[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})

import warnings
warnings.filterwarnings('ignore')
```

DS1 Lecture 26

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

Last time:

1. Bayesian inference
    - text messaging data
2. Building a model for bayesian inference

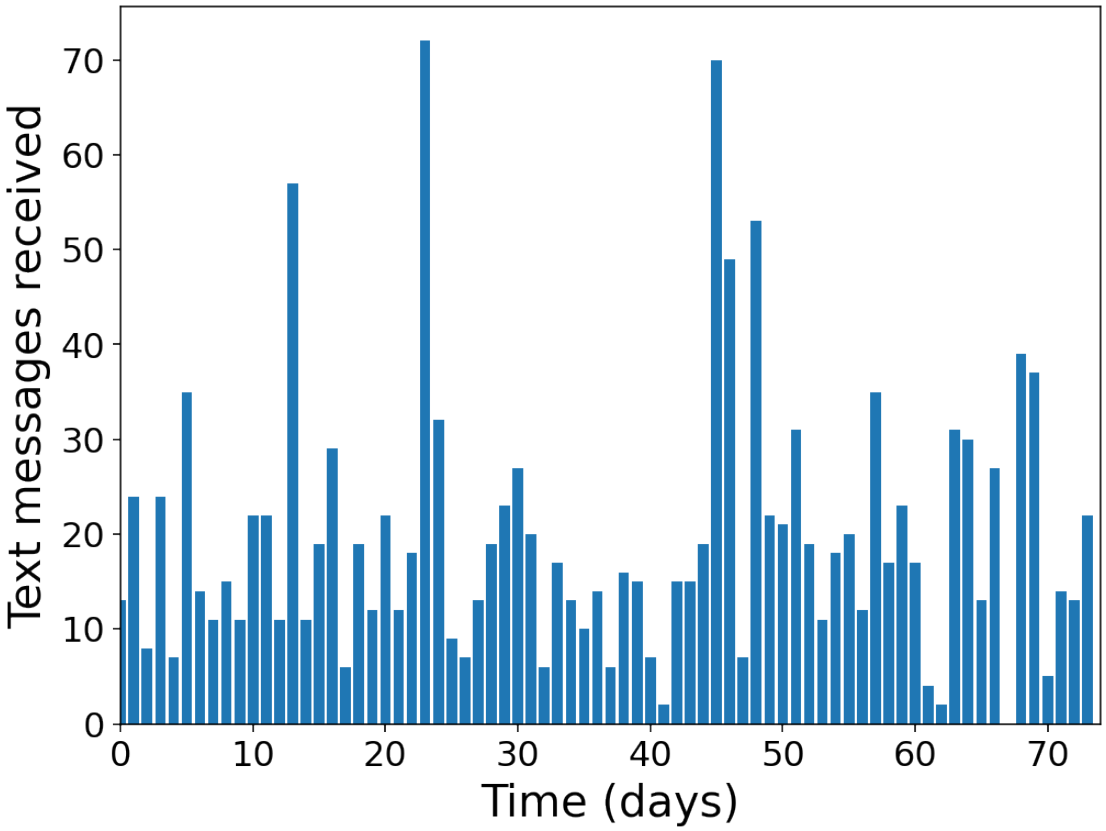
Today:

1. Intro to Markov Chain Monte Carlo (MCMC)
    - How to sample from a posterior you can't compute!

Bayesian inference

The dataset:

[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);
```



