In [2]:
```
# ECON 289 Problem set 3
# Instructor: Ben Brooks
# Spring 2023

# This problem set has a series of cells with different programming tasks. You will be
# asked to run code that I have written, and also add and run your own code. Add your
# code between lines that look like this:

# ----------------------------------------

# To complete the problem set, add your own code, run all the cells, and then submit
# a copy of the notebook on canvas. The easiest way to do so is to select "print
# preview" from the file menu, and then save the new page that opens as a pdf document.

# Please work together to complete the problem set. Also, remember, Google
# is your friend. Only ask me for help after you have looked for the
# answer on stack overflow.
```

In [3]:
```
# Insert code to load gurobi, numpy, matplotlib pyplot, and mplot3d, as we did in pset 2.

# ----------------------------------------

import gurobipy as gp
from gurobipy import GRB

import matplotlib.pyplot as plt

import numpy as np

from mpl_toolkits import mplot3d
%matplotlib notebook

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

In [4]:

```
# We will solve the Myersonian auction design problem when there are
# two bidders with values that are independently and uniformly distributed
# on an evenly spaced grid in [0,1].

# Start by setting up the model. Create an index set and a set of 51 evenly
# spaced values. Add variables corresponding to the direct allocation q1[v1,v2]
# q2[v1,v2] and transfers t1[v1,v2] and t2[v1,v2]. Make sure that the transfers are
# free variables!

# ----------------------------------------
numVals=51
K=range(0,numVals)

V={k:k/(numVals-1) for k in K};

f={k:1/numVals for k in K};

model = gp.Model()

model.Params.Method = 2 # Barrier algorithm
model.Params.Crossover = 0 # Disable crossover
model.Params.OutputFlag = 1 # Enable output

q1=model.addVars(K,K)
q2=model.addVars(K,K)
t1=model.addVars(K,K,lb=-GRB.INFINITY)
t2=model.addVars(K,K,lb=-GRB.INFINITY)

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

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-03-14
Set parameter Method to value 2
Set parameter Crossover to value 0
```

In [5]:

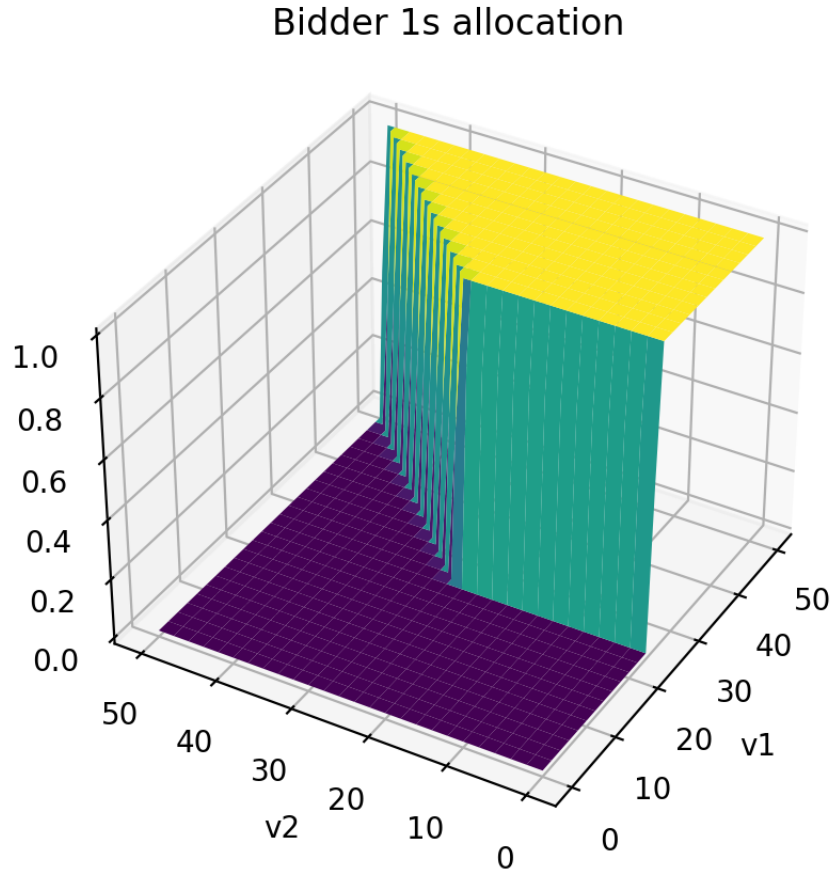
```
# Now add incentive constraints and participation constraints for each bidder.
# In particular, there should be a constraint that in expectation across v2, bidder
# 1 gets a non-negative interim payoff from reporting truthfully, and the interim payoff from
# reporting truthfully is greater than that from any misreport.

# -----------------------------------------

ir1 = model.addConstrs(sum((V[v1]*q1[v1,v2]-t1[v1,v2])*f[v2] for v2 in K)>=0 for v1 in K)
ir2 = model.addConstrs(sum((V[v2]*q2[v1,v2]-t2[v1,v2])*f[v1] for v1 in K)>=0 for v2 in K)
ic1 = model.addConstrs(sum((V[v1]*(q1[v1,v2]-q1[vhat,v2])-(t1[v1,v2]-t1[vhat,v2]))*f[v2] for v2 in K)>=0
ic2 = model.addConstrs(sum((V[v2]*(q2[v1,v2]-q2[v1,vhat])-(t2[v1,v2]-t2[v1,vhat]))*f[v1] for v1 in K)>=0

# -----------------------------------------

# Next add the feasibility constraints.

# -----------------------------------------

feas = model.addConstrs(q1[v1,v2]+q2[v1,v2]<=1 for v1 in K for v2 in K)

# -----------------------------------------

# Finally, set the objective to revenue maximization, and solve the model.

# -----------------------------------------

model.setObjective(sum((t1[v1,v2]+t2[v1,v2])*f[v1]*f[v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

