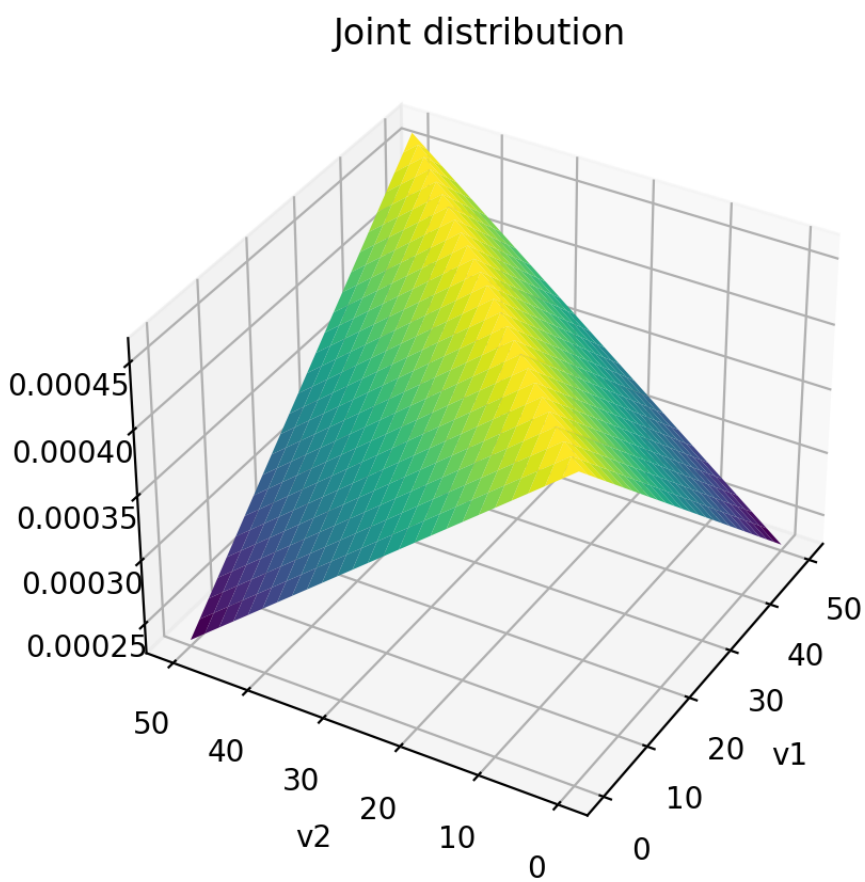
In [1]:
```
# ECON 289 Problem set 4:
# Ex post implementation and robust mechanism design
# Instructor: Ben Brooks
# Spring 2023

# 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.
```

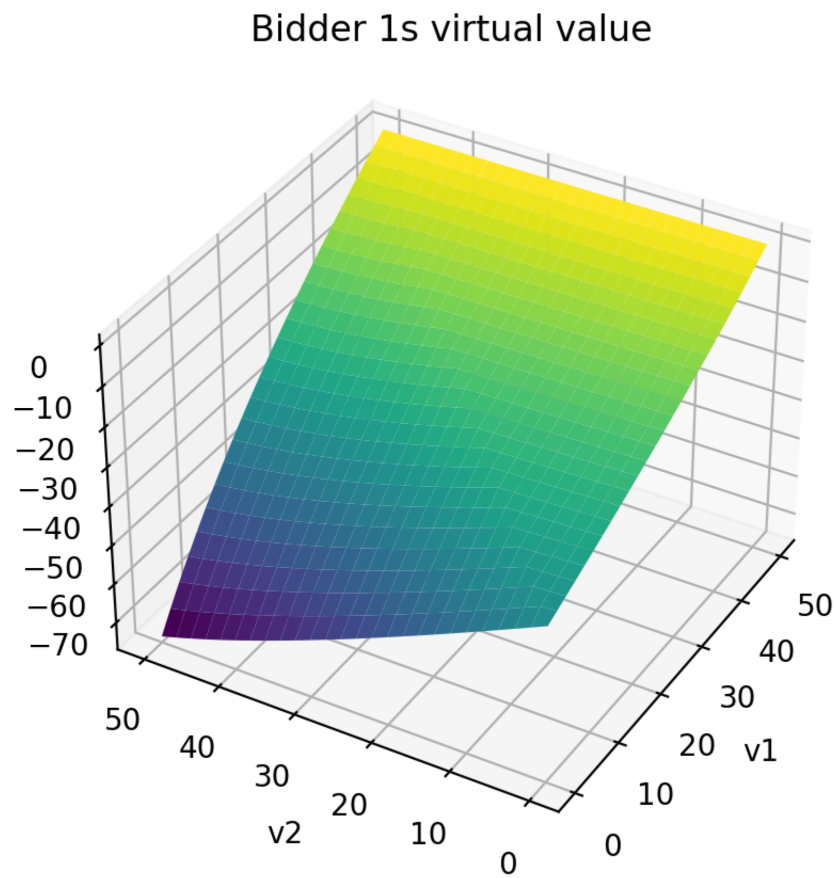
In [2]:
```
# Insert code to load gurobi, numpy, matplotlib pyplot, and mplot3d.

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

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 [3]:
```
# We will solve the ex post auction design problem when there are
# two bidders with values that are correlated, but satisfy Chung and Ely's
# generalized regularity condition.

