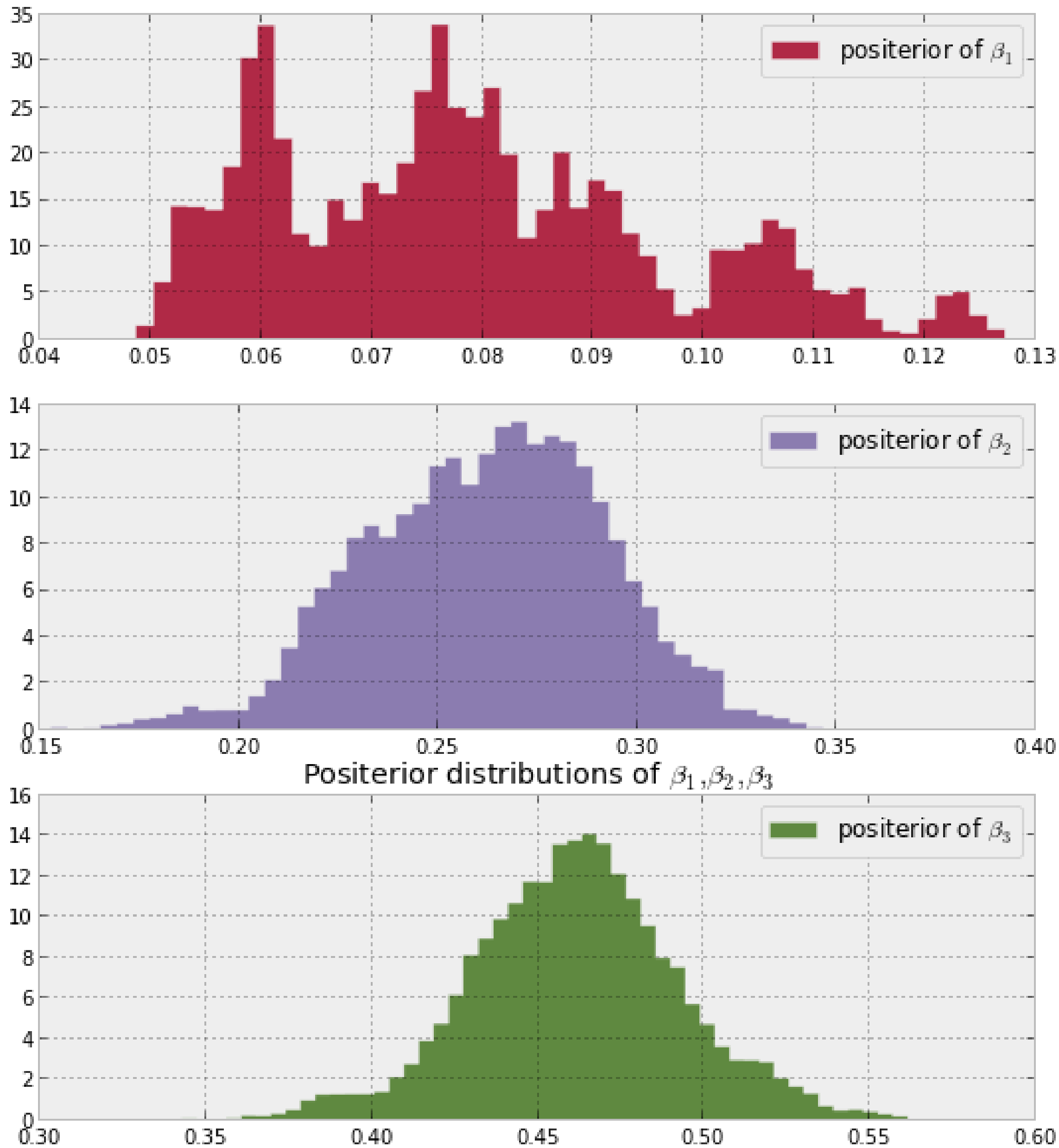
```
model = mc.Model( [observations, beta, exp_lin_comb] )



#mysterious code to be explained in Chapter 3
map_ = mc.MAP( model )
map_.fit()
mcmc = mc.MCMC( model )
mcmc.sample( 1000000, 900000, 2 )
```

```
[***************100%*****************]  1000000 of 1000000 complete
```

```
figsize(9, 10)
#histogram of the samples:
plt.figure()

plt.subplot(311)
plt.title( r"Positerior distributions of $\beta_1, \beta_2, \beta_3$" )
plt.hist( beta_samples[:, 1],histtype='stepfilled', bins = 50, alpha = 0.85, \
        label = r"positerior of $\beta_1$", color = "#A60628",normed = True  )
plt.legend()

plt.subplot(312)
plt.hist( beta_samples[:, 2], histtype='stepfilled', bins = 50, alpha = 0.85, \
        label = r"positerior of $\beta_2$", color = "#7A68A6",normed = True)
plt.legend()

plt.subplot(313)
plt.hist( beta_samples[:, 3], bins = 50, alpha = 0.85,
        label = r"positerior of $\beta_3$", \
         color="#467821", normed = True, histtype='stepfilled' )
plt.legend()
```

```python
print
```

```
<matplotlib.text.Text at 0x193df1d0>
```



What can we say? It looks like all positerior distributions are strictly positive. The odd shape of the posterior of β_1 is noteworthy.

