Assignment 3: Building Models with Stan

```
import numpy as np
import scipy as sp
import scipy.stats as sts
import matplotlib.pyplot as plt

import pystan
```

Task 1.1: Call Center

```
# Load the data set containing durations between calls arriving at the call
# center during 1 day. All values are in minutes.
waiting_times_day = np.loadtxt('call_center.csv')

# Display some basic information about the data set.
print('Size of data set:', len(waiting_times_day))
print('First 3 values in data set:', waiting_times_day[:3])
print('Sum of data set:', sum(waiting_times_day))

# Make 24 empty lists, one per hour.
waiting_times_per_hour = [[] for _ in range(24)]

# Split the data into 24 separate series, one for each hour of the day.
current_time = 0
for t in waiting_times_day:
    current_hour = int(current_time // 60)
    current_time += t
    waiting_times_per_hour[current_hour].append(t)
```

```
Size of data set: 5856
First 3 values in data set: [30.   3.4  3.2]
Sum of data set: 1441.6838153800093
```

```
call_center_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> num_intervals;  // number of call intervals
    real<lower=0> waiting_time[num_intervals];  // waiting time per interval
    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 posterior
// distributions over all parameters.
parameters {
    real<lower=0> lambda;  // call rate
}

// 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);  // prior over lambdas
    for(i in 1:num_intervals) {
        waiting_time[i] ~ exponential(lambda);  // likelihood function
    }
}

"""
```

```
hour_index = 13

call_center_data = {
    "num_intervals": len(waiting_times_per_hour[hour_index]),
    "waiting_time": waiting_times_per_hour[hour_index],
    "alpha": 1,
    "beta": 0.25,
}
```

```
stan_model_call_center = pystan.StanModel(model_code=call_center_stan_code)
```

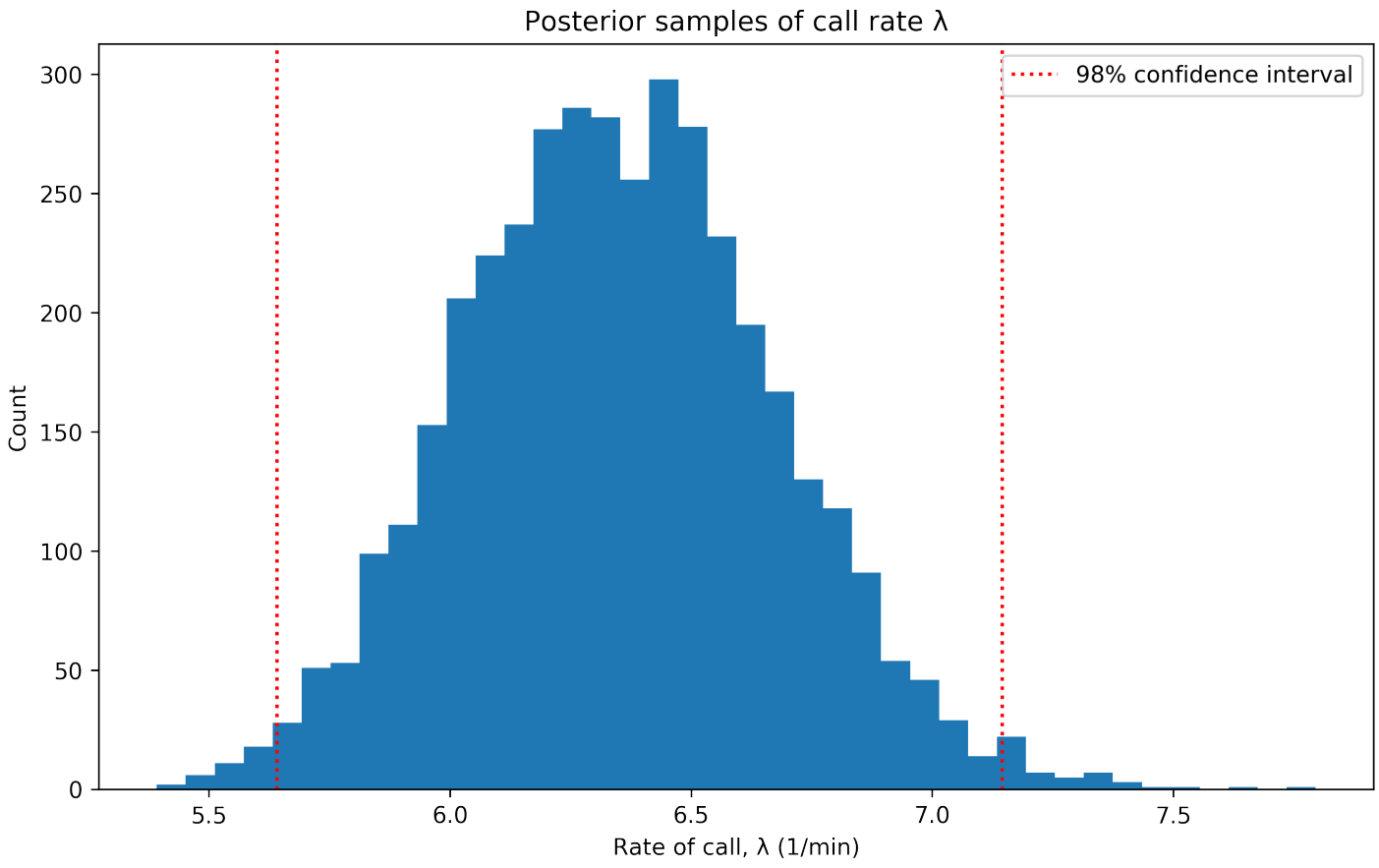
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_5ecbf1e6c285fa2e1535028ff15f
```