Statistical model for our hypothesis

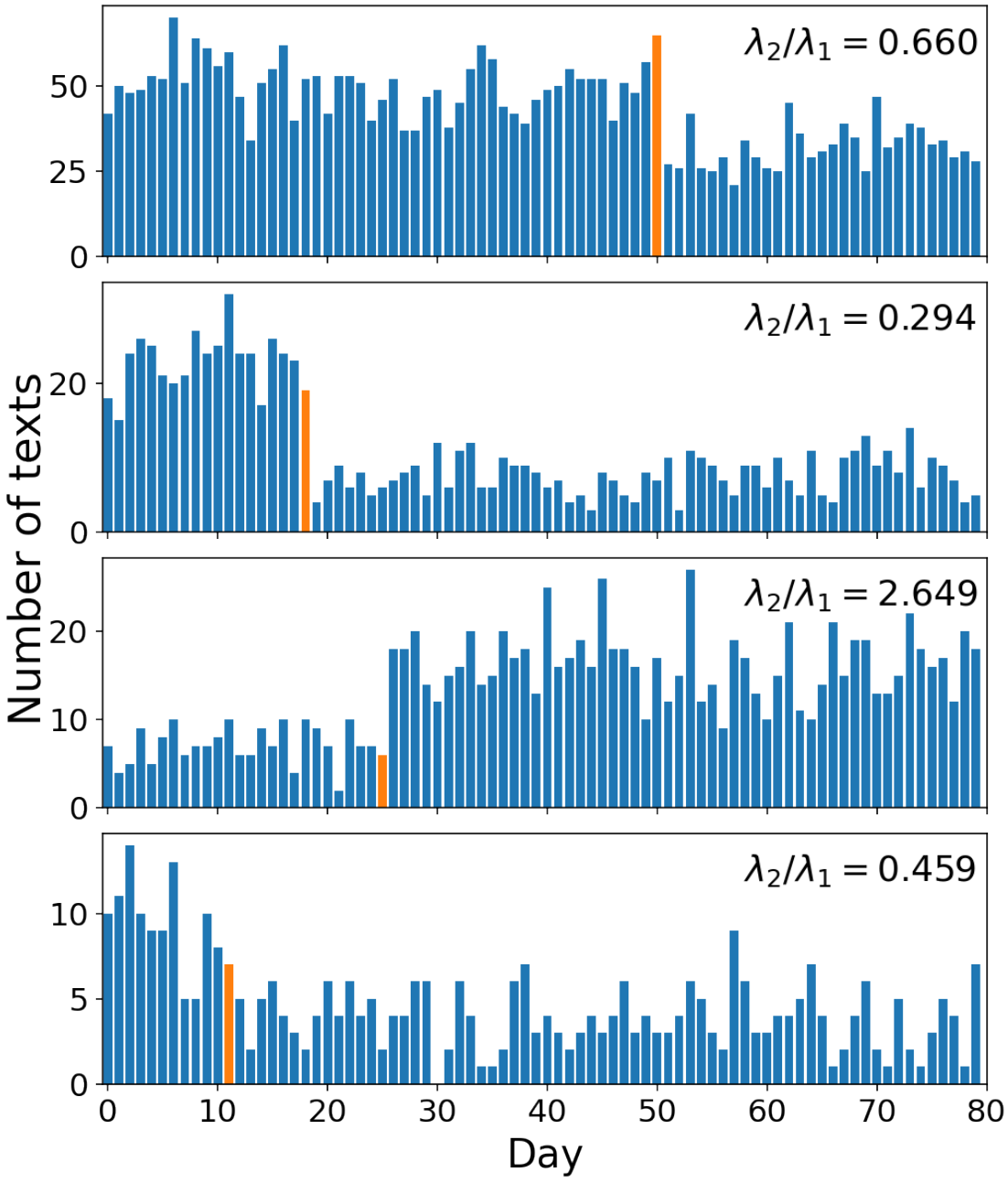
[derived last time]

Here's some examples of synthetic data drawn from our statistical model:

[3]:
```
def gen_artificial_sms_dataset(lambda_1, lambda_2, tau):
    # vector of counts drawn from Pois(lambda_1) when < tau
    # and Pois(lambda_2) >= tau
    data_b = np.random.poisson(lam=lambda_1, size=tau)
    data_a = np.random.poisson(lam=lambda_2, size=80-tau)

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

[4]:
```
list_L1 = [50.01, 23.87,  6.19, 8.70]
list_L2 = [33.02,  7.02, 16.40, 3.99]
list_Ta = [   51,    19,    26,   12]

