Economic Valuation of Environmental Change

Module 5.2: Choice Experiments: Simulation

Book chapters: Koop Ch. 5 (for MH), KPT Ch. 11 (for MH)

In [1]:
```
# Import packages we'll need for this module
################################################
from numpy import *  # numpy is used a lot in Python, and some load it "as np" - but then
# every time we use a numpy command, so I prefer not to use a prefix in this case
from numpy.linalg import inv  #not really necessary since we already imported the entire n
# but makes life easier taking inverses, else we would have to type "linalg.inv" all the t
from numpy.linalg import det #same for determinant
import matplotlib.pyplot as plt
from scipy.stats import invgamma #for draws from inverse gamma
from scipy.stats import norm #for evaluating normal priors for betas
from scipy.optimize import minimize #needed for MLE routine within the GS
from sklearn.neighbors import KernelDensity as KD #for smoth plotting of the (empirical) d
import pandas as pd #for creating data frames and output tables
import math #for pi
from scipy.special import gammaln #for evaluating the multivariate t-density - same s Mat
```

Simulate data

In [2]:
```
# Set random seed so you always get identical results when re-running an (unchanged) scrip
################################################################################
random.seed(37)

N = 1000 #number of individuals
J=3

#X1 = concatenate((random.normal(1.2,1,size=(N,1)),random.normal(0.6,1,size=(N,1))),axis=1
#X2 = concatenate((random.normal(1,1,size=(N,1)),random.normal(1.1,1,size=(N,1))),axis=1)
#X3 = concatenate((random.normal(1.1,1,size=(N,1)),random.normal(2.1,1,size=(N,1))),axis=1

X1 = concatenate((random.normal(0.1,1,size=(N,1)),random.normal(0.1,1,size=(N,1))),axis=1)
X2 = concatenate((random.normal(1.1,1,size=(N,1)),random.normal(1.1,1,size=(N,1))),axis=1)
X3 = concatenate((random.normal(2.1,1,size=(N,1)),random.normal(2.1,1,size=(N,1))),axis=1)

k = shape(X1)[1] #get column dimension

btrue=array([1,-1]) #same for all equations, as required for an unlabeled experiment
btrue.shape=(k,1)

# draw errors from the EV-type I distribution (aka "Gumbel")
eps=random.gumbel(0,1,size=(N,J)) #one column per choice option, #N by J
#note: based on the numpy documentation, python uses the "max" version of the Gumbel, as i

#quick check: column means should be colose to 0.57722, var(eps)*6/pi^2 should be close to
# mtest = eps.mean(0) #mean over rows  #looks good
# Vtest =(eps.var(0))*(6/math.pi**2)   #looks good

e1=eps[:,0:1]
e2=eps[:,1:2]
e3=eps[:,2:3]

U1=X1 @ btrue + e1  #indirect utilities for each option
U2=X2 @ btrue + e2
U3=X3 @ btrue + e3
```

```
#stack each person's J utilities as one long column

intU=concatenate((U1.T,U2.T,U3.T),axis=0) #J by N, correct - I checked against U1,U2,U3 us
Ustack=intU.T.reshape(N*J,1) #need to transpose int first to get desired result - stacking
#"reshape" always collect row by row by default, looks good

#now put each person's triplet of utilities in a separate cell, so each cel has a 3 by 1 a

# break ystar cell into lb sub-vectors of length gs by 1
Ucell=Ustack.reshape(N,J,1)
# first dimension: number of "cells"
# second dimension: number of rows in each cell
# third dimension: number of columns in each cell

y=zeros([N,1]) #will collect winning option for each person, "1," "2," or "3"

for i in range(0,N):
    int1=Ucell[i]
    #find location of maximum
    jm=argmax(int1)
    y[i]=jm+1 #(so we get "1" for the first option and not zero, just a personal preferenc

#check a few cells for accuracy
#print(Ucell[1],y[1])
#print(Ucell[10],y[10])
#print(Ucell[100],y[100])

#capture number of 1's,2's, and 3's in y to make sure we have a balanced outcome
#where each option is chosen a reasonable number of times - play with means of X to achiev

l1=sum(where(y==1,y,0))      #where.. means: where y==1, keep y, set all others to 0
l2=sum(where(y==2,y,0))/2    #since we're keeping 2's
l3=sum(where(y==3,y,0))/3

print(l1,l2,l3)
#319.0 334.0 347.0 - good enough

save("output/ClogitData", array([y,Ucell,X1,X2,X3,btrue], dtype=object), allow_pickle = Tr
# this gets rid of the "depreciated" warning message...
# to load, use: [y,bidvec,X,btrue,sig2true] = load("output\probitBidsData.npy", allow_pick
```

```
319.0 334.0 347.0
```