```
results_call_center = stan_model_call_center.sampling(data=call_center_data)
```

```
lambda_samples_call_center = results_call_center.extract()['lambda']
conf_int_call_center = np.quantile(lambda_samples_call_center, [0.01, 0.99])
print("98% confidence interval is: [{}, {}]".format(conf_int_call_center[0], conf_ir
```

```
98% confidence interval is: [5.639676852859057, 7.145063523011561]
```

```
plt.figure(figsize=(10, 6), dpi=330)
plt.hist(lambda_samples_call_center, bins=40)
plt.ylabel('Count')
plt.xlabel('Rate of call, λ (1/min)')
plt.title('Posterior samples of call rate λ')
plt.axvline(conf_int_call_center[0], color="red", linestyle=':', label='98% confider
plt.axvline(conf_int_call_center[1], color="red", linestyle=':')
plt.legend()
plt.show()
```



Task 1.2: Normal Likelihood with Normal-Inverse-Gamma Prior

```
data = np.array([3.54551763569501, 4.23799861761927, 4.72138425951628, -0.6922653203
print(len(data), "data")
```

```
200 data
```

```
normal_likelihood_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> data_len;
    real<lower=-1000> normal_likelihood_data[data_len];
    real<lower=0> mu;
    real<lower=0> nu;
    real<lower=0> alpha;
    real<lower=0> beta;
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterior
// distributions over all parameters.
parameters {
    real<lower=-1000> x;
    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 {
    sigma2 ~ inv_gamma(alpha, beta);
    x ~ normal(mu, sqrt(sigma2 / nu));
    for (i in 1:data_len)
        normal_likelihood_data[i] ~ normal(x, sqrt(sigma2));
}

"""
```

```
stan_model_normal_likelihood = pystan.StanModel(model_code=normal_likelihood_stan_co
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_ff6032204bc21e80061c0fadd9bb
```

```
normal_likelihood_data = {
    "data_len": len(data),
    "normal_likelihood_data": data,
    "mu": 0,
    "nu": 0.054,
    "alpha": 1.12,
    "beta": 0.4
}
```

