```
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
}

parameters {
  real mu;
  real<lower=0> tau;
  vector[J] theta_tilde;
}

transformed parameters {
  vector[J] theta = mu + tau * theta_tilde;
}

model {
  mu ~ normal(0, 5);
  tau ~ normal(0, 5);
  theta_tilde ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
```

```
import numpy as np__
import tensorflow as tf__
import tensorflow_probability as tfp__
tfd__ = tfp__.distributions
tfb__ = tfp__.bijectors

class eight_schools_model(tfd__.Distribution):

  def __init__(self, J, y, sigma):
    self.J = J
    self.y = tf__.cast(y, tf__.float64)
    self.sigma = tf__.cast(sigma, tf__.float64)

  def log_prob_one_chain(self, params):
    target = 0
    # Data
    J = self.J
    y = self.y
    sigma = self.sigma
    # Parameters
    mu = tf__.cast(params[0], tf__.float64)
    tau = tf__.cast(params[1], tf__.float64)
    theta_tilde = tf__.cast(params[2], tf__.float64)
    # Target log probability computation
    theta = mu + (tau * theta_tilde)
    target += tf__.reduce_sum(tfd__.Normal(tf__.cast(0, tf__.float64),
                                           tf__.cast(5, tf__.float64)).log_prob(mu))
    target += tf__.reduce_sum(tfd__.Normal(tf__.cast(0, tf__.float64),
                                           tf__.cast(5, tf__.float64)).log_prob(tau))
    target += tf__.reduce_sum(tfd__.Normal(tf__.cast(0, tf__.float64),
                                           tf__.cast(1, tf__.float64)).log_prob(theta_tilde))
    target += tf__.reduce_sum(tfd__.Normal(theta, sigma).log_prob(y))
    return target

  def log_prob(self, params):
    return tf__.vectorized_map(self.log_prob_one_chain, params)

  def parameter_shapes(self, nchains__):
    J = self.J
    y = self.y
    sigma = self.sigma
    return [(nchains__, ), (nchains__, ), (nchains__, J)]

  def parameter_bijectors(self):
    J = self.J
    y = self.y
    sigma = self.sigma
    return [tfb__.Identity(),
            tfb__.Chain([tfb__.Shift(tf__.cast(0, tf__.float64)), tfb__.Exp()]),
            tfb__.Identity()]

  def parameter_names(self):
    return ["mu", "tau", "theta_tilde"]
```