ISTA 421/521 – Homework 4

Due: Monday, October 29, 5pm

20 pts total for Undergrads, 25 pts total for Grads

Ken Youens-Clark

Graduate

Note: I worked with Kai Blumberg and Matt Miller.

1. [5 points; **Required only for Graduates**] Adapted from **Exercise 3.12** of FCMA p.135:

   When performing a Bayesian analysis of the Olympics data, we assumed that σ^2 was known. If instead we assume that $\mathbf{w}$ is known and an inverse Gamma prior is placed on σ^2,

   $$p(\sigma^2|\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}(\sigma^2)^{-\alpha-1}\exp\left\{-\frac{\beta}{\sigma^2}\right\},$$

   then the posterior over σ^2 will also be inverse Gamma. Derive the parameters for the posterior belief in the variance.

   Solution.

   The posterior will be a Gaussian over $\mathbf{w}$ times the given prior:

$$
\begin{aligned}
p(\mathbf{w}|\mu,\sigma^2)p(\sigma^2|\alpha,\beta) &= \frac{1}{(2\pi)^{N/2}|\sigma^2\mathbf{I}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{X}\mathbf{w}-\mathbf{t})^\top(\sigma^2\mathbf{I})^{-1}(\mathbf{X}\mathbf{w}-\mathbf{t})\right\} \\
&\times \frac{\beta^\alpha}{\Gamma(\alpha)}(\sigma^2)^{-\alpha-1}\exp\left\{-\frac{\beta}{\sigma^2}\right\} \\
&= \left(\frac{1}{(2\pi)^{N/2}|\sigma^2\mathbf{I}|^{1/2}}\times\frac{\beta^\alpha}{\Gamma(\alpha)}(\sigma^2)^{-\alpha-1}\right)\exp\left\{-\frac{1}{2}(\mathbf{X}\mathbf{w}-\mathbf{t})^\top(\sigma^2\mathbf{I})^{-1}(\mathbf{X}\mathbf{w}-\mathbf{t})-\frac{\beta}{\sigma^2}\right\} \\
&= \left(\frac{1}{(2\pi)^{N/2}}\times\frac{1}{|\sigma^2\mathbf{I}|^{1/2}}\times\frac{\beta^\alpha}{\Gamma(\alpha)}\times(\sigma^2)^{-\alpha-1}\right)\exp\left\{-\frac{1}{2\sigma^2}(\mathbf{X}\mathbf{w}-\mathbf{t})^\top(\mathbf{X}\mathbf{w}-\mathbf{t})-\frac{\beta}{\sigma^2}\right\} \\
&= \left(\frac{1}{(2\pi)^{N/2}}\times\frac{1}{|\sigma^2\mathbf{I}|^{1/2}}\times\frac{\beta^\alpha}{\Gamma(\alpha)}\times(\sigma^2)^{-\alpha-1}\right)\exp\left\{-\frac{1}{2\sigma^2}((\mathbf{X}\mathbf{w})^\top-\mathbf{t}^\top)(\mathbf{X}\mathbf{w}-\mathbf{t})-\frac{\beta}{\sigma^2}\right\} \\
&= \left(\frac{1}{(2\pi)^{N/2}}\times\frac{1}{|\sigma^2\mathbf{I}|^{1/2}}\times\frac{\beta^\alpha}{\Gamma(\alpha)}\times(\sigma^2)^{-\alpha-1}\right)\exp\left\{\frac{-\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w}-2\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}+\mathbf{t}^\top\mathbf{t})-\beta}{\sigma^2}\right\} \\
&= \left(\frac{1}{|\sigma^2\mathbf{I}|^{1/2}}\times(\sigma^2)^{-\alpha-1}\right)\exp\left\{\frac{-\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w})+\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}-\frac{1}{2}\mathbf{t}^\top\mathbf{t}-\beta}{\sigma^2}\right\} \\
&= \left(\frac{(\sigma^2)^{-\alpha-1}}{(\sigma^2)^{D/2}}\right)\exp\left\{\frac{-\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w})+\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}-\frac{1}{2}\mathbf{t}^\top\mathbf{t}-\beta}{\sigma^2}\right\} \\
&= \left((\sigma^2)^{-\alpha-1}(\sigma^2)^{-D/2}\right)\exp\left\{\frac{-\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w})+\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}-\frac{1}{2}\mathbf{t}^\top\mathbf{t}-\beta}{\sigma^2}\right\} \\
&= \left((\sigma^2)^{(-\alpha-D/2-1)}\right)\exp\left\{\frac{-\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w})+\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}-\frac{1}{2}\mathbf{t}^\top\mathbf{t}-\beta}{\sigma^2}\right\} \\
\hat{\alpha} &= \alpha + D/2 \\
\hat{\beta} &= -\frac{1}{2}((\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w})+\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}-\frac{1}{2}\mathbf{t}^\top\mathbf{t}-\beta
\end{aligned}
$$