# build 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 i in range(4):
```

```
    L1,L2= list_L1[i],list_L2[i] # lambdas
    tau = list_Ta[i]             # tau
    data = gen_artificial_sms_dataset(L1,L2,tau)

    ax = list_axes[i]
    ax.bar(np.arange(80), data, color='C0', ec='none')
    ax.bar(tau-1, data[tau-1], color='C1', ec='none')

    ax.text(0.725, 0.85,
            "$\lambda_2/\lambda_1 = %0.3f$" % (L2/L1),
            fontsize=18,
            horizontalalignment='left',
            verticalalignment='center',
            transform = ax.transAxes
           )
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'
        );
```



Bayesian inference. Take your (meaningful?) priors, update them with the likelihoods (using the observed data) to get a function proportional to the posterior. Then **drawing samples** from the posterior gives us distributions of our parameters that incorporate your knowledge and uncertainty in the underlying model

Markov chain monte carlo using PyMC

- Later we will work through how to use this!

First, let's check that PyMC is installed:

[5]:
```
import pymc3 as pm
print(pm.__version__)
```

```
3.8
```

(You should be able to install PyMC yourself using Conda if you don't have it already.)

Now let's start building the statistical model using PyMC data structures for random and deterministic variables:

[6]:
```
alpha = 1.0 / C.mean() # Recall C is the variable that
                       # holds our txt counts
```

Recall, the model we're using consists of a discrete uniform variable and two exponential variable (our **priors**) "glued" together into a Poisson variable (our **likelihood**) via a step function (our λ_t):

[7]:
```
mod = pm.Model()
with mod:
    # Prior for tau:
    tau = pm.DiscreteUniform('tau', 0, T)

    # Prior for lambda1, lambda2 (note shape=2):
    lam  = pm.Exponential('lam', lam=alpha, shape=2)
    grp = (np.arange(T) > tau) * 1 # 0 if t <= tau, 1 if t > tau

    # Likelihood, Poisson w/ time-dependent rate:
    y_obs = pm.Poisson('y_obs', mu=lam[grp], observed=C)

    # algorithm(s) for sampling from posterior:
    #step1 =  pm.Slice([tau])
    step2 = pm.Metropolis([lam])
    #step2 = pm.NUTS() # default, No U-Turn Sampler

    # sample from the posterior using one or more "step" methods:
    trace = pm.sample(step=[step2], draws=15000, chains=1,tune=1000)
```

```
Sequential sampling (1 chains in 1 job)
CompoundStep
>Metropolis: [lam]
>Metropolis: [tau]
Sampling chain 0, 0 divergences: 100%|      | 16000/16000 [00:04<00:00,
3660.53it/s]
Only one chain was sampled, this makes it impossible to run some convergence
checks
```

(We'll cover the details of sampling algorithms soon.)

Visualizing the trace

Now that we have our sample (the **trace**), let's plot it and take a look.

First, extract the individual model parameters $(\tau, \lambda_1, \lambda_2)$ using methods associated with `trace`:

[8]:
```
tau_samples = trace.get_values('tau',burn=400,thin=5)

lam_samples = trace.get_values('lam',burn=400,thin=5)
lam1_samples = lam_samples[:,0]
```

```
lam2_samples = lam_samples[:,1]