# -----------------------------------------

# What is the approximate value you get for maximum revenue?
```

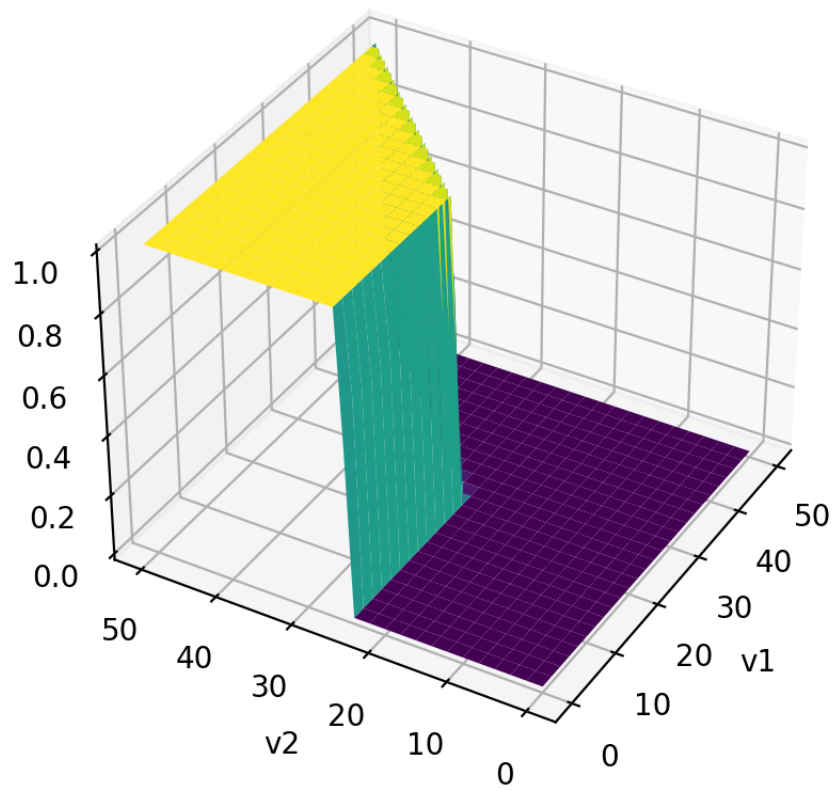
```
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 7905 rows, 10404 columns and 1045704 nonzeros
Model fingerprint: 0x8fbfe2fb
Coefficient statistics:
  Matrix range     [4e-04, 1e+00]
  Objective range  [4e-04, 4e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 205 rows and 5202 columns
Presolve time: 0.37s
Presolved: 7700 rows, 5202 columns, 525300 nonzeros
Ordering time: 0.60s

Barrier statistics:
 AA' NZ     : 4.987e+05
 Factor NZ  : 4.951e+06 (roughly 40 MB of memory)
 Factor Ops : 8.209e+09 (less than 1 second per iteration)
 Threads    : 4

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.31106282e+00  1.57763273e+01  6.92e+01 1.00e-01  3.48e+00     2s
   1  -1.37249243e-01  2.11895683e+01  2.97e+01 2.27e-02  1.28e+00     2s
   2   1.40469486e+00  1.99647425e+01  6.23e+00 1.39e-03  2.35e-01     2s
   3   2.33165772e-01  1.81976098e+01  8.76e-01 6.99e-04  5.29e-02     2s
   4   2.99455286e-01  7.22007332e+00  1.20e-01 9.27e-06  1.15e-02     2s
   5   2.83395133e-01  1.21271927e+00  7.81e-03 1.04e-09  1.25e-03     3s
   6   3.29221514e-01  5.85698150e-01  3.21e-03 2.46e-10  3.40e-04     3s
   7   3.95549523e-01  4.51909524e-01  7.03e-04 4.75e-11  7.44e-05     3s
   8   4.22612075e-01  4.26989240e-01  1.30e-05 1.34e-12  5.56e-06     3s
   9   4.24770982e-01  4.24893147e-01  1.37e-07 5.00e-16  1.54e-07     3s
  10   4.24835855e-01  4.24836815e-01  2.20e-08 6.31e-16  1.21e-09     3s
  11   4.24836599e-01  4.24836602e-01  4.00e-09 5.24e-16  3.24e-12     4s

Barrier solved model in 11 iterations and 3.68 seconds (3.54 work units)
Optimal objective 4.24836599e-01
```

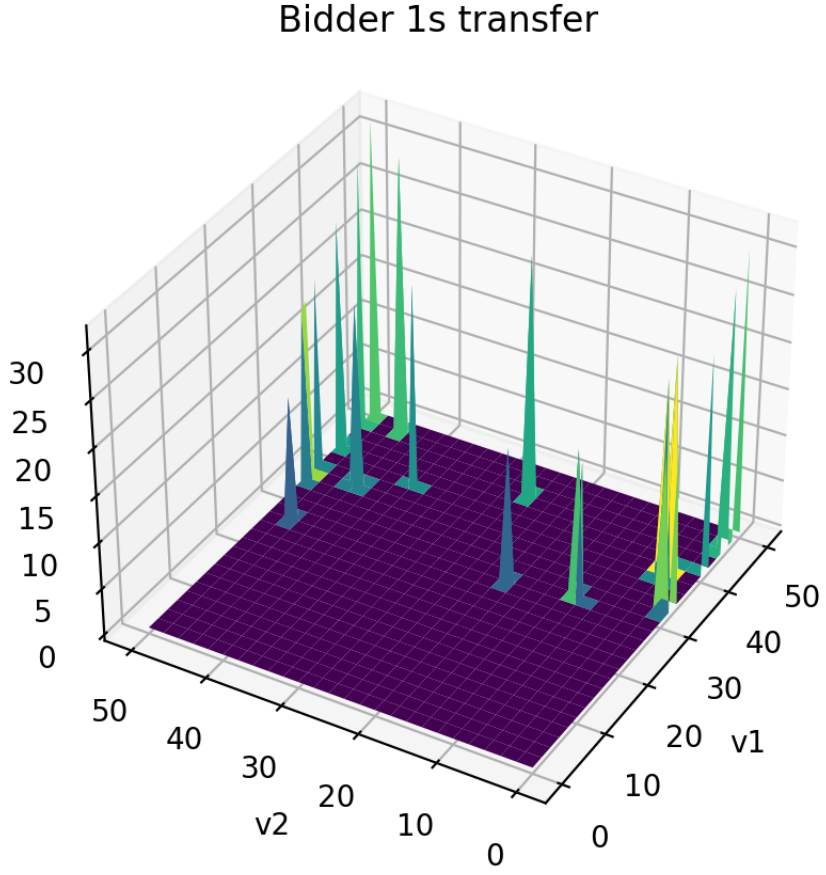
In [6]:
```
# Now let's explore the solution. Plot the optimal allocation and transfers.

