Economic Valuation of Environmental Change

Module 5.4: Choice Experiments: Application 1:

Red Tide air quality forecast in SW Florida

Book chapters: PR Ch. 19, CBB CH. 5

LaTex commands

Background

Blooms of *Karenia brevis*, commonly referred to as "Red Tide" (RT), in southwest Florida are known for causing respiratory irritation and illness in humans via aerosolized toxins (when cells are broken up by wind and waves), among other environmental impacts.

These blooms have also become more frequent, severe, and longer-lasting in recent years, affecting all sorts of outdoor activities and thus the daily life of locals, as well as many aspects of Florida's tourism industry. Local governments have spent considerable amounts on mitigating or possibly preventing blooms using scientific and engineering tools, but somewhat limited attention has been given to the design of improved air quality forecasts.

We hypothesize that more temporally and spatially more refined forecasts would help the local population to adapt to red tide conditions by timing and siting outdoor activities to avoid exposure to high toxin concentrations during a bloom.

A group of VT researchers thus teamed up with Mote Marine Laboratories (MML) in Sarasota, FL - the premier scientific clearinghouse for RT research - to implement a pilot project geared towards the development of an improved forecasting system.

At the heart of these activities was a survey based choice experiment (CE) to understand the individual (and combined) value of different forecasting attributes to the underlying population.

At a broader level, we argued that a better understanding of the societal values of an improved forecast would give policymakers guidance as to the optimal levels of investment to develop such a system. In other words, we wanted to find out if societal benefits would outweigh (expected / estimated) costs of implementation.

Background materials

The paper coming out of this research has been conditionally accepted at *Marine Resource Economics*. It includes all relevant details on focus groups, survey design, experimental design, and econometric modeling.

Here is are the links to the paper and the survey instrument:

- RT paper
- RT survey

As shown in the paper, close to 90% of the target population (five SW-FL gulf coast counties) engaged in some form of outdoor activities in the 12 months preceding the survey.

The average household spends approximately 16 hours / week on outdoor activities, and an additional 8-9 hours in outside areas of their house or property.

Thus, it is clear that the typical 5-county household is at a **high risk of exposure** to RT toxins if it wants to follow its typical outdoor lifestyle. This also suggests that a better forecast could indeed be helpful and relevant for the majority of stakeholders.

The survey also confirmed that past RT blooms have hampered these activities to some extent. At the extreme end of the **impact range**, people have sold their coastal home and moved inland, sold their boat or water gear, and even gave up coastal jobs or volunteer work to avoid exposure to RT toxins.

In sum, RT-impacted air quality is indeed a **recurring and pervasive problem** in that area, and our project is thus well-targeted.

Estimation

In [20]:

```
# 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 
# but makes life easier taking inverses, else we would have to type "linalg.inv" all the 
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) 
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
```

In [21]:

```
#read in csv data
#######################################
dataf=pd.read_csv("data\RTdata.csv")
# this comes in as a dataframe, thus the "f" suffix

# Contents of data
#################################################################

# 1  id                 running respondent id (12 rows / person)
# 2  set                choice set  (1 through 4)
# 3  idset              id x set (running id for triplets of rows)
# 4  option             choice option (= "alternative," or "profile") (1 through 3)
# 5  origBlock          original choice block (= survey version x rotation)
# 6  block              survey version (1-5)
# 7  rotation           choice set rotation within block (1-4)
# 8  income             approx. HH income, dollars
# 9  cov                forecast coverage (6 or 12 miles)
# 10 acc1               forecast accuracy, first 12 hours (50,75,100)
# 11 acc2               forecast accuracy, second 12 hours (50,75,100)
# 12 bid                price / bid ($; 0 (SQ), 5, 15, 25, 35)
# 13 vote               indicator for chosen alternative (1-3)
# 14 choice             vote translated to binary (1=chosen)
# 15 allNO              chose SQ for all 4 questions
```

```
# 16 protNO            1= protest NO response
# 17 badFollow         1= ANY problematic follow-up response
# 18 genBad            1 = combo of protNO and badFollow
# 19 badFollow2        1 = disagr. on confident, actual/same vote or ownmind
# 20 genBad2           1 = combo of protNO and badFOllow2 - USE THIS!!!
# 21 sq                SQ indicator (3rd option)
# 22 covacc1           linear interaction cov / acc1
# 23 covacc2           linear interaction cov / acc2
# 24 acc1acc2          linear interaction acc1 / acc2
# 25 cov12             basic dummy for coverage=12
# 26 acc175            basic dummy for acc1=75
# 27 acc1100           basic dummy for acc1=100
# 28 acc275            basic dummy for acc2=75
# 29 acc2100           basic dummy for acc2=100
# 30 cov12acc175       binary interaction cov12 dummy with acc175 dummy
# 31 cov12acc1100
# 32 cov12acc275
# 33 cov12acc2100
# 34 cov12ec          effect code variable for coverage=12
# 35 acc175ec
# 36 acc1100ec
# 37 acc275ec
# 38 acc2100ec
# 39 cov12acc175ec
# 40 cov12acc1100ec
# 41 cov12acc275ec
# 42 cov12acc2100ec
# 43 existused
# 44 hrsout          total hours per week spent outside by all HH members
# 45 hrsyardc        total hours per week spent outside around house by HH

# note the data are already in long format, which each row corresponding to a single choic
# rows corresponding to a single choice set - third row is always the SQ (here just zeros
# Thus, the data already has the form of our "ybig" and "Xbig" from the simulated model.

display(dataf)
```