# Create an index set and a set of 51 evenly
```

```
# spaced values. Create a joint distribution f[v1,v2] such that the density is proportional
# to 2-|v1-v2|.

# (Hint: You can create dictionaries with multidimensional indices, e.g.,
# D={(foo,bar): foo*bar for foo in F for bar in B}, which you can then access
# as D[foo,bar].)

# Check if this distribution satisfies Chung and Ely's regularity condition by plotting the
# virtual value of bidder 1 as a function of (v1,v2). What is the efficient surplus?

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

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

totalProb = sum(2-abs(V[v1]-V[v2])  for v1 in K for v2 in K)
f={(v1,v2):(2-abs(V[v1]-V[v2]))/totalProb for v1 in K for v2 in K};

F = np.array([[f[v1,v2] for v1 in K] for v2 in K])

vv1 = np.array([[V[v1]-sum(f[v,v2] for v in K if v > v1)/f[v1,v2] 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, F, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Joint distribution');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, vv1, cmap='viridis')

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

```
print('The virtual value is non-decreasing in the bidder''s own value, so CE regularity is satisfied.')

print(f'The efficient surplus is {sum(max(V[v1],V[v2])*f[v1,v2] for v1 in K for v2 in K)}')
# ------------------------------------------
```



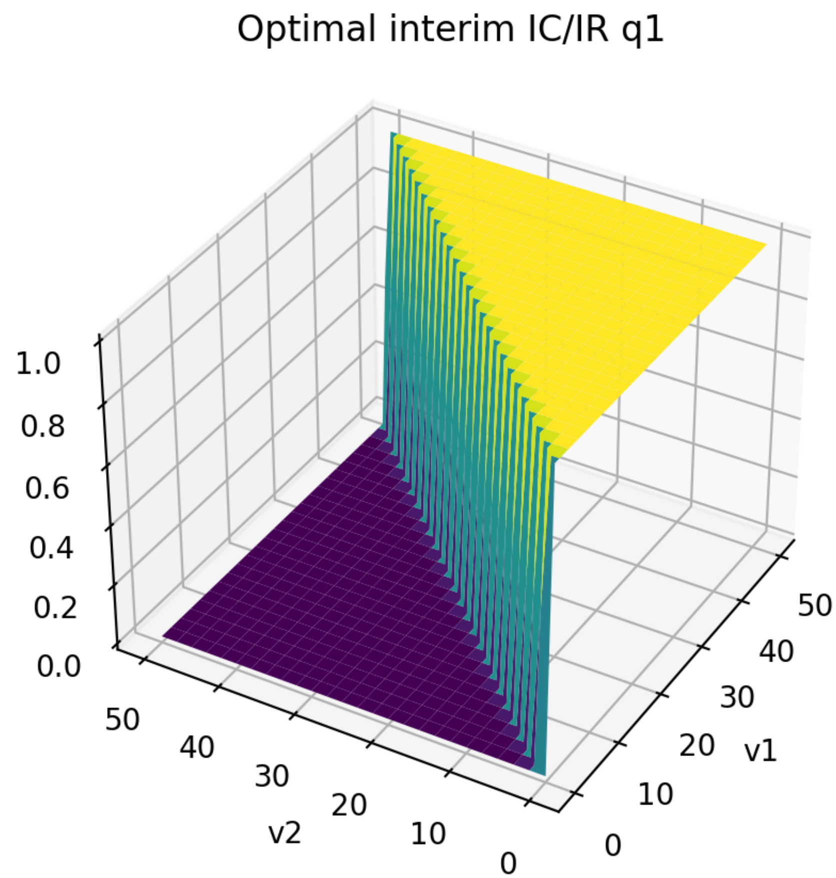



```
The virtual value is non-decreasing in the bidders own value, so CE regularity is satisfied.
The efficient surplus is 0.652519685039366
```