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

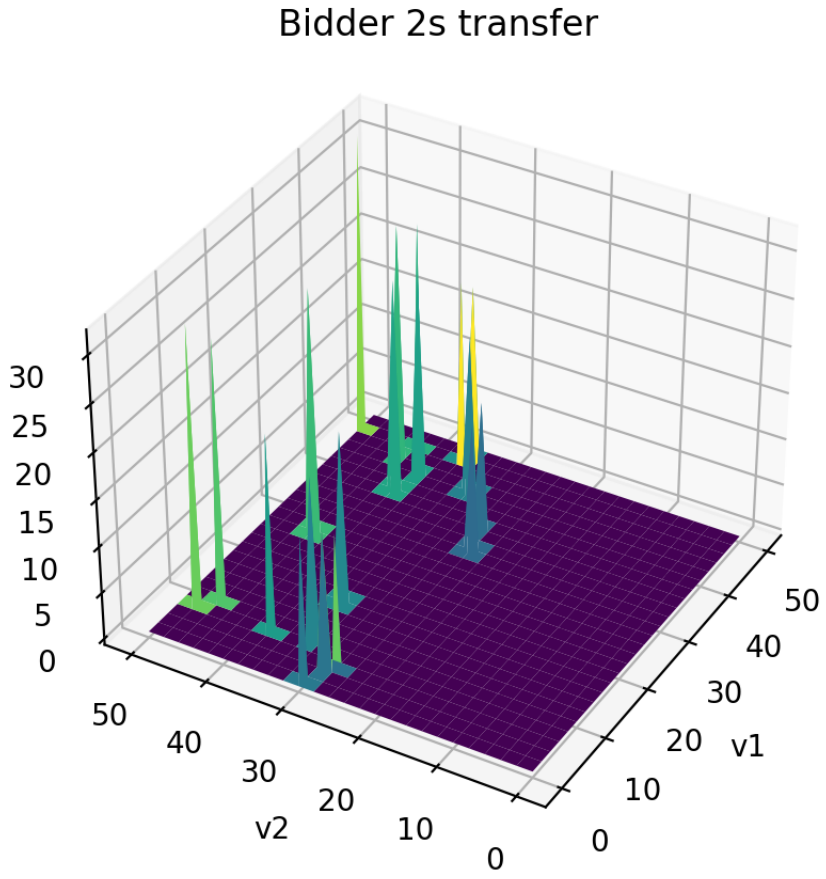
```
Q1 = np.array([[q1[v1,v2].X for v1 in K] for v2 in K])
Q2 = np.array([[q2[v1,v2].X for v1 in K] for v2 in K])
T1 = np.array([[t1[v1,v2].X for v1 in K] for v2 in K])
T2 = np.array([[t2[v1,v2].X for v1 in K] for v2 in K])

X, Y = np.meshgrid(K,K)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Q1, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 1''s allocation');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Q2, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 2''s allocation');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, T1, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 1''s transfer');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, T2, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 2''s transfer');
ax.view_init(35,210)
```