	id	set	idset	option	origBlock	block	rotation	income	cov	acc1	...	acc1100ec	acc275ec	acc2100ec
0	1	1	1	1	2.4	2	4	162500	6	100	...	1	1	0
1	1	1	1	2	2.4	2	4	162500	12	50	...	-1	-1	-1
2	1	1	1	3	2.4	2	4	162500	0	0	...	0	0	0
3	1	2	2	1	2.4	2	4	162500	12	100	...	1	0	1
4	1	2	2	2	2.4	2	4	162500	6	50	...	-1	-1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6019	502	3	2007	2	2.3	2	3	87500	12	50	...	-1	-1	-1
6020	502	3	2007	3	2.3	2	3	87500	0	0	...	0	0	0
6021	502	4	2008	1	2.3	2	3	87500	6	100	...	1	1	0
6022	502	4	2008	2	2.3	2	3	87500	12	50	...	-1	-1	-1
6023	502	4	2008	3	2.3	2	3	87500	0	0	...	0	0	0

6024 rows × 45 columns

In [22]:
```
######################################################################
# It is convenient to "clean" the data while we're still in a dataframe
######################################################################
```

```
###############################################################
# eliminate protest responses
###############################################################
df1 = dataf[dataf['genBad2'] == 0]


# We ned to convert the dataframe to an array for further processing
#####################################################################
data = df1.to_numpy()

N=1472 #number of (presumed) independent choice occasions (368 individuals @ 4 occasions)
J=3 #number of choice options, including SQ

ybig=data[:,13:14] #14th column, 0/1 indictor for each choice option
Xbig = concatenate((data[:,20:21],data[:,24:29],data[:,11:12]),axis=1)
k=shape(Xbig)[1] #get column dimension

Xbig.shape=(N*J,k)
ybig.shape=(N*J,1)

# Contents of Xbig
#############################
# 1 sq                  SQ dummy (flags SQ option)
# 2 cov12               basic dummy for coverage=12
# 3 acc175              basic dummy for acc1=75
# 4 acc1100             basic dummy for acc1=100
# 5 acc275              basic dummy for acc2=75
# 6 acc2100             basic dummy for acc2=100
# 7 bid

# check if means are same as stata
#print(mean(ybig))
#tt=Xbig.mean(0)
#print(tt) #OK, all good
```

In [36]:

```
#TUNERS
##############
r1 = 10000  #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=30  #degrees of freedom for t-distribution
betadraw=0.1*ones((k,1)) #something not too extreme, relatively close to zero to avoid "lo
```

In [37]:

```
# run GS
##########################################################################
random.seed(37) #don't forget to set the random seed

%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
16000
17000
18000
19000
20000
```

In [38]:

```
# import the "kdiagnostics" function from your "functions" folder
%run functions/kdiagnostics.ipynb
#
# now execute the function
diagnostics=kdiagnostics(betamat)
```

