```python
#!/usr/bin/python

import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plot

class gaussian_peak:
    def __init__(self, height, mean, var):
        self.height = height
        self.mean = mean
        self.var = var

    def func(self, x):
        return self.height * np.exp(-np.square(x-self.mean)/(2*self.var))

def update_var_est(samp_vec, mean_est):
    n = float(len(samp_vec))
    v = sum([np.square(x-mean_est) for x in samp_vec])
    #Scipy.stats implementation of inv gamma is a one parameter, need to feed
second value as scale
    return st.invgamma.rvs(n/2, scale=v/2)

def update_mean_est(samp_vec, var_est):
     n = float(len(samp_vec))
     #Scale variable is standard deviation
     return st.norm.rvs(np.mean(samp_vec), scale=np.sqrt(var_est/n))

def update_height_est(samp_vec, samp_scaled):
    #Get values for linear regression estimation
    ys = np.matrix(samp_vec).transpose()
    xs = np.matrix(samp_scaled).transpose()
    vbeta = 1./(xs.transpose()*xs)
    beta = vbeta*xs.transpose()*ys

    #used pp 580 of Bayesian data Analysis, Second Edition to do sampling from
dist
    s_2 = (ys-xs*beta).transpose()*(ys-xs*beta)
    var = s_2/st.chi2.rvs(1)

    #.item() is to convert from 1x1 matrix back to scalar
    return st.norm.rvs(beta, scale=np.sqrt(vbeta*var)).item()

def plotter(results):
    plot.figure()
    plot.plot(results["height"], label="$A: " +
`round(np.mean(results["height"]),2)` + "$")
    plot.title("Height results for n=1500")
    plot.ylabel("Height")
    plot.xlabel("Iteration")
    plot.axis([0, 1000, 0, 50])
    plot.legend()

    plot.figure()
    plot.plot(results["mean"], label="$\mu: " +
`round(np.mean(results["mean"]),2)` + "$")
    plot.title("Mean results for n=1500")
    plot.ylabel("Mean")
    plot.xlabel("Iteration")
    plot.legend()

    plot.figure()
    plot.plot(results["var"], label="$\sigma^2: " +
`round(np.mean(results["var"]),2)` + "$")
    plot.title("Variance results for n=1500")
    plot.ylabel("Variance")
    plot.xlabel("Iteration")
    plot.axis([0, 1000, 0, 20])
    plot.legend()

def estimate(rand_vec, samp_vec):
    mean_est = np.random.random()
    var_est = np.random.random()
    results = {"height":[], "mean":[], "var":[]}
    run = 1000
    for i in range(run):
        var_est = update_var_est(rand_vec, mean_est)
        mean_est = update_mean_est(rand_vec, var_est)

        unscaled = gaussian_peak(1, mean_est, var_est)
        approx_vec = [unscaled.func(x) for x in rand_vec]
        height_est = update_height_est(samp_vec, approx_vec)
        results["height"].append(height_est)
        results["mean"].append(mean_est)
        results["var"].append(var_est)

    plotter(results)
    return map(np.mean, ([results["height"], results["mean"], results["var"]]))

#means spaced from 1 to 100
sample_count = 1500
h = 27.50
m = 19.15
v = 4.05

rand_vec = sorted(m + np.sqrt(v)*np.random.randn(sample_count))
actual = gaussian_peak(h, m, v)
samp_vec = [actual.func(x) for x in rand_vec]

[h2, m2, v2] = estimate(rand_vec, samp_vec)
print [h, m, v]
print [h2, m2, v2]

approx = gaussian_peak(h2, m2, v2)
result_vec = [approx.func(x) for x in rand_vec]
plot.figure()
plot.plot(rand_vec, samp_vec, "b",
          label="A:" + `round(h,2)` + "\n$\mu$:" + `round(m,2)` + "\n$\sigma^2$:"
+ `round(v,2)`)
plot.plot(rand_vec, result_vec, "r",
          label="A:" + `round(h2,2)` + "\n$\mu$:" + `round(m2,2)` +
"\n$\sigma^2$:" + `round(v2,2)`)
plot.title("Comparison of estimated and actual values")
plot.ylabel("$Ae^{\\frac{-(x-\mu)}{2\sigma^2}}$", fontsize=22)
plot.xlabel("Input x")
plot.legend()
plot.show()
```