```
normal_likelihood_results = stan_model_normal_likelihood.sampling(data=normal_likeli
print(normal_likelihood_results.stansummary(pars=['x', 'sigma2'], probs=[0.025, 0.97
```

```
Inference for Stan model: anon_model_ff6032204bc21e80061c0fadd9bbf570.
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%  97.5%  n_eff   Rhat
x        3.06  2.3e-3   0.13    2.8   3.33   3463    1.0
sigma2   3.61    0.02   0.35   2.98   4.34    549    1.0

Samples were drawn using NUTS at Sat Oct 19 15:35:06 2019.
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).
```

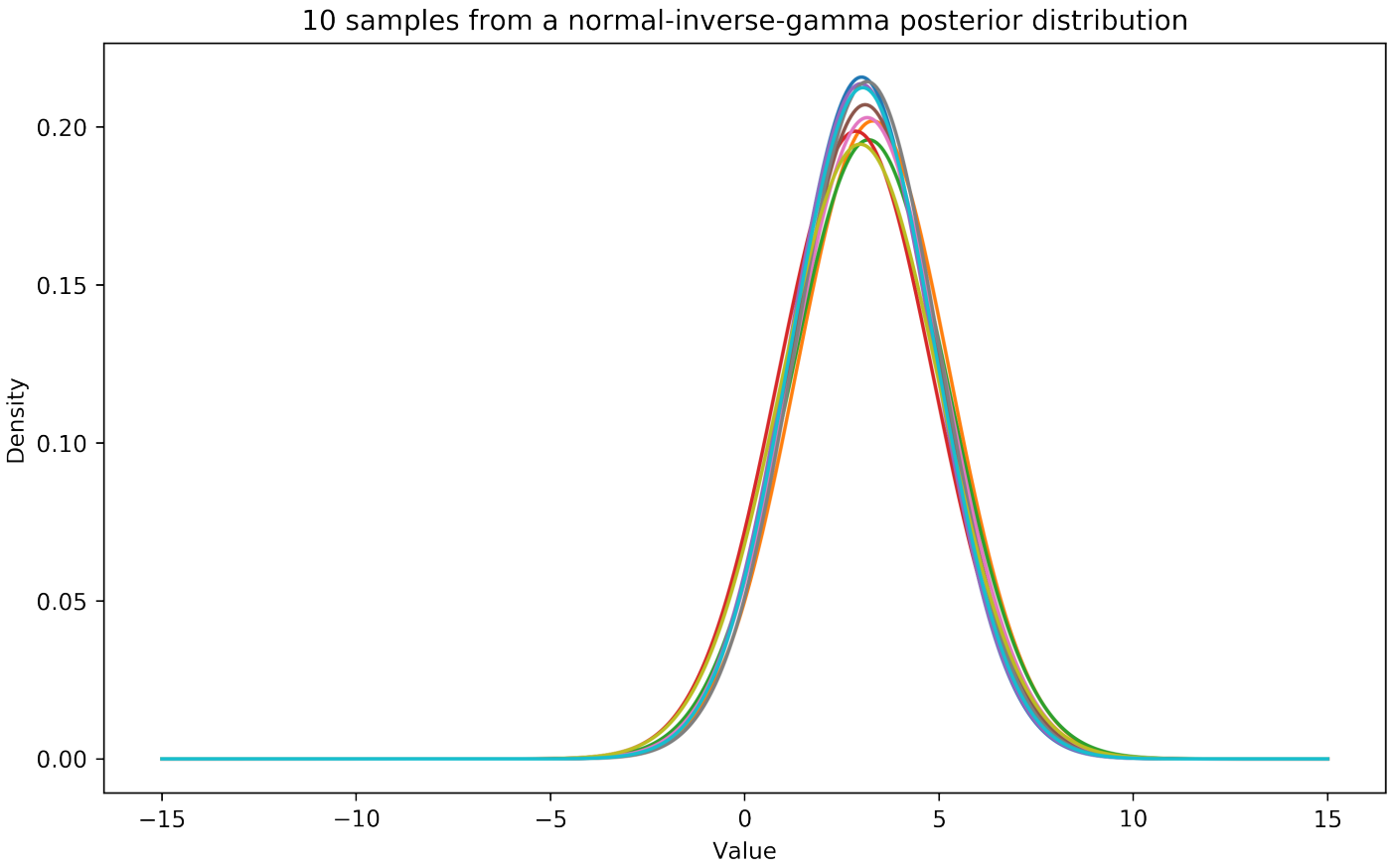
```
raw_normal_likelihood_results = normal_likelihood_results.extract()

