PHP 2530: BAYESIAN STATISTICAL METHODS
HOMEWORK II PYTHON APPENDIX

NICK LEWIS

Packages

```
import numpy as np #useful math functions and everything else
from numpy.random import rand #Uniform(0,1). just makes writing more succinct
import pandas as pd #Use this to create dataframes for nice manipulations
import matplotlib.pyplot as plt #for plotting our histograms and contours
import statsmodels.formula.api as smf #for linear regression in Problem 5
import statsmodels.api as sm
import matplotlib.gridspec as gridspec #Package to help with plotting

#lets us use probability distributions like t, beta, gamma,etc.
from scipy.stats import dirichlet, norm, gamma, poisson,t, beta, chi2, binom

'''
NOTES:
 When using np.var from numpy package, set ddof = 1.
np.var = sum^n_j=1 (X_j - np.mean(X))^2 / (n-ddof)
ddof is automatically set to 0 so you need to set ddOf=1. Likewise with np.std
'''
```

Problem 1

```
### PROBLEM 1 (BDA 3rd Ed. Exercise 3.2)

### METHOD 1: SAMPLE DIRECTLY FROM THE THETAS, THEN MANUALLY CALCULATE
pre_theta = dirichlet.rvs([295, 308, 39], size=10000, random_state=1)
post_theta = dirichlet.rvs([289, 333, 20], size=10000, random_state=1)

#Distribution of those who preferred bush to Dukakis before the debate
pre_alpha = pre_theta[:,0] / (pre_theta[:,0] + pre_theta[:,1])
#Distribution of those who preferred bush to Dukakis after the debate
post_alpha =  post_theta[:,0] / (post_theta[:,0] + post_theta[:,1])

#Gives us density of those who prefer Bush to Dukakis
diff = post_alpha - pre_alpha

plt.hist(x = diff, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
plt.ylabel('Frequency')
plt.xlabel(r'$\alpha_{post}$' + ' - ' + r'$\alpha_{pre}$')
plt.title('Frequency of Those Who Prefer Bush to Dukakis')
plt.show()

print(f"The posterior probability of a shift towards Bush is {100*np.mean(diff > 0)}%")

### METHOD 2: SAMPLE DIRECTLY FROM THE DISTRIBUTION OF ALPHA
```

```python
#Distribution of those who preferred bush to Dukakis before the debate
pre_alpha = beta.rvs(a=295, b=308, size=10000)
#Distribution of those who preferred bush to Dukakis after the debate
post_alpha = beta.rvs(a=289, b=333, size=10000)

#Gives us density of those who prefer Bush to Dukakis
diff = post_alpha - pre_alpha

plt.hist(x = diff, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
plt.ylabel('Frequency')
plt.xlabel(r'$\alpha_{post}$' + ' - ' + r'$\alpha_{pre}$')
plt.title('Frequency of Those Who Prefer Bush to Dukakis')
plt.show()

print(f"The posterior probability of a shift towards Bush is {100*np.mean(diff > 0)}%")
```

Problem 2

```python
### PROBLEM 2 (BDA 3rd Ed. Exercise 3.3)

#treatment group
#sample size, mean and standard deviation
n_t = 36; mean_t = 1.173; sd_t = 0.20 / np.sqrt(n_t)
#distribution of treatment group mean
mu_t = t.rvs(df = n_t-1, loc = mean_t, scale = sd_t, size = 10000)

#control group
#sample size, mean and standard deviation
n_c = 32; mean_c = 1.013; sd_c = 0.24 / np.sqrt(n_c)
#distribution of control group mean
mu_c = t.rvs(df = n_c-1, loc = mean_c, scale = sd_c, size = 10000)

# Our difference in means
mu = mu_t - mu_c

plt.hist(x = mu, bins='auto', color='yellow', alpha=0.7, rwidth=0.85)
plt.xlabel(r'$\mu_{treated}$' + ' - ' +  r'$\mu_{control}$')
plt.ylabel('Frequency')
plt.title('Histogram of Mean Difference Between Treatment and Control Group')
plt.show()