Let's perform some prediction. As we modeled the parameter λ_i in a Poisson distribution, we can find the *expected rate* of deaths. We are taking the average over the posterior distributions:

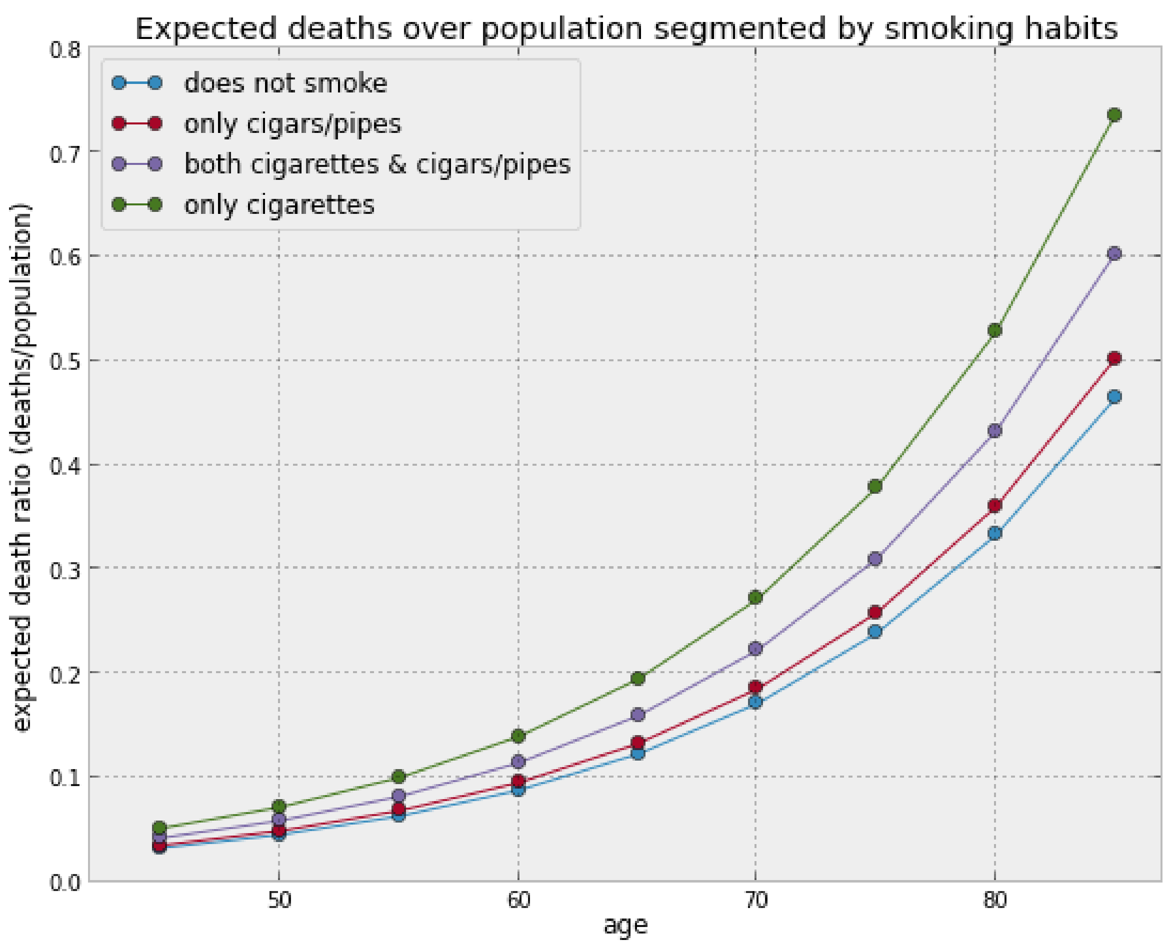
We compute `np.dot( data, beta_samples.T )`. This produces a very large matrix which is each posterior sample multiplied with the each data:

$$\beta_j^T x_i, \quad \text{for all } i, \text{for all } j$$

we then exponentiate this matrix and take the mean over all j (over the posterior samples).

```
figsize(9, 7)
rates = np.mean( np.exp( np.dot( data, beta_samples.T ) ), axis = 1 )

colors = ["#348ABD", "#A60628", "#7A68A6", "#467821"]
labels = ["does not smoke",
          "only cigars/pipes",
          "both cigarettes & cigars/pipes",
          "only cigarettes"]

for i in range(4):
    plt.plot( 40 + 5*data[9*i:9*(i+1),0], rates[9*i:9*(i+1)], marker = 'o',
         color = colors[i], label = labels[i], lw=1)

plt.legend(loc="upper left")
plt.xlabel( "age" )
plt.ylabel("expected death ratio (deaths/population)" )
plt.title( "Expected deaths over population segmented by smoking habits")
plt.xlim( 42, 87 )
```

```
(42, 87)
```