In [4]:
```
# Now create the Gurobi model. 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! Add the feasibility, interim IR, and interim IC constraints. Compute optimal
# revenue, across all interim IC and IR mechanisms. Plot the optimal allocation.
# How do you interpret the result?

# ------------------------------------------
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)
# t1=model.addVars(K,K,lb=0)
# t2=model.addVars(K,K,lb=0)

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

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,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])

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('Optimal interim IC/IR q1');
ax.view_init(35,210)

print('The allocation is efficient, but the auctioneer can exploit correlation in types to extract all of
# ---------------------------------------------
```

```
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
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: 0x584143c1
Coefficient statistics:
  Matrix range     [5e-06, 1e+00]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 203 rows and 102 columns
Presolve time: 0.42s
Presolved: 7702 rows, 10302 columns, 1040402 nonzeros
Ordering time: 0.70s

Barrier statistics:
 Free vars  : 2601
 AA' NZ     : 4.988e+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   6.59359198e-06  3.86718750e-02  5.77e-01 1.19e-01  5.63e-02     2s
   1   2.73929178e-01  1.44887531e+01  9.87e-03 1.15e-03  4.83e-03     2s
   2   3.63550430e-01  7.14901108e-01  0.00e+00 1.24e-05  1.12e-04     2s
   3   6.28896765e-01  6.67728936e-01  0.00e+00 9.31e-07  1.23e-05     3s
   4   6.52132026e-01  6.52846305e-01  0.00e+00 1.80e-08  2.25e-07     3s
   5   6.52519291e-01  6.52520021e-01  0.00e+00 9.80e-11  2.28e-10     3s
   6   6.52519685e-01  6.52519685e-01  0.00e+00 9.75e-14  2.28e-13     3s

Barrier solved model in 6 iterations and 3.25 seconds (2.40 work units)
Optimal objective 6.52519685e-01
```



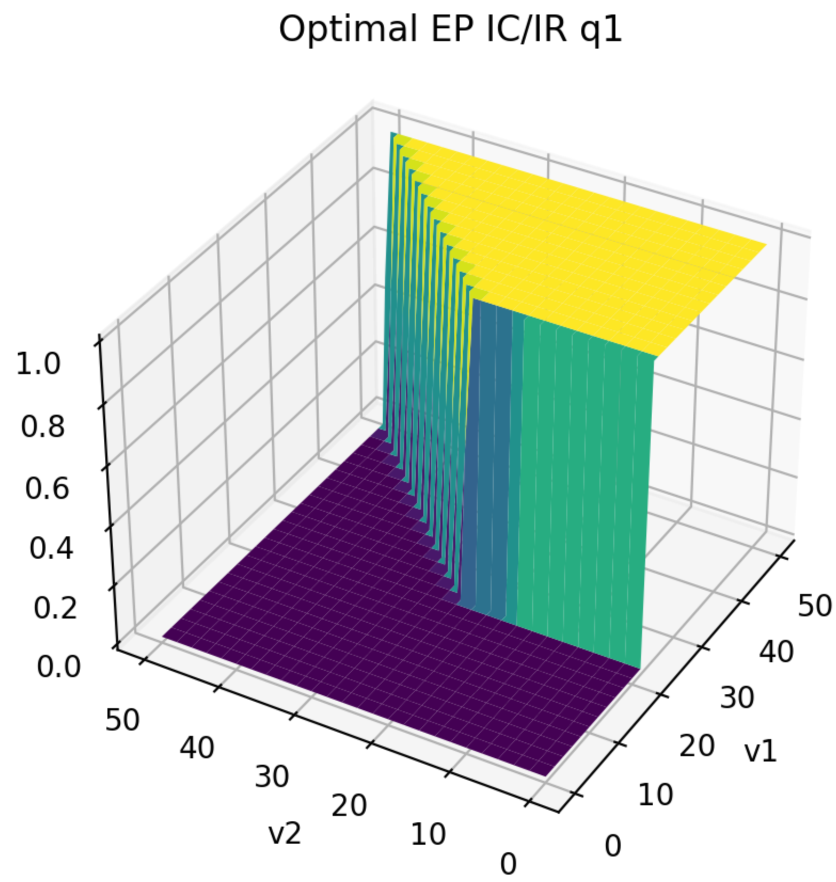
The allocation is efficient, but the auctioneer can exploit correlation in types to extract all of the s urplus using side bets.