num_samples = 10
rand_idx = np.random.choice(range(4000), num_samples) # number of draws
x_samples = raw_normal_likelihood_results['x'][rand_idx]
sigma2_samples = raw_normal_likelihood_results['sigma2'][rand_idx]
```

```
plt.figure(figsize=(10, 6), dpi=330)
plot_x = np.linspace(-15, 15, 500)

for i in range(num_samples):
    plot_y = sts.norm.pdf(plot_x, loc=x_samples[i], scale=np.sqrt(sigma2_samples[i])
    plt.plot(plot_x, plot_y)

plt.ylabel('Density')
plt.xlabel('Value')
plt.title('%i samples from a normal-inverse-gamma posterior distribution' % num_samp
plt.show()
```



Task 1.3: Log-normal HRTEM data

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

```
500 data, min: 0.050529, max: 3.365314
```

```
hrtem_stan_code = """

// The data block contains all known quantities - typically the observed
// data and any constant hyperparameters.
data {
    int<lower=1> data_len;
    real<lower=0> hrtem_data[data_len];
    real<lower=0> mu;
    real<lower=0> nu;
    real<lower=0> alpha;
    real<lower=0> beta;
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterior
// distributions over all parameters.
parameters {
    real<lower=0> x;
    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 {
    sigma2 ~ inv_gamma(alpha, beta);
    x ~ normal(mu, sqrt(sigma2 / nu));
    for (i in 1:data_len)
        hrtem_data[i] ~ normal(x, sqrt(sigma2));
}

"""
```

```
stan_model_hrtem = pystan.StanModel(model_code=hrtem_stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_113a1aa8e14a85988c8e9b622908
```

```
hrtem_data = {
    "data_len": len(log_data),
    "hrtem_data": log_data,
    "mu": 2.3,
    "nu": 0.1,
    "alpha": 2,
    "beta": 5
}
```

```
hrtem_results = stan_model_hrtem.sampling(data=hrtem_data)
print(hrtem_results.stansummary(pars=['x', 'sigma2'], probs=[0.025, 0.975]))
```

```
Inference for Stan model: anon_model_113a1aa8e14a85988c8e9b622908aa26.
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%  97.5%  n_eff   Rhat
x        1.89  5.3e-4   0.03   1.83   1.95   3383    1.0
sigma2    0.5  5.4e-4   0.03   0.44   0.56   3405    1.0