2. [6 points] Adapted from **Exercise 4.2** of FCMA p.163:

   In Chapter 3, we computed the posterior density over r, the probability of a coin giving heads, using a beta prior and a binomial likelihood. Recalling that the beta prior, with parameters α and β, is given by

$$p(r|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}r^{\alpha-1}(1-r)^{\beta-1}$$

   and the binomial likelihood, assuming y heads in N throws, is given by

$$p(y|r,N) = \binom{N}{y}r^y(1-r)^{N-y}\ ,$$

   compute the Laplace approximation to the posterior. (Note, you should be able to obtain a closed-form solution for the MAP value, $\hat{r}$, by getting the log posterior, differentiating (with respect to r), equating to zero and solving for r.)

Solution.

$$
\begin{aligned}
f &= p(r|\alpha,\beta)p(y|r,N) \\
&\propto r^{\alpha+y_N-1}(1-r)^{\beta+N-y_N-1} \\
&\propto r^{\delta-1}(1-r)^{\gamma-1} \\
&\text{where } \delta = y_N+\alpha \text{ and } \gamma = N-y_N+\beta \\
\log(f) &= \log(r)(\delta-1)+\log(1-r)(\gamma-1) \\
\frac{\partial \log(f)}{\partial r} &= \frac{\delta-1}{r}-\frac{\gamma-1}{1-r}=0 \\
\frac{\delta-1}{r} &= \frac{\gamma-1}{1-r} \\
(\delta-1)(1-r) &= r(\gamma-1) \\
\delta-r\delta-1+r &= r\gamma-r \\
2r-r\delta-r\gamma &= 1-\delta \\
r(2-\delta-\gamma) &= 1-\delta \\
\hat{r} &= \frac{1-\delta}{2-\delta-\gamma} \\
\hat{r} &= \frac{1-(y_N+\alpha)}{2-(y_N+\alpha)-(N-y_N+\beta)} \\
\hat{r} &= \frac{1-y_N-\alpha}{2-y_N-\alpha-N+y_N-\beta} \\
\hat{r} &= \frac{1-y_N-\alpha}{2-\alpha-N-\beta} \\
\frac{\partial \log(f)}{\partial r} &= \frac{\delta-1}{r}-\frac{\gamma-1}{1-r} \\
&= r^{-1}(\delta-1)-(1-r)^{-1}(\gamma-1) \\
\frac{\partial^2 \log(f)}{\partial r} &= -r^{-2}(\delta-1)-(1-r)^{-2}(\gamma-1) \\
&= -\frac{y_N+\alpha-1}{r^2}-\frac{(N-y_N+\beta-1)}{(1-r)^2}
\end{aligned}
$$

The Laplace approximation is $\mathcal{N}(\mu,\Sigma)$ where

$$
\begin{aligned}
\mu &= \hat{r} \\
\Sigma^{-1} &= -\left(\frac{\partial^2 \log(f)}{\partial r}\right)\Big|_{\hat{r}}
\end{aligned}
$$

3. [4 points] Adapted from **Exercise 4.3** of FCMA p.163:

   In the previous exercise you computed the Laplace approximation to the true beta posterior. In this problem, plot both the true beta posterior and the Laplace approximation for the following three parameter settings:

   1. $\alpha = 5$, $\beta = 5$, $N = 20$, and $y = 10$,
   2. $\alpha = 3$, $\beta = 15$, $N = 10$, and $y = 3$,
   3. $\alpha = 1$, $\beta = 30$, $N = 10$, and $y = 3$.