print(tau_samples)
print(lam1_samples)
```

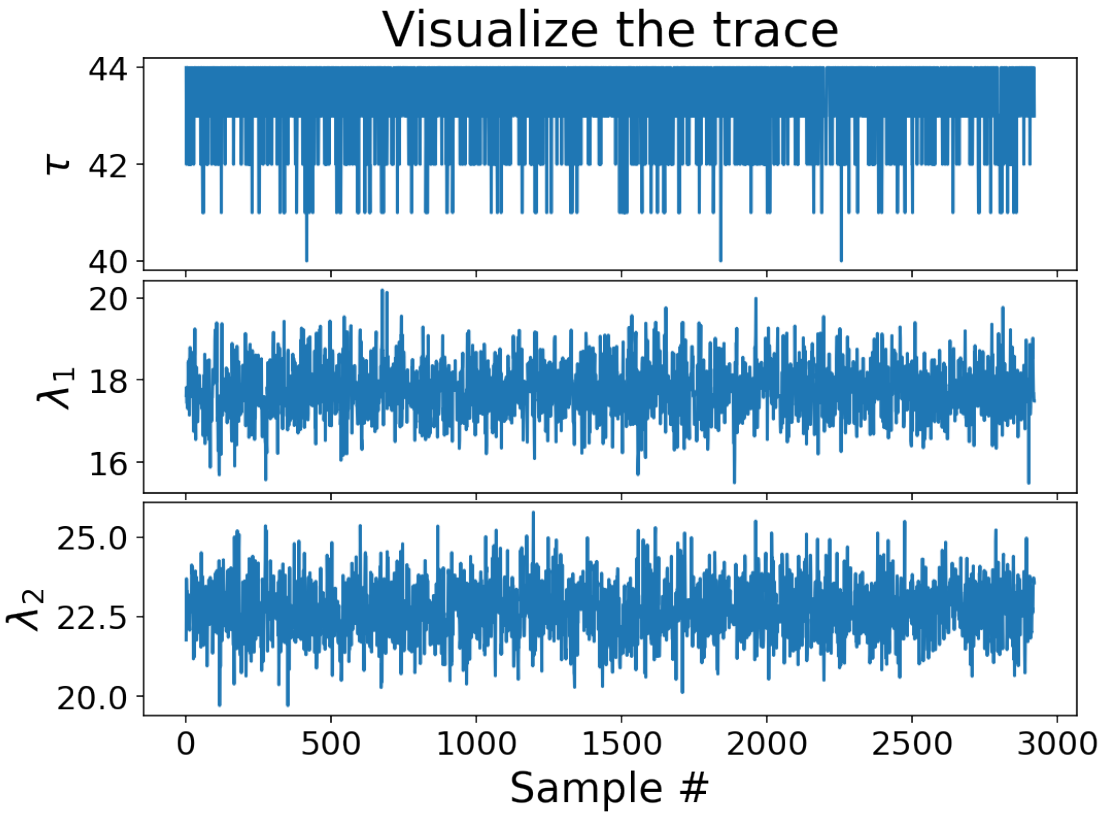
```
[44 42 44 … 43 44 43]
[17.62468733 17.81919266 17.45445327 … 17.67794085 17.48950451
 17.48950451]
```

(We will cover burn-in and thinning in the future.)

- These numpy arrays are *empirical* samples from the underlying posterior. We managed to compute them without actually knowing the posterior…

Now plot the trace:

[9]:
```
fig, axes = plt.subplots(3,1,sharex=True)

axes[0].plot(tau_samples)
axes[0].set_ylabel(r'$\tau$')

axes[1].plot(lam1_samples)
axes[1].set_ylabel(r'$\lambda_1$')

axes[2].plot(lam2_samples)
axes[2].set_ylabel(r'$\lambda_2$')
axes[2].set_xlabel("Sample #")