In [5]:
```
# Now compute the revenue maximizing mechanism subject to ex post IC and IR constraints.

# How does the result change? How do you interpret the result?

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

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

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,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])

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('Optimal EP IC/IR q1');
ax.view_init(35,210)

print('The optimal auction now rations the good. The outcome is inefficient, and the seller is unable to

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

```
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 273105 rows, 10404 columns and 1045704 nonzeros
Model fingerprint: 0x01b79677
Coefficient statistics:
```

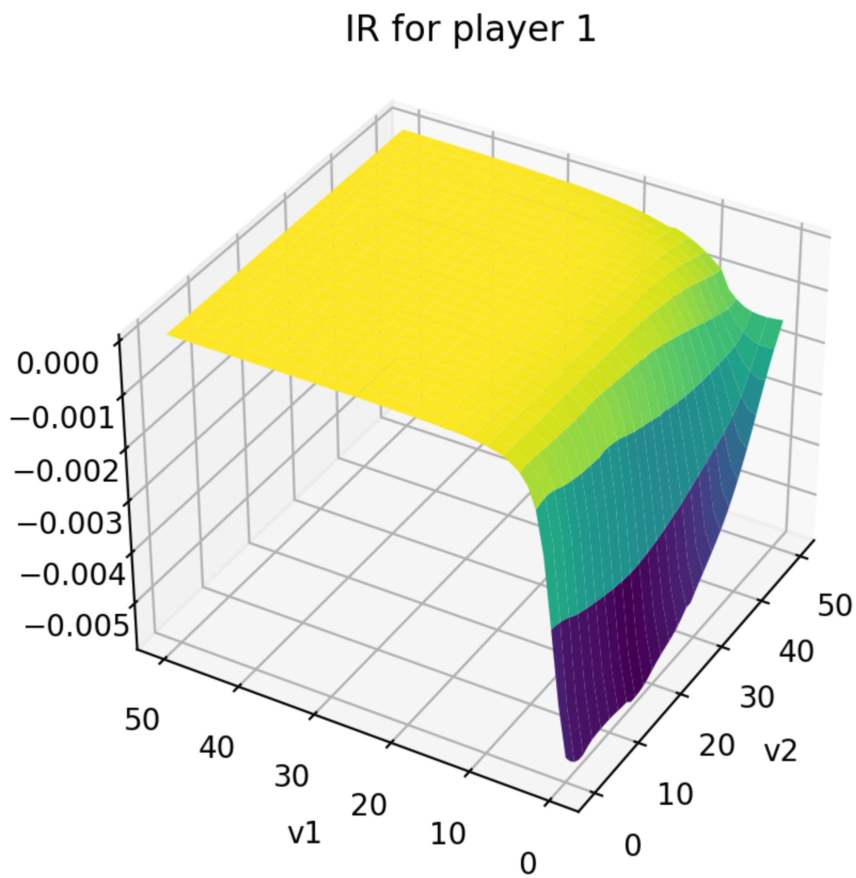
```
  Matrix range     [2e-02, 1e+00]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 102 rows and 5305 columns
Presolve time: 0.68s
Presolved: 10302 rows, 267800 columns, 1040400 nonzeros
Ordering time: 0.11s

Barrier statistics:
 AA' NZ     : 2.588e+05
 Factor NZ  : 8.069e+05 (roughly 60 MB of memory)
 Factor Ops : 3.032e+08 (less than 1 second per iteration)
 Threads    : 4

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   3.25418566e+01  0.00000000e+00  3.43e+00 0.00e+00  1.01e-02     2s
   1   2.30791124e+01  1.21433918e-02  1.38e+00 3.73e-02  4.02e-03     2s
   2   2.18829223e+01  8.33300569e-02  3.28e-01 1.15e-02  1.24e-03     2s
   3   1.25809526e+01  1.32049332e-01  2.87e-02 2.47e-03  2.14e-04     2s
   4   4.01719090e+00  2.69699881e-01  4.89e-03 1.09e-04  3.79e-05     2s
   5   1.70507016e+00  2.80339920e-01  1.75e-03 2.91e-05  1.39e-05     2s
   6   5.11307910e-01  2.99655999e-01  2.37e-04 5.55e-17  2.01e-06     2s
   7   4.96302395e-01  3.22595636e-01  2.12e-04 5.55e-17  1.71e-06     2s
   8   4.52400458e-01  3.60294375e-01  1.06e-04 5.55e-17  9.22e-07     2s
   9   4.27750494e-01  3.88415090e-01  3.25e-05 5.55e-17  3.70e-07     2s
  10   4.21143535e-01  4.11102526e-01  1.13e-05 5.55e-17  1.04e-07     2s
  11   4.19308879e-01  4.17872019e-01  2.64e-06 8.33e-17  1.77e-08     2s
  12   4.18993177e-01  4.18813040e-01  5.36e-07 8.33e-17  2.77e-09     2s
  13   4.18893204e-01  4.18883994e-01  3.19e-08 8.33e-17  1.54e-10     2s
  14   4.18885679e-01  4.18885480e-01  2.52e-09 6.66e-16  1.48e-12     2s
  15   4.18885595e-01  4.18885595e-01  2.41e-11 6.66e-16  1.48e-15     2s