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

# What is the optimal allocation?

# How about the transfers? Do you notice a lot of structure?
```



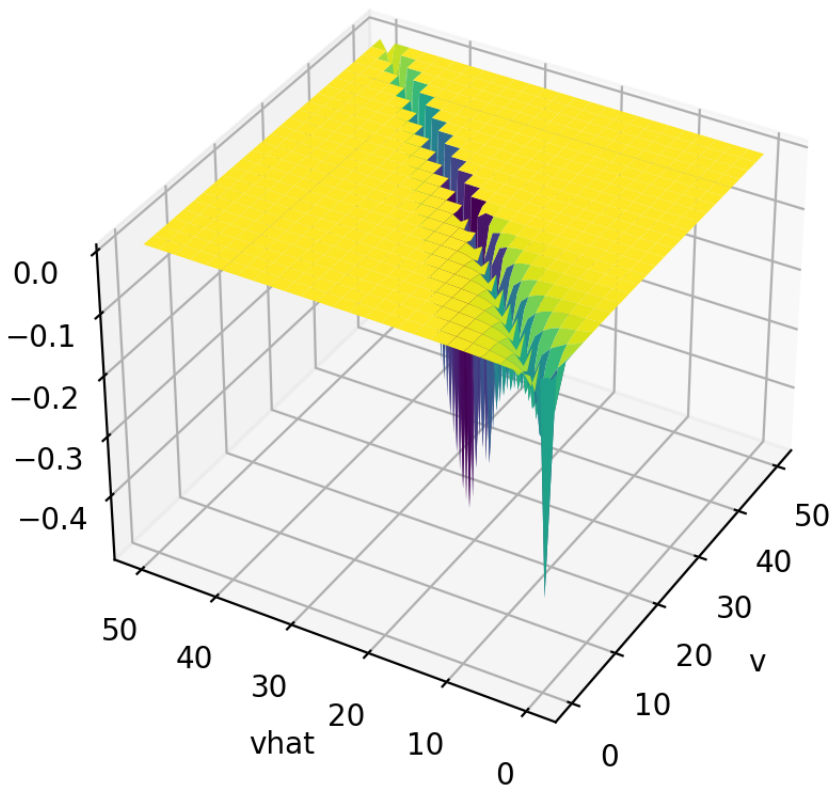

Bidder 2s allocation
1.0
0.8
0.6
0.4
0.2
0.0
50
40
30
20
10
0
v2
0
10
20
30
40
50
v1

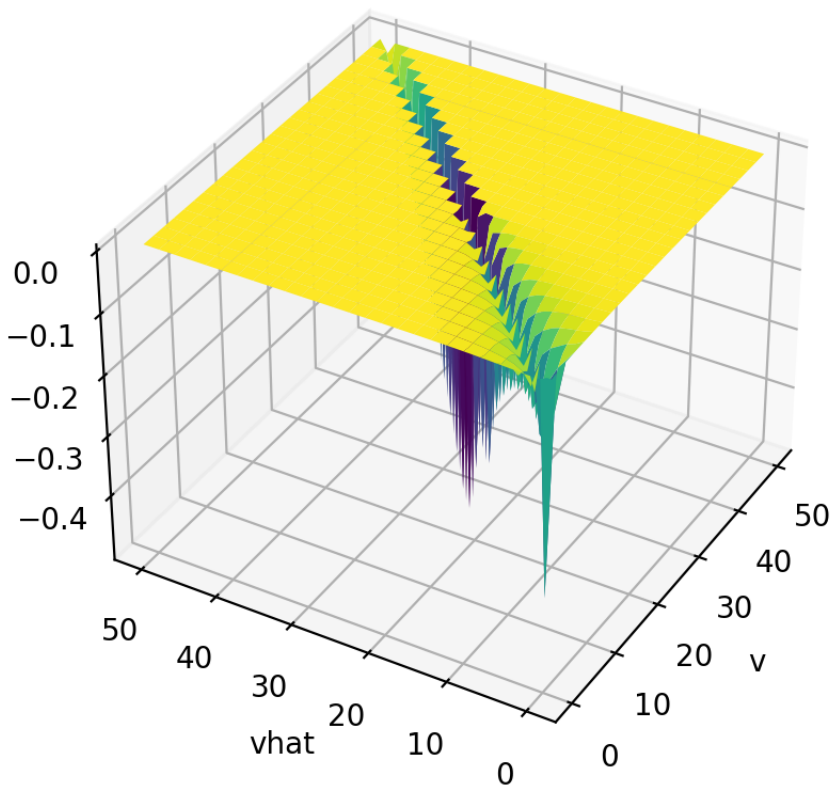

Bidder 1s transfer
30
25
20
15
10
5
0
50
40
30
20
10
0
v2
50
40
30
20
10
0
v1

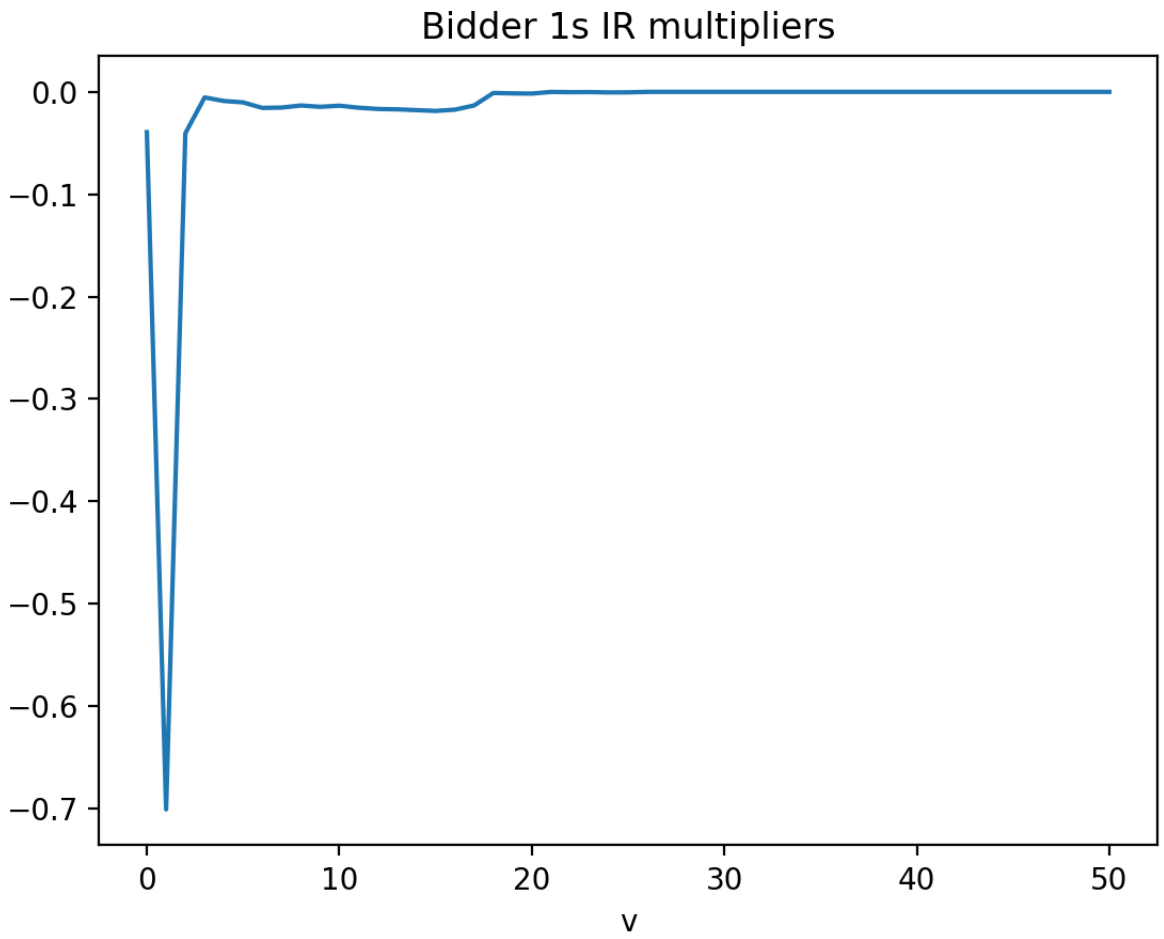


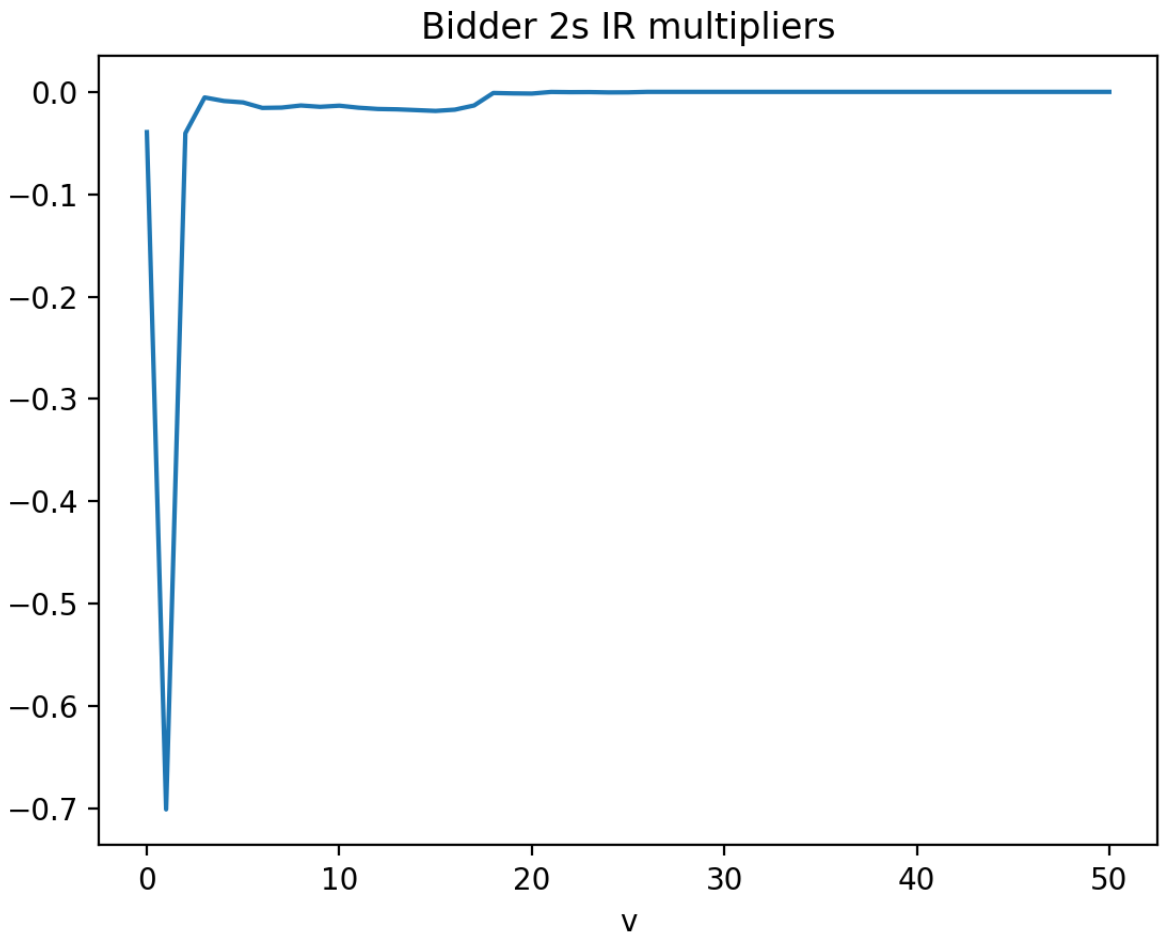
In [7]:
```
# Now plot the multipliers on the incentive and participation constraints.