Be sure to clearly indicate the values in your plot captions. Include how the two distributions (the true beta posterior and the Laplace approximation) compare in each case. Include the python script you use to generate these plots; the script should be named `plot_laplace_approx.py`. **Suggestion**: for plotting the beta and Gaussian (Normal) distributions, you can use `scipy.stats.beta` and `scipy.stats.normal` to create the beta and Gaussian random variables, and use the `pdf(x)` method for each to generate the curves. Note that for `scipy.stats.normal`, the mean is the location (`loc`) parameter, and the sigma is the `scale` parameter. Also, `scipy.stats.normal` expects the scale parameter to be the standard deviation (i.e., take the square root: `math.sqrt(x)`) of the variance you'll compute for the Laplace approximation.

Solution.

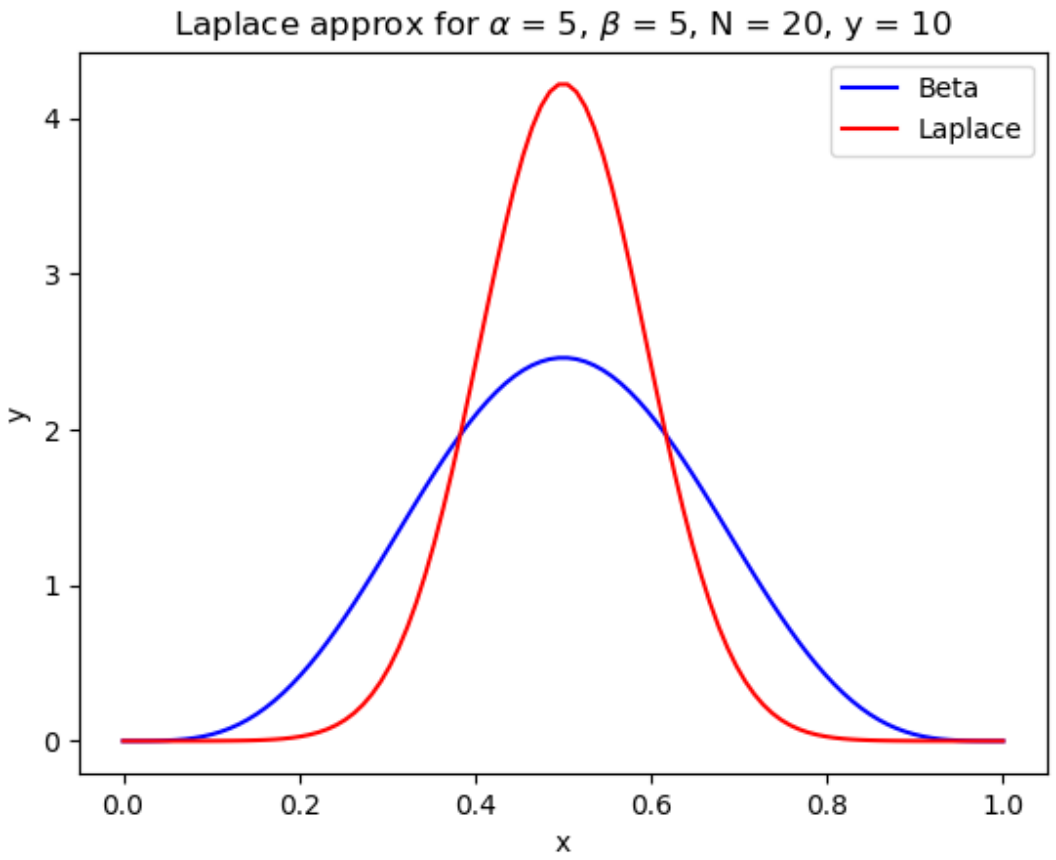

Figure 1: Laplace approximation of Beta distibution for $\alpha = 5, \beta = 5, N = 20, y = 10$.

Here the Laplace approximation can easily match the mode but overshoots the intensity. Overall this is a lackluster fit.