Barrier solved model in 15 iterations and 2.37 seconds (1.20 work units)
Optimal objective 4.18885595e-01
```



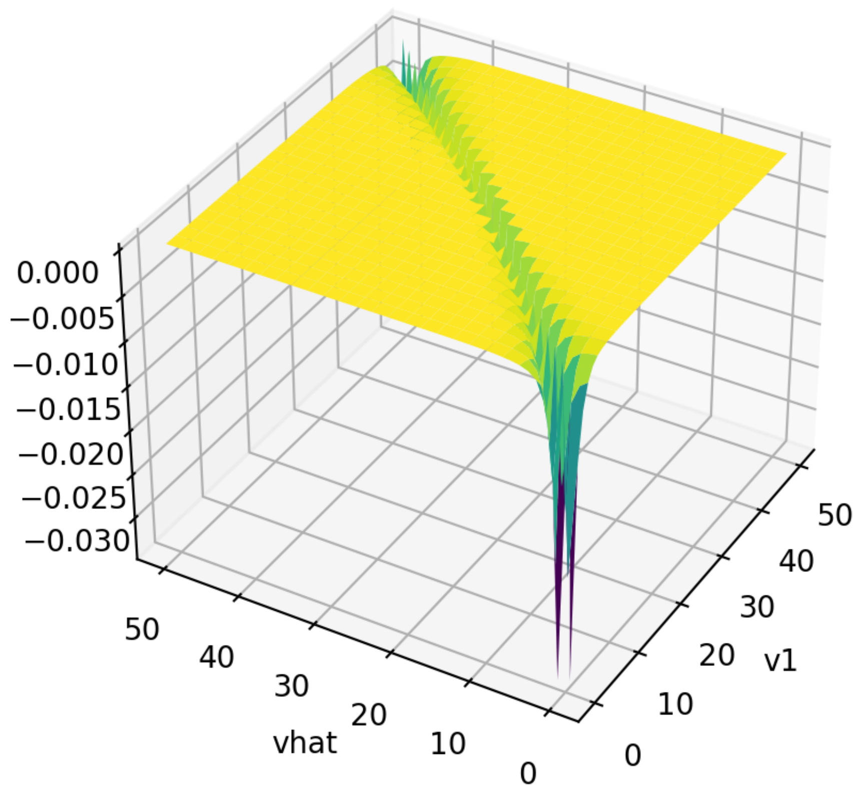
The optimal auction now rations the good. The outcome is inefficient, and the seller is unable to extrac
t all of the surplus, in spite of the correlation in types.

In [6]:

```
# Let us examine the optimal multipliers. Plot the IR and IC multipliers for
# bidder 1. For the IC multipliers, do this for v2=V[10]. Which IC multipliers
# appear to be non-zero?

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


IR1=np.array([[ir1[v1,v2].Pi for v1 in K] for v2 in K])
IC1=np.array([[(ic1[10,v1,vhat].Pi if v1 != vhat else 0) for v1 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, IR1, cmap='viridis')

ax.set_xlabel('v2')
ax.set_ylabel('v1')
ax.set_title('IR for player 1');
ax.view_init(35,210)


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

ax.set_xlabel('v1')
ax.set_ylabel('vhat')
ax.set_title('IC for player 1 when v2=V[10]');
ax.view_init(35,210)

print('It appears that only local-downward incentive constraints bind. Lots of IR constraints bind, but c

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


IR for player 1
0.000
−0.001
−0.002
−0.003
−0.004
−0.005
50
40
30
20
10
0
v1
50
40
30
20
10
0
v2

IC for player 1 when v2=V[10]



It appears that only local-downward incentive constraints bind. Lots of IR constraints bind, but only fo
r values below the level at which they are excluded from the auction.