#mean and variance of our difference along with confidence interval bounds
print(f"Our mean for the difference is {np.round(mu.mean(),3)}")
print(f"Our standard deviation for the difference is {np.round(mu.std(ddof=1),3)}")
print(f"Our credible interval for our difference in means  two groups is "
      f"{np.round(np.percentile(mu,[2.5,97.5]),3)}")
```

Problem 3

```python
#PROBLEM 3 (BDA 3rd Ed. Exercise 3.5)

#data
w = np.array([10,10,12,11,9])
```

```python
#create grid for mu, log(sigma)
A=1000 #technically don't need this many, but Python works super fast
moo = np.linspace(start = 1, stop = 20, num = A)
lsig = np.linspace(start = -3, stop = 3, num = A)

#PART A : Assume unrounded measurements
def unrounded(a, b, x):
    '''
    Parameters:
        a - grid space for mean parameter
        b - grid space for standard deviation parameter
        x - data vector
    Returns:
        log posterior function
    '''
    #sample size, mean and variance  for data vector
    n = len(x); v = x.mean(); s = x.var(ddof=1)
    b = np.exp(b)     # translate log(sigma) back to sigma;
    #using p(mu,log(sigma)|y), the prior on p(log(sigma)) propto 1
    logpost = -n*np.log(b) - ( ((n-1)*s + n*(v-a)**2) / (2*b*b) )
    return( logpost  )

unrounded_post = unrounded(a = moo[None,:], b = lsig[:,None], x=w)

#turn into unnormalized denstiy
unrounded_post = np.exp( unrounded_post )
#normalize the posterior
unrounded_post  = unrounded_post  / unrounded_post .sum( )

#PART B: posterior without rounding
def rounded(x, a, b):
    '''
    Parameters:
        a - grid space for mean parameter
        b - grid space for standard deviation parameter
        x - data vector
    Returns:
        natural log of unnormalized posterior
    '''
    b = np.exp(b) #swith log sigma back to sigma

    dummy = np.array(b) #dummy variable to test axis to take sum over
    axes = len(dummy.shape)-1 if len(dummy.shape) > 1 else 0

    e = 1e-200 #python not good with really small values, so add this correction
    #upper part; lower part
    upper = norm.cdf((x + 0.5),loc=a,scale=b) #upper part of likelihood
    lower = norm.cdf((x - 0.5),loc=a,scale=b) #lower part of likelihood
    logpost = np.sum(np.log(upper-lower+e),axis=axes) #sums over all data values
    return( logpost )

'''
NOTE: w[:,None,None] creates 3 dimensions. First dimension corresponds to
the entries of w. Last two are matrices the size of our grid.
```

```python
'''

'''
NOTE: This matrix is built similarly to a plot. mu corresponds to the x axis (col)
log sigma corresponds to the y axis (rows).
'''

rounded_post = rounded(x=w, a = moo[:,None], b = lsig[:,None,None])

#turn into unnormalized denstiy
rounded_post = np.exp( rounded_post )
#normalize the posterior
rounded_post  = rounded_post  / rounded_post .sum( )


## PART C:    COMPARING MEANS, VARIANCES AND CONTOUR PLOTS

#FIRST, SAMPLE FROM DISTRIBUTIONS:

#simulated points from marginal posteriors (unrounded)
B = 10000
#sample size, sample mean, sample variance
r = len(w); mu_w = w.mean(); var_w = w.var(ddof=1)

#marginal posterior pdf's for mu and sigma.
sig_unrounded = np.sqrt( ((r-1)*var_w) / (chi2.rvs(df = r - 1, size = B)) )
mu_unrounded = norm.rvs(loc = mu_w, scale = sig_unrounded/np.sqrt(r), size = B)

'''
Note: there is no nice posterior distribution for these so we have to try
something else. unravel matrix going row to row instead of column to column.