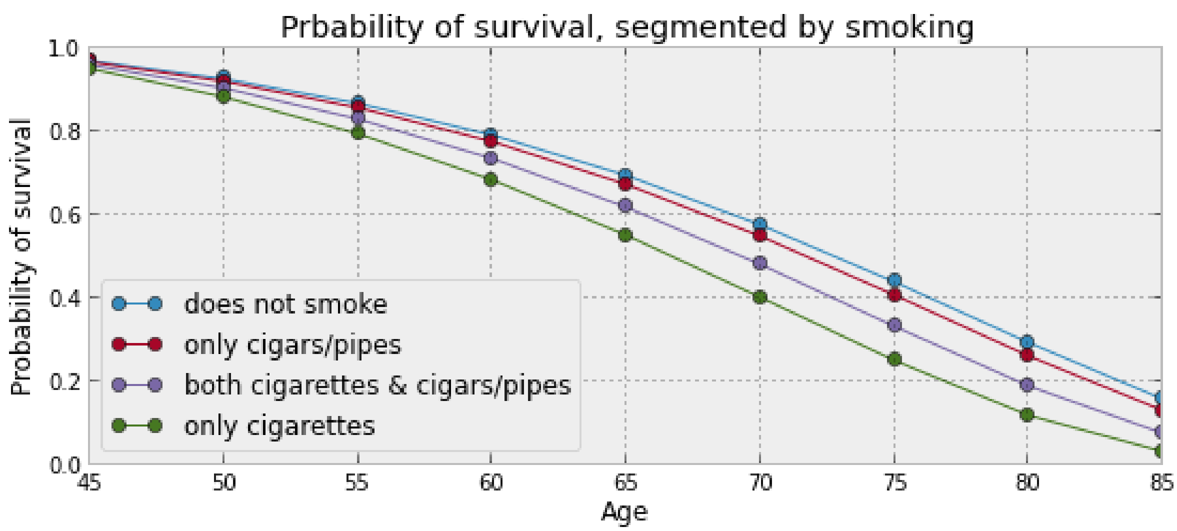
Clearly smoking affects how long you live.

We can interpret these rates are probabilities, for example, 0.032 is the rate of a non-smokers dying at age 40, which is equilivant to the probablity of a randomly selected non-smoker aged 40-45 will die. Hence 1- 0.032 = 0.968 is the probability he/she survives to 45. Hence, (1- 0.032)(1 - 0.045) is the probability a non-smoker, aged 40 will survive until 50, etc. Let's plot this.

```
figsize( 9, 3.5 )
```

```python
prob_survival  = 1-rates

colors = ["#348ABD", "#A60628", "#7A68A6", "#467821"]
labels = ["does not smoke",
          "only cigars/pipes",
          "both cigarettes & cigars/pipes",
          "only cigarettes"]

for i in range(4):
    _p = prob_survival[ 9*i:9*(i+1) ].cumprod()
    plt.plot( 40 + 5*data[9*i:9*(i+1),0], _p, marker = 'o',
         color = colors[i], label = labels[i], lw=1)

plt.legend(loc="lower left")
plt.xlabel("Age")
plt.ylabel("Probability of survival")
plt.title("Prbability of survival, segmented by smoking")
```

```
<matplotlib.text.Text at 0x21a350f0>
```



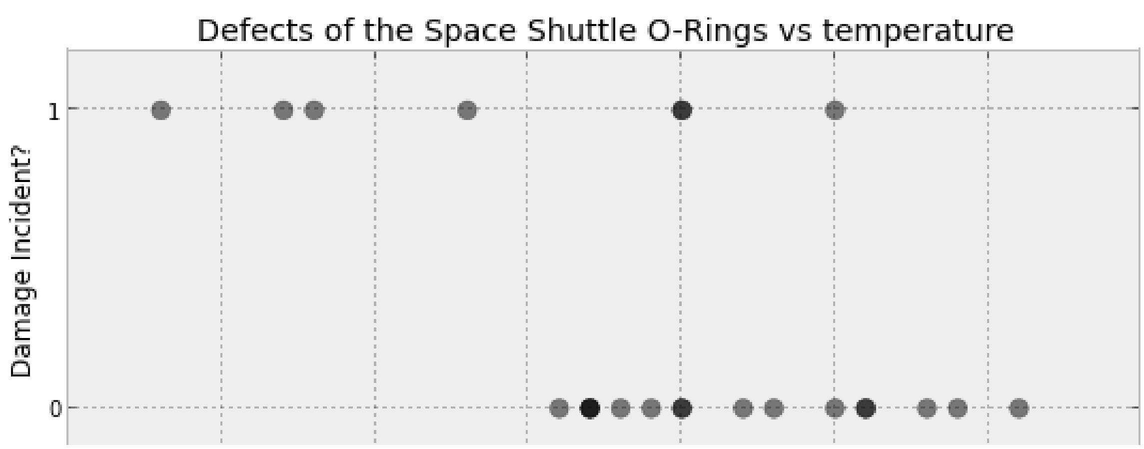
It appears that smoking cigarettes will almost certainly kill you before age 85, whereas not smoking provides you with 18% chances to live this long. So, for you young folks out there, which path do you want to take?