# ----------------------------------------
IC1 = np.array([[ic1[v,vhat].Pi for v in K] for vhat in K])
IC2 = np.array([[ic2[v,vhat].Pi for v in K] for vhat in K])
IR1 = np.array([ir1[v].Pi for v in K])
IR2 = np.array([ir2[v].Pi for v in K])

X, Y = np.meshgrid(K,K)

fig = plt.figure()
```

```
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, IC1, cmap='viridis')

ax.set_xlabel('v')
ax.set_ylabel('vhat')
ax.set_title('Bidder 1''s IC multipliers');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, IC2, cmap='viridis')

ax.set_xlabel('v')
ax.set_ylabel('vhat')
ax.set_title('Bidder 2''s IC multipliers');
ax.view_init(35,210)

fig, ax = plt.subplots()
plt.plot(K,IR1)
ax.set_xlabel('v')
ax.set_title('Bidder 1''s IR multipliers');
plt.show()

fig, ax = plt.subplots()
plt.plot(K,IR2)
ax.set_xlabel('v')
ax.set_title('Bidder 2''s IR multipliers');
plt.show()

# ----------------------------------------

# What do you notice about which IC multipliers are non-zero?

# What about the IR multipliers?
```


Bidder 1s IC multipliers
0.0
−0.1
−0.2
−0.3
−0.4
50
40
30
20
10
0
vhat
0
10
20
30
40
50
v


Bidder 2s IC multipliers
0.0
−0.1
−0.2
−0.3
−0.4
50
40
30
20
10
0
vhat
0
10
20
30
40
50
v


Bidder 1s IR multipliers
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
10
20
30
40
50
v


Bidder 2s IR multipliers
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
10
20
30
40
50
v

In [11]:

```
# Resolve the previous model, but drop all of the incentive constraints except for those
# associated with misreporting the next lower signal, and drop all of the individual
# rationality constraints, except for the lowest valuation. What happens to the value?
# What do you conclude from this exercise?

# ----------------------------------------
numVals=51
K=range(0,numVals)

V={k:k/(numVals-1) for k in K};

f={k:1/numVals for k in K};

model = gp.Model()

model.Params.Method = 2 # Barrier algorithm
model.Params.Crossover = 0 # Disable crossover
model.Params.OutputFlag = 1 # Enable output

q1=model.addVars(K,K)
q2=model.addVars(K,K)
t1=model.addVars(K,K,lb=-GRB.INFINITY)
t2=model.addVars(K,K,lb=-GRB.INFINITY)


ir1 = model.addConstrs(sum((V[v1]*q1[v1,v2]-t1[v1,v2])*f[v2] for v2 in K)>=0 for v1 in K if v1==0)
ir2 = model.addConstrs(sum((V[v2]*q2[v1,v2]-t2[v1,v2])*f[v1] for v1 in K)>=0 for v2 in K if v2==0)
ic1 = model.addConstrs(sum((V[v1]*(q1[v1,v2]-q1[vhat,v2])-(t1[v1,v2]-t1[vhat,v2]))*f[v2] for v2 in K)>=0
ic2 = model.addConstrs(sum((V[v2]*(q2[v1,v2]-q2[v1,vhat])-(t2[v1,v2]-t2[v1,vhat]))*f[v1] for v1 in K)>=0

feas = model.addConstrs(q1[v1,v2]+q2[v1,v2]<=1 for v1 in K for v2 in K)

model.setObjective(sum((t1[v1,v2]+t2[v1,v2])*f[v1]*f[v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()
# ----------------------------------------
```

```
Set parameter Method to value 2
Set parameter Crossover to value 0
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 2703 rows, 10404 columns and 25704 nonzeros
Model fingerprint: 0x25cbb435
Coefficient statistics:
  Matrix range     [4e-04, 1e+00]
  Objective range  [4e-04, 4e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 2703 rows and 10404 columns
Presolve time: 0.02s
Presolve: All rows and columns removed

