In [1]:
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import pystan
```

1. Call Center Data

Data: waiting times for the 13th hour of a day in a call center
Prior distribution: Gamma distribution with $\alpha = 1$ and $\beta = 0.25$
Likelihood function: exponential with parameter λ
Parameters: rate λ
Posterior: Gamma distribution over λ

In [2]:
```
## import the dataset (code from call_center_solution.ipynb)
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)

# use just the 13th hour of the day
waiting_times_hour = waiting_times_per_hour[13]
```

In [3]:
```
# define the data for the 13th hour of the day
call_center_data = {
    '13': {
        'alpha': 1,  # fixed prior hyperparameters for the
        'beta': 0.25,   # gamma distribution
        'num_calls': len(waiting_times_hour),  # number of calls coming in
        'waiting_times': waiting_times_hour} # data set on waiting times
    }
```

In [4]:
```
# define stan code
calls_stan_code = """

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

// All unknown quantities, in this case the waiting time lambda
parameters {
    real lambd;  // rate lambda for the exponential
}

// The model block contains all probability distributions in the model.
model {
    lambd ~ gamma(alpha, beta);  // prior over lambda
    for(i in 1:num_calls) {
        waiting_times[i] ~ exponential(lambd);  // likelihood function
    }
}

"""
```

In [5]:
```
# define stan model
calls_stan_model = pystan.StanModel(model_code=calls_stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_50fe82232f7cd8b2736c8b3bf1959587 NOW.
```

In [6]:
```
# evaluate model with the dataset and print parameter values
calls_stan_results = calls_stan_model.sampling(data=call_center_data['13'])
print(calls_stan_results.stansummary(pars=['lambd'], probs=[0.01, 0.5, 0.99]))
```

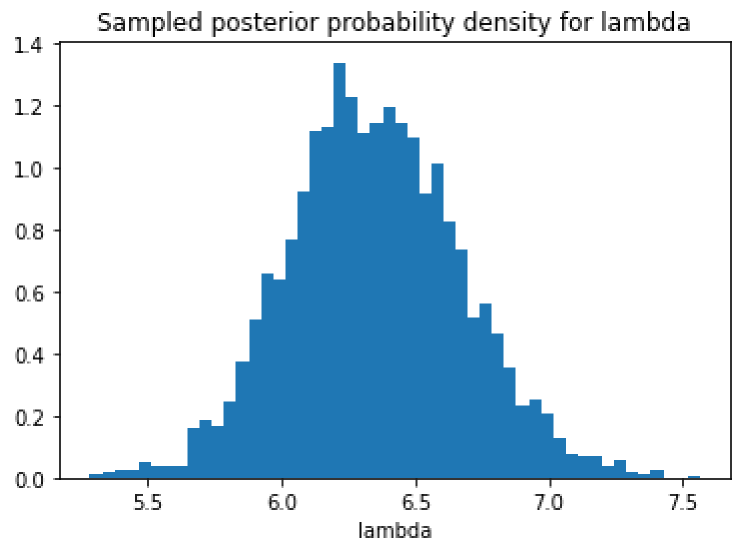
```
Inference for Stan model: anon_model_50fe82232f7cd8b2736c8b3bf1959587.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd     1%    50%    99%  n_eff   Rhat
lambd   6.35  8.0e-3   0.32   5.61   6.34   7.15   1618    1.0

Samples were drawn using NUTS at Fri Oct 16 00:42:46 2020.
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 [7]:
```
# plot a histogram of the distribution of lambdas from the model
posterior_calls_samples = calls_stan_results.extract()
plt.hist(posterior_calls_samples['lambd'], bins=50, density=True)
plt.xlabel("lambda")
plt.title('Sampled posterior probability density for lambda')
print(
    "Posterior 98% confidence interval for lambda:",
    np.percentile(posterior_calls_samples['lambd'], [1, 99]))
plt.show()
```

```
Posterior 98% confidence interval for lambda: [5.61525608 7.14944725]
```



2. Normal likelihood with normal-inverse-gamma prior

Prior distribution: normal inverse gamma distribution
Likelihood function: normal with parameters x and sigma2
Parameters: mean x and variance sigma2
Posterior: normal inverse gamma

In [8]:
```
#define dataset
raw_norm_inv_gamma_data = np.array([3.54551763569501, 4.23799861761927, 4.72138425951628, -0.69226532036
norm_inv_gamma_data = {
        'mu': 0,  # prior mean centered at 0
        'nu': 0.054,    # nu indicates the uncertainty of the prior mean
        'alpha' : 1.12, # alpha and beta govern the marginal prior over the variance
        'beta' : 0.4,
        'data_length': len(raw_norm_inv_gamma_data),  # number of data points
        'norm_data': raw_norm_inv_gamma_data} # data set
```