axes[0].set_title("Visualize the trace")
plt.tight_layout(h_pad=0)
```



It's a **good idea** to visualize the trace (traceplot) in this way, to see if there is anything **weird** going on, like patterns.

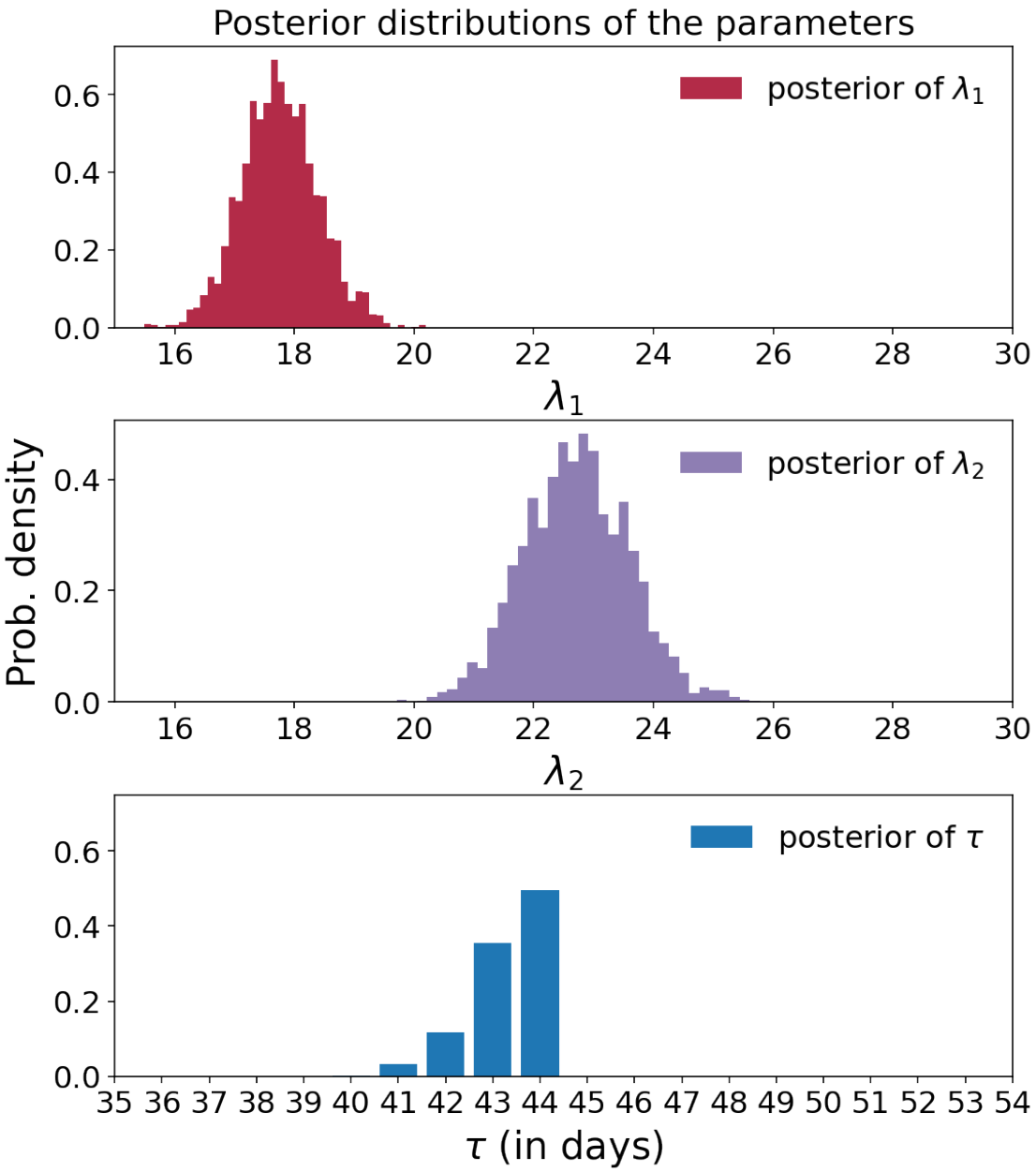
- We want our models to be iid draws from the posterior, so if there are any correlations, trends, or other non-random patterns, then we know that we have not achieved **convergence** and we should be **skeptical** of our results.

Posterior distributions

Plot the distributions of the parameters → posteriors!!

[10]:
```
fig = plt.figure(figsize=(8,9))

# histogram for lambda1:
plt.subplot(311)
plt.hist(lam1_samples, histtype='stepfilled', bins='auto', alpha=0.85,
         label="posterior of $\lambda_1$", color="#A60628", density=True)
plt.legend(loc="upper right", frameon=False);
plt.title(r"Posterior distributions of the parameters", fontsize=18);
plt.xlim([15, 30])
plt.xlabel("$\lambda_1$"); # actually behind the next subplot..


# histogram for lambda2:
plt.subplot(312)
```

```
plt.hist(lam2_samples, histtype='stepfilled', bins='auto', alpha=0.85,
         label="posterior of $\lambda_2$", color="#7A68A6", density=True)
plt.legend(loc="upper right", frameon=False);
plt.xlim([15, 30])
plt.xlabel("$\lambda_2$");
plt.ylabel("Prob. density");


# histogram for tau
plt.subplot(313)

import collections # tau is integer-valued so let's count instead of bin...
taus,Ntaus = zip(*sorted( collections.Counter(tau_samples).items() ))
Ptaus = [1.0*n/sum(Ntaus) for n in Ntaus]

plt.bar(taus,Ptaus, color='C0', label=r"posterior of $\tau$", align='center');
plt.xticks(np.arange(80));