Preliminaries, priors, and tuners

In [3]:

```
########################################
# Preliminaries, priors, tuners
########################################

# For the GS it will be convenient to create a J x 1 yi-vector (that flags which option wa
# each person, then stack them

#so we want the first row of X1, the first row of X2, and the first row of X3 stacked into
# then repeat for each person

Xbig=zeros([N*J*k,1]).reshape(N,J,k) #start with all zeros, in the correct shape
ybig=zeros([N*J,1]).reshape(N,J,1) #same for ybig

for i in range(0,N):
    Xbig[i,0:1,:]=X1[i,:]
    Xbig[i,1:2,:]=X2[i,:]
    Xbig[i,2:3,:]=X3[i,:]

    ir=y[i] #capture 1,2, or 3 index, return simple scalar
```

```
    #tried to simply index ybig by ir, but no success, thus the lengthy if-sequence instea
    if ir==1:
        ybig[i,0]=1
    elif ir==2:
        ybig[i,1]=1
    else:
        ybig[i,2]=1

#check a few cells - OK
#Xbig[0]  #and compare to X1,X2,X3 in variable inspector
#ybig[0]  #and compare to y

#now reduce to a 2-dimensional array
Xbig.shape=(N*J,k)
ybig.shape=(N*J,1)
#looks OK


#TUNERS
##############
r1 = 5000  #burn-ins, be generous for limited dep. variable problems
r2 = 10000 #keepers
R = r1 + r2
#
#PRIORS:
###################
#for beta:
mu0 = zeros((k,1))

V0 = 100*identity(k)

tau=1 #tuner for variance in t-distribution
v=10  #degrees of freedom for t-distribution
betadraw=2*ones((k,1)) #somehting not too extreme, and not too close to the truth
```

Run Gibbs Sampler

In [4]:

```
# run GS
#############################################################################
%run functions/gs_clogit.ipynb  #actual GS function
#
# now execute the function
[betamat,accept]=gs_clogit(Xbig,ybig,k,J,N,r1,r2,mu0,V0,tau,v,betadraw)
```

```
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
```

In [5]:

```
# import the "kdiagnostics" function from your "functions" folder
%run functions/kdiagnostics.ipynb
#
```

```
# now execute the function
diagnostics=kdiagnostics(betamat)
```

In [6]:

```
# convert diagnostics matrix to data frame for plotting
#############################################################
myframe = pd.DataFrame(diagnostics)
myframe.index = pd.Index(["b1", "b2"])
myframe.columns = ["post.mean", "post.std", "p(>0)", "nse", "IEF", "M*", "CD"]

#myframe = frame.style.format("{:,.3f}") #this sets all entries to 3 decimals

# this is more slective:
myframeNice = myframe.style.format({"post.mean": "{:,.3f}",
                                    "post.std": "{:,.3f}",
                                    "p(>0)": "{:,.3f}",
                                    "nse": "{:,.3f}",
                                    "IEF": "{:,.3f}",
                                    "CD": "{:,.3f}",
                                    "M*": "{:,.0f}"})
display(myframeNice)
#print(frame) #produces a raw-looking table, this is nicer
```

	post.mean	post.std	p(>0)	nse	IEF	M*	CD
b1	0.998	0.054	1.000	0.001	1.161	8,613	0.633
b2	-0.938	0.052	0.000	0.001	1.167	8,571	-0.679

The acceptance rate is:

In [9]:

```
print(round(accept,2))
```

```
0.93
```

In [11]:

```
save("output/simResults", array([betamat,accept], dtype=object), allow_pickle = True)
# this gets rid of the "depreciated" warning message...
# to load, use: [betamat,accept] = load("output\simResults.npy", allow_pickle = True)
```