In [9]:
```
# define the stan model
norm_inv_gamma_stan_code = """

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

// All unknown quantities, in this case the mean and standard deviation of the data
parameters {
    real x;  // mean of the data
    real<lower=0> sigma2; // variance of the data
}

// The model block contains all probability distributions in the model.
model {
    sigma2 ~ inv_gamma(alpha, beta);  // prior over mean
    x ~ normal(mu,sqrt(sigma2/nu)); //prior over variance
    for(i in 1:data_length) {
        norm_data[i] ~ normal(x,sqrt(sigma2));  // likelihood function
    }
}

"""
```

In [10]:
```python
# evaluate stan model
norm_inv_gamma_stan_model = pystan.StanModel(model_code=norm_inv_gamma_stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_775fbb51b12e7e2ff371e277ab8fbfec NOW.
```

In [11]:
```python
# input data to stan model and print the results
norm_inv_gamma_stan_results = norm_inv_gamma_stan_model.sampling(data=norm_inv_gamma_data)
print(norm_inv_gamma_stan_results.stansummary(pars=['x','sigma2'], probs=[0.025, 0.5, 0.975]))
```

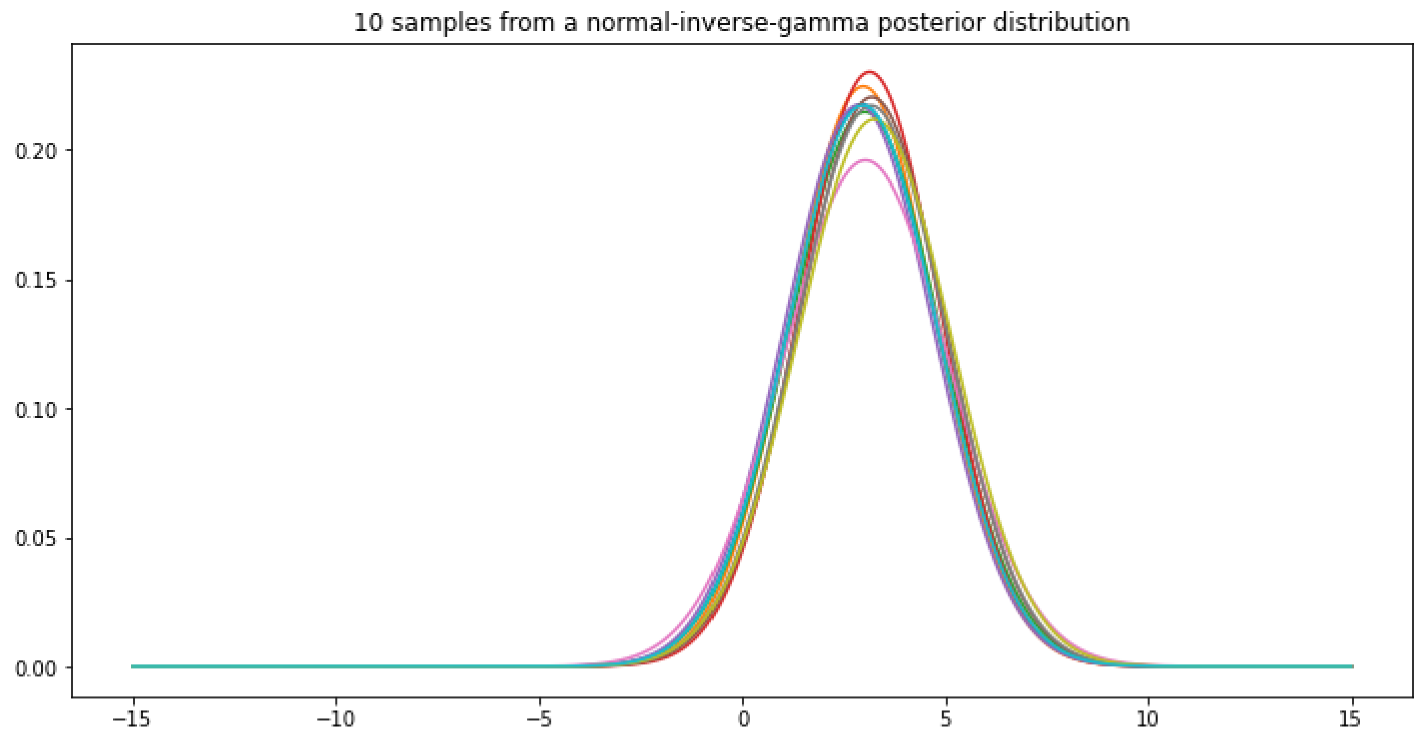
```
Inference for Stan model: anon_model_775fbb51b12e7e2ff371e277ab8fbfec.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean     sd   2.5%    50%  97.5%  n_eff   Rhat
x        3.06  2.4e-3   0.14    2.8   3.06   3.33   3163    1.0
sigma2   3.62  6.7e-3   0.36   2.99   3.59   4.36   2851    1.0

Samples were drawn using NUTS at Fri Oct 16 00:43:47 2020.
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 [12]:
```
# extract results from stan and get 10 random samples
posterior_norm_inv_gamma_samples = norm_inv_gamma_stan_results.extract()
samples_i = np.random.randint(4000,size=10)
xs = posterior_norm_inv_gamma_samples['x'][samples_i]
sigma2s = posterior_norm_inv_gamma_samples['sigma2'][samples_i]