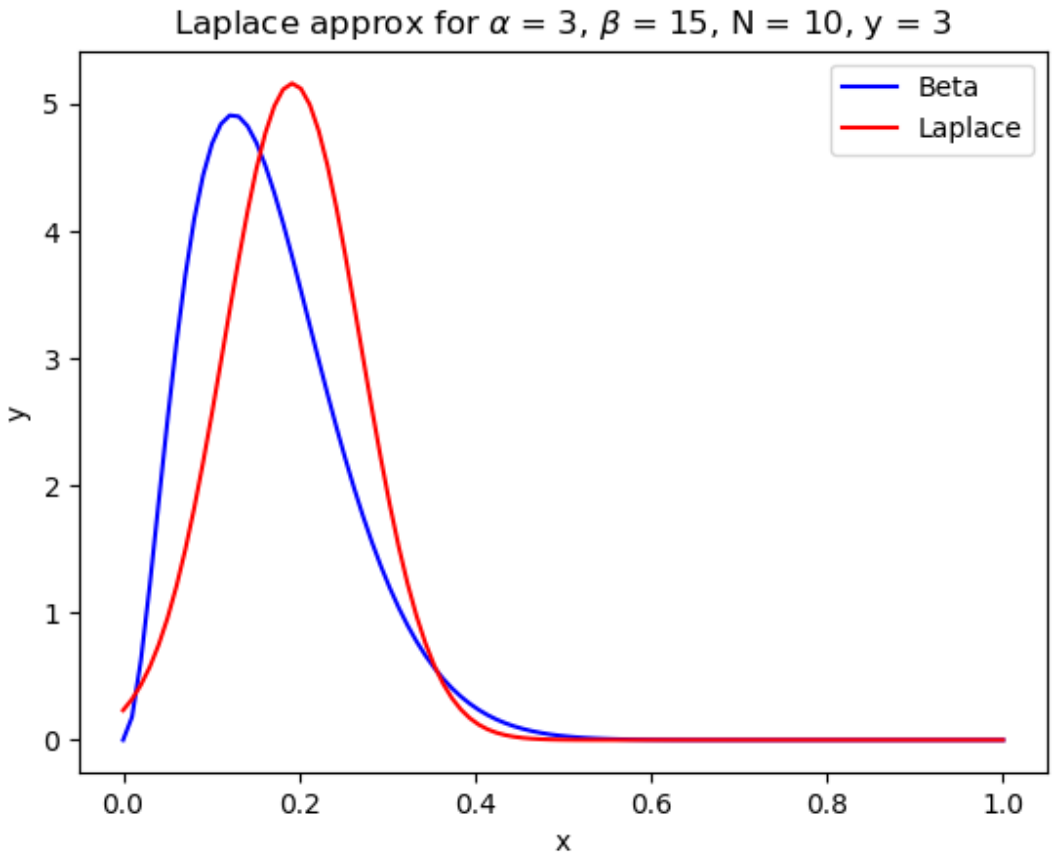

Figure 2: Laplace approximation of Beta distibution for $\alpha = 3, \beta = 15, N = 10, y = 3$.

The approximation seems quite good in shape though it slightly misses the peak in both location and size.