In [7]:

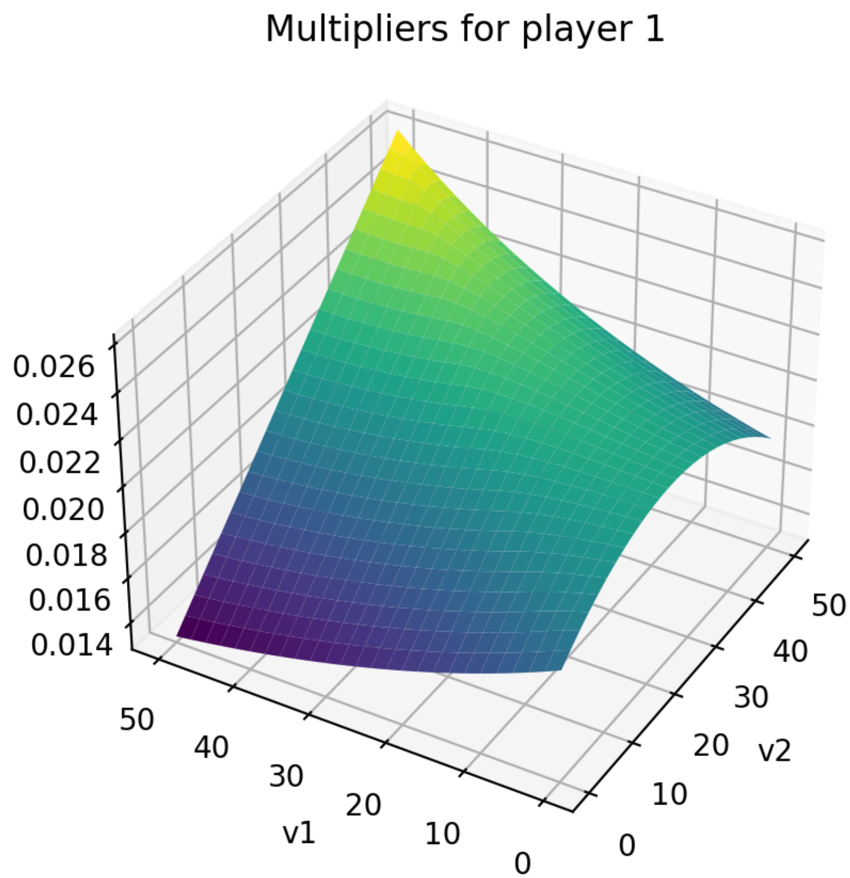
```
# Based on the previous simulation, and what we know of the optimal auction in the IPV
# setting, we may guess that the solution will not change if we drop all but the ex post IR
# constraints for the lowest type and local downward IC. Do this, recompute the
# solution, verify that the value hasn't changed.

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

model = gp.Model()

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

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(V[0]*q1[0,v2]-t1[0,v2]>=0 for v2 in K)
ir2 = model.addConstrs(V[0]*q2[v1,0]-t2[v1,0]>=0 for v1 in K)
ic1 = model.addConstrs((V[v1]*(q1[v1,v2]-q1[v1-1,v2])-(t1[v1,v2]-t1[v1-1,v2]))>=0 for v2 in K for v1 in K
ic2 = model.addConstrs((V[v2]*(q2[v1,v2]-q2[v1,v2-1])-(t2[v1,v2]-t2[v1,v2-1]))>=0 for v1 in K for v2 in K

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,v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

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

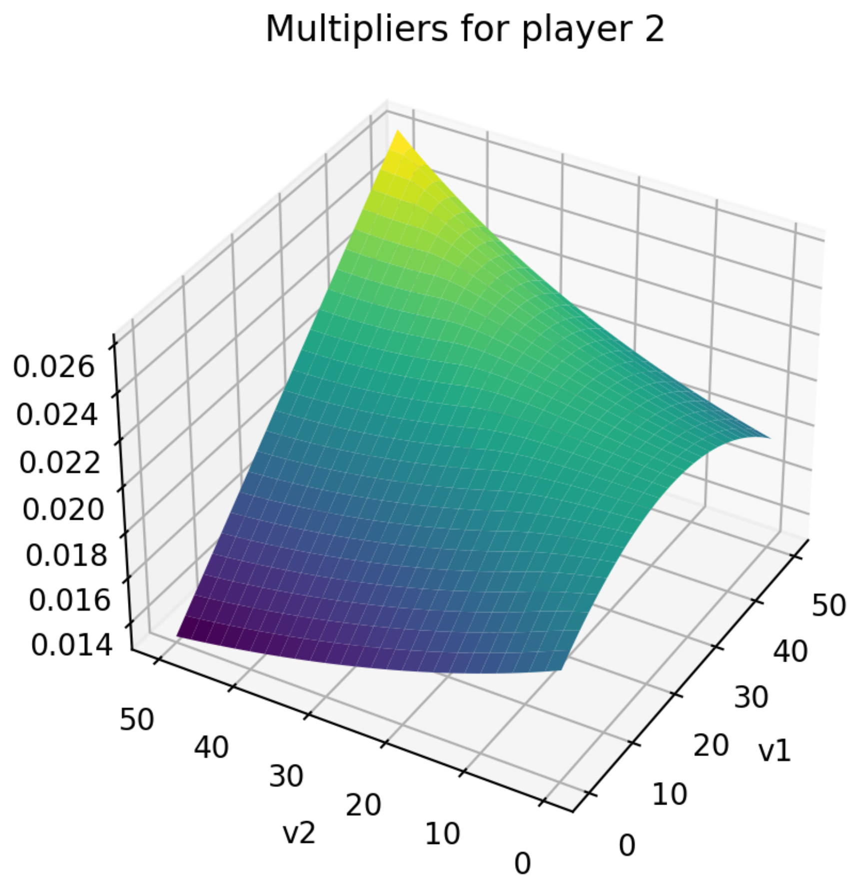
```
Set parameter Method to value 1
Set parameter Crossover to value 0
Set parameter Presolve to value 0
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 7803 rows, 10404 columns and 25704 nonzeros
Model fingerprint: 0x31d16290
Coefficient statistics:
  Matrix range     [2e-02, 1e+00]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                          0s
    7263    4.1888560e-01   0.000000e+00   0.000000e+00      0s