Barrier solved model in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective 4.24836601e-01
```

In [9]:

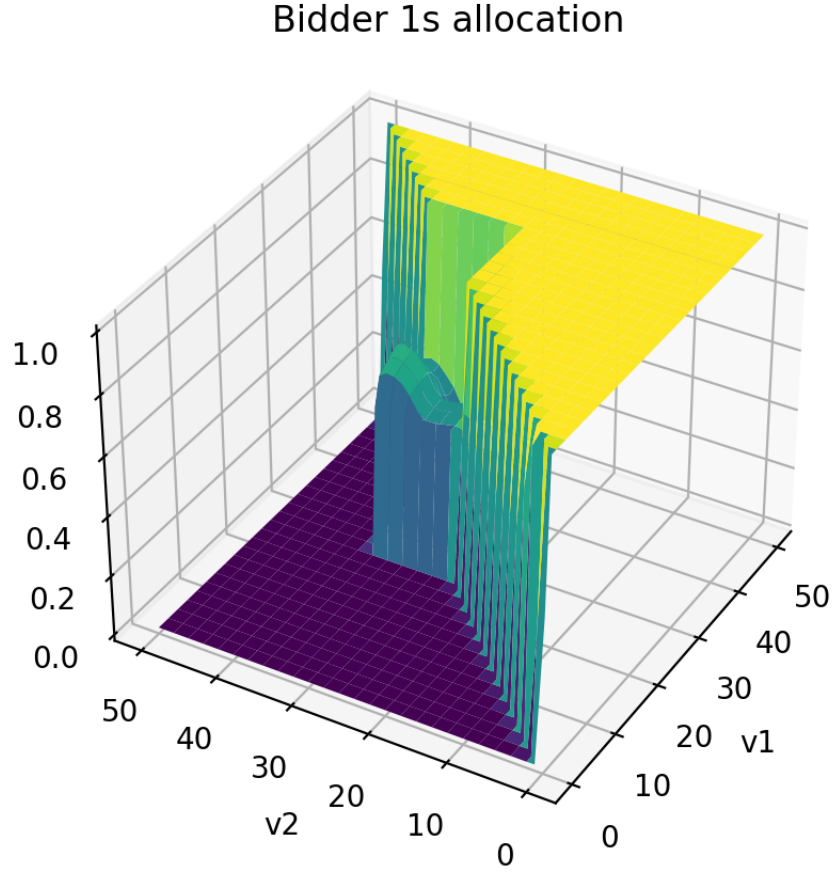
```python
# Redo the exercise where with probability 1/4, the value is uniformly
# distributed on an evenly spaced grid in [0,1], and with probability 1/2,
# the value is 1/2. Also, restrict the seller to mechanisms that always
# allocate the good. Just plot the optimal allocation and IC multipliers.

# ----------------------------------------

numVals=51
K=range(0,numVals)

V={k:k/(numVals-1) for k in K};

f={k:1/(4*numVals) for k in K};
f[(numVals-1)/2]+= 3/4;

model = gp.Model()

model.Params.Method = 2 # Barrier algorithm
model.Params.Crossover = 0 # Disable crossover
model.Params.OutputFlag = 1 # Enable output

q1=model.addVars(K,K)
q2=model.addVars(K,K)
t1=model.addVars(K,K,lb=-GRB.INFINITY)
t2=model.addVars(K,K,lb=-GRB.INFINITY)
```

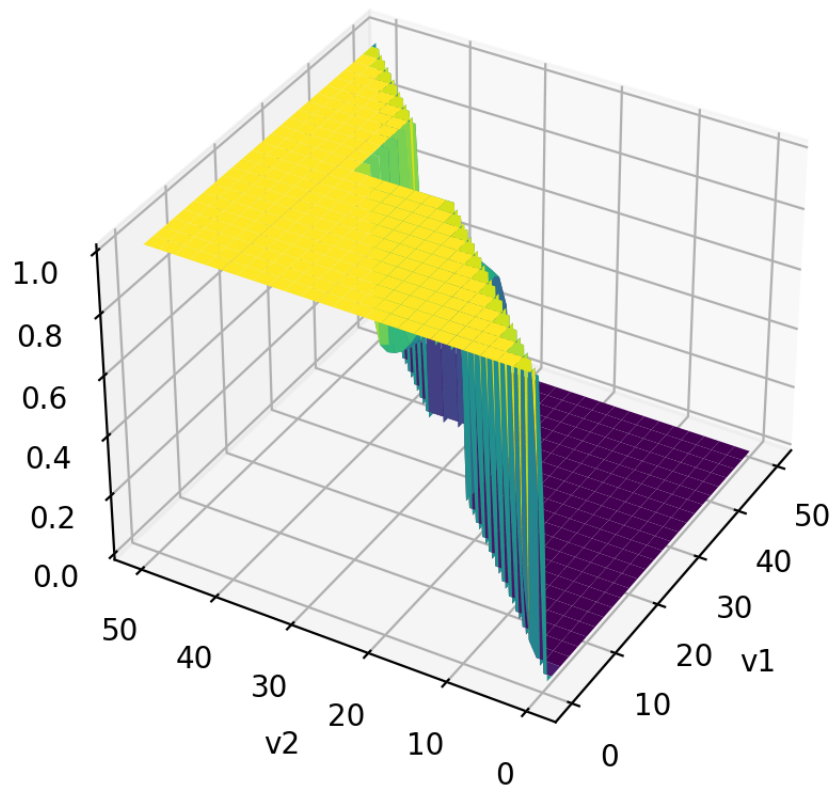
```
ir1 = model.addConstrs(sum((V[v1]*q1[v1,v2]-t1[v1,v2])*f[v2] for v2 in K)>=0 for v1 in K)
ir2 = model.addConstrs(sum((V[v2]*q2[v1,v2]-t2[v1,v2])*f[v1] for v1 in K)>=0 for v2 in K)
ic1 = model.addConstrs(sum((V[v1]*(q1[v1,v2]-q1[vhat,v2])-(t1[v1,v2]-t1[vhat,v2]))*f[v2] for v2 in K)>=0
ic2 = model.addConstrs(sum((V[v2]*(q2[v1,v2]-q2[v1,vhat])-(t2[v1,v2]-t2[v1,vhat]))*f[v1] for v1 in K)>=0

feas = model.addConstrs(q1[v1,v2]+q2[v1,v2]==1 for v1 in K for v2 in K)

model.setObjective(sum((t1[v1,v2]+t2[v1,v2])*f[v1]*f[v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

Q1 = np.array([[q1[v1,v2].X for v1 in K] for v2 in K])
Q2 = np.array([[q2[v1,v2].X for v1 in K] for v2 in K])

X, Y = np.meshgrid(K,K)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Q1, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 1''s allocation');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Q2, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Bidder 2''s allocation');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, T1, cmap='viridis')

IC1 = np.array([[ic1[v,vhat].Pi for v in K] for vhat in K])
IC2 = np.array([[ic2[v,vhat].Pi for v in K] for vhat in K])

