[1]:
```
%matplotlib inline
import matplotlib

# make figures bigger:
import numpy as np
import matplotlib.pyplot as plt
import random
import scipy, scipy.stats

# make figures better:
font = {'weight':'normal','size':20}
matplotlib.rc('font', **font)
matplotlib.rc('figure', figsize=(8.0, 6.0))
matplotlib.rc('xtick', labelsize=16)
matplotlib.rc('ytick', labelsize=16)
matplotlib.rc('legend',**{'fontsize':16})
matplotlib.rc('image',cmap='viridis')  # change default colormap
```

DS1 Lecture 24

James Bagrow, james.bagrow@uvm.edu, http://bagrow.com

Last time:

1. Intro to Bayesian Inference
    - Checking whether a coin is fair
2. Using posterior distributions for model selection: Bayes Factor
    - Unbiased (simple) model vs. Biased (complex) model (online)

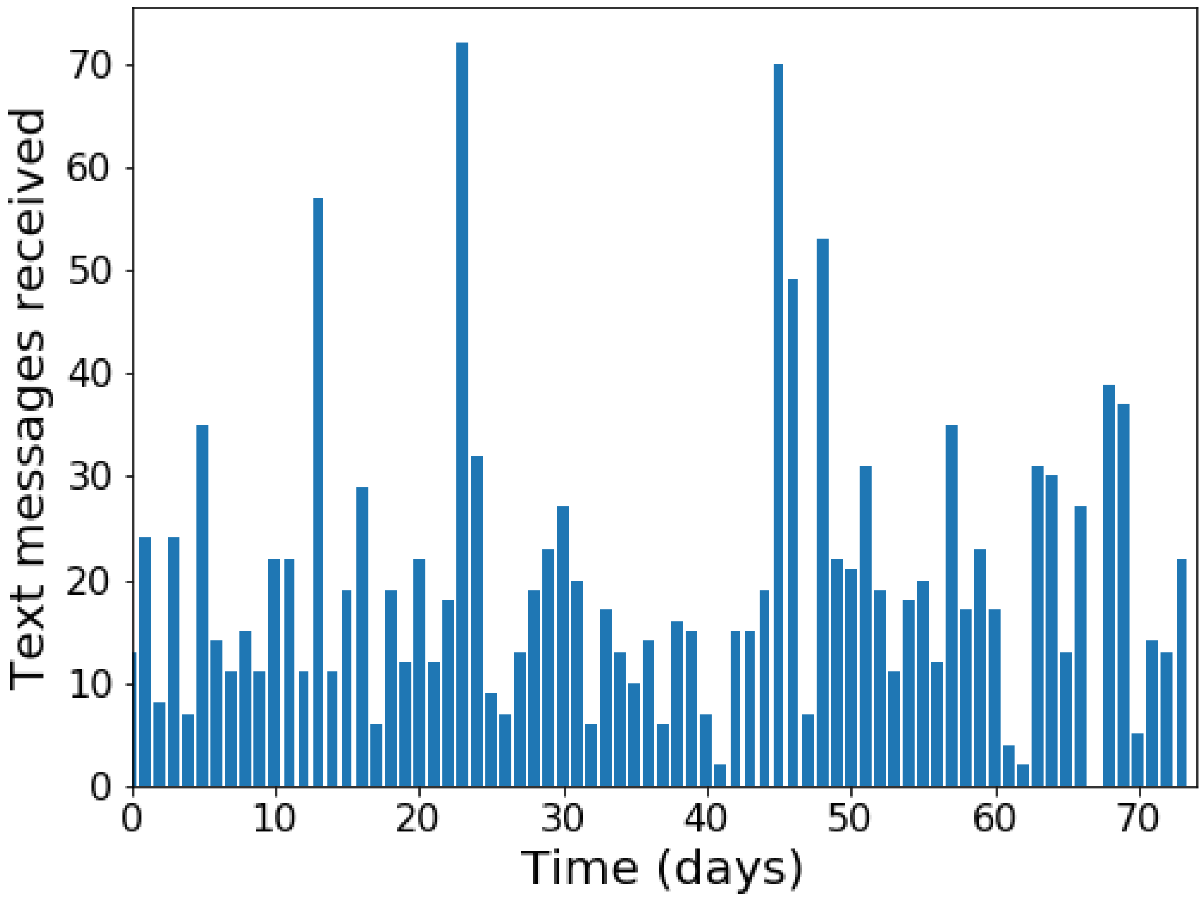
Today:

1. Bayesian inference
    - text messaging data

Bayesian inference

We have some data from a cell phone user, the number of text messages received per day, during a time period:

[2]:
```
C = np.loadtxt("txtdata.csv")
T = len(C)

plt.bar(np.arange(T), C, color='C0', ec='none')
plt.xlabel("Time (days)")
plt.ylabel("Text messages received")
plt.xlim(0, T)
plt.show()
```



We want to see: Does the **rate** (messages per day) over the course of the data?

- If it does change, when does it change? Does it change suddenly or gradually?

Visually, it looks pretty steady...