Samples were drawn using NUTS at Sat Oct 19 15:57:48 2019.
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).
```

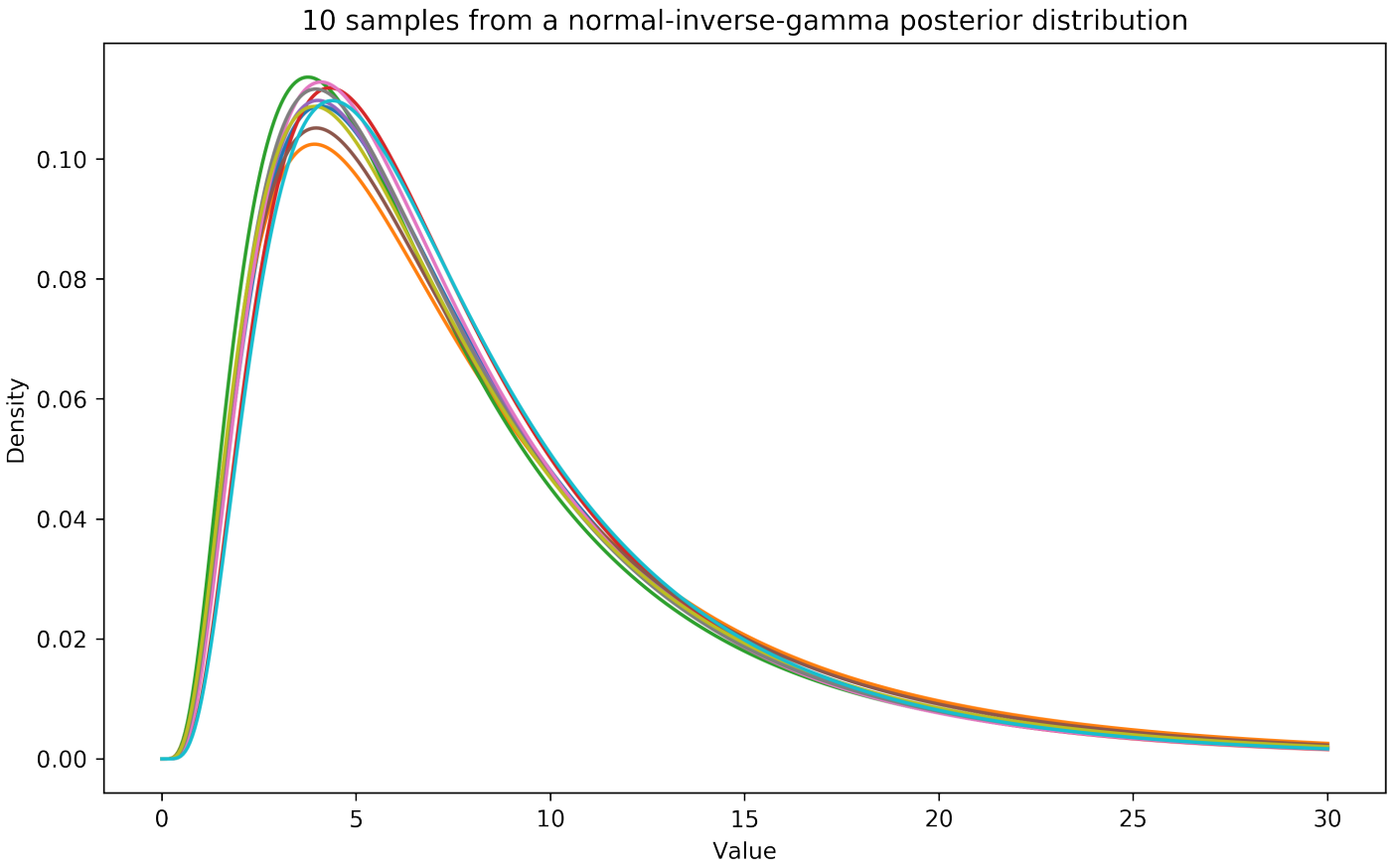
```
raw_hrtem_results = hrtem_results.extract()

num_samples = 10
rand_idx = np.random.choice(range(4000), num_samples) # number of draws
x_samples_hrtem = raw_hrtem_results['x'][rand_idx]
sigma2_samples_hrtem = raw_hrtem_results['sigma2'][rand_idx]
```

```
plt.figure(figsize=(10, 6), dpi=330)
plot_x = np.linspace(0, 30, 500)

for i in range(num_samples):
    plot_y = sts.lognorm.pdf(plot_x, np.sqrt(sigma2_samples_hrtem[i]), scale=np.exp(
    plt.plot(plot_x, plot_y)

plt.ylabel('Density')
plt.xlabel('Value')
plt.title('%i samples from a normal-inverse-gamma posterior distribution' % num_samp
plt.show()
```


10 samples from a normal-inverse-gamma posterior distribution
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
5
10
15
20
25
30
Value