In [39]:

```
# convert diagnostics matrix to data frame for plotting
##############################################################
myframe = pd.DataFrame(diagnostics)
myframe.index = pd.Index(["SQ", "band=12", "acc1=75%", "acc1=100%", "acc2=75%", "acc2=100%
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
SQ	-0.803	0.105	0.000	0.001	1.127	8,870	0.086
band=12	0.111	0.094	0.880	0.001	1.170	8,547	1.346
acc1=75%	0.264	0.130	0.980	0.001	1.115	8,967	0.647
acc1=100%	0.939	0.143	1.000	0.002	1.163	8,596	0.524
acc2=75%	0.222	0.095	0.990	0.001	1.170	8,549	0.015
acc2=100%	-0.007	0.128	0.477	0.001	1.130	8,847	0.315
price	-0.047	0.005	0.000	0.000	1.229	8,139	-1.722

The acceptance rate is:

In [40]:

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

```
0.92
```

In [41]:

```
save("output/RTResults", 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)
```

Marginal WTP

The marginal WTP, also referred to as "implicit price" for each attribute effect captured in the model is obtained by dividing the corresponding attribute coefficient by the negative value of the price coefficient.

Let's capture these marginal WTP values, along with their HPDIs and show them in a separate table.

In [55]:

```
# extract attribute effects and price from betamat
##########################################################
attmat=betamat[1:k-1,:] #rows 2 through k-1
bprice=-betamat[k-1:k,:] #last row

# replicate price row and divide
###########################################################
pricemat=tile(bprice,(k-2,1)) #replicate bprice k-2 times in the row dimension
margmat=attmat/pricemat #still 5 by 10000

# Get HPDI bounds
########################################################
%run functions/khpdi.ipynb #call function
# short loop to get bounds for all cases
katt=5 #number of attribute effects
hpdimat=zeros([katt,2]) #first column for lower bound, second for upper

for i in range(0,katt):
    int1=margmat[i:i+1,:].T #needs to be column vector
    [L,U]=khpdi(int1,0.05,1000)
    hpdimat[i,0]=L
    hpdimat[i,1]=U

postmean = mean(margmat,axis=1)
postmean.shape=(5,1)
outmat=concatenate((hpdimat[:,0:1],postmean,hpdimat[:,1:2]),axis=1)

# convert HPDI matrix to data frame for plotting
##############################################################
myframe = pd.DataFrame(outmat)

myframe.index = pd.Index(["band=12", "acc1=75%", "acc1=100%", "acc2=75%", "acc2=100%"])
myframe.columns = ["lower bound", "post. mean", "upper bound"]

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

# this is more slective:
myframeNice = myframe.style.format({"lower bound": "{:,.3f}",
                                    "post. mean": "{:,.3f}",
                                    "upper bound": "{:,.3f}"})
display(myframeNice)
#print(frame) #produces a raw-looking table, this is nicer
# OK, same as Matlab's - just checking...
```

	lower bound	post. mean	upper bound
band=12	-1.448	2.281	6.057
acc1=75%	0.211	5.643	11.050

	lower bound	post. mean	upper bound
acc1=100%	14.846	20.164	24.985
acc2=75%	0.733	4.786	8.910
acc2=100%	-5.725	-0.179	5.171

Predictions

Let's derive the PPDs of total WTP for all meaningful attribute combinations (where acc2 does not exceed acc1), and display the mean along with HPDI bounds.

In [82]:

```
# generate each possible forecast scenario
###########################################

x1= array([[0, 0, 0, 0, 0, 0]]) #double-bracket forces this to be a row vector
x2= array([[0, 0, 1, 0, 0, 0]])
x3= array([[0, 0, 1, 0, 1, 0]])
x4= array([[0, 0, 0, 1, 0, 0]])
x5= array([[0, 0, 0, 1, 1, 0]])
x6= array([[0, 0, 0, 1, 0, 1]])
x7= array([[0, 1, 0, 0, 0, 0]])
x8= array([[0, 1, 1, 0, 0, 0]])
x9= array([[0, 1, 1, 0, 1, 0]])
x10=array([[0, 1, 0, 1, 0, 0]])
x11=array([[0, 1, 0, 1, 1, 0]])
x12=array([[0, 1, 0, 1, 0, 1]])

X1=concatenate((x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12),axis=0) #12 by 6
X0=concatenate((ones([shape(X1)[0],1]),zeros([shape(X1)[0],shape(X1)[1]-1])),axis=1) #jus

int1=(X1-X0) @ betamat[0:k-1,:] #12 by 10000

#replicate price coeff. draws 12 times in the row dimension
lammat=tile(bprice,(12,1))

#generate 12 PPDs for total WTP
WTPmat=int1/lammat

# Get HPDI bounds
########################################################
# short loop to get bounds for all cases
kp=12 #number of attribute effects
hpdimat=zeros([kp,2]) #first column for lower bound, second for upper

for i in range(0,kp):
    int1=WTPmat[i:i+1,:].T #needs to be column vector
    [L,U]=khpdi(int1,0.05,1000)
    hpdimat[i,0]=L
    hpdimat[i,1]=U

postmean = mean(WTPmat,axis=1)
postmean.shape=(12,1)
outmat=concatenate((hpdimat[:,0:1],postmean,hpdimat[:,1:2]),axis=1)

# convert HPDI matrix to data frame for plotting
###############################################################
myframe = pd.DataFrame(outmat)

myframe.index = pd.Index(["6,50,50", "6,75,50", "6,75,75", "6,100,50", "6,100,75", "6,100,
                          "12,50,50", "12,75,50", "12,75,75", "12,100,50", "12,100,75", "12
myframe.columns = ["lower bound", "post. mean", "upper bound"]
```

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

# this is more slective:
myframeNice = myframe.style.format({"lower bound": "{:,.3f}",
                                "post. mean": "{:,.3f}",
                                "upper bound": "{:,.3f}"})
display(myframeNice)
#print(frame) #produces a raw-looking table, this is nicer
# OK, same as Matlab's - just checking...
```

	lower bound	post. mean	upper bound
6,50,50	12.317	17.359	22.772
6,75,50	18.089	23.002	28.297
6,75,75	22.041	27.788	33.255
6,100,50	32.274	37.523	43.114
6,100,75	36.498	42.309	48.486
6,100,100	32.068	37.344	42.952
12,50,50	15.057	19.640	24.544
12,75,50	20.227	25.283	30.028
12,75,75	25.374	30.069	34.804
12,100,50	34.814	39.804	44.653
12,100,75	39.825	44.590	49.588
12,100,100	34.854	39.625	44.877

Aggregate predictions

As a final step, let's derive the aggregate WTP per year for all 835,000 households that live in the 5-county research area.

Let's do this for the least (6,50,50) desirable, and most (12,100,75) desirable forecast scenario. We will plot the corresponding PPDs along with HPDI bounds.

In [87]:

```
#predict agg WTP, in millions
aggmat=0.835*concatenate((WTPmat[0:1,:],WTPmat[10:11,:]),axis=0) #2 by 10000

# Get HPDI bounds
#######################################################
# short loop to get bounds for all cases
kagg=2 #number of attribute effects
hpdimat=zeros([kagg,2]) #first column for lower bound, second for upper

for i in range(0,kagg):
    int1=aggmat[i:i+1,:].T #needs to be column vector
    [L,U]=khpdi(int1,0.05,1000)
    hpdimat[i,0]=L
    hpdimat[i,1]=U
```

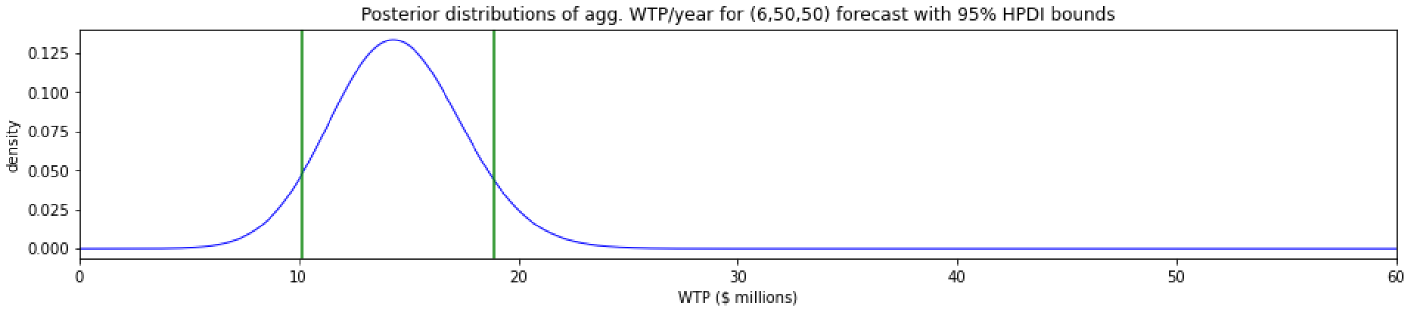
In [96]:

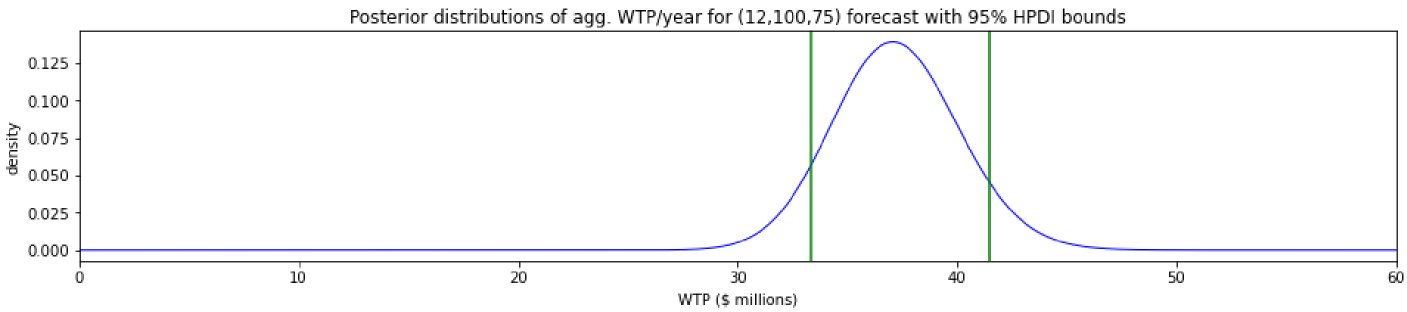
```
# get kernel density estimates for each coefficient for smooth plotting
#######################################################################
```

```
yS1=aggmat[0:1,:] #low-level forecast
yS2=aggmat[1:2,:] #high-level forecast

L1=hpdimat[0,0]
U1=hpdimat[0,1]

L2=hpdimat[1,0]
U2=hpdimat[1,1]

x01 = linspace(-20,100,r2)[:,newaxis]
kde1 = KD(kernel='gaussian', bandwidth=2).fit(yS1.T) #re-shape to column vector
logdens1 = kde1.score_samples(x01) #needs 2-D array

x02 = linspace(-20,100,r2)[:,newaxis] #np. newaxis (or short: newaxis in our case) turns i
kde2 = KD(kernel='gaussian', bandwidth=2).fit(yS2.T) #re-shape to column vector
logdens2 = kde2.score_samples(x02) #needs 2-D array
```

In [103…

```
# Initiate Figure
############################################################
fig,ax = plt.subplots(2,1,figsize=(16,8))
#
# subplot (1,1): posteriors for "low-level forecast"
#####################################################
ax[0].plot(x01,exp(logdens1),'b-', lw=1, label='PPD')
ax[0].set_xlim([0,60])
ax[0].axvline(x= L1,color='g') #add lower bound line
ax[0].axvline(x= U1,color='g') #add upper bound line
ax[0].set_xlabel('WTP ($ millions)') #the "r" is needed to render latex in graph labels an
ax[0].set_ylabel('density')
ax[0].set_title('Posterior distributions of agg. WTP/year for (6,50,50) forecast with 95%
#ax[0].legend()
#
# subplot (1,1): posteriors for "high-level forecast"
#########################################
ax[1].plot(x02,exp(logdens2),'b-', lw=1, label='PPD')
ax[1].set_xlim([0,60])
ax[1].axvline(x= L2,color='g') #add lower bound line
ax[1].axvline(x= U2,color='g') #add upper bound line
ax[1].set_xlabel('WTP ($ millions)') #the "r" is needed to render latex in graph labels an
ax[1].set_ylabel('density')
ax[1].set_title('Posterior distributions of agg. WTP/year for (12,100,75) forecast with 95
#ax[1].legend()
#
# adjust spacing between subplots
plt.subplots_adjust(wspace=0.1, hspace=0.8)
```





References:

Moeltner, K., T. Fanara, H. Foroutan, R. Hanlon, V. Lovko, S. Ross, and D. Schmale III, "Harmful algal blooms and toxic air: The economic value of improved forecasts," paper presented at the annual meetings of the European Association of Environmental and Resource Economists (EAERE), virtual, Jun. 25, 2021.