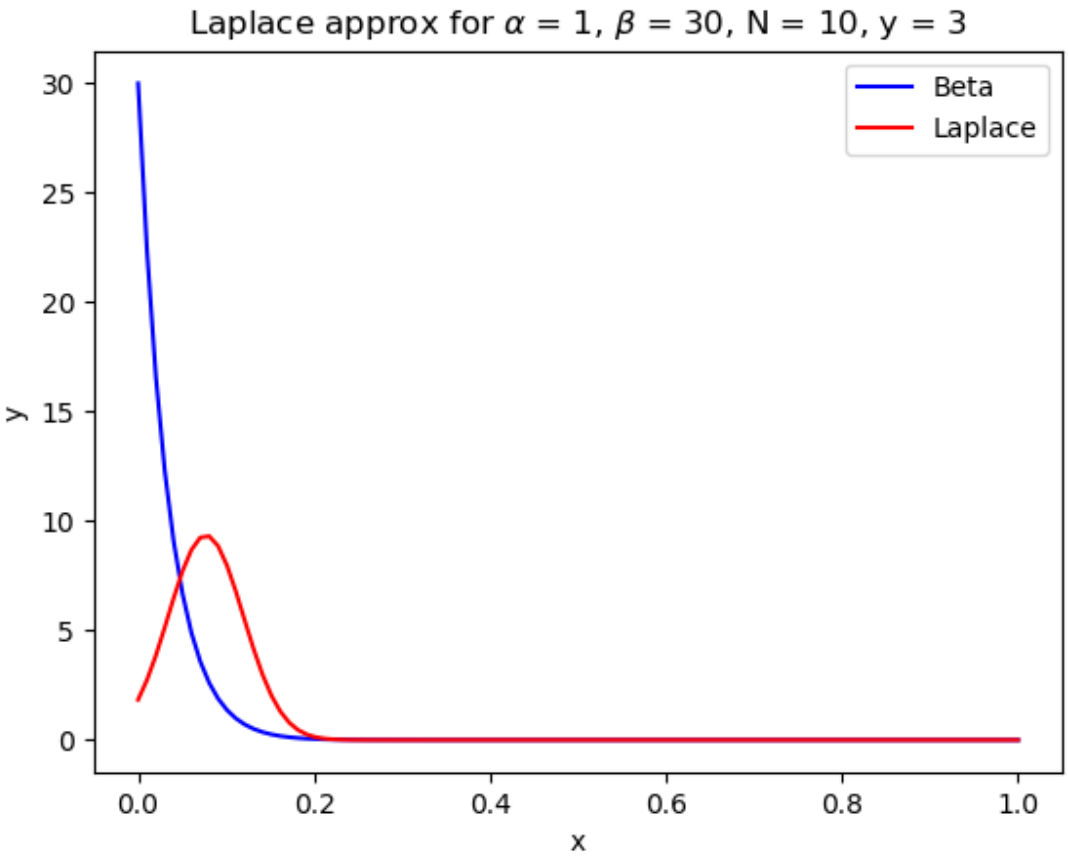

Figure 3: Laplace approximation of Beta distibution for $\alpha = 1, \beta = 30, N = 10, y = 3$.

The approximation is not able to show the true Beta form's heavily skewed distribution to the left.

Following is my code to plot these:

```
#!/usr/bin/env python3
"""
Author : Ken Youens-Clark <kyclark@email.arizona.edu>
Date   : 2018-10-25
Purpose: Plot Laplace estimation of beta distribution
"""

import argparse
import sys
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import beta, norm


# --------------------------------------------------
def get_args():
```

```
    """get args"""
    parser = argparse.ArgumentParser(
        description='Plot Laplace estimation of beta distribution',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-a',
        '--alpha',
        help='alpha value',
        metavar='int',
        type=int,
        default=5)

    parser.add_argument(
        '-b', '--beta', help='beta value', metavar='int', type=int, default=5)

    parser.add_argument(
        '-n',
        '--num_samples',
        help='n value',
        metavar='int',
        type=int,
        default=20)

    parser.add_argument(
        '-y', '--num_y', help='y value', metavar='int', type=int, default=10)

    parser.add_argument(
        '-o',
        '--outfile',
        help='Output file',
        metavar='str',
        type=str,
        default=None)

    parser.add_argument(
        '-d', '--debug', help='Talk about (pop music)', action='store_true')

    parser.add_argument(
        '-N', '--no_viz', help='Do not show pictures', action='store_true')

    return parser.parse_args()


# --------------------------------------------------
def warn(msg):
    """Print a message to STDERR"""
    print(msg, file=sys.stderr)


# --------------------------------------------------
```

```
def die(msg='Something bad happened'):
    """warn() and exit with error"""
    warn(msg)
    sys.exit(1)


# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    alpha_val = args.alpha
    beta_val = args.beta
    n_val = args.num_samples
    y_val = args.num_y
    out_file = args.outfile

    def debug(msg):
        if args.debug:
            warn(msg)

    debug('$\alpha = {} \beta = {} N = {} y = {}$'.format(
        alpha_val, beta_val, n_val, y_val))
    r_hat = (1 - y_val - alpha_val) / (2 - alpha_val - n_val - beta_val)
    debug('r_hat = {}'.format(r_hat))

    t1 = (y_val + alpha_val - 1) / np.power(r_hat, 2)
    t2 = (n_val - y_val + beta_val - 1) / np.power((1 - r_hat), 2)
    d2 = (-1 * t1) - t2
    sigma = -1 / d2
    debug('d2 = {}'.format(d2))
    debug('sigma = {}'.format(sigma))

    beta_dist = beta(alpha_val, beta_val)
    x = np.linspace(0, 1, 100)

    plt.figure()
    plt.plot(x, beta_dist.pdf(x), 'blue')

    debug(norm.pdf(x, loc=r_hat, scale=np.sqrt(sigma)))
    plt.plot(x, norm.pdf(x, loc=r_hat, scale=np.sqrt(sigma)), 'red')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(['Beta', 'Laplace'])

    tmpl = r'Laplace approx for $\alpha$ = {}, $\beta$ = {}, N = {}, y = {}'
    title = tmpl.format(alpha_val, beta_val, n_val, y_val)
    plt.title(title)

    if out_file:
        ext = '.png'
```

```
        if not out_file.endswith(ext):
            out_file += ext

        plt.savefig(out_file, fmt='png')

    if not args.no_viz:
        plt.show()


# --------------------------------------------------
if __name__ == '__main__':
    main()
```