Example: Challenger Space Shuttle Disaster

On January 28, 1986, the twenty-fifth flight of the U.S. space shuttle pro-gram ended in disaster when one of the rocket boosters of the Shuttle Challenger exploded shortly after lift-off, killing all seven crew members. The presidential commission on the accident concluded that it was caused by the failure of an O-ring in a field joint on the rocket booster, and that this failure was due to a faulty design that made the O-ring unacceptably sensitive to a number of factors including outside temperature. Of the previous 24 flights, data were available on failures of O-rings on 23, (one was lost at sea), and these data were discussed on the evening preceding the Challenger launch, but unfortunately only the data corresponding to the 7 flights on which there was a damage incident were considered important and these were thought to show no obvious trend. The data are shown below (see [1]):

```python
figsize( 9, 3.5 )
challenger_data = np.genfromtxt("chp2data/challenger_data.csv", skip_header = 1, \
                                usecols=[1,2], missing_values="NA", delimiter=",")
#drop the NA values
challenger_data = challenger_data[ ~np.isnan(challenger_data[:,1]) ]

#plot it, as a function of tempature (the first column)
print "Temperature, O-Ring failure"
print challenger_data

plt.scatter( challenger_data[:,0], challenger_data[:,1], s = 75, color="k",
 alpha = 0.5)
plt.yticks([0,1])
plt.ylabel("Damage Incident?")
plt.xlabel("Outside temperature (Farhenhit)" )
plt.title("Defects of the Space Shuttle O-Rings vs temperature")
print
```

```
Temperature, O-Ring failure
[[ 66.    0.]
 [ 70.    1.]
 [ 69.    0.]
 [ 68.    0.]
 [ 67.    0.]
 [ 72.    0.]
 [ 73.    0.]
 [ 70.    0.]
 [ 57.    1.]
 [ 63.    1.]
 [ 70.    1.]
 [ 78.    0.]
 [ 67.    0.]
 [ 53.    1.]
 [ 67.    0.]
 [ 75.    0.]
 [ 70.    0.]
 [ 81.    0.]
 [ 76.    0.]
 [ 79.    0.]
 [ 75.    1.]
 [ 76.    0.]
 [ 58.    1.]]
```



It looks clear that *the probability* of damage incidents occuring increases as the outside temperature decreases. We are interested in modeling the probability here because it does not look like there is a strict threshold between temperature and a damage incident.

We need a function of temperature, call it $p(t)$, that is bounded between 0 and 1 (so as to model a probability) and changes from 0 to 1 as we decrease temperature. There are actually many such functions, but the most popular choice is the *logistic function.*

$$p(t) = \frac{1}{1 + e^{\;\beta t}}$$

In this model, β is the variable we are uncertain about. Below is the function plotted for $\beta = 1, 3, -5$.

```
def logistic( x, beta):
    return 1.0/( 1.0 + np.exp( beta*x) )
x = np.linspace( -4, 4, 100 )
plt.plot(x, logistic( x, 1), label = r"$\beta = 1$")
plt.plot(x, logistic( x, 3), label = r"$\beta = 3$")
plt.plot(x, logistic( x, -5), label = r"$\beta = -5$")
plt.legend()
```

<matplotlib.legend.Legend at 0x2226cda0>

But something is missing. In the plot above, the probability changes quickly only near zero, but in our data above the probability changes around 65 to 70. We need to add a *bias* term to our logistic function:

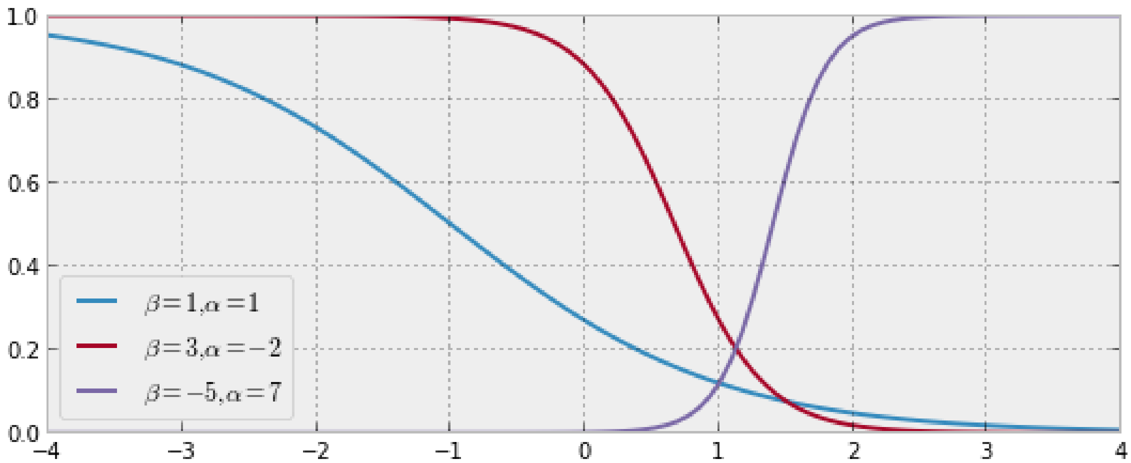
$$p(t) = \frac{1}{1 + e^{\;\beta t + \alpha}}$$

This means we can change where the curve changes. Some plots are below, with differing α.

```
def logistic( x, beta, alpha):
    return 1.0/( 1.0 + np.exp( beta*x + alpha) )
```

```python
x = np.linspace( -4, 4, 100 )
plt.plot(x, logistic( x, 1, 1), label = r"$\beta = 1, \alpha = 1$")
plt.plot(x, logistic( x, 3, -2), label = r"$\beta = 3, \alpha = -2$")
plt.plot(x, logistic( x, -5, 7), label = r"$\beta = -5, \alpha = 7$")
plt.legend(loc="lower left")
```

```
<matplotlib.legend.Legend at 0x2224aa20>
```



Adding a constant term α amounts to shifting the curve left or right (hence why it called a *bias.*)

Let's start modeling this in PyMC. The β, α paramters have no reason to be positive, bounded or relativly large, so there are best modeled by a Normal random variable.

```python
import pymc as mc

temperature = challenger_data[:,0]
defect_ind= challenger_data[:,1]

beta = mc.Normal( "beta", 0, 0.001, value = 0 )
alpha = mc.Normal( "alpha", 0, 0.001, value = 0 )

@mc.deterministic
def p( temp = temperature, alpha = alpha, beta = beta):
    return 1.0/( 1. + np.exp( beta*temperature + alpha) )
```

We have our probabilities, but how do we connect them to our observed data? A *Bernoulli* random variable with parameter p (denoted $\text{Ber}(p)$, is a random variable that takes value 1 with probability p, and 0 else. Thus, our model can look like:

$$\text{Defect Incident} \sim \text{Ber}(\ p(t)\ )$$

where $p(t)$ is our logistic function. Notice in the above code we had to set the values of `beta` and `alpha` to 0. The reason for this is that if `beta` and `alpha` are very large, they make `p` equal to 1 or 0. Unfortunatly, `mc.Bernoulli` does not like probabilities of 0 or 1, though they are mathematically well-defined probabilties. So by setting the values to `0`, we set `p` to a resonable starting value. This has no effect on our results, nor does it mean we are including information in our prior. It is simply a computational caveat in PyMC.

```
p.value
```

```
array([ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5])
```

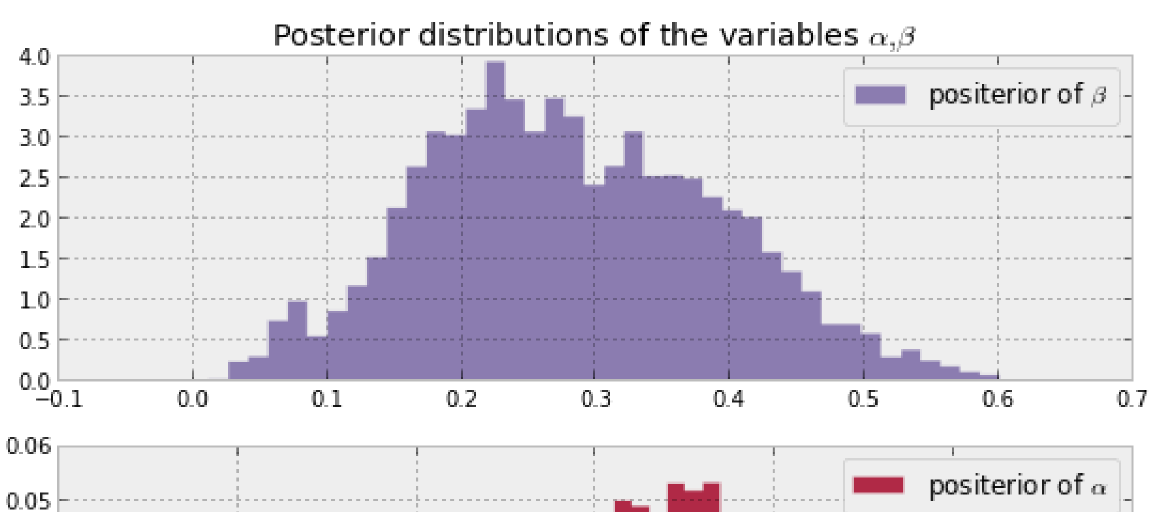
```
observed = mc.Bernoulli( "bernoulli_obs", p, \
                        value = defect_ind, observed=True)

model = mc.Model( {"observed":observed, "beta":beta, "alpha":alpha} )

#mysterious code to be explained in Chapter 3
map_ = mc.MAP(model)
map_.fit()
mcmc = mc.MCMC( model )
mcmc.sample( 260000, 220000, 3 )
```