Here's a quick calculation, mean rate of messages averaged over the beginning and ending of the data:

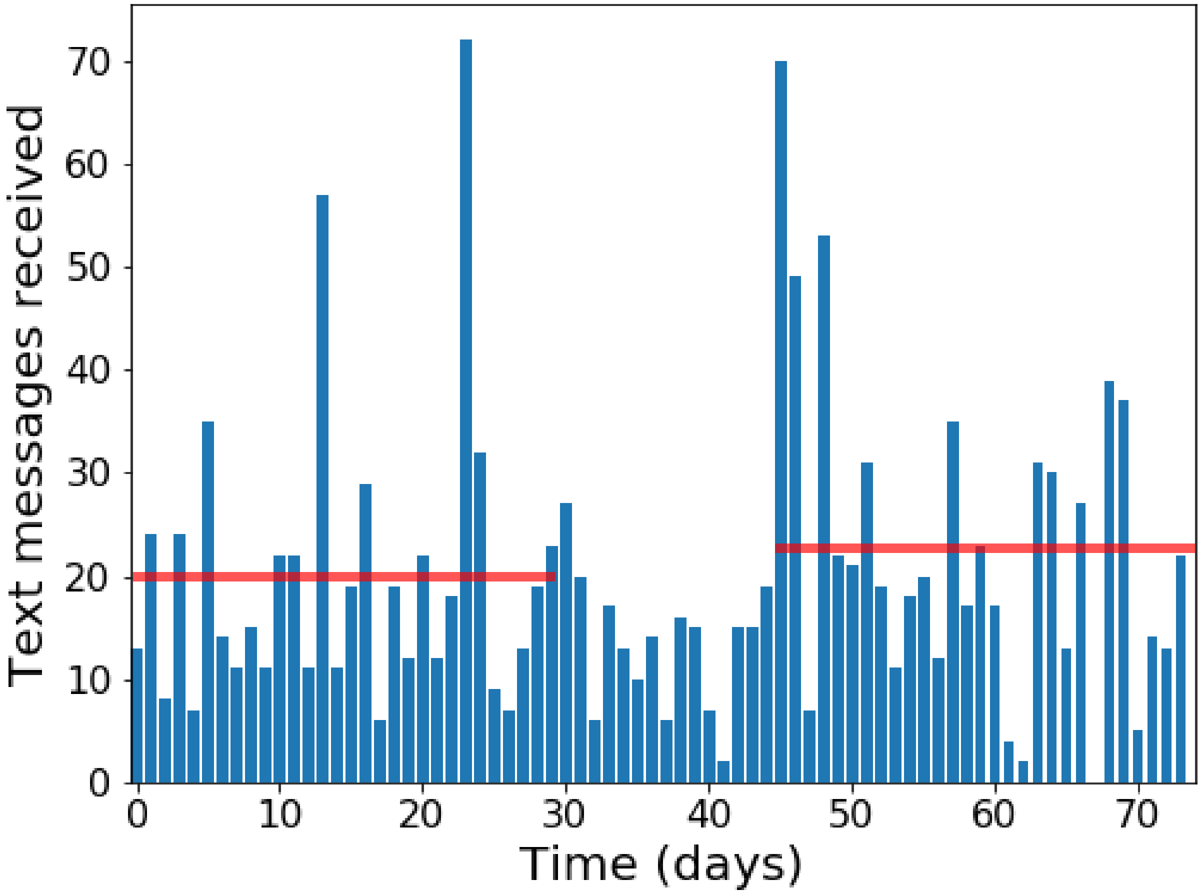
[3]:
```
print("mean rate during first month =", C[ :30].mean(), "msgs/day")
print("mean rate during last month  =", C[-30:].mean(), "msgs/day")
```

```
mean rate during first month = 19.9 msgs/day
mean rate during last month  = 22.7 msgs/day
```

And let's visualize these rates:

[4]:
```
plt.bar(np.arange(T), C, color='C0', ec='none')

# add mean rate for first and last 30 days:
n = 30
plt.plot(np.arange(n),         n*[C[:n].mean() ], 'r-', lw=4, alpha=0.667 )
plt.plot(np.arange(T-n+1,T+1), n*[C[-n:].mean()], 'r-', lw=4, alpha=0.667 )

plt.xlabel("Time (days)")
plt.ylabel("Text messages received")
```

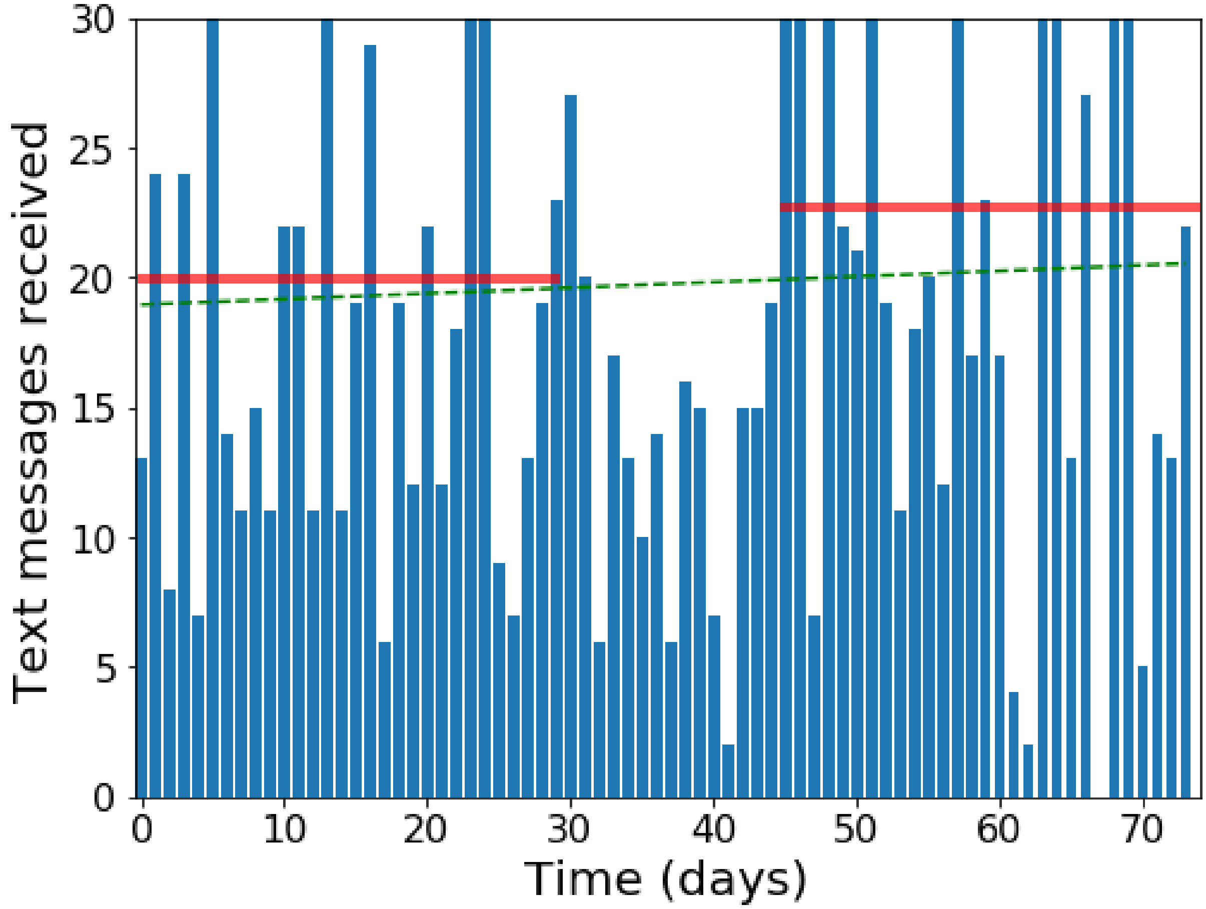
```
plt.xlim(0-0.5, T)
plt.show()
```



Let's zoom in a bit:

[5]:
```
plt.bar(np.arange(T), C, color='C0', ec='none')

# add mean rate for first and last 30 days:
n = 30
plt.plot(np.arange(n),          n*[C[:n].mean() ], 'r-', lw=4, alpha=0.667 )
plt.plot(np.arange(T-n+1,T+1), n*[C[-n:].mean()], 'r-', lw=4, alpha=0.667 )

# and also, add a linear model!
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(np.arange(T), C)
plt.plot(np.arange(T), slope*np.arange(T)+intercept, 'g--')
print("p-value (linear model) =", p_value)

plt.xlabel("Time (days)")
plt.ylabel("Text messages received")
plt.xlim(0-0.5, T)
plt.ylim(0,30) # ***
plt.show()
```

```
p-value (linear model) = 0.7801884386142313
```



Hmm, ≈ 3 messages per day difference... Hmm... ....

Statistical model for our hypothesis

[notes on board about statistical model, poisson distributions, rates, proof of poisson as limit of binomial]

Let's **simulate** this thing.

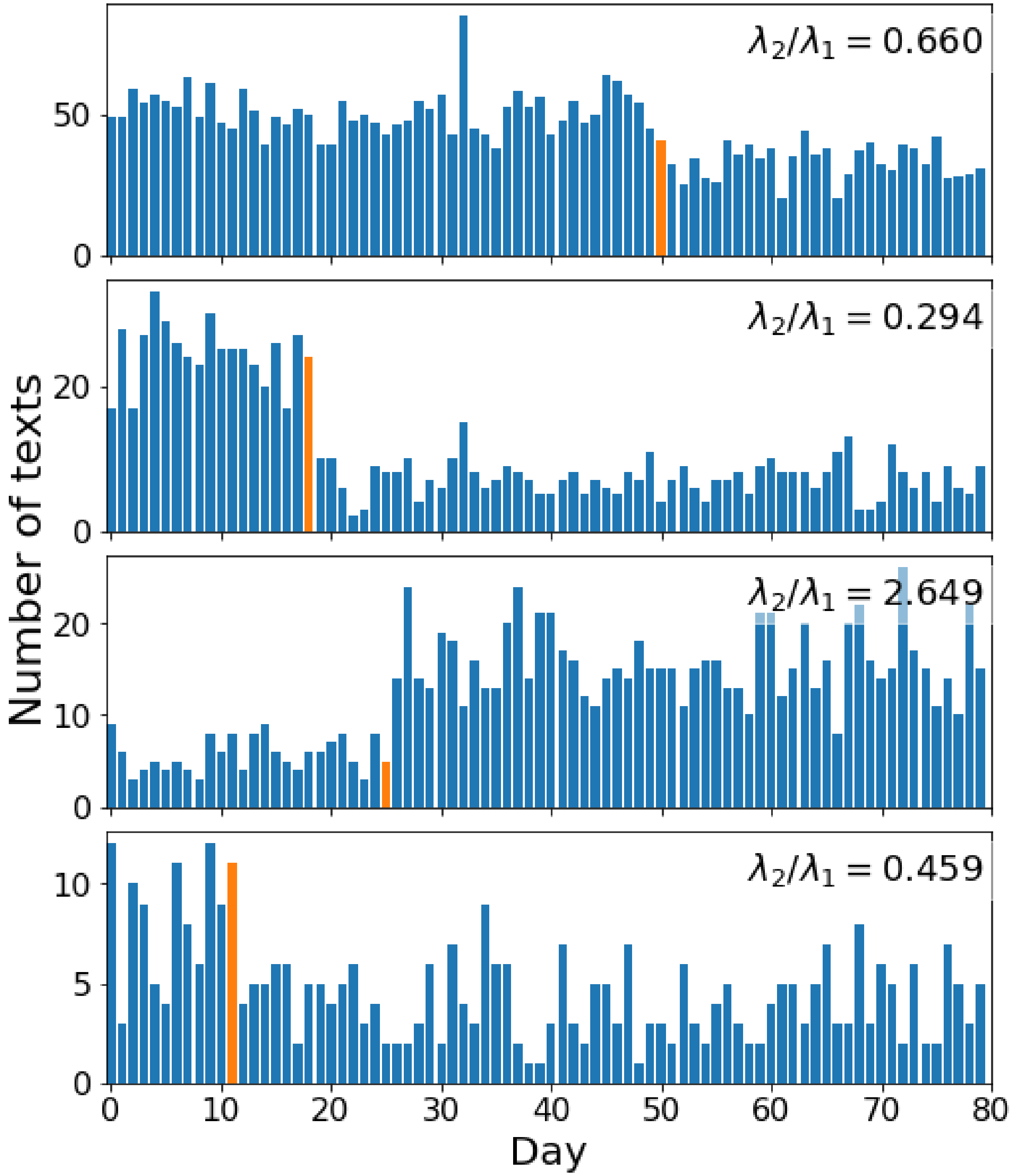
First, a function that generates data from the model:

[6]:
```python
def gen_artificial_sms_dataset(lambda_1, lambda_2, tau, length=80):
    """vector of counts drawn from Pois(lambda_1) when t < tau
    and Pois(lambda_2) when t >= tau.
    """
    data_b = np.random.poisson(lam=lambda_1, size=tau)
    data_a = np.random.poisson(lam=lambda_2, size=length-tau)

    return np.append(data_b, data_a)
```

Now let's generate **artificial text data** following our model. We'll pick some values of λ_1, λ_2, and τ, and make a plot for each:

[7]:

```python
# specify FOUR models (lam1, lam2, tau):
models = [
    (50.01, 33.02, 51),
    (23.87,  7.02, 19),
    ( 6.19, 16.40, 26),
    ( 8.70,  3.99, 12)
]

# set up a figure with 4 plots stacked in a row:
fig, list_axes = plt.subplots(4, 1, sharex=True)
fig.set_size_inches(8, 10)
fig.subplots_adjust(hspace=0.1)

for ax,(L1,L2,tau) in zip(list_axes, models):
    data = gen_artificial_sms_dataset(L1,L2,tau)

    ax.bar(np.arange(len(data)), data, color='C0', ec='none')
    ax.bar(tau-1, data[tau-1],  color='C1', ec='none')

    txt = ax.text(0.725, 0.85,
            "$\lambda_2/\lambda_1 = {:0.3f}$".format(L2/L1),
            fontsize=18,
            horizontalalignment='left',
            verticalalignment='center',
            transform=ax.transAxes
           )
    txt.set_bbox(dict(color='white', alpha=0.5))

# clean up the plots:
plt.xlabel("Day")
plt.xlim(-0.5,80)
fig.text(0.06, 0.5, 'Number of texts',
         fontsize=22,
         ha='center', va='center',
         rotation='vertical'
        );
```



Of course, we aren't GIVEN the parameters λ_1, λ_2, and τ. We need to infer them. How? Bayes!

Posterior!

`Pr(model | data) = Pr(data | model) Pr(model) / Pr(data)`

We want to get `Pr(M | D)` to find which parameters (λ_1, λ_2, and τ) have high probability given the observed data.

If we are **given values** of the parameters we can compute:

```
Likelihood * Prior = Pr(data | model) Pr(model) ~ Posterior
```

This is **almost** what we need—we are missing P(data). Next time we will fill in the gap to perform inference.

For now let's make some pictures to see priors and posteriors. * Simplify: We will **fix** τ and only look at λ_1 and λ_2.

Priors

We will look at two priors for the λ's

Uniform Prior:

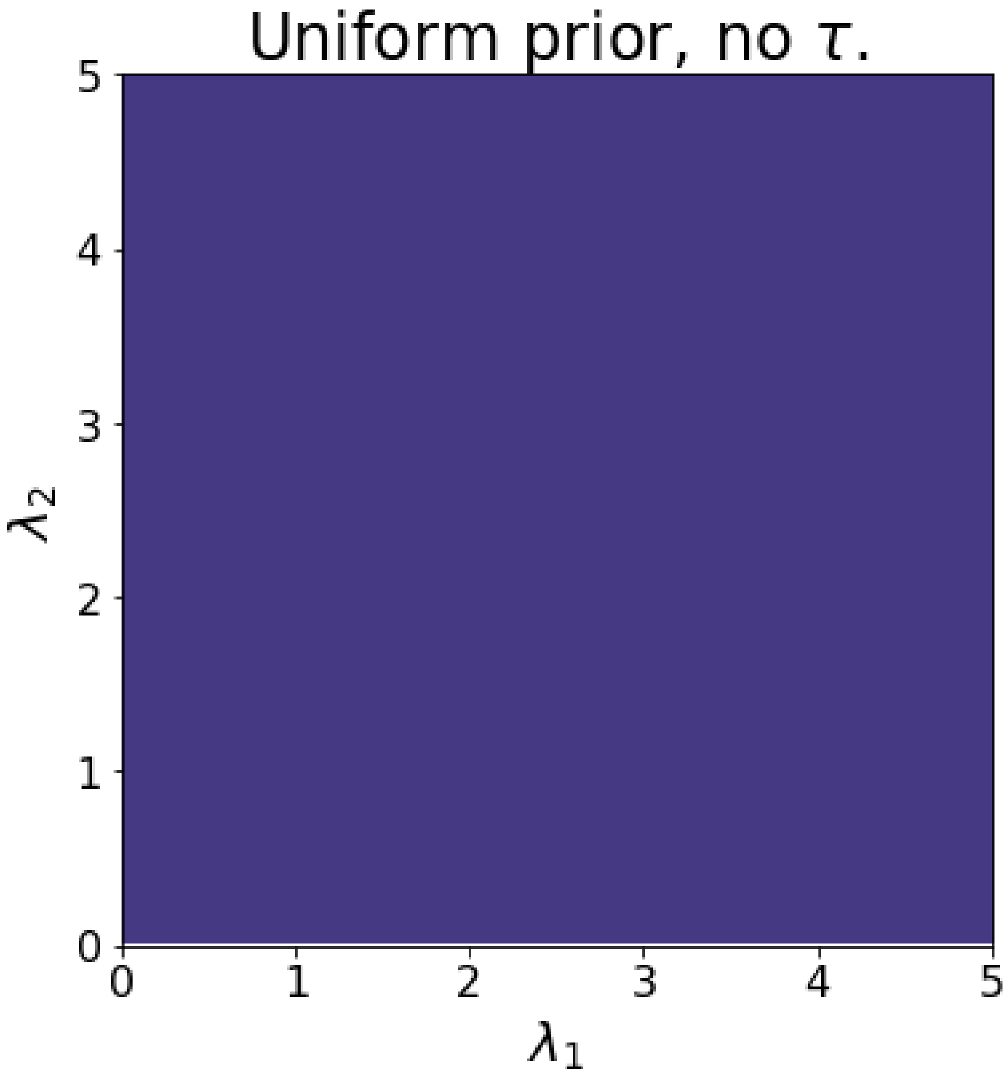
All models (values of λ's) over a certain range are equally likely before we look at the data.

- λ_1 ~ Uniform(0,5)
- λ_2 ~ Uniform(0,5)

(This is simpler than the prior we used on the board.)

Plot the prior:

[8]:
```
# double-loop of 100 values for lambdas:
lam1 = lam2 = np.linspace(0, 5, 100) # vectors
L1, L2 = np.meshgrid(lam1, lam2)     # for 3d plots

# compute probabilities:
Uniform_Prior = 0.2*0.2*np.ones(L1.shape)
#uni_L1 = scipy.stats.uniform.pdf(L1, loc=0, scale=5)
#uni_L2 = scipy.stats.uniform.pdf(L2, loc=0, scale=5)
#Prior = np.dot(uni_L1[:, None], uni_L2[None, :])

# top-down (matrix) view:
plt.imshow(Uniform_Prior, interpolation='none', origin='lower',
           vmax=1, vmin=-0.15,
           extent=(0, 5, 0, 5)
          );

plt.xlim(0, 5); plt.xlabel("$\lambda_1$")
plt.ylim(0, 5); plt.ylabel("$\lambda_2$")
plt.title(r"Uniform prior, no $\tau$.");
```

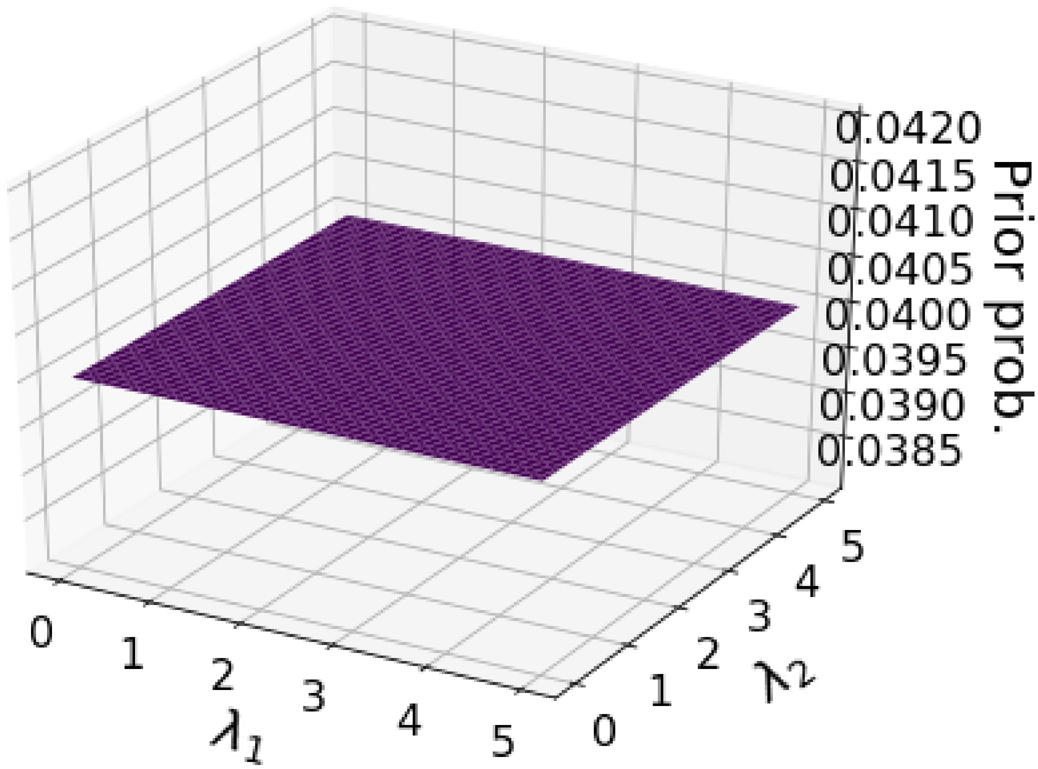


(In practice we wouldn't make such a visualization, and we would probably want to include a *colorbar*, but here's it's meant to compare with what comes next.)

Boring... Here is a 3D view:

[9]:
```
from mpl_toolkits.mplot3d import Axes3D


# 3D view:b
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(L1, L2, Uniform_Prior, cmap=plt.cm.viridis)
ax.set_xlabel("\n$\lambda_1$")
ax.set_ylabel("\n$\lambda_2$");

ax.zaxis.set_rotate_label(False) # disable automatic rotation
ax.set_zlabel("\nPrior prob.", linespacing=-3, rotation=-90)
```

```
ax.dist = 11 # camera distance
```



Also pretty boring…

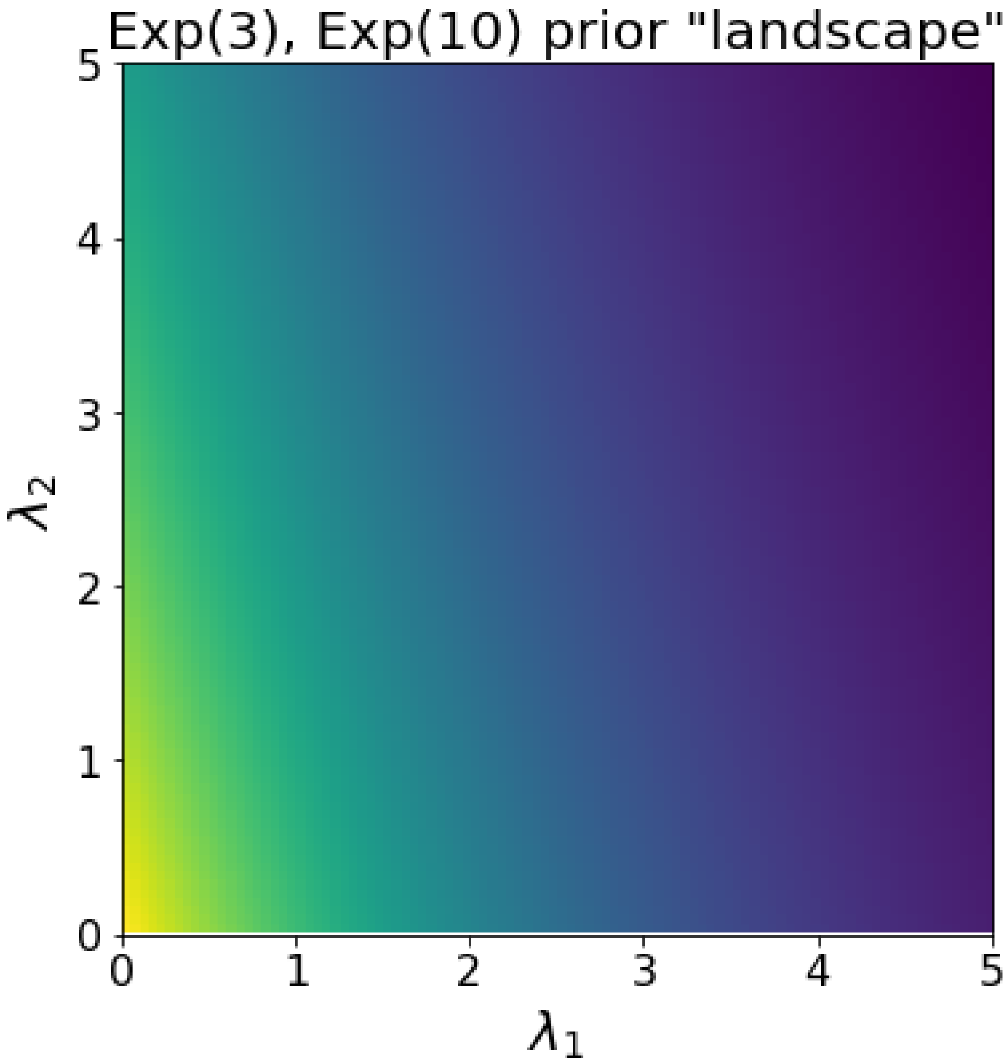
Exponential Prior:

Here we now suppose that smaller values of λ are "more likely" than larger values, all else being equal.

Example:

- $\lambda_1 \sim \text{Exp}(1/3)$ mean$(\lambda_1) = 3$
- $\lambda_2 \sim \text{Exp}(1/10)$ mean$(\lambda_2) = 10$

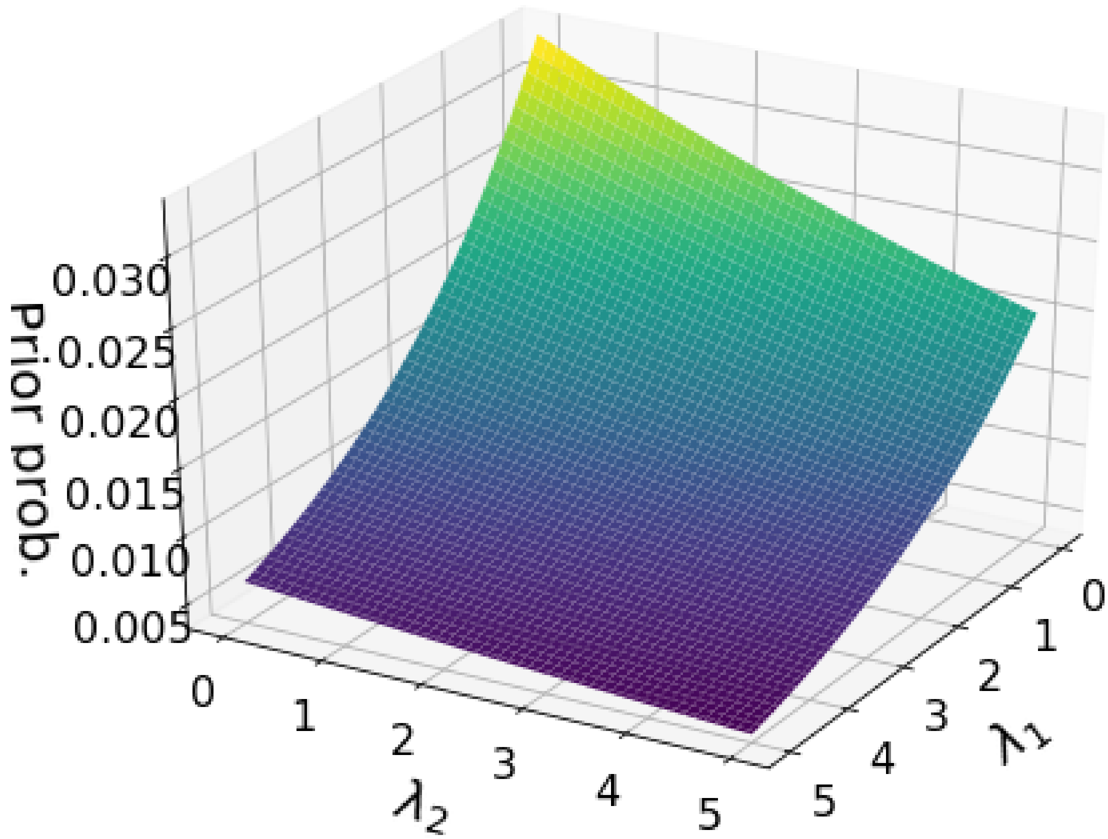
[10]:
```
exp_L1 = scipy.stats.expon.pdf(lam1, scale=3)
exp_L2 = scipy.stats.expon.pdf(lam2, scale=10)
Exp_Prior = np.dot(exp_L2[:, None], exp_L1[None, :])

# plot:
im = plt.imshow(Exp_Prior, interpolation='none', origin='lower',
                extent=(0, 5, 0, 5))

plt.title("Exp(3), Exp(10) prior \"landscape\"",fontsize=20)
plt.xlabel("$\lambda_1$")
plt.ylabel("$\lambda_2$");
```



(Like before, needs a colorbar.)

And here the 3d view is not so boring:

[11]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(L1, L2, Exp_Prior, cmap=plt.cm.viridis)
ax.view_init(azim=390)

ax.set_xlabel("\n$\lambda_1$")
ax.set_ylabel("\n$\lambda_2$")

ax.set_zlabel("Prior prob.", labelpad=11);
```



Where does the knowledge of the prior come from? Past experience, physical reasons, etc.

- Uniform prior: we have no way "a priori" to assume any value of the parameter over another value.
- Exponential prior: we think both parameters are more likely to come from small values than from large values.

Posteriors

Now comes the data!

The data *warps* the height of the probability surface via the likelihood. This is Bayes Theorem.

Let's make some fake data from a (secretly known) model. We will at first draw a few values and see how the posterior changes for both priors.

[12]:
```
# create the observed data

# sample size of data we observe, trying varying this (keep it less than 100 ;)
N = 3 # 2N total datapoints, N on either side of tau
# the true parameters, but of course we do not see these values...
lambda_1_true = 1
lambda_2_true = 3

# Generate count data, given the above true lambdas:
# (could also have used gen_artificial_sms_dataset...)
```

```
data_b = scipy.stats.poisson.rvs(lambda_1_true, size=(N, 1))
data_a = scipy.stats.poisson.rvs(lambda_2_true, size=(N, 1))
data = np.append(data_b, data_a) # this is our synthetic c_t
```

And generate the likelihoods for each pair of λ's:

[13]:
```
# parameter sweep over lambdas to get likelihoods...
L1 = L2 = np.linspace(0.01, 5, 100)

Pois = scipy.stats.poisson.pmf # Pois(x,L) = L^x exp(-L) / x!

like_db = np.array([Pois(data_b, lam1).prod() for lam1 in L1])
like_da = np.array([Pois(data_a, lam2).prod() for lam2 in L2])
likelihood = np.dot(like_da[:, None], like_db[None, :]) # matrix
```

[14]:
```
if N < 5:
    print(data)
    print(Pois(data, 0.3))
    print()
    print(Pois(data, 0.3).prod())
```

```
[0 2 1 2 1 4]
[7.40818221e-01 3.33368199e-02 2.22245466e-01 3.33368199e-02
 2.22245466e-01 2.50026149e-04]

1.0167431302704622e-08
```

Put it into a little *reusable* function:

[15]:
```
def gen_data_likelihood(N):
    data_b = scipy.stats.poisson.rvs(lambda_1_true, size=(N, 1))
    data_a = scipy.stats.poisson.rvs(lambda_2_true, size=(N, 1))
    data = np.append(data_b, data_a)

    like_db = np.array([Pois(data_b, lam1).prod() for lam1 in L1])
    like_da = np.array([Pois(data_a, lam2).prod() for lam2 in L2])
    likelihood = np.dot(like_da[:, None], like_db[None, :]) # matrix

    return data, likelihood
```

Now, plot:

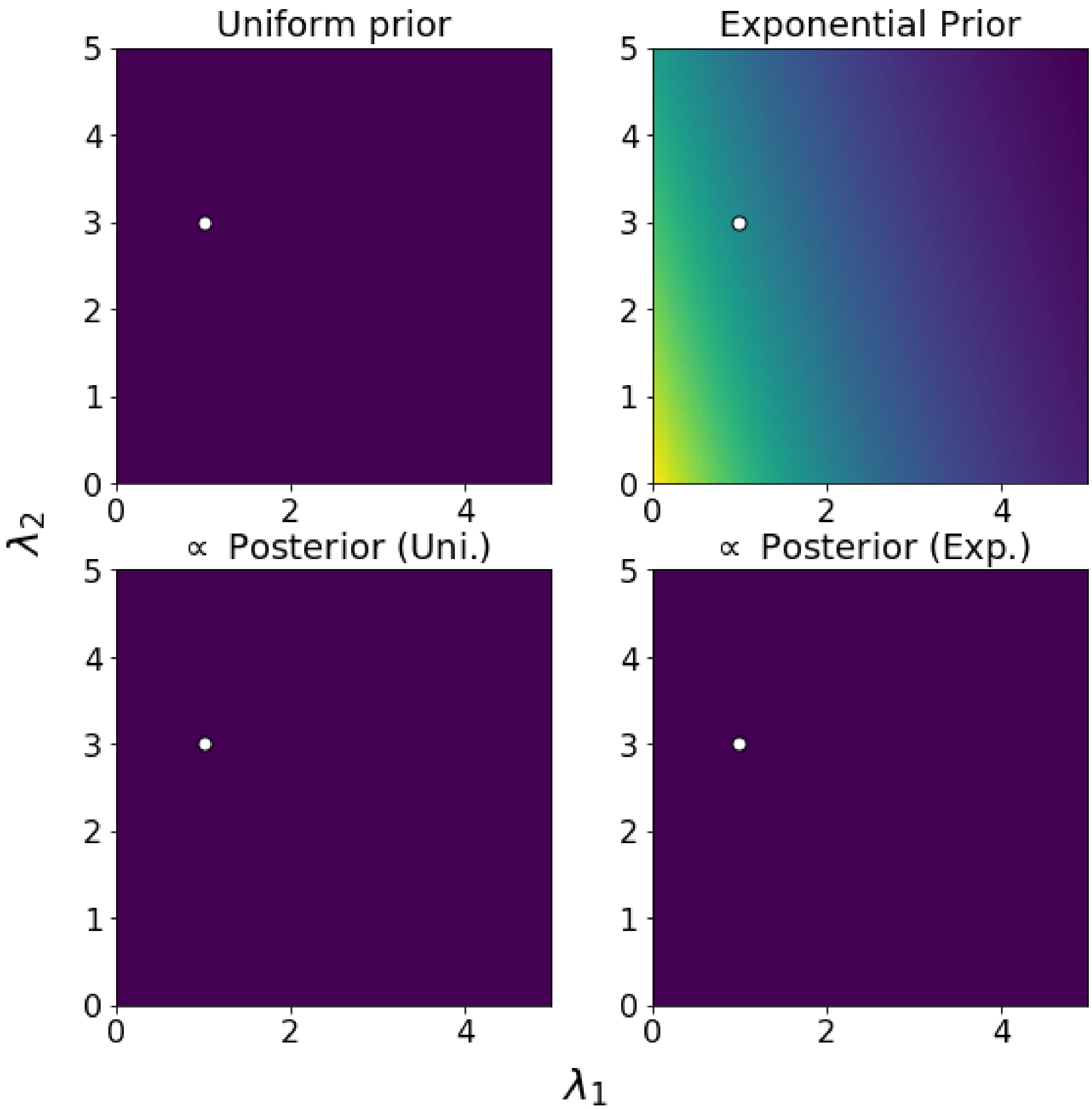
[16]:
```
def plot_mat(X,Y,Z, fig=None, extent=None):
    if fig is None:
        fig = plt.gcf() # get current fig
    ax = fig.gca()
    ax.imshow(Z, interpolation='none', origin='lower',
              extent=extent)
    if extent is not None:
        ax.set_xlim(extent[0:2])
        ax.set_ylim(extent[2:] )

def plot_truth(x_true,y_true, fig=None):
```

```
    if fig is None:
        fig = plt.gcf() # get current fig
    ax = fig.gca()
    ax.scatter(x_true, y_true, c="w", s=50, edgecolor="k",zorder=1e9)
```

[17]:
```
def each_plot(subplot, Y, title=None):
    plt.subplot(subplot)
    plot_mat(L1, L2, Y, extent=extent)
    plot_truth(lambda_1_true, lambda_2_true)
    plt.title(title, fontsize=18)


def label_axes():
    fig.text(0.5, 0.06, r"$\lambda_1$",
             fontsize=22,
             ha='center', va='center')
    fig.text(0.06, 0.5, r"$\lambda_2$",
             fontsize=22,
             ha='center', va='center',
             rotation='vertical');
```

Now, *really* plot:

[21]:
```
fig = plt.figure(figsize=(9,9))
extent = (0,5,0,5) # = (xmin,xmax,ymin,ymax)

N = 1000 # 10,20,100,200,1000 -- there are 2N datapoints
data, likelihood = gen_data_likelihood(N)

each_plot(221, Uniform_Prior,              title='Uniform prior')
each_plot(223, Uniform_Prior * likelihood, title=r"$\propto$ Posterior (Uni.)")

each_plot(222, Exp_Prior,              title=r"Exponential Prior")
each_plot(224, Exp_Prior * likelihood, title=r"$\propto$ Posterior (Exp.)")

label_axes()
```



Now let's go back and see how things look when there's more data! * [Rerun for different values of `N`]

Remarks:

- These functions are only *proportional* to the posterior because we have ignored the denominator in Bayes Thm.
- The posterior becomes more sharply peaked as N increases
- Sometimes the "bulk" of the posterior will not include the secretly known true value (λ_1, λ_2), but it will be close to (λ_1, λ_2).
- The uniform and exponential priors look very different, but their corresponding posteriors look very similar
- The real *cheat* here is the absence of τ.

Bayesian inference. Take your (meaningful?) priors, update them with the likelihoods (using the observed data) to get a function proportional to the posterior.

- Of course, **parameter sweeps** like we've done (looping over many combinations of λ_1 and λ_2) are prohibitive in practice. Instead, **drawing samples** from the posterior gives us distributions of our parameters that incorporate your knowledge and uncertainty in the underlying model. But how to do this?

References

This dataset and some of this code is taken from a nice online book, Probabilistic Programming and Bayesian Methods for Hackers.

Next time:

Sampling from the posterior!