# clean up
plt.legend(loc="upper right", frameon=False);
plt.ylim([0, .75]);
plt.xlim([35, len(C) - 20]);
plt.xlabel(r"$\tau$ (in days)");
plt.tight_layout(h_pad=0)
plt.show()
```



Does this answer the question of whether the rate changes?

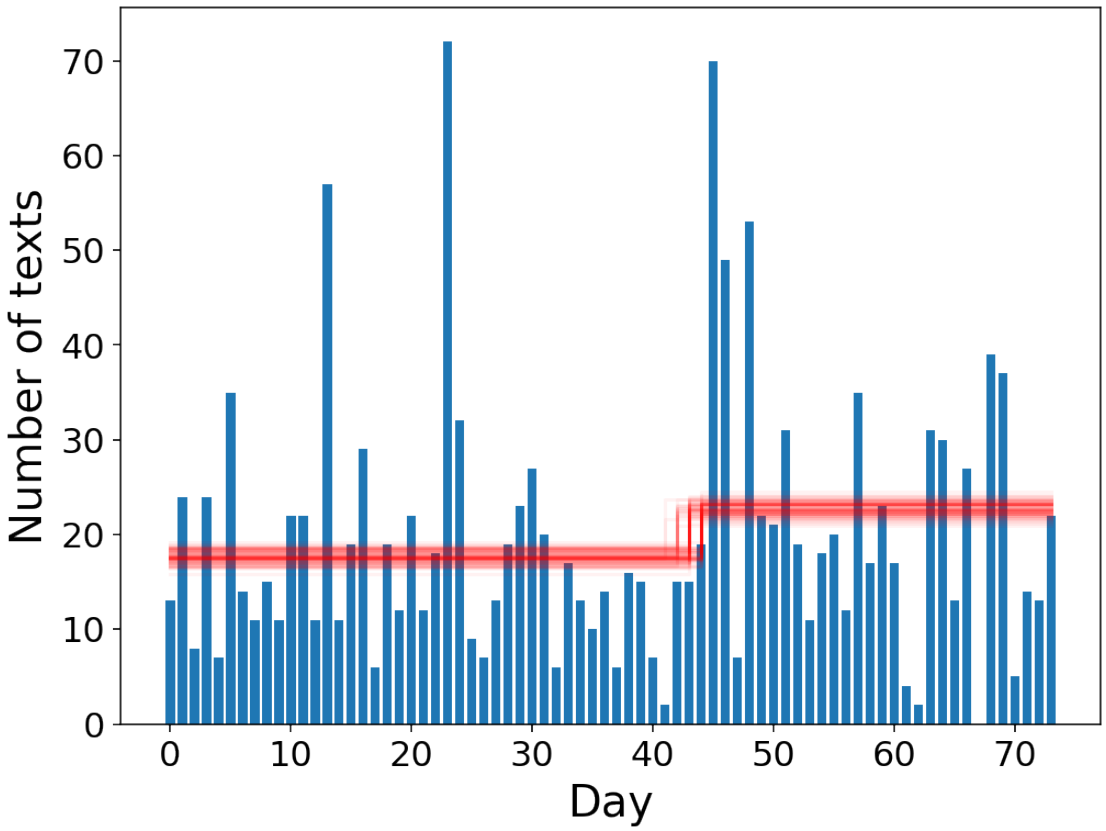
Yes!

More over, the narrow spread of values of τ gives us very strong evidence for *when* the rate changes.

Remember, in Bayesian inference you do not have a single model, you have distributions of models. In this problem, each "model" is really a triple of parameters $(\lambda_1, \lambda_2, \tau)$.

Here's a plot of λ_t for a few models:

[11]:
```
plt.bar(np.arange(T), C, label='count')
plt.xlabel('Day')
plt.ylabel('Number of texts')

def rate_plot(tau,lam1,lam2, color='red',alpha=0.05):
    grp = (np.arange(T) > tau) * 1
    lambda_t = lam1*(1-grp) + lam2*grp
    line = plt.step(np.arange(T), lambda_t,
                    color=color, alpha=alpha)

for t,(lam1,lam2) in zip(tau_samples[:100],lam_samples[:100]):
    rate_plot(t, lam1, lam2)
```



Summary

- We've seen the basics for taking our statistical model, implementing it in Python using a library called PyMC, and then computationally drawing samples of the model's posterior using MCMC.

Next time

- Doing MCMC yourself
- How MCMC works