4. [4 points] Adapted from **Exercise 4.4** of FCMA p.164:

   Given the expression for the area of a circle, $A = \pi r^2$, and *using only uniformly distributed random variates*, devise a sampling approach for estimating π. Describe your method in detail and provide your script to do the estimation – this script should be called `pi_sample_estimate.py`. Report your estimate based on 1 million samples to 6 decimal places. (NOTE: You do *not* need to use Metropolis-Hastings to compute this.)

   Solution.

   Here is the code I wrote:

```
#!/usr/bin/env python3
"""
Author:  Ken Youens-Clark
Date:    October 22, 2018
Purpose:

A circle of radius R is inscribed in a square with sides 2R. The area
of the circle is pi*R^2 and the area of the square is (2R)^2 or 4R^2.
Therefore the ratio of the areas is pi/4.

To estimate pi, we will choose N samples from a uniform distribution
between 0 and the radius of the circle (1). We can then use the
Pythagorean theorem (a^2 + b^2 = c^2) (via the math.hypot function
that computes Euclidean distance) to find the distance from the origin
(0,0). If this is less than the radius squared (which is just one
here), then the point falls within the circle.  Because the ratio
of the area of the circle to the square is pi/4, multiply
the number found to be within the circle by 4 and then divide by the
number of samples to estimate pi.

Cf. https://en.wikipedia.org/wiki/Approximations_of_%CF%80
"""

import argparse
import sys
from random import random
from math import hypot
```

```
# --------------------------------------------------
def get_args():
    """get args"""
    parser = argparse.ArgumentParser(
        description='Estimate pi',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-n',
        '--num_samples',
        help='Number of samples',
        metavar='int',
        type=int,
        default='1000000')

    return parser.parse_args()


# --------------------------------------------------
def warn(msg):
    """Print a message to STDERR"""
    print(msg, file=sys.stderr)


# --------------------------------------------------
def die(msg='Something bad happened'):
    """warn() and exit with error"""
    warn(msg)
    sys.exit(1)


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    num_samples = args.num_samples

    if num_samples < 1:
        die('-n ({}) cannot be less than 1'.format(num_samples))

    count = 0
    for _ in range(0, num_samples):
        x, y = random(), random()
        if hypot(x, y) <= 1:
            count += 1

    print('pi ~ {:.06f}'.format(count * 4 / num_samples))


# --------------------------------------------------
if __name__ == '__main__':
```

```
    main()
```

Here is the code with the default value of 1M samples:

```
$ ./pi_sample_estimate.py
pi ~ 3.138772
$ ./pi_sample_estimate.py
pi ~ 3.140552
$ ./pi_sample_estimate.py
pi ~ 3.141736
$ ./pi_sample_estimate.py
pi ~ 3.140920
```

It's interesting to compare to fewer samples:

```
$ ./pi_sample_estimate.py -n 1000
pi ~ 3.156000
$ ./pi_sample_estimate.py -n 10000
pi ~ 3.146000
$ ./pi_sample_estimate.py -n 100000
pi ~ 3.135040
```

5. [6 points] Adapted from **Exercise 4.6** of FCMA p.164:

   Assume that we observe N vectors of attributes, $\mathbf{x}_1, ..., \mathbf{x}_N$, and associated integer counts $t_1, ..., t_N$. A Poisson likelihood would be suitable:

$$p(t_n|\mathbf{x}_n, \mathbf{w}) = \frac{f(\mathbf{x}_n; \mathbf{w})^{t_n} \exp\{-f(\mathbf{x}_n; \mathbf{w})\}}{t_n!},$$

   where $f(\mathbf{x}_n; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}_n$. Assuming a zero-mean Gaussian prior on $\mathbf{w}$ with constant diagonal covariance of σ^2, derive the gradient and Hessian of the posterior. Using these, express the parameter update rules for (a) gradient *ascent* (because we're maximizing) update (in class we looked at Widrow-Hoff, which is typically expressed for *descent*), and (b) Newton-Raphson.