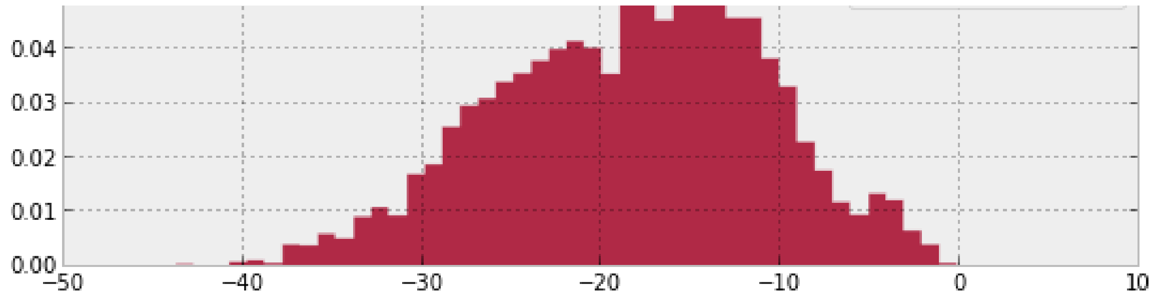
```
[****************100%*****************]  260000 of 260000 complete
```

We have trained our model on the observed data, now we can sample values from the positerior. Let's look at the positerior distributions for α and β:

```
alpha_samples = mcmc.trace( 'alpha' )[:]
beta_samples = mcmc.trace( 'beta' )[:]

figsize(9, 6)

#histogram of the samples:
plt.subplot(211)
plt.title(r"Posterior distributions of the variables $\alpha, \beta$")
plt.hist( beta_samples, histtype='stepfilled', bins = 45, alpha = 0.85, \
        label = r"positerior of $\beta$", color = "#7A68A6",normed = True )
plt.legend()

plt.subplot(212)
plt.hist( alpha_samples, histtype='stepfilled', bins = 45, alpha = 0.85, \
        label = r"positerior of $\alpha$", color = "#A60628",normed = True )
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2549f400>
```





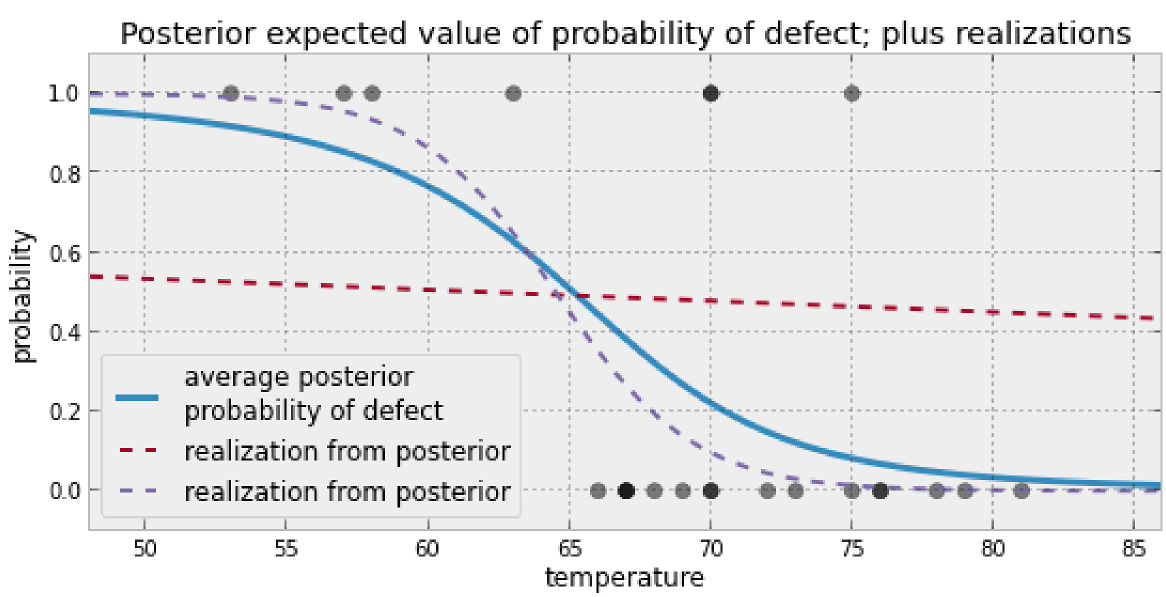
Regarding the spread of the data, we are very uncertain about what the paramters may be(though considering the low sample size and the large overlap of defects to no-defects this behaviour is perhaps expected). The alternative, frequentist-logistic regression would return two scalars that are likely very different from the true values.

Next, let's look at the *expected probability* for a specific value of the temperature.

```
t = np.linspace( temperature.min() - 5, temperature.max()+5, 50 )
linear_combination = np.dot( beta_samples[:,None], t[None,:]) + alpha_samples[:,None]
p = 1.0/(1.0 + np.exp( linear_combination ) )
mean_prob_t = p.mean(axis=0)
```

```
figsize( 9, 4)
plt.plot( t, mean_prob_t, lw = 3, label = "average posterior \nprobability \
of defect")
plt.plot( t, p[0, :], ls="--",label="realization from posterior" )
plt.plot( t, p[-2, :], ls="--", label="realization from posterior"  )
plt.scatter( temperature, defect_ind, color = "k", s = 50, alpha = 0.5 )
plt.title("Posterior expected value of probability of defect; plus realizations")
plt.legend(loc= "lower left")
plt.ylim( -0.1, 1.1 )
plt.xlim( t.min(), t.max() )
plt.ylabel("probability")
plt.xlabel("temperature")
```

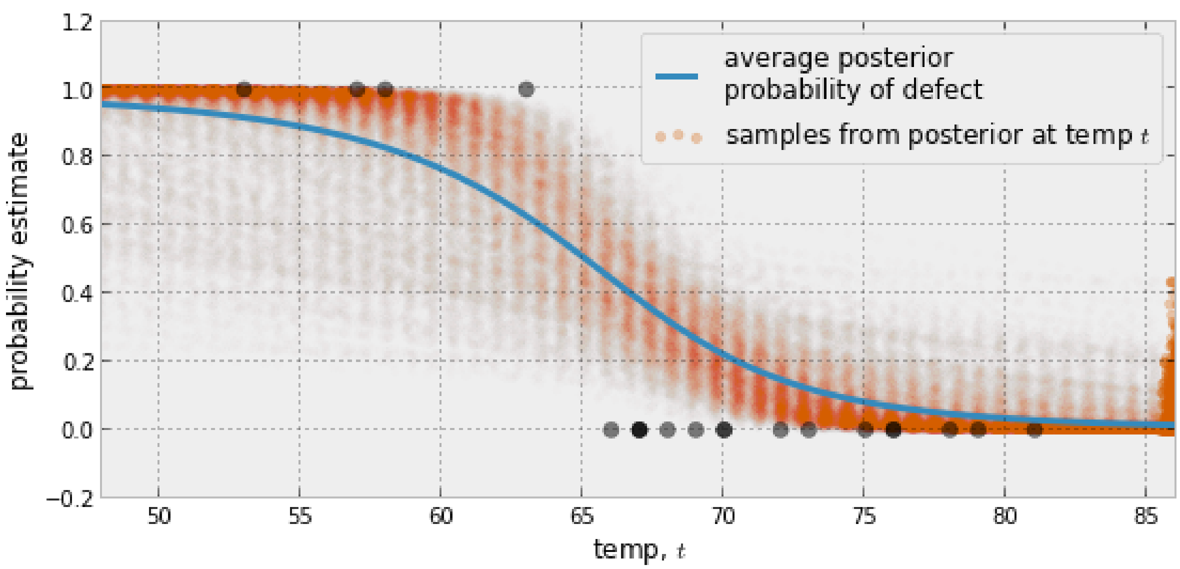
```
<matplotlib.text.Text at 0x25d4ff28>
```



Above we also plotted two possible realizations of what the actual underlying system might be. Both are as equally likely as any other. Notice they end to deviate from the expected value line around 60 degrees. An interesting question is for what temperatures are we most uncertain about the probability being? Below we plot the expected value line and and the associated distribution at each temperature.

```
plt.plot( t, mean_prob_t, lw = 3, label = "average posterior \nprobability \
of defect")
n = 2000
for i, _t in enumerate(t[:-1]):
    plt.scatter( _t*ones( n ) + 0.20*np.random.randn(n), p[:n,i], alpha = 0.009,\
                 color = "#D55E00")

#what's a better way to only have a for loop and include it in the legend? and
# have it somewhat show. You can see an artifact at t = 85.
plt.scatter( t[-1]*ones( n ) + 0.20*np.random.randn(n), p[:n,-1], alpha = 0.3,\
        color = "#D55E00", label = "samples from posterior at temp $t$")

plt.xlim( t.min(), t.max() )
plt.legend()
plt.scatter( temperature, defect_ind, color = "k", s = 50, alpha = 0.5 )
plt.xlabel("temp, $t$")
plt.ylabel("probability estimate" )
```

<matplotlib.text.Text at 0x25aef278>

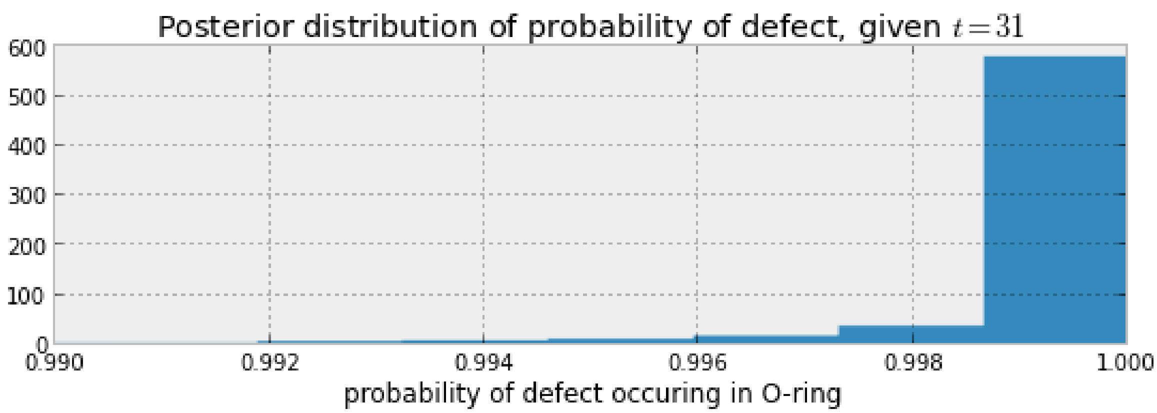


We can see that as the temperature nears 60 degrees, the distributions spread out over [0,1] quickly. As we pass 70 degrees, the distributions tighten again. This can give us insight about how to proceed next: we should probably test more O-rings around 60-65 temperature to get a better estimate of probabilities in that range.

What about the day of the Challenger disaster?

On the day of the Challenger disaster, the outide temperature was 31 degrees fahrenheit. What is the positerior distribution of a defect, given this temperature? The distribution it plotted below. It looks almost garunteed that the Challenger was going to be subject to defective O-rings.

```
figsize(9, 2.5)
t = 31
p = beta_samples*31 + alpha_samples
prob_31 = 1.0/(1.0 + exp(p) )

plt.xlim( 0.99, 1)
plt.hist( prob_31, bins =160, normed = True, histtype='stepfilled' )
plt.title( "Posterior distribution of probability of defect, given $t = 31$")
plt.xlabel( "probability of defect occuring in O-ring" )

print
```

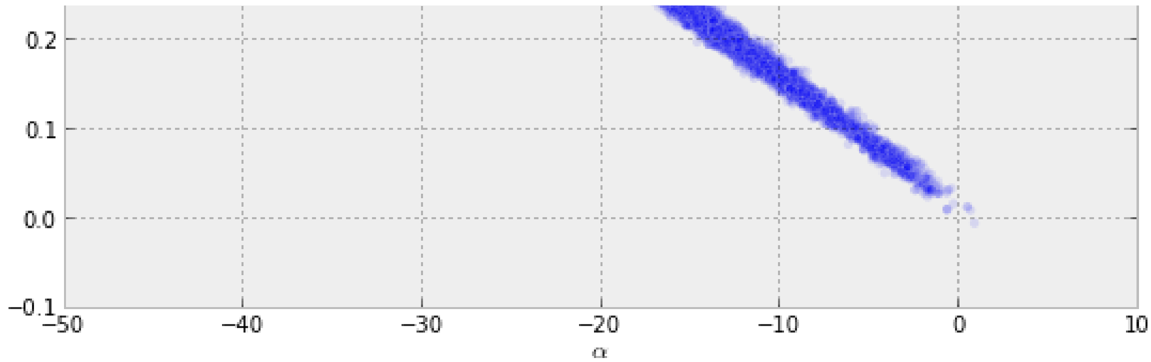


Exercises

1. Try plotting α samples versus β samples.

```
#type your code here.

plt.scatter( alpha_samples, beta_samples, alpha = 0.1 )
plt.title( "Why does the plot look like this?" )
plt.xlabel( r"$\alpha$")
plt.ylabel( r"$\beta$")
```

```
<matplotlib.text.Text at 0x25a65438>
```



2. How would add a prior belief that smoking causes more deaths in first example?

References

- [1] Dalal, Fowlkes and Hoadley (1989),JASA, 84, 945-957.
- [2] German Rodriguez. Datasets. In WWS509. Retrieved 30/01/2013, from http://data.princeton.edu/wws509/datasets/#smoking.