Solved in 7263 iterations and 0.05 seconds (0.02 work units)
Optimal objective  4.188855952e-01
```

In [8]:

```
# Plot the IR and local downward IC multipliers for bidder 1
# as a function of [v1,v2], normalized so that the multipliers sum
# to 1 across v2, for every v1.

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

# First create array with the IC/IR multipliers
# NB Important to assign 0.0 so that numpy doesn't make this an
# array of integers! Wasted too much time trying to debug that.
IR1=np.array([ir1[v2].Pi for v2 in K])
IR2=np.array([ir2[v1].Pi for v1 in K])

mult1 = np.array([[np.double(0) for v1 in K] for v2 in K])
mult2 = np.array([[np.double(0) for v1 in K] for v2 in K])
for vi in K:
    for vj in K:
        if vi > 0:
            mult1[vi][vj] = -ic1[vj,vi].Pi
            mult2[vi][vj] = -ic2[vj,vi].Pi
        else:
            mult1[vi][vj] = -ir1[vj].Pi
            mult2[vi][vj] = -ir2[vj].Pi

# Now normalize so that the sum across v2 is 1.
```

```
for vi in K:
    totalProb = sum(mult1[vi,vj] for vj in K)
    for vj in K:
        mult1[vi,vj]/=totalProb
    totalProb = sum(mult2[vi,vj] for vj in K)
    for vj in K:
        mult2[vi,vj]/=totalProb

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

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, mult1, cmap='viridis')

ax.set_xlabel('v2')
ax.set_ylabel('v1')
ax.set_title('Multipliers for player 1');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, mult1, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Multipliers for player 2');
ax.view_init(35,210)

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


Multipliers for player 1
0.026
0.024
0.022
0.020
0.018
0.016
0.014
50
40
30
20
10
0
v1
50
40
30
20
10
0
v2



Multipliers for player 2
0.026
0.024
0.022
0.020
0.018
0.016
0.014
50
40
30
20
10
0
v2
50
40
30
20
10
0
v1

In [9]:

```
# Now set up a new version of the interim IC problem, where
# you use the normalized multipliers as the beliefs of the players,
# and solve for optimal revenue. What is the value of the solution?
# How do you interpret this result?

# -------------------------------------------
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])*mult1[v1,v2] for v2 in K)>=0 for v1 in K)
ir2 = model.addConstrs(sum((V[v2]*q2[v1,v2]-t2[v1,v2])*mult2[v2,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]))*mult1[v1,v2] for v2 ir
ic2 = model.addConstrs(sum((V[v2]*(q2[v1,v2]-q2[v1,vhat])-(t2[v1,v2]-t2[v1,vhat]))*mult2[v2,v1] for v1 ir

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,v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

print('The value is the same as for the ex post IC/IR problem.')

print('The interpretation is that with these particular non-common prior beliefs, the designer is not abl
# -------------------------------------------
```

```
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: 0x47d0b9d6
Coefficient statistics:
  Matrix range     [4e-04, 1e+00]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 203 rows and 102 columns
Presolve time: 0.41s
Presolved: 7702 rows, 10302 columns, 1040402 nonzeros
Ordering time: 0.72s

Barrier statistics:
 Free vars  : 2601
 AA' NZ     : 4.988e+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   4.35335567e-06  3.73046875e-02  1.73e+00 1.19e-01  1.18e-01     2s
   1  -2.19090541e+00  8.63778416e-01  1.22e+00 8.21e-02  9.17e-02     2s
   2  -6.09881119e+01  1.15620269e+00  1.15e+00 6.54e-02  1.52e-01     3s
   3  -4.71673123e+02  1.57644091e+00  3.44e-01 2.94e-02  1.56e+00     3s
   4  -1.40672616e+02  7.43875381e-01  0.00e+00 7.44e-03  6.42e-01     3s
   5  -1.23942722e+00  4.75249750e-01  0.00e+00 1.82e-03  9.53e-03     3s
   6   4.13078838e-01  4.19818795e-01  0.00e+00 3.12e-05  5.48e-05     3s
   7   4.18838650e-01  4.18945715e-01  0.00e+00 1.25e-06  1.40e-06     4s
   8   4.18884801e-01  4.18886189e-01  0.00e+00 3.65e-09  9.64e-09     4s
   9   4.18885594e-01  4.18885595e-01  3.71e-11 1.88e-13  4.22e-12     4s

Barrier solved model in 9 iterations and 4.08 seconds (3.24 work units)
Optimal objective 4.18885594e-01
```

