Assignment 3

Using Stan & other exercises

Implement models in Stan

Implement each of the models below using Stan and produce the results or plots requested for each model. You have seen each of these models before in class.The goal of this exercise is to learn how to implement different types of parameters, likelihood functions, and prior distributions using Stan. Stan always generates samples for estimating posterior distributions, while we used conjugate distributions in class. Check that your results from Stan's samples match the results we computed in class.

Link to notebook: https://github.com/hueyning/CS146-repo/blob/master/Assigment%203/Assignment%203.ipynb (https://github.com/hueyning/CS146-repo/blob/master/Assigment%203/Assignment%203.ipynb)

In [107]:
```
#import libraries
import pystan
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
sns.set()
```

1. Call center data set — exponential likelihood with gamma prior.

Estimate the number of calls per minute for the 11th hour of the call center data set.

Results to compute:

- Posterior 95% confidence interval over λ (check that it matches results in the solution notebook below)

- Histogram of posterior λ samples

Resources for you to use:

- Data set: call_center.csv

- Solution for class activity (call_center_solution.ipynb)

In [108]:
```
waiting_times_day = np.loadtxt('call_center.csv')

# Split the data into 24 separate series, one for each hour of the day
current_time = 0
waiting_times_per_hour = [[] for _ in range(24)]  # Make 24 empty lists,
 one per hour

for t in waiting_times_day:
    current_hour = int(current_time // 60)
    current_time += t
    waiting_times_per_hour[current_hour].append(t)

hour_index = 11
waiting_times_hour = waiting_times_per_hour[hour_index]

# For Stan we provide all known quantities as data, namely the observed
 data
# and our prior hyperparameters.
call_center_data = {
                      'waiting_times': waiting_times_hour, #length of ea
ch waiting time
                      'N': len(waiting_times_hour), # number of waiting
 times in the data set
                      'alpha': 1,
                      'beta': 0.25
                   }
```

In [71]:
```
# We have to tell Stan what data to expect, what our parameters are and
 what
# the likelihood and prior are. Since the posterior is just proportional
 to
# the product of the likelihood and the prior, we don't distinguish betw
een
# them explicitly in the model below. Every distribution we specify is
# automatically incorporated into the product of likelihood * prior.

call_center_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> N; // number of waiting times logged
    real<lower=0> waiting_times[N];  // number of waiting times for the
 11th hour
    real<lower=0> alpha;  // fixed prior hyperparameter
    real<lower=0> beta;   // fixed prior hyperparameter
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterio
r
// distributions over all parameters.
parameters {
    real<lower=0> lambda;  // call rate - the parameter of the exponenti
al likelihood
}

// The model block contains all probability distributions in the model.
// This of this as specifying the generative model for the scenario.
model {
  lambda ~ gamma(alpha, beta);  // gamma prior
  for(i in 1:N) {
    waiting_times[i] ~ exponential(lambda);  // likelihood function
  }
}

"""
```

In [72]:
```
call_center_stan_model = pystan.StanModel(model_code=call_center_stan_co
de)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_3b9cfeff352dfec
559c05c2ae0f248e6 NOW.
```

In [73]:

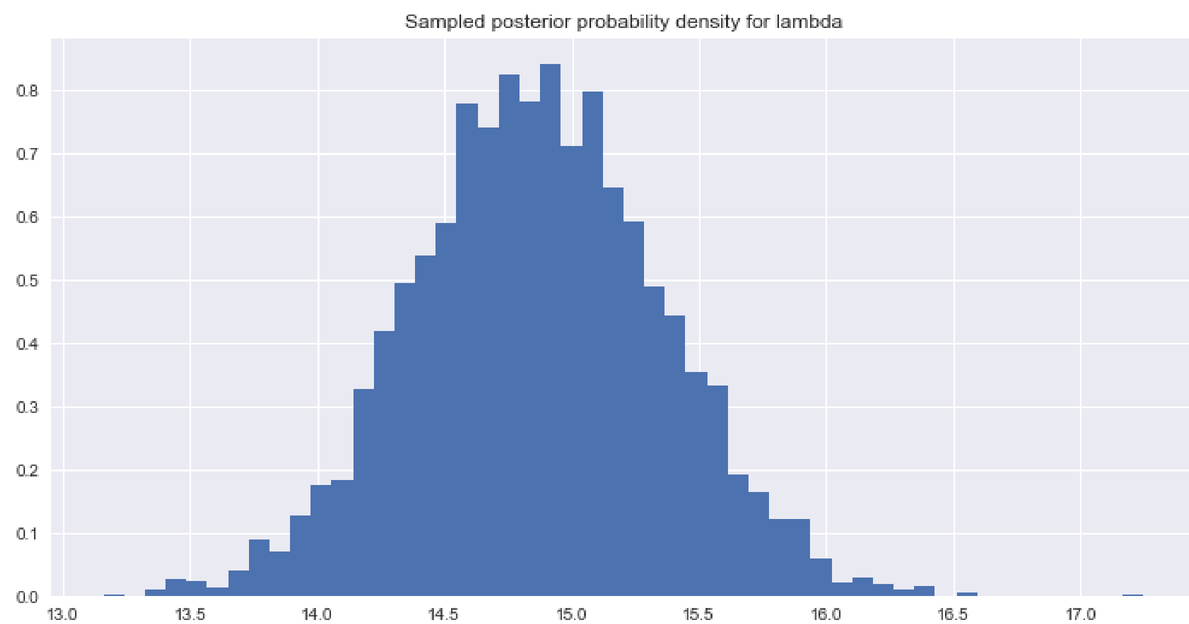
```
# Fit the model to the data. This will generate samples from the posteri
or over
# all parameters of the model.

call_center_stan_results = call_center_stan_model.sampling(data=call_cen
ter_data)
print(call_center_stan_results)
```

```
Inference for Stan model: anon_model_3b9cfeff352dfec559c05c2ae0f248e6.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff
  Rhat
lambda  14.87    0.01   0.49  13.92  14.54  14.86  15.19  15.84   1379
   1.0
lp__   1516.4    0.02    0.7 1514.5 1516.2 1516.7 1516.8 1516.9   1596
   1.0

Samples were drawn using NUTS at Fri Oct 19 11:07:04 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [109]:

```
# Finally, we can extract the samples generated by Stan so that we
# can plot them or calculate any other functions or expected values
# we might be interested in.

call_center_posterior_samples = call_center_stan_results.extract()
plt.figure(figsize=(12, 6))
plt.hist(call_center_posterior_samples['lambda'], bins=50, density=True)
plt.title('Sampled posterior probability density for lambda')
print(
    "Posterior 95% confidence interval for lambda:",
    np.percentile(call_center_posterior_samples['lambda'], [2.5, 97.5]))
plt.show()
```

```
Posterior 95% confidence interval for lambda: [13.92512194 15.83381684]
```



2. Normal likelihood with normal-inverse-gamma prior.

Results to compute:

• 95% posterior confidence intervals for the mean μ and variance σ of the data.

• Take 10 samples from your posterior over μ and σ and plot the normal distributions corresponding to them. See Task 3 in the solutions below — you should produce a plot similar the one you find there.

Resources for you to use:

• Data and solution for class activity (normal_inverse_gamma_solution.ipynb)

In [76]:

```
dat = np.array([3.54551763569501, 4.23799861761927, 4.72138425951628, -
0.692265320368236, 3.04473513808788, 3.10721270732507, 3.42982225852764,
 3.12153903971176, 3.60532628639808, 2.46561737557325, 1.64059465916131,
 2.4621623937158, 2.76744495617481, 2.11580054750407, 5.14077208608354,
4.90288499104252, 1.43357579078348, 4.78997817363558, 1.93633438207439,
2.43698838097178, 3.95389148701877, 2.4242295507716, 2.90256268679023,
2.90931728045901, 0.658072819386888, 3.05946763895983, 3.42615331539605,
 2.68842833004417, 2.35850130765166, 2.20014998540933, 4.73846511350084,
 4.19839721414451, 2.11805510171691, -0.572742936038015, 0.3894139820106
23, 3.87846130744249, 1.34057656890858, 0.7235748351719, 5.1104236984017
4, 4.00747556696571, 3.18080956726965, 3.24677964069676, 5.1154659863626
, 1.80276616697155, 0.305877679021404, -0.449168307882718, 4.63705561194
774, 1.37783714058301, 4.9608149859515, 6.7764195802069, 1.7551552292239
9, 7.04457337435215, 0.625185284955128, 2.25130734369064, 2.197701781192
55, 2.16858257249432, 6.25367644481438, 0.116081323476489, 2.06315857864
341, 1.82409781471718, 5.15226741230987, 2.03408231293173, -1.1245085433
7596, 5.03511270642234, 2.03841989653263, 5.80911741751597, 2.3171812878
3245, 4.97575010580997, 3.34262752222776, -0.786983904253601, 0.77736235
9850013, 0.975825009321195, 3.76354577515958, 7.27215002907876, 1.354040
89480189, 3.76567940257157, 3.48573993343334, 1.85976988586156, 1.935670
61960716, 5.31071812003942, 2.96832987672751, 3.32378908637275, 2.616319
60054551, 5.80897964052825, 4.95215217171488, 1.32036772796131, 3.799325
42233371, 3.08108492766309, 2.6734110081666, -0.14251851138521, 2.487443
75588965, 3.98463042123415, 6.32781680028, 4.0029172024315, 4.2321036945
9457, 1.71412938967325, 5.16492114963802, 2.53409673107906, 4.7734696397
3334, 3.34088878725551, 4.77681472750664, 3.81135755590976, 1.1405426998
3137, 1.42057452397702, 0.132142311125433, 7.12577254064672, 4.854220127
81764, 4.15745720676399, 4.48763147363348, 1.56060322283629, 2.648217615
42887, 1.26655351354548, 4.48497722937931, 4.3286302403783, 4.2615767951
2625, 4.0597558651364, 5.14051109132496, 2.5660348362221, 1.107640138186
17, 0.386889523012303, 3.54150473246237, 3.57480214382351, 1.95150869584
847, 2.70688970563118, 2.47971849820016, 6.50838037000679, 4.01511556826
974, 1.11562740835344, 5.02637639472439, 4.38184491686864, 5.60423144047
386, 2.40067408379298, 5.7849941378344, 2.37225791084559, 6.860314659102
73, 4.09214858239736, 6.85994063692621, 3.62202415158781, -1.11220646958
158, 3.73920971696866, 3.24533871512216, 1.28724203643002, 0.29115254177
3164, 0.368630935755111, 6.71607270510525, 5.42278455200833, 5.351884161
19281, 2.305874586163, -1.85878097203032, 2.69877382351447, 4.8412186055
0417, 4.40973060799391, 5.04399320650774, 2.68632252661298, 6.0653161065
9912, 3.11881325011993, 3.45532087005125, 3.08442259840346, 4.4356442413
6733, 2.84252623135804, 1.50536798885106, 1.48868622407603, 2.0732283761
5663, 2.5476910210998, 5.66941808257884, 2.16731067416426, 2.49843958833
905, 3.94586413879977, 0.316433764679541, -0.608937441815983, 2.59434365
58557, 1.05516869528337, 2.1447601332725, 6.65846634141906, 2.1771555267
834, 5.23953812029442, 3.53629759842647, 6.03263538017003, 3.85739159396
599, 5.95093453004638, 1.12856987160476, 3.5559912886093, 2.219748642444
89, 3.38471394882135, -1.90805399279409, 3.5113699258973, 4.493199554123
46, 5.10507952638867, 1.08277895384184, 4.58403638422759, 1.373049944268
24, 4.17566975753523, 3.36454182510378, 0.177136582644021, 2.91337423388
405, 3.22796455457526, 2.80124198378441, 1.95189718582788, 3.37659263896
246, -1.6463045238231])

# For Stan we provide all known quantities as data, namely the observed
 data
# and our prior hyperparameters.
norm_inv_gamma_data = {
```

```
                    'dat': data,
                    'N': len(data),
                    'mu': 0,
                    'nu': 0.054,
                    'alpha': 1.12,
                    'beta': 0.4
                  }
```

In [83]:

```
norm_inv_gamma_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> N; // length of data
    real dat[N];  // data
    real<lower=0> alpha;  // fixed prior hyperparameter
    real<lower=0> beta;   // fixed prior hyperparameter
    real<lower=0> mu;   // fixed prior hyperparameter
    real<lower=0> nu;   // fixed prior hyperparameter
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterio
r
// distributions over all parameters.
parameters {
    real new_mu;
    real<lower=0> sigma2;
}

// The model block contains all probability distributions in the model.
// This of this as specifying the generative model for the scenario.
model {
    new_mu ~ normal(mu, sqrt(sigma2/nu));
    sigma2 ~ inv_gamma(alpha, beta);
    for(i in 1:N) {
        dat[i] ~ normal(new_mu, sqrt(sigma2));  // likelihood function
  }
}

"""
```

In [84]:

```
norm_inv_gamma_stan_model = pystan.StanModel(model_code=norm_inv_gamma_s
tan_code)
```

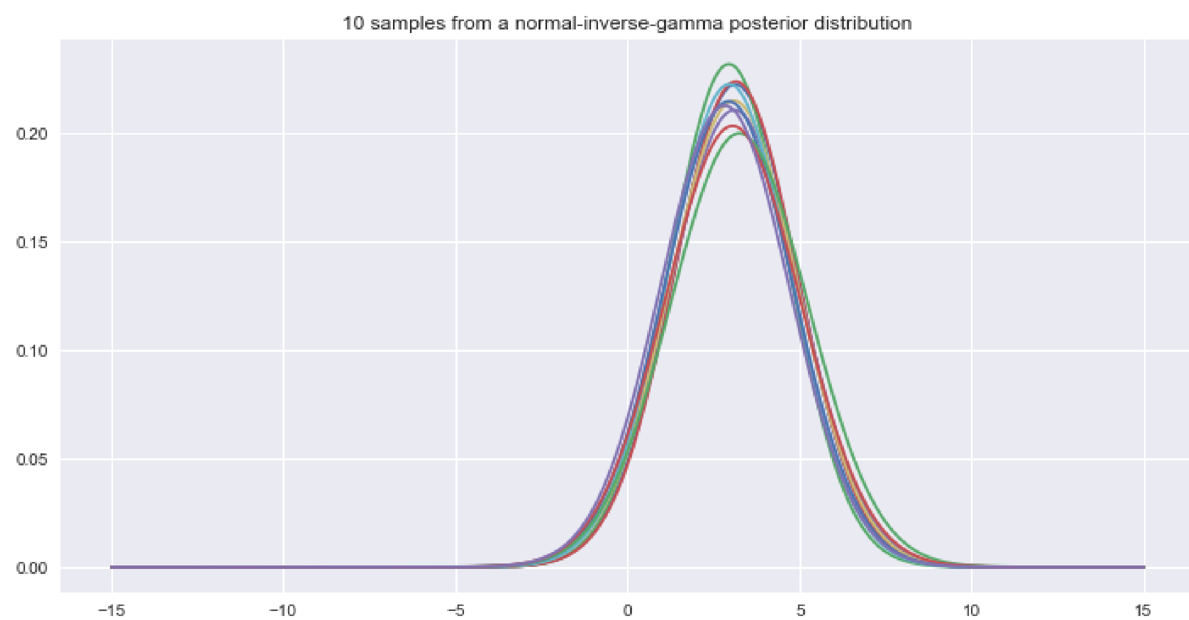
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_0e7626ec078a97e
df40b6a3435e8a217 NOW.
```

In [85]:
```
norm_inv_gamma_stan_results = norm_inv_gamma_stan_model.sampling(data=no
rm_inv_gamma_data)
print(norm_inv_gamma_stan_results)
```

```
Inference for Stan model: anon_model_0e7626ec078a97edf40b6a3435e8a217.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

          mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff
  Rhat
new_mu    3.07  2.2e-3   0.14   2.79   2.98   3.07   3.16   3.34   3759
   1.0
sigma2    3.62  6.2e-3   0.36    3.0   3.36   3.59   3.85   4.38   3314
   1.0
lp__    -233.2    0.02   0.99 -235.9 -233.5 -232.8 -232.5 -232.2   1678
   1.0

Samples were drawn using NUTS at Fri Oct 19 11:19:50 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [91]:
```
norm_inv_gamma_posterior_samples = norm_inv_gamma_stan_results.extract()
print(
    "Posterior 95% confidence interval for mean μ:",
    np.percentile(norm_inv_gamma_posterior_samples['new_mu'], [2.5, 97.5
])
)
print(
    "Posterior 95% confidence interval for variance σ:",
    np.percentile(norm_inv_gamma_posterior_samples['sigma2'], [2.5, 97.5
]),
)

# Generate 10 samples from the posterior
num_samples = 10
mu_samples = np.random.choice(norm_inv_gamma_posterior_samples['new_mu'
],num_samples)
var_samples = np.random.choice(norm_inv_gamma_posterior_samples['sigma2'
],num_samples)

# Plot the normal distributions corresponding to the samples
plt.figure(figsize=(12, 6))
plot_x = np.linspace(-15, 15, 500)
for i in range(num_samples):
    plot_y = stats.norm.pdf(plot_x, loc=mu_samples[i], scale=np.sqrt(var
_samples[i]))
    plt.plot(plot_x, plot_y)
plt.title('%i samples from a normal-inverse-gamma posterior distributio
n' % num_samples)
plt.show()
```

```
Posterior 95% confidence interval for mean μ: [2.7939755  3.33914918]
Posterior 95% confidence interval for variance σ: [3.00350978 4.3776847
2]
```



3. Log-normal HRTEM data. Normal likelihood log-transformed data and using a normal-inverse-gamma prior.

Results to compute:

- 95% posterior confidence intervals for the μ and variance σ of the log-transformed data. (Should match results under Task 3 of the solutions.)

- Take 10 samples from your posterior over μ and σ and plot the log-normal distributions corresponding to them. See Task 5 in the solutions below — you should produce a plot similar the one you find there, but with 10 pdfs rather than one.

Resources for you to use:

- Data set: hrtem.csv (remember to log-transform the data)

- Solution for class activity (hrtem_solution.ipynb)

In [96]:
```
# Load data: read the particle sizes (in nanometers) from a CSV file.
data = np.loadtxt('hrtem.csv')
print('%i data, min: %f, max: %f' % (len(data), min(data), max(data)))
log_data = np.log(data)
print('log data, min: %f, max: %f' % (min(log_data), max(log_data)))
```

```
500 data, min: 1.051827, max: 28.942578
log data, min: 0.050529, max: 3.365314
```

In [98]:
```
hrtem_data = {
                'dat': log_data,
                'N': len(log_data),
                'mu': 2.3,
                'nu': 0.1,
                'alpha': 2,
                'beta': 5
              }
```

In [99]:
```
hrtem_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> N; // length of data
    real dat[N];  // data
    real<lower=0> alpha;  // fixed prior hyperparameter
    real<lower=0> beta;   // fixed prior hyperparameter
    real<lower=0> mu;   // fixed prior hyperparameter
    real<lower=0> nu;   // fixed prior hyperparameter
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterio
r
// distributions over all parameters.
parameters {
    real new_mu;
    real<lower=0> sigma2;
}

// The model block contains all probability distributions in the model.
// This of this as specifying the generative model for the scenario.
model {
    new_mu ~ normal(mu, sqrt(sigma2/nu));
    sigma2 ~ inv_gamma(alpha, beta);
    for(i in 1:N) {
        dat[i] ~ normal(new_mu, sqrt(sigma2));  // likelihood function
  }
}

"""
```

In [100]:
```
hrtem_stan_model = pystan.StanModel(model_code=hrtem_stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_0e7626ec078a97e
df40b6a3435e8a217 NOW.
```

In [102]:
```
hrtem_stan_results = hrtem_stan_model.sampling(data=hrtem_data)
print(hrtem_stan_results)
```

```
Inference for Stan model: anon_model_0e7626ec078a97edf40b6a3435e8a217.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff
  Rhat
new_mu     1.89  5.3e-4   0.03   1.83   1.87   1.89   1.91   1.95   3427
   1.0
sigma2      0.5  5.6e-4   0.03   0.44   0.47   0.49   0.52   0.56   3115
   1.0
lp__     -76.01    0.02   0.97 -78.62 -76.38 -75.72  -75.3 -75.04   1753
   1.0

Samples were drawn using NUTS at Fri Oct 19 12:55:44 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [106]:

```
hrtem_posterior_samples = hrtem_stan_results.extract()
print(
    "Posterior 95% confidence interval for mean μ:",
    np.percentile(hrtem_posterior_samples['new_mu'], [2.5, 97.5])
)
print(
    "Posterior 95% confidence interval for variance σ:",
    np.percentile(hrtem_posterior_samples['sigma2'], [2.5, 97.5]),
)

# Generate 10 samples from the posterior
num_samples = 10
mu_samples = np.random.choice(norm_inv_gamma_posterior_samples['new_mu'
],num_samples)
var_samples = np.random.choice(norm_inv_gamma_posterior_samples['sigma2'
],num_samples)

# Plot the normal distributions corresponding to the samples
plt.figure(figsize=(12,6))
plot_x = np.linspace(0, 30, 200)
for i in range(num_samples):
    plot_y = stats.lognorm.pdf(plot_x, np.sqrt(var_samples[i]), scale=np
.exp(mu_samples[i]))
    plt.plot(plot_x, plot_y)
plt.title('%i samples from a normal-inverse-gamma posterior distributio
n' % num_samples)
plt.show()
```

```
Posterior 95% confidence interval for mean μ: [1.83188172 1.95254423]
Posterior 95% confidence interval for variance σ: [0.43637221 0.5580363
8]
```