# plot the normal distributions corresponding to the samples
plt.figure(figsize=(12, 6))
plot_x = np.linspace(-15, 15, 500)
for i in range(len(xs)):
    plot_y = stats.norm.pdf(plot_x, loc=xs[i], scale=np.sqrt(sigma2s[i]))
    plt.plot(plot_x, plot_y)
plt.title('%i samples from a normal-inverse-gamma posterior distribution' % len(xs))
plt.show()
```

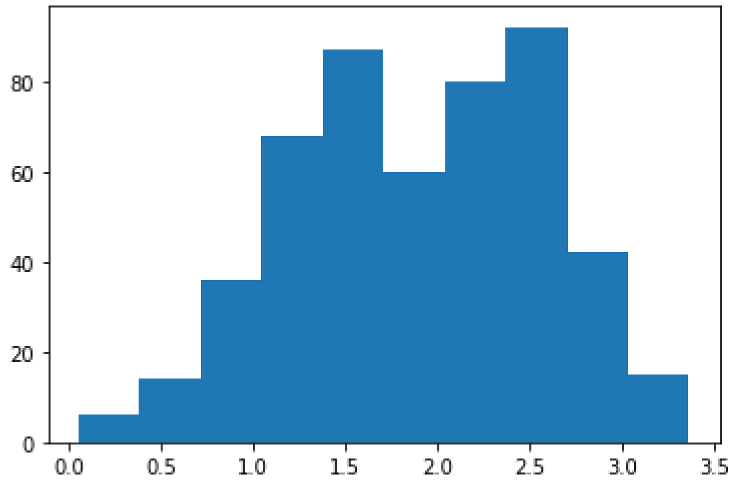


3. Log-normal HRTEM data

Data: particle sizes in nanometers
Prior distribution: normal inverse gamma distribution
Likelihood function: normal with parameters x and sigma2
Parameters: mean x and variance sigma2
Posterior: normal inverse gamma

In [13]:
```
raw_hrtem = np.loadtxt('hrtem.csv')
log_hrtem = np.log(raw_hrtem)
hrtem_data = {
        'mu': 2.3,
        'nu': 0.1,
        'alpha' : 2,
        'beta' : 5,
        'data_length': len(log_hrtem),  # number of data points
        'hrtem_data': log_hrtem} # data set
```

In [14]:
```
plt.hist(log_hrtem)
plt.show()
```



In [15]:
```
hrtem_stan_code = """

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

// All unknown quantities, in this case the mean and standard deviation of the data
parameters {
    real x;  // mean of the data
    real<lower=0> sigma2; // variance of the data
}

// The model block contains all probability distributions in the model.
model {
    sigma2 ~ inv_gamma(alpha, beta);  // prior over mean
    x ~ normal(mu,sqrt(sigma2/nu)); //prior over variance
    for(i in 1:data_length) {
        hrtem_data[i] ~ normal(x,sqrt(sigma2));  // likelihood function
    }
}

"""
```

In [16]:
```
hrtem_stan_model = pystan.StanModel(model_code=hrtem_stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_0c47c1ab3212c3dd8140560a3a5e5e9f NOW.
```

In [17]:
```
hrtem_stan_results = hrtem_stan_model.sampling(data=hrtem_data)
print(hrtem_stan_results.stansummary(pars=['x','sigma2'], probs=[0.025, 0.5, 0.975]))
```

```
Inference for Stan model: anon_model_0c47c1ab3212c3dd8140560a3a5e5e9f.
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%    50%  97.5%  n_eff   Rhat
x        1.89  5.4e-4   0.03   1.83   1.89   1.96   3453    1.0
sigma2    0.5  5.4e-4   0.03   0.44    0.5   0.56   3407    1.0

Samples were drawn using NUTS at Fri Oct 16 00:44:58 2020.
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 [18]:
```
# extract 10 samples from the distribution
posterior_hrtem_gamma_samples = hrtem_stan_results.extract()
hrtem_samples_i = np.random.randint(4000,size=10)
hrtem_xs = posterior_hrtem_gamma_samples['x'][hrtem_samples_i]
hrtem_sigma2s = posterior_hrtem_gamma_samples['sigma2'][hrtem_samples_i]

# plot the log-normal together with the data
plt.figure(figsize=(12, 6))
plot_x = np.linspace(0, 30, 500)
plt.hist(raw_hrtem, bins=20, density=True, alpha =0.5)
for i in range(len(xs)):
    plot_y = stats.lognorm.pdf(plot_x, np.sqrt(hrtem_sigma2s[i]), scale= np.exp(hrtem_xs[i]))
    plt.plot(plot_x, plot_y)
plt.title('%i samples of the posterior log-normal pdf' % len(hrtem_xs))
plt.show()
```