   The following facts will help in the derivation. First, keep in mind that although $\mathbf{w}$ and $\mathbf{x}_n$ are vectors (of the same dimension), their dot product, $\mathbf{w}^\top \mathbf{x}_n$, is a *scalar* value. This means you can take the partial derivative of $\log \mathbf{w}^\top \mathbf{x}_n$ with respect to $\mathbf{w}$. Also, remember that the Hessian is a matrix representing the second partial derivatives of the gradient with respect to itself (see Comment 2.6 of p.73), and the second derivative will involve the *transpose* of the partial derivative with respect to $\mathbf{w}$. So, e.g., as part of taking the second derivative, if you are taking the transpose derivative part of $\mathbf{w}^\top \mathbf{x}_n$, as follows:

$$\frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial \mathbf{w}} = \mathbf{x}_n \quad \text{and} \quad \frac{\partial(\mathbf{w}^\top \mathbf{x}_n)}{\partial \mathbf{w}^\top} = \mathbf{x}_n^\top$$

   Solution.

The Gaussian prior

$$p(\mathbf{w}|\mu,\sigma^2) = \frac{1}{(2\pi)^{N/2}|\sigma^2\mathbf{I}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{w}\ -\mu)^\top(\sigma^2\mathbf{I})^{-1}(\mathbf{w}\ -\mu)\right\}$$

Since $\mu = 0$

$$= \frac{1}{(2\pi\sigma^2)^{D/2}}\exp\left\{-\frac{1}{2\sigma^2}\mathbf{w}^\top\mathbf{w}\right\}$$

$$f = p(t_n|\mathbf{x}_n,\mathbf{w})p(\mathbf{w}|\mu,\sigma^2) = \frac{(\mathbf{w}^\top\mathbf{x}_n)^{t_n}\exp\{-\mathbf{w}^\top\mathbf{x}_n\}}{t_n!}\times\frac{1}{(2\pi\sigma^2)^{D/2}}\exp\left\{-\frac{1}{2\sigma^2}\mathbf{w}^\top\mathbf{w}\right\}$$

$$\log(f) = t_n\log(\mathbf{w}^\top\mathbf{x}_n) + (-\mathbf{w}^\top\mathbf{x}_n) - \log(t_n!) + \log(\frac{1}{(2\pi\sigma^2)^{D/2}}) - \frac{1}{2\sigma^2}\mathbf{w}^\top\mathbf{w}$$

$$\frac{\partial\log(f)}{\partial\mathbf{w}} = t_n\mathbf{x}_n(\mathbf{w}^\top\mathbf{x}_n)^{-1} - \mathbf{x}_n - \frac{1}{\sigma^2}\mathbf{w}$$

$$\frac{\partial^2\log(f)}{\partial\mathbf{w}\partial\mathbf{w}^\top} = -(t_n\mathbf{x}_n^\top\mathbf{x}_n(\mathbf{w}^\top\mathbf{x}_n)^{-2}) - \frac{1}{\sigma^2}$$

(a) Widrow-Hoff update:

$$w_{n+1} = w_n + \alpha\left(\frac{\partial\log(f)}{\partial\mathbf{w}}\right)$$

$$= w_n + \alpha\left(t_n\mathbf{x}_n(\mathbf{w}^\top\mathbf{x}_n)^{-1} - \mathbf{x}_n - \frac{1}{\sigma^2}\mathbf{w}\right)$$

(b) Newton-Raphson update:

$$w_{n+1} = w_n - \left(\frac{\partial^2\log(f)}{\partial\mathbf{w}\partial\mathbf{w}^\top}\right)^{-1}\frac{\partial\log(f)}{\partial\mathbf{w}}$$

$$= w_n - \left(-(t_n\mathbf{x}_n^\top\mathbf{x}_n(\mathbf{w}^\top\mathbf{x}_n)^{-2}) - \frac{1}{\sigma^2}\right)^{-1}\left(t_n\mathbf{x}_n(\mathbf{w}^\top\mathbf{x}_n)^{-1} - \mathbf{x}_n - \frac{1}{\sigma^2}\mathbf{w}\right)$$