This way we sample (mu, sigma) jointly instead of separately.

Since Python unravels the matrix row to row, but the posterior matrix is built
via logsig (rows) vs mu (col), we instead repeat the entries of mu, and repeat
logsig
'''

#np.repeat repeats the vector; np.tile repeats the entries
mu_grid =  np.tile(moo,len(lsig))
lsig_grid = np.repeat(lsig,len(moo))

samples = np.random.choice(rounded_post.size,size=B,p=rounded_post.ravel())

#need to add random jitter (see pg 76 of BDA book, 3rd Ed.)

#step sizes for our grids
d_moo = np.diff(moo)[0]/2
d_lsig =np.diff(lsig)[0]/2

mu_rounded = mu_grid[samples] -d_moo + (d_moo)*rand(B)
sig_rounded = np.exp( lsig_grid[samples] -d_lsig + d_lsig*rand(B))

#LASTLY, WE PLOT THE CONTOURS
```

```python
'''
NOTE: Contours in python go by the values in the matrix, not the quantiles.
Therefore we use a certain scheme to plot the contour lines. We take a grid for
values between the min and max of the posterior, then find the values
corresponding to the quantiles we want.
'''
#contour levels
lev = [0.0001, 0.001, 0.01,.025,0.05,0.25,0.50,0.75,0.90,0.95]
cont = np.quantile(np.linspace(rounded_post.min(),rounded_post.max(),10000),lev)

fig, (ax1,ax2) = plt.subplots(1, 2,figsize=(10,5))

#unrounded contour
ax1.contour(moo,lsig,unrounded_post,levels=cont,colors='red')
ax1.scatter(mu_unrounded, np.log(sig_unrounded), zorder = 1)
ax1.set_ylabel("log(" + r'$\sigma$' +")", fontsize = 20)
ax1.set_ylim(-3,3)
ax1.set_xlabel( r'$\mu$', fontsize = 20)
ax1.set_xlim(3,18)
ax1.set_title('Unrounded Posterior Distribution', fontsize = 18)
#plt.show()

#rounded contour
ax2.contour(moo, lsig, rounded_post,levels=cont,colors='red')
ax2.scatter(mu_rounded, np.log(sig_rounded), zorder = 1)
ax2.set_ylabel("log(" + r'$\sigma$' +")", fontsize = 20)
ax2.set_ylim(-3,3)
ax2.set_xlabel( r'$\mu$', fontsize = 20)
ax2.set_xlim(3,18)
ax2.set_title('Rounded Posterior Distribution', fontsize = 18)
#plt.show()
fig.tight_layout()

### All of this sets up the dataframe so we can look at summary statistics
df_stats  = {
        'Unrounded mu': mu_unrounded,
        'Unrounded sigma': sig_unrounded,
        'Rounded mu':mu_rounded,
        'Rounded sigma': sig_rounded
                  }
df_stats = pd.DataFrame(df_stats)

df_stats  = {
        'Mean': df_stats.mean(axis=0),
        'Variance': df_stats.var(axis=0),
        '2.5%': df_stats.quantile(0.025,axis=0),
        '50%': df_stats.quantile(0.50,axis=0),
        '97.5%': df_stats.quantile(0.975,axis=0)
                  }
df_stats = np.round(pd.DataFrame(df_stats),4)

#so we can see all the data
pd.set_option("display.max_columns", 6)
```

```
print(df_stats)

#part d

'''
NOTE: The Inverse cdf method for the normal distribution works as follows:
1). Let F be the normal cdf. F:[a,b] -> [F(a),F(b)], so F^-1:[F(a),F(b)] -> [a,b].
2). Note [F(a),F(b)]= F(a) + (F(b)-F(a))*[0,1] so F^-1(F(a) + (F(b)-F(a))*[0,1])
    maps those values to [a,b].
'''
#This calculates the cdfs.
up = norm.cdf(w[None,:] + 0.5, loc = mu_rounded[:,None], scale = sig_rounded[:,None])
down = norm.cdf(w[None,:] - 0.5, loc = mu_rounded[:,None], scale = sig_rounded[:,None])

#this is equivalent to part 2 in the above notes
invcdf_samps = down + (up-down)*rand( B*len(w) ).reshape(B,len(w))
Z = norm.ppf(invcdf_samps,loc = mu_rounded[:,None], scale = sig_rounded[:,None])

print(f"Our Posterior mean is {np.round(((Z[:,1]-Z[:,0])**2).mean(),3)}")
```

Problem 4

```
### Problem 4 (BDA 3rd Ed., Exercise 3.8)

#Data for this problem
#y-bikes for streets w/ bike lanes;v- vehicles for streets w/ bike lanes
y = np.array([16, 9, 10, 13,19, 20, 18, 17,35, 55])
v = np.array([58,90, 48, 57, 103, 57, 86,112, 273, 64])
n_y = v + y

#z-bikes for streets w/o bike lanes;v- vehicles for streets w/o bike lanes
z = np.array([12, 1, 2, 4, 9, 7, 9, 8])
w = np.array([113, 18, 14, 44,208, 67, 29, 154])
n_z = w+z


alphas = np.linspace(start=0.001,stop=100,num=1000)
betas = np.linspace(start=0.001,stop=100,num=1000)

#posteriors

post_y = np.prod(beta.pdf(x = (y/n_y),
                          a = alphas[:,None], b = betas[:,None,None]),axis=2)
post_y = post_y / post_y.sum()
post_z = np.prod(beta.pdf(x = (z/n_z),
                          a = alphas[:,None], b = betas[:,None,None]),axis=2)
post_z = post_z / post_z.sum()

#Posterior draws

samples_y = np.random.choice(post_y.size,size=1000,p=post_y.ravel())
samples_z = np.random.choice(post_z.size,size=1000,p=post_z.ravel())

#should add random jitter,but I don't want to
```

```python
alpha_y_post = np.tile(alphas,len(betas))[samples_y]
beta_y_post = np.repeat(betas,len(alphas))[samples_y]

alpha_z_post = np.tile(alphas,len(betas))[samples_z]
beta_z_post = np.repeat(betas,len(alphas))[samples_z]

#Posterior predictive draws of proportional difference

prop_diff1 = ( beta.rvs(size=1000,a=alpha_y_post,b = beta_y_post) -
              beta.rvs(size=1000,a=alpha_z_post,b = beta_z_post))

print(f"The Difference in proportions is {np.round(prop_diff1.mean(),3)}")

#Method 2: FOCUS ON NUMBER OF BIKES INSTEAD OF PROPORTIONS

#parameters for the beta priors
a2 = 5; b2 = 5
#Posterior distribution for y
theta_y = beta.rvs(a=a2+y.sum(), b=b2+n_y.sum()-y.sum(), size=1000)
#Posterior distribution for z
theta_z = beta.rvs(a=a2+z.sum(), b=b2+n_z.sum()-z.sum(), size=1000)

#Posterior Predictive samples (y tilde | y) for our two models

#posterior predictive draws. Sum up all traffic
y_bikes = binom.rvs(n = n_y[None,:], p = theta_y[:,None])
z_bikes = binom.rvs(n = n_z[None,:], p = theta_z[:,None])

#proportions of vehicles
y_prop = y_bikes / n_y[np.newaxis,:]
z_prop = z_bikes / n_z[np.newaxis,:]

'''
Note: Take the row means. This is equivalent to taking the mean of each column
and provides us with the mean proportion of bikes on residential streets w/ or
w/o bike lanes.
'''
prop_diff2 = y_prop.mean(axis=1) - z_prop.mean(axis=1)

print(f"The Difference in proportions is {np.round(prop_diff2.mean(),3)}")

#METHOD 3: Separate Bikes and Vehicles

#Parameters for our gamma priors (y- bicycles, vehichles. z - bicyles, vehicles)
a_b = 15; a_v = 85;b = 1

#Posterior distributions for y, v based off choice of prior
theta_by = gamma.rvs(a=a_b+y.sum(), scale=1/(b+len(y)), size=1000)
theta_vy = gamma.rvs(a=a_v+v.sum(), scale=1/(b+len(v)), size=1000)

#Posterior distributions for z,w based off choice of prior
theta_bz = gamma.rvs(a=a_b+z.sum(), scale=1/(b+len(z)), size=1000)
theta_vz = gamma.rvs(a=a_v+w.sum(), scale=1/(b+len(w)), size=1000)
```

```
#sum of the rates
yrate = poisson.rvs(size=1000,mu = theta_by)+poisson.rvs(size=1000,mu = theta_vy)
zrate = poisson.rvs(size=1000,mu = theta_bz)+poisson.rvs(size=1000,mu = theta_vz)

#this is the proportion of bikes that we see
y_bikes = poisson.rvs(size=1000,mu = theta_by) / yrate
z_bikes = poisson.rvs(size=1000,mu = theta_bz) / zrate

#Taking difference of proportions between street w/ bike lane vs without
prop_diff3 = y_bikes - z_bikes

print(f"The Difference in proportions is {np.round(prop_diff3.mean(),3)}")

#Part d: Histograms

gs = gridspec.GridSpec(9, 9)
gs.update(wspace=0.5)

fig = plt.subplots(figsize=(10,5))
ax3 = plt.subplot(gs[:4, :4])
ax3.hist(x = prop_diff1, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
ax3.axvline(prop_diff1.mean(), color='k', linestyle='dashed', linewidth=2)
ax3.set_xlabel(r'$\mu_{y}$' + ' - ' +  r'$\mu_{z}$')
ax3.set_ylabel('Frequency')
ax3.set_title('Method 1', fontsize = 18)

ax4 = plt.subplot(gs[:4, 4:8])
ax4.hist(x = prop_diff2, bins='auto', color='red', alpha=0.7, rwidth=0.85)
ax4.axvline(prop_diff2.mean(), color='k', linestyle='dashed', linewidth=2)
ax4.set_xlabel(r'$\mu_{y}$' + ' - ' +  r'$\mu_{z}$')
ax4.set_title('Method 2', fontsize = 18)

ax5 = plt.subplot(gs[5:, 2:6])
ax5.hist(x = prop_diff3, bins='auto', color='green', alpha=0.7, rwidth=0.85)
ax5.axvline(prop_diff3.mean(), color='k', linestyle='dashed', linewidth=2)
ax5.set_xlabel(r'$\mu_{y}$' + ' - ' +  r'$\mu_{z}$')
ax5.set_ylabel('Frequency')
ax5.set_title('Method 3', fontsize = 18)

plt.suptitle('Rate Difference Between Residential Streets w/ and w/o Bike Lanes'
             , y = 1.05,fontsize=20)
```

Problem 5

```
#Problem 5 (BDA 3rd Ed., Exercise 3.12)
#number of fatal accidents between 1976-1985

### STORES OUR INFORMATION
df  = {
       'Accidents':np.array([24, 25, 31, 31, 22, 21, 26, 20, 16, 22]),
       'Deaths': np.array([734, 516, 754, 877, 814, 362, 764, 809, 223, 1066]),
       'Year': np.arange(1,11),
       'Death Rate':  np.array([0.19, 0.12, 0.15, 0.16,
                                0.14, 0.06, 0.13, 0.13, 0.03, 0.15])
```

```
                    }
df = pd.DataFrame(df)

#PART B: INFORMATIVE PRIOR
# number of draws
W = 1000
#grid for alpha and beta
alpha = np.linspace(start = 10, stop = 70, num = W)
betas =  np.linspace(start= -5, stop = 5, num = W)

#a = alpha, b = beta
def priors(a,b):
    prior = gamma.pdf(x=a,a=50,scale=1)*norm.pdf(x=b,loc = 0,scale = np.sqrt(0.5))
    return(prior)

prior = priors(a = alpha[None,:],b = betas[:,None])
plt.contour(alpha, betas, prior, colors='red')
plt.ylabel(r'$\beta$')
plt.ylim (-2,2)
plt.xlabel(r'$\alpha$')
plt.xlim(30,70)
plt.title('Contour of Informative Prior')
plt.show()

#e, Linear Regression

'''
NOTE: This regression gives you the mode of the posterior under the uniform prior
'''


fit = smf.glm(formula='Accidents ~ Year', data=df,
              family=sm.families.Poisson(link = sm.families.links.identity()) ).fit()
print(fit.params)
print(fit.cov_params())

fit = smf.ols(formula='Accidents ~ Year', data=df).fit()
print(fit.params)
print(fit.cov_params())

#a = alpha, b = beta, t = time vector, y= #number of fatal accidents
def flight_post(a,b,t,y):
    '''
    Parameters:
        a - grid space for alpha
        b - grid space for beta
        t - time data
        y - number of fatal accidents
    Returns:
        natural log of unnormalized psoterior density
    '''
    #Uniform Prior 1_{alpha+beta*t>0}
    flat = a+b*t; flat[flat<=0] = 1e-200
    dummy = np.array(b) #dummy variable to test axis to take sum over
    axes = len(dummy.shape)-1 if len(dummy.shape) > 1 else 0

    #log posterior
    log_post = np.sum(y*np.log(flat) - (flat),axis=axes)

    return log_post

flights = flight_post(a=alpha[:,None],b=betas[:,None,None],
           t=np.asarray(df['Year']),y=np.asarray(df['Accidents']))

#turn into unnormalized denstiy
flights = np.exp( flights )
#normalize the posterior
flights  = flights  / flights.sum( )

alpha_post = np.repeat(alpha,len(betas))
beta_post = np.tile(betas,len(alpha))

Fl = 1000 #Number of samples to draw
samples = np.random.choice(flights.size, size=Fl, p = flights.ravel(order="F"))

#add some random jitter so the variables are continous random variables
d_alpha = np.diff(alpha)[0]/2
d_beta = np.diff(betas)[0]/2

alpha_post = alpha_post[samples] -d_alpha + (d_alpha)*rand(Fl)
beta_post = beta_post[samples] -d_beta + (d_beta)*rand(Fl)

#contour levels for flight posterior
lev = [0.0001, 0.001, 0.01,.025,0.05,0.25,0.50,0.75,0.90,0.95]
flight_cont = np.quantile(np.linspace(flights.min(),flights.max(),10000),lev)
# This is another Posterior plot, but this has the simulated points built on top of it
plt.contour(alpha, betas, flights,levels=flight_cont,colors='red')
plt.scatter(alpha_post,beta_post,zorder=1)
plt.ylim (-4,3)
plt.ylabel(r'$\beta$')
plt.xlim(15,50)
plt.xlabel(r'$\alpha$')
plt.title('Posterior Distribution with Simulated Points')
plt.show()

#Now we plot our histogram
plt.hist(x = alpha_post + beta_post*11, bins='auto', color='green',rwidth=0.85)
plt.xlabel(r'$\alpha$'+'+' +'1986'+ r'$\beta$')
plt.ylabel('Frequency')
plt.title('Frequency of Expected Fatal Accidents 1986')
plt.show()

#95% confidence interval
pos = poisson.rvs(alpha_post + beta_post*11, size = Fl)
print("Our 95% predictive interval for the expected number "
      f"of accidental crashes in 1986 is {np.quantile(pos, [0.025, 0.975])}")
```