X, Y = np.meshgrid(K,K)
```

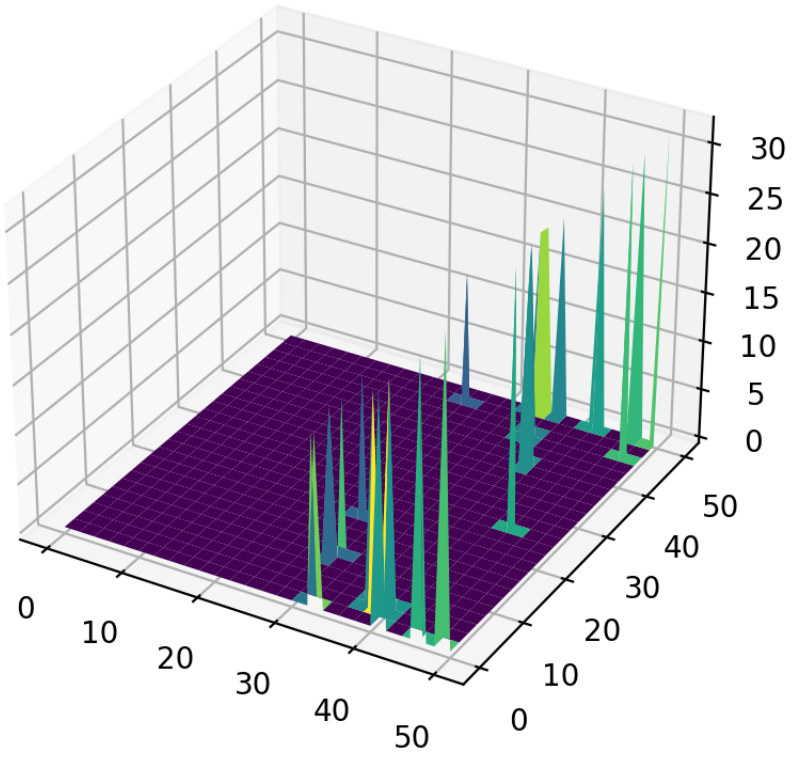
```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, IC1, cmap='viridis')

ax.set_xlabel('v')
ax.set_ylabel('vhat')
ax.set_title('Bidder 1''s IC multipliers');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, IC2, cmap='viridis')

ax.set_xlabel('v')
ax.set_ylabel('vhat')
ax.set_title('Bidder 2''s IC multipliers');
ax.view_init(35,210)

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

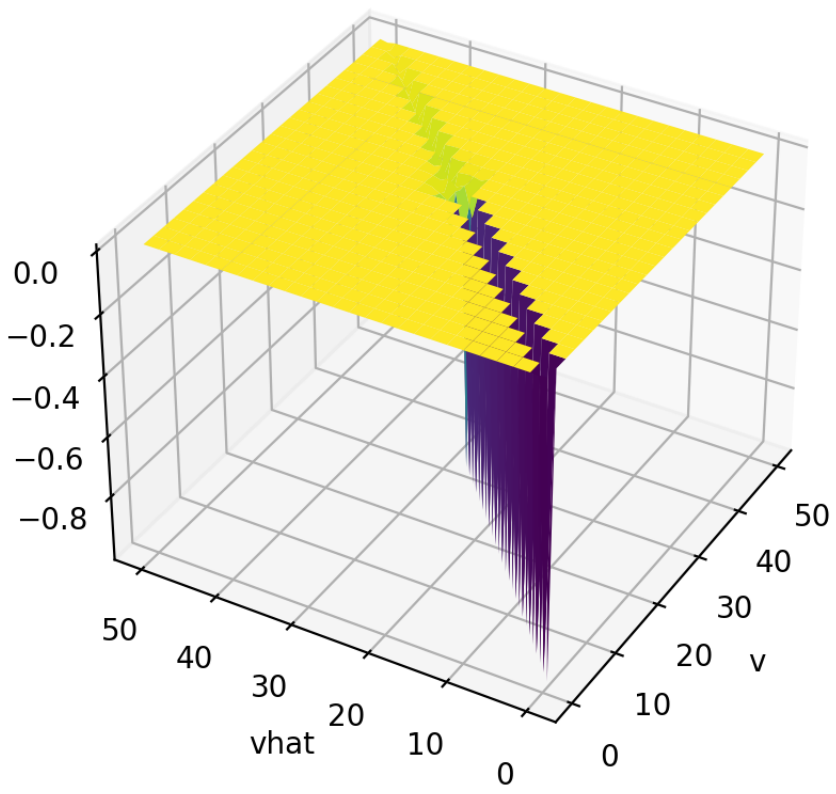
```
Set parameter Method to value 2
Set parameter Crossover to value 0
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 7905 rows, 10404 columns and 1045704 nonzeros
Model fingerprint: 0x06aa5814
Coefficient statistics:
  Matrix range     [1e-04, 1e+00]
  Objective range  [2e-05, 6e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 2705 rows and 7701 columns
Presolve time: 0.45s
Presolved: 5200 rows, 2703 columns, 525400 nonzeros
Ordering time: 0.32s

Barrier statistics:
 AA' NZ     : 7.015e+06
 Factor NZ  : 1.327e+07 (roughly 100 MB of memory)
 Factor Ops : 4.489e+10 (roughly 1 second per iteration)
 Threads    : 4

                  Objective                Residual
```