The value is the same as for the ex post IC/IR problem.
The interpretation is that with these particular non-common prior beliefs, the designer is not able to extract all of the surplus through side bets.

In [10]:

```python
# In our unit on epistemic game theory, we showed that a type space
# satisfies the CPA if and only if there is no trade. If there is an admissible trade,
# then a third party could use the players as a money pump, by charging a fee to
# whichever player has a strictly positive interim expected payment, and then scaling
# up the transfers to extract arbitrarily large fees. Check if the players can be
# used as a money pump for the Chung-Ely type space. In particular, solve the linear
# program of maximizing total revenue, subject to each type having a non-negative
# surplus. What do you find? Can the players be exploited as a money pump?
# How do you reconcile this with your previous findings?

# (Hint: For models that are infeasible or unbounded, Gurobi will tell you which is the case
# if you run the simplex algorithm and change the "InfUnbdInfo" parameter to 1.)

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

model = gp.Model()

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

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

ir1 = model.addConstrs(sum(-t1[v1,v2]*mult1[v1,v2] for v2 in K)>=0 for v1 in K)
ir2 = model.addConstrs(sum(-t2[v1,v2]*mult2[v2,v1] for v1 in K)>=0 for v2 in K)

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

print('The model is unbounded, indicating that the players can indeed be used as a money pump. But this i

model = gp.Model()

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

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

```
ir1 = model.addConstrs(sum(-t1[v1,v2]*mult1[v1,v2] for v2 in K)>=0 for v1 in K)
ir2 = model.addConstrs(sum(-t2[v1,v2]*mult2[v2,v1] for v1 in K)>=0 for v2 in K)
ic1 = model.addConstrs(sum((-(t1[v1,v2]-t1[vhat,v2]))*mult1[v1,v2] for v2 in K)>=0 for v1 in K for vhat i
ic2 = model.addConstrs(sum((-(t2[v1,v2]-t2[v1,vhat]))*mult2[v2,v1] for v1 in K)>=0 for v2 in K for vhat i

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


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

```
Set parameter Method to value 2
Set parameter Crossover to value 1
Set parameter InfUnbdInfo to value 1
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 102 rows, 5202 columns and 5202 nonzeros
Model fingerprint: 0xad2df1ed
Coefficient statistics:
  Matrix range     [1e-02, 3e-02]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve removed 1 rows and 1 columns
Presolve time: 0.01s
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 5202
 AA' NZ     : 0.000e+00
 Factor NZ  : 1.020e+02 (roughly 2 MB of memory)
 Factor Ops : 1.020e+02 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -0.00000000e+00 -0.00000000e+00  0.00e+00 9.84e-02  1.00e-02     0s
   1   6.91769492e+03 -0.00000000e+00  0.00e+00 1.92e-02  3.89e-04     0s
   2   6.00981012e+09 -0.00000000e+00  6.58e-05 1.88e-02  4.86e-07     0s
   3   1.27075797e+10 -0.00000000e+00  1.05e-04 1.92e-02  1.64e-08     0s
   4   1.96248522e+10 -0.00000000e+00  2.40e-04 1.92e-02  1.64e-11     0s
   5   2.65422776e+10 -0.00000000e+00  2.52e-04 1.92e-02  4.72e-17     0s
```

```
Barrier performed 5 iterations in 0.03 seconds (0.00 work units)
Numerical trouble encountered

Model may be infeasible or unbounded.  Consider using the
homogeneous algorithm (through parameter 'BarHomogeneous')

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                          0s

Solved in 102 iterations and 0.04 seconds (0.00 work units)
Unbounded model
The model is unbounded, indