Groups:

Group 1:
Amelia Camino
Jake Magee
Amanda Muratore
Julissa Sarmiento

Group 2:
Finian Ashmead
Francis Burk
Mykola Chernyashevskyy
Mohamed Ismail
Ehteshamul Karim

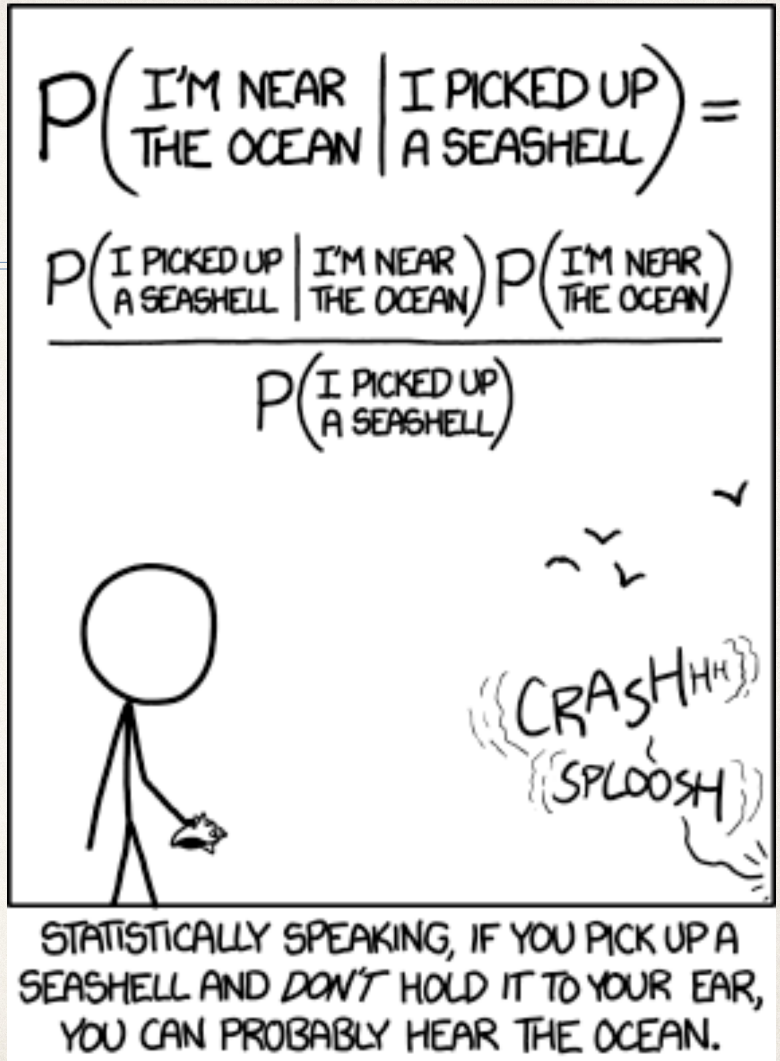
Group 3:
Cullen Abelson
Alia Dawood
Mira Salman
Yunchong Zhang

Group 4:
Nathalie Chicoine
Lauren Elicker
Yoki Salcedo
Marcos Tamargo-Arizmendi

Probability Distributions

Statistics and Data Science

Spring 2025



http://xkcd.com/1236/

Goals for today: you should be able to…

- **Lecture 5 notebook:** Explain what the binomial distribution is and apply it
- Choose priors for a binomial distribution
- **Lecture 6 notebook**: Perform Bayesian statistical inference with the binomial distribution
- Use np.where() and np.isfinite()
- **Reminder: Homework due Friday!**

Review: key rules of probability to remember

- ➡If events A and B are ***independent*** then: $p(A \text{ AND } B) = p(A) \times p(B)$
- ➡The ***conditional*** probability A is true, given that B is true:
  $p(A \mid B) = p(A \text{ AND } B)/p(B)$
- ➡$p(A \mid B) = p(A)$ if - and only if - A and B are independent.
- ➡In general, $p(A \text{ OR } B) = p(A) + p(B) - p(A \text{ AND } B)$
- ➡***Marginalization:*** $p(A) = \Sigma\, p(A \mid B_i)\, p(B_i)$

Let's generalize from coins and dice to more complicated situations…

- We'll come back to applications of Bayes' theorem, but need more background to go further.
- Till now, we've mostly talked about probabilities with just a few possibilities, equally likely. However, we can instead consider arbitrary probabilities as a function of continuous variables: a ***probability density function*** or PDF (for functions of discrete variables, the term is ***probability mass function*** or PMF).
- Probability density functions just have to be nonnegative everywhere, and integrate to 1 (or have sum 1 in the discrete case). For a continuous PDF $f(x)$,

$$p(a < x < b) = \int_a^b f(x)\ dx$$

- Sometimes, the term ***probability distribution*** is used instead of PDF.

More on distributions

- It is sometimes helpful to look at the fraction of the integral of *f(x)* which is below some value. This is the ***cumulative distribution function*** or CDF, generally written *F(x)*:

$$F(x) = \int_{-\infty}^{x} f(y)\, dy$$

- Another useful thing to look at can be the expectation value of some function *g(x)*:

$$\mathbf{E}g = \int_{-\infty}^{\infty} g(x)\, f(x)\, dx$$

- The expectation value is the value of *g(x)*, weighted by the probability of each possible x.

Expectation values

- Ex.: suppose I'll pay you \$5 if a coin comes up heads, and you pay me \$5 if it comes up tails. What is the expectation value of your winnings after a coin toss?
- Some important expectation values are based on the moments of the distribution (remember basic mechanics...):
  - The ***mean*** is, simply, the first moment of f(x) - compare to the center of mass in mechanics. It provides a measure of the ***location*** or ***center*** of $f(x)$.

$$\mu = \mu_1 = \mathrm{E}x = \int_{-\infty}^{\infty} x\, f(x)\, dx$$

Variance and standard deviation (*of a probability distribution*)

- We often are interested in the width, not just central value, of a probability distribution (e.g. an error bar). We can measure this with a second moment (compare to the moment of inertia about the center of mass):

$$\sigma^2 = \mu_2 = \mathrm{E}(x-\mu)^2 = \int_{-\infty}^{\infty} (x-\mu)^2 \; f(x) \; dx$$

- In statistics, μ_2 is called the ***variance***, and is equal to the ***standard deviation*** squared. σ provides a measure of width in the same units as x.
- We can also construct statistics to describe other moments: the ***skewness*** = μ_3 / σ^3 = $\mathbf{E}(x\text{-}\mu)^3 / \sigma^3$ describes how asymmetric a distribution is, the ***kurtosis*** = μ^4 / σ^4 describes how flattened it is, etc. A Gaussian distribution has 0 skewness and kurtosis of 3.

Although $f(x)$ can take arbitrary form (if nonnegative and normalized to have integral 1), there are dozens of well-studied cases.

- Let's start, again, with dice rolls. What is the probability of getting M ones when we roll N dice? Open up today's jupyter notebook...
- E.g.: if we roll 100 dice, how many 1's will we observe, in total?

```
import numpy.random as random
import numpy as np
nsims=int(1E5)
prob=1/6.
is_one=(random.rand(????,100) < prob)
ndice=100
# plot a histogram of the total # of 1's from each sim
plt.hist(np.sum(is_one[:,0:ndice],????) )
```

- We'd like to try this with ndice=2,5,10,50,100, and do a few repeats, plotting each time. This can get to be a pain retyping things or copying and pasting -- so I've made a function for you in a module file.

Using another module

- Last week, we wrote a function to give the result of 2 coin flips.
- This time, I made a module for you that rolls dice: **dice.py**
- Download **dice.py** from Courseweb to a directory in your `PYTHONPATH:` e.g., `~/python/`
- Open the file up in vscode to see what is in it…

Contents of dice.py

```
import numpy.random as random

import numpy as np

import matplotlib.pyplot as plt
```

Contents of dice.py

```
def rolldice(nsims):
# nsims is number of simulations to do
nsims =int(nsims)
    prob=1/6.
    is_one=(random.rand(nsims,100) < prob)
# generate nsims sets of 100 rolls
    ndice_array=[2,5,10,25,100]

    for i,ndice in enumerate(ndice_array):
        plt.figure(i) # create a new figure for each plot
        plt.hist( np.sum(is_one[:,0:ndice],axis=1),
range=(-0.5,ndice+0.5),bins=(ndice + 1))
        plt.title(str(ndice) + ' dice')
 # convert ndice to a string with str(), then use that to title the plot
```

Running the function

- **Import dice and run `dice.rolldice()`, e.g., with 50_000 simulations.**

What would we expect to see?

- Let p = the probability of getting a 1 (=1/6.)
- Each roll of the dice is independent of all others, so
  $p(A \text{ AND } B)=p(A)\, p(B)$ for each combination of die rolls
- e.g. probability of rolling a 1 on the first roll, the second, etc. N times must be
  p^N
- probability of a 1 on the first M rolls, and non-1 all the rest would be:
  $p^M(1-p)^{N-M}$, since each roll is independent
  - the same must be true for *any* specific ordering of the rolls that gives M ones total, as we'll have to have M factors of p and N factors of (1-p)

How many ways can we get M ones?

- If we have N things, there are $N!$ (N factorial) different ways of ordering them.
- However, any case with a 1 coming up on dice 1, 3, and 5 only, say, is indistinguishable, and we shouldn't double-count them.
- There are $M!$ ways to reorder the M cases of p that are indistinguishable, and then we can still reorder the N-M cases of (1-p) - so there are a total of $C(N,M)=N!/(M!(N-M)!)$
- So, summing up the probability of each case with M ones (since they are mutually exclusive), we find:

  $prob(M\ ones) = C(N,M)\ p^M\ (1-p)^{N-M}$

The Binomial Distribution

- We didn't really use the fact that we're looking at dice anywhere in that derivation. In general, if there is a probability p of success, and we do N trials, then:

  $prob(M\ successes) = C(N,M)\ p^M\ (1-p)^{N-M}$

- This formula defines the binomial distribution. This is the distribution that controls coin tosses (like in the homework), or any other set of independent events of fixed probability.
- It has mean = <x>:

  $\mu = N\ p$

- and variance = $\langle x^2 \rangle - (\langle x \rangle)^2$:

  $\sigma^2 = N\ p\ (1-p)$

Testing our distributions

- We expect that the number of ones we get will follow a binomial distribution with N=ndice. Let's test this by modifying the rolldice function in dice.py:

1) Does the data have mean $\mu = N\,p$?

- In python, we can check this with either np.mean or np.sum:

```
print(f'ndice: {ndice}')
print(f'np.mean: {np.mean( np.sum(is_one[:,
     0:ndice],axis =1) ) }')
print(f'np.sum: {np.sum( np.sum(is_one[:,0:ndice],axis=1) )*1./nsims:.4f}')
print(f'Expected mean: {ndice*prob:.4f} ')
```

- Note: some good links on f-string formatting: https://realpython.com/python-f-strings/ , https://fstring.help/ , http://cissandbox.bentley.edu/sandbox/wp-content/uploads/2022-02-10-Documentation-on-f-strings-Updated.pdf

Be sure to reload the module when you make changes!

Testing our distributions

2) Does the data have variance $\sigma^2 = N * p * (1\text{-}p)$?

- In Python, we can check this with `np.std()` or `np.var()`:

```
print(f'np.std**2:{np.std( np.sum(is_one[:,
    0:ndice],axis=1) )**2:.4f}')
print(f'np.var: {np.var( np.sum(is_one[:,
    0:ndice],axis=1) ) :.4f}')
print(f'Expected variance: {ndice*prob*(1-prob):.4f}')
#Then to make things look prettier print a blank line
# after each set of numbers:
print('')
```

Be sure to reload the module when you make changes!

Results

- Did that all check out? If so, let's make rolldice also plot the predicted distributions. First, we need to add an import:

```
from scipy.misc import factorial,comb
```

- Then, after:

```
plt.hist( np.sum(n_ones[:,0:ndice],axis=1),range=(-0.5,ndice+0.5),bins=(ndice + 1) )
```

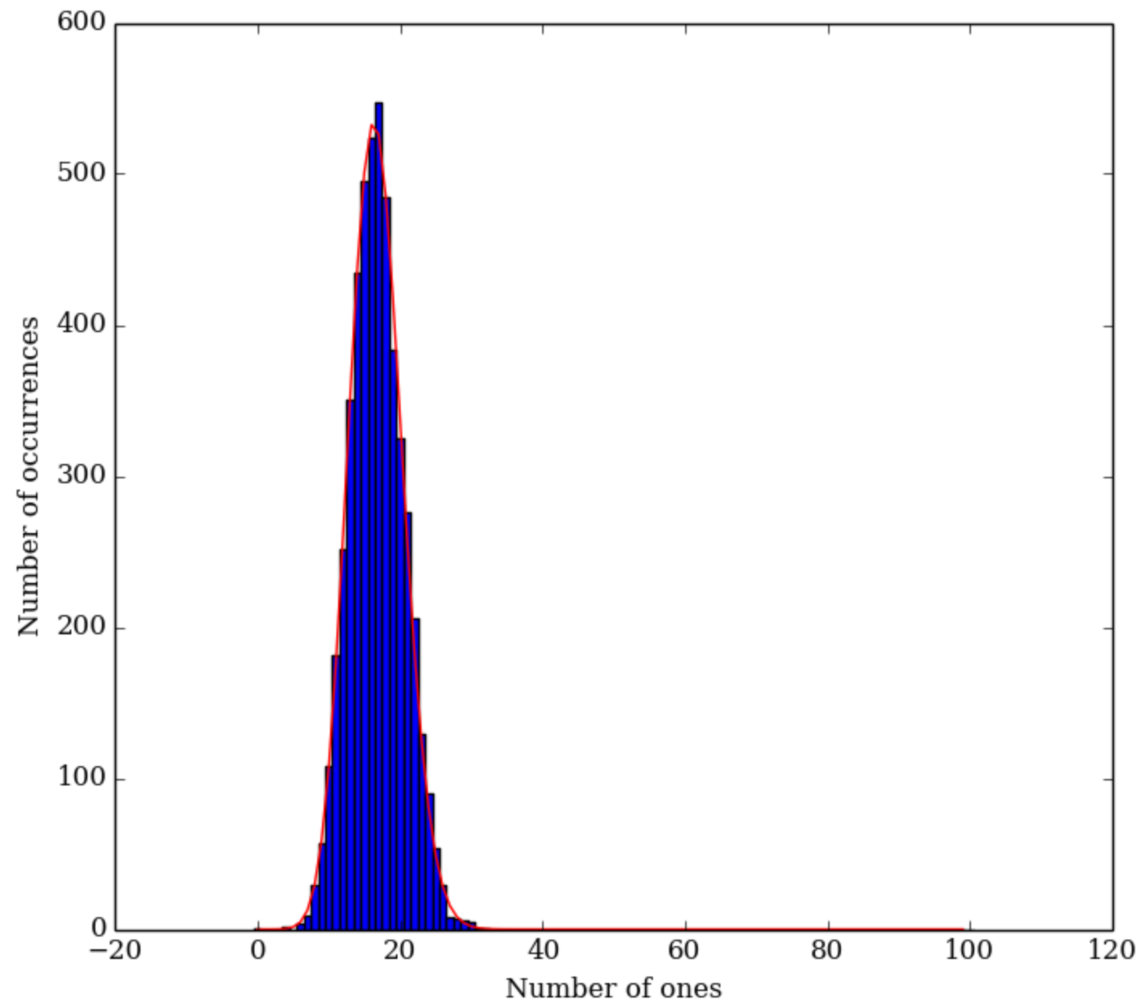
- add:

```
x=np.arange(ndice)
plt.plot(x,nsims*factorial(ndice)/factorial(x)/    \
 factorial(ndice-x)*prob**x*(1-prob)**(ndice-x),'r-')
```

- or:

```
plt.plot(x,nsims*comb(ndice,x)*prob**x*(1-prob)**(ndice-x),'ro')
```

It works!



600
500
400
300
200
100
0
Number of occurrences
−20
0
20
40
60
80
100
120
Number of ones

We could have used a scipy function instead

- `scipy` has a class (i.e., type of object), `scipy.stats.binom`, that allows you to calculate pretty much anything you'd want about a binomial distribution.
- It offers many subsidiary functions; `scipy.stats.binom.pmf(x,n,prob)` provides the probability of getting `x` occurrences out of `n` trials if the probability of an occurrence is `prob` (note that `x` can be an array!)
- alternatively, you can set up an object that is a member of the binomial class and inherits all of its functions ("methods"), but set up to assume n trials and probability prob, with e.g.

```
a = stats.binom(n,prob)
```

and then get the PMF (the discrete equivalent of a PDF) via

```
a.pmf(x)
```

We could have also used a scipy function

- `import scipy.stats` and look at the help information for `binom`. **Now modify your function** to add a curve to your plots showing the expected number of ones out of the simulation for each value of x using `stats.binom`; this should be `nsims` times the probability for one simulation...
- Note: you can actually look at the code for binom! Just do:

```
??scipy.stats.binom
```

Back to Bayesian statistics

- Suppose we do N independent experiments, and get a positive result M times. What have we learned about the true probability that we get a positive result, p?
- Recall Bayes' theorem:

$$prob(B \mid A) = prob(A \mid B)\, prob(B) / prob(A)$$

- So: $prob(p \mid M\ successes) = prob(M\ successes \mid p)\, prob(p) / prob(M\ successes)$
- For a binomial distribution,

$$prob(M\ successes \mid p) = C(N,M)\, p^M (1-p)^{N-M}$$

- What is the prior, $prob(p)$?

Choosing a prior

- *prob(p | M successes)* $\propto$ $p^M(1-p)^{N-M}$ *prob(p)* ; what's prob(p)?
- We have a number of options:

1) Often, we don't have any information about p, so we want an "uninformative" or "***noninformative***" prior . One option is to choose a prior that can make no difference at all: a uniform prior. In that case, *prob(p)* is a constant for all eligible *p* (0<p<1).

- A distribution uniform between *a* and *b* has mean $\mu = (a+b)/2$ and variance $\sigma^2 = (b-a)/12$
- Note that a uniform prior technically isn't a PDF if it's uniform over all possible parameter values (e.g. $-\infty < y < \infty$), as it doesn't integrate to 1. Most calculations still work, though; we call a prior that is not a PDF an ***improper prior***.

Choosing a prior

- *prob(p | M successes)* $\propto$ $p^{M}(1-p)^{N-M}$ *prob(p)* ; what's prob(p)?

2) A uniform prior sounds like a good no-information assumption... but suppose we had a different set of parameters.

- A prior uniform in p would not be uniform in *log(p)*,*sqrt(p)*, etc.; changing variables changes the effect of the prior. A uniform prior is not the only one to encode 'no information'.
- Often, probabilities are either close to 0 or close to 1 (e.g.: will Pitt win their next basketball game? The answer should be yes or no…) In such cases, one reasonable guess is Haldane's prior:

prob(p) $\propto$ $p^{-1}(1-p)^{-1}$

- This is an improper prior (due to its infinite integral), even over the range [0,1], with equal probability of p~0 and p~1.
- With this prior, *prob(p | 1 success in 2 trials)* will be a uniform distribution between 0 and 1.

Choosing a prior

3) It is possible to construct a prior which has the same effect on the posterior distribution under simple transformations of variables - this is called a Jeffreys prior (**NOT** Jeffrey's prior). This sort of prior can be constructed from a Fisher matrix (which will turn out to be derivable from the likelihood function).

- For a binomial process, the Jeffreys prior is $prob(p) \propto p^{-1/2}(1-p)^{-1/2}$
- Note that this is intermediate between (geometric mean of) the Haldane prior and a uniform prior
- Note: the book erroneously mislabels the Haldane and Jeffreys priors in section 2.3!

What difference does the prior make?

- Suppose you observe a set of 8 different early-type (red sequence / non-star-forming / elliptical/S0) galaxies and observe AGN signatures from three of them. What can we conclude about the probability p that a randomly chosen early-type galaxy has an AGN?
- The likelihood $prob(observation \mid p)$ should correspond to a binomial distribution, with 8 trials and 3 coming up 'AGN': $\propto p^3(1-p)^5$
- For the prior, $prob(p)$, we'll try all three possibilities for a binomial prior

$$prob(p \mid observation) = prob(observation \mid p)\ prob(p)\ /\ prob(observation)$$

Plotting a distribution in Python

- Before we plotted functions of x, where x took whole-number values. Instead we want to consider probabilities at equally-spaced values between 0 and 1, inclusive.

```
p=np.linspace(0.,1.,501)
```

So what does prob(observation | p) look like?

```
likelihood=p**3 * (1-p)**5
plt.plot(likelihood)
```

- Now let's set up our priors: Haldane, Jeffreys, and uniform

```
prior_h=1/p/(1-p)
prior_j=np.sqrt(prior_h)
prior_u=p*0.+1.
```

- These are plotted in the notebook. Which one would you expect to have the greatest impact on the posterior probability density function?

So what does prob(p | observation) look like?

- Now, we can make plots of the relative probability of each value of p, as a function of p.
- **Use a y range from 0 to 0.03, and plot the likelihood and the posterior assuming each different prior, using different lines or plot symbols for each.**

Normalizing probabilities

- Which of these was most strongly peaked?

```
plt.plot(p,likelihood*prior_h,'b-')
plt.plot(p,likelihood*prior_j,'r--')
plt.plot(p,likelihood*prior_u,'g-.')
```

- We can tell most easily if we normalize them all to give PDFs - i.e., to have integral 1.
- This will require us to do some calculus -- numerically.

Interpolation and calculus in Python

- We often want to calculate the integral or derivative of some function. There are many ways to do this in Python; a simple recipe:

1) create arrays of values of x and the evaluated function f(x), where f(x) is the function you want to integrate or differentiate.

2) Use `scipy.interpolate.interp1d()` to create a Python function (just like `np.sin(x)`, etc.) that interpolates between the tabulated values of x and f(x). E.g.:

```
import scipy.interpolate as interpol
x = np.linspace(-np.pi,3*np.pi,100)
# we want the interpolation table to extend beyond bounds we will use
x_fine = np.linspace(0,2*np.pi,10_000)
f = np.cos(x)
interp_f = interpol.interp1d(x,f,kind='cubic')
# interp_f is a new Python function!
plt.plot( x_fine,interp_f(x_fine),'r-' )
```

Interpolation and calculus

3) To integrate: one routine is `scipy.integrate.quad(function name, lower limit, upper limit)`. E.g.:

```
import scipy.integrate as integrate
print(f'{integrate.quad(interp_f,0,np.pi/2) = }')
```

- Note that `integrate.quad` returns a tuple (of the integral and its uncertainty) !

- To differentiate: one routine is `scipy.misc.derivative(function name,x value[s],dx=[dx value for calculations])`. E.g.:

```
import scipy.misc as misc
der = misc.derivative(interp_f,x_fine,dx=1E-3)
plt.plot(x_fine,der,'b--')
```

Normalizing probabilities (cont'd)

- So we can do:

```
posterior_u = interpol.interp1d(p,likelihood*prior_u,kind='cubic')
norm_u=(integrate.quad(posterior_u,0.,1.))[0]
```

- and similarly for priors h and j (Haldane & Jeffreys).

- We can then check how this worked:

```
print(f'{norm_u = } , {norm_j = } , {norm_h = }')
```

- Where did things go wrong?

Ways to avoid the problem

- 2 ways to proceed:

1) avoid division by zero via algebra:

```
prob_u=likelihood
prob_h=p**2*(1-p)**4
prob_j=p**2.5*(1-p)**4.5
posterior_u = interpol.interp1d(p,prob_u,kind='cubic')
norm_u=(integrate.quad(posterior_u,0.,1.))[0]
...
```

- and similarly for priors h and j (Haldane & Jeffreys).

- Then we can plot normalized posterior PDFs … (e.g., `posterior_h / norm_h`)

Second option: `np.isfinite()`

- 2) The problem was that we divided by zero, so some elements of `prior_h` & `prior_j` were illegal numbers (NaN). We can test if a given number is finite with `np.isfinite(x);` it returns True if the number is finite, False if not.

```
print(np.sum(np.isfinite(p)==False))
print(np.sum(np.isfinite(prior_h)==False))
```

- Since, for a dataset that is not all one value, the posterior probability should be zero at both p=0 and p=1 (**WHY**?), we can safely make the prior 0 at those points. We could use logic to deduce the right element #s, but let's be lazy.

`np.where()`

- One of the most powerful functions in numpy is np.where().
- It returns the array indices where some condition is true. e.g.:

```
test=np.array([1,2,3,4,5])
print(np.where(test == 3))
print(np.where(test > 2))
```

- We can use the result just like we can specify any other set of array indices (e.g. [0:i]):

```
print(test[np.where(test > 2)])
```

- Alternatively, you can use a logical expression to slice the array, with similar results:

```
print(test[ test > 2 ])
```

Using `np.where` with `np.isfinite`

- So we can get the array indices where prior_h blows up with :

```
whbad=np.where( np.isfinite(prior_h)==False)
print(whbad)
```

- **Let's repair the problem:**

```
prior_h[whbad]=0
prior_j[whbad]=0
posterior_h = interpol.interp1d(p,likelihood*prior_h,kind='cubic')
posterior_j = interpol.interp1d(p,likelihood*prior_j,kind='cubic')

norm_h=integrate.quad(posterior_h,0.,1.)[0]
norm_j=integrate.quad(posterior_j,0.,1.)[0]
```

- and plot again:

```
plt.plot(p,likelihood*prior_u/norm_u)
...
```

Some things to discuss with your group

- **Where does the posterior peak in each case? How does this compare to the observed fraction (3/8)?**

- **For which prior do you get the tightest constraint on *p*?** (Hint: since the integral is one, a tighter distribution will have a higher peak, when they are normalized into PDFs)

What happens if we have more data?

- Montero-Dorta et al. 2008 found that, of 710 early-type galaxies, 213 were AGN and 497 were not. What is the probability distribution for the true probability an early-type galaxy is an AGN, given each prior?
- **What do we want for likelihood? Do the priors change?**
- **Redo the normalizations and plot the normalized posterior for the new likelihood**

prob(p | observation) = prob(observation | p) prob(p) / prob(observation)

Using previous results as priors

- The Uniform, Haldane, and Jeffreys priors are all uninformative priors - i.e., priors we might choose knowing absolutely nothing about p.
- Suppose we want to use our sample of 8 galaxies to estimate the AGN fraction
- Going in, we've read Montero-Dorta et al., and we can use their result as a prior:
    - $prob(p) \propto p^{213} (1-p)^{497}$

prob(p | observation) = prob(observation | p) prob(p) / prob(observation)

Results as priors

- Then: $prob(p \mid obs) = prob(obs \mid p)\, prob(p) \,/\, prob(obs)$

$$\propto (p^3\,(1-p)^5)\ x\ (\ p^{213}\,(1-p)^{497}\ )$$

- Notice that:

1) This is the same result we'd get if we took the result of our observation as a prior, and then added the information from Montero-Dorta et al. - which is 'prior' and which is 'data' doesn't matter.

2) This is exactly the same result as if we had 216 AGN and 502 non-AGN; i.e., evaluated everything as one dataset

What difference does our sample make?

- Our prior is:

```
prior_obs=x**213.*(1-x)**497.
```

  - **Calculate its normalization to get a pdf!** No need to worry about checking for nonfinite elements here...
- Our likelihood is:

```
likelihood=x**3*(1-x)**5
```

  - **Calculate its normalization to get a pdf!**

```
posterior = likelihood*prior_obs
```

  - **Calculate its normalization too!**
- **Now plot up the normalized likelihood, the normalized prior, and the normalized posterior probability, with different linestyles and/or colors (solid, dashed, red, etc.)**

Coming up with your own priors

- One simple way to produce a prior for a binomial distribution is to produce a guess as to the right answer, and how many observations that guess is 'worth'.
- E.g. if in your experience about half of galaxies are AGN, but you're not too sure about that, you might consider it worth a few observations : prior $\propto p^{1.5}(1-p)^{1.5}$
- Or you might know that ~25% of galaxies overall are AGN, but be unsure whether that overall number is relevant for the particular sample you are working on : prior $\propto p^{1}(1-p)^{3}$
- Or you might be really sure that 25% is the right universal answer, whatever your sample tells you: prior $\propto p^{250}(1-p)^{750}$