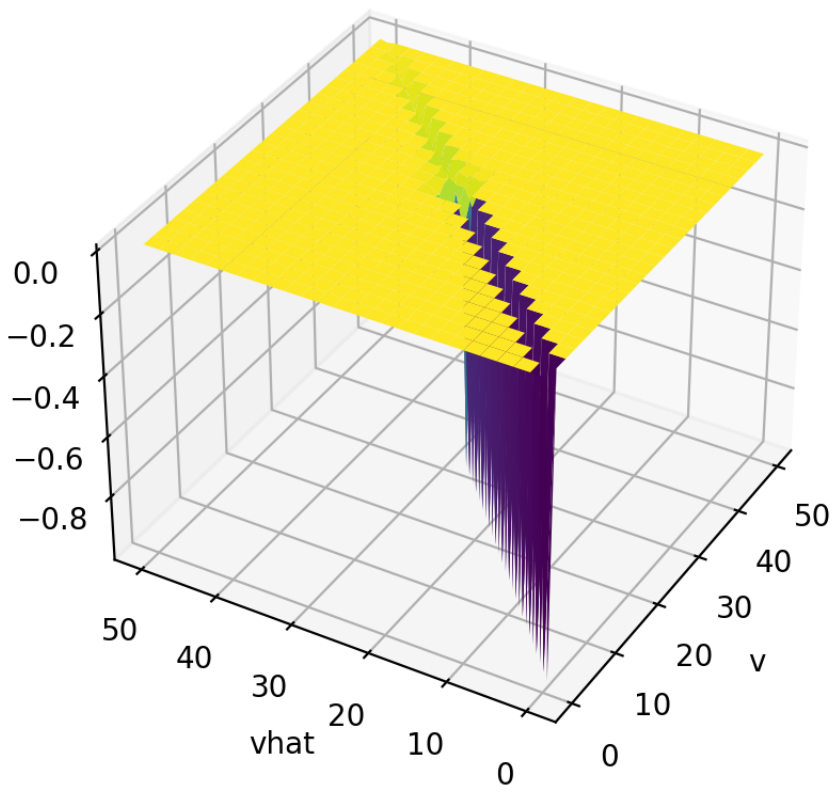
```
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -1.18761208e+00  1.75433007e+00  5.29e+02 0.00e+00  1.25e+00     3s
   1  -1.64359607e+00  2.41960146e+00  1.16e+02 2.53e-02  3.13e-01     3s
   2  -5.12112922e+00  2.53324636e+00  2.19e+01 1.69e-03  6.74e-02     4s
   3   5.37414045e-01  2.11179827e+00  2.51e+00 2.22e-15  7.99e-03     4s
   4   5.40727838e-01  1.53398472e+00  1.10e+00 7.39e-15  3.81e-03     5s
   5   4.38538383e-01  1.06502868e+00  2.65e-01 1.28e-15  1.54e-03     6s
   6   4.54544902e-01  5.79010258e-01  2.82e-02 1.61e-15  2.37e-04     6s
   7   4.53767880e-01  4.78502479e-01  7.80e-03 2.11e-15  4.92e-05     7s
   8   4.54535660e-01  4.62819050e-01  4.03e-03 3.48e-15  1.88e-05     8s
   9   4.55740546e-01  4.56839298e-01  1.23e-04 4.63e-15  1.87e-06     8s
  10   4.56012112e-01  4.56298579e-01  2.32e-05 3.27e-15  4.75e-07     9s
  11   4.56048061e-01  4.56203172e-01  1.54e-05 3.54e-15  2.62e-07    10s
  12   4.56083210e-01  4.56146443e-01  6.70e-06 4.70e-15  1.08e-07    10s
  13   4.56095045e-01  4.56135321e-01  4.34e-06 4.22e-15  6.86e-08    11s
  14   4.56098804e-01  4.56123499e-01  2.32e-06 3.28e-15  4.15e-08    12s
  15   4.56104438e-01  4.56120327e-01  1.63e-06 5.42e-15  2.70e-08    12s
  16   4.56108379e-01  4.56118277e-01  9.77e-07 4.35e-15  1.67e-08    13s
  17   4.56112588e-01  4.56115128e-01  1.53e-07 5.35e-15  4.11e-09    14s
  18   4.56113618e-01  4.56114459e-01  6.15e-08 3.16e-15  1.39e-09    14s
  19   4.56113981e-01  4.56114017e-01  1.92e-09 7.13e-15  5.70e-11    15s
  20   4.56113990e-01  4.56113995e-01  7.21e-09 7.75e-15  5.79e-12    16s

Barrier solved model in 20 iterations and 15.57 seconds (25.67 work units)
Optimal objective 4.56113990e-01
```


Bidder 1s allocation
1.0
0.8
0.6
0.4
0.2
0.0
50
40
30
20
10
0
v2
50
40
30
20
10
0
v1


Bidder 2s allocation
1.0
0.8
0.6
0.4
0.2
0.0
50
40
30
20
10
0
v2
50
40
30
20
10
0
v1


30
25
20
15
10
5
0
0
10
20
30
40
50
0
10
20
30
40
50


Bidder 1s IC multipliers
0.0
−0.2
−0.4
−0.6
−0.8
50
40
30
20
10
0
vhat
0
10
20
30
40
50
v


Bidder 2s IC multipliers
0.0
−0.2
−0.4
−0.6
−0.8
50
40
30
20
10
0
vhat
50
40
30
20
10
0
v

In [ ]:

In [12]:

```
# Once again, resolve the model, with only the incentive constraints
# associated with misreporting the next lower signal, and individual
# rationality for the lowest valuation. What happens to the value?
# How do your conclusions differ from the previous example?

# ----------------------------------------

numVals=51
K=range(0,numVals)

V={k:k/(numVals-1) for k in K};

f={k:1/(4*numVals) for k in K};
f[(numVals-1)/2]+= 3/4;

model = gp.Model()

model.Params.Method = 2 # Barrier algorithm
model.Params.Crossover = 0 # Disable crossover
model.Params.OutputFlag = 1 # Enable output

q1=model.addVars(K,K)
q2=model.addVars(K,K)
t1=model.addVars(K,K,lb=-GRB.INFINITY)
t2=model.addVars(K,K,lb=-GRB.INFINITY)

ir1 = model.addConstrs(sum((V[v1]*q1[v1,v2]-t1[v1,v2])*f[v2] for v2 in K)>=0 for v1 in K if v1==0)
ir2 = model.addConstrs(sum((V[v2]*q2[v1,v2]-t2[v1,v2])*f[v1] for v1 in K)>=0 for v2 in K if v2==0)
ic1 = model.addConstrs(sum((V[v1]*(q1[v1,v2]-q1[vhat,v2])-(t1[v1,v2]-t1[vhat,v2]))*f[v2] for v2 in K)>=0
ic2 = model.addConstrs(sum((V[v2]*(q2[v1,v2]-q2[v1,vhat])-(t2[v1,v2]-t2[v1,vhat]))*f[v1] for v1 in K)>=0

feas = model.addConstrs(q1[v1,v2]+q2[v1,v2]==1 for v1 in K for v2 in K)

model.setObjective(sum((t1[v1,v2]+t2[v1,v2])*f[v1]*f[v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

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

```
Set parameter Method to value 2
Set parameter Crossover to value 0
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 2703 rows, 10404 columns and 25704 nonzeros
Model fingerprint: 0xd952c9e2
Coefficient statistics:
  Matrix range     [1e-04, 1e+00]
  Objective range  [2e-05, 6e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 2703 rows and 10404 columns
Presolve time: 0.01s
Presolve: All rows and columns removed

Barrier solved model in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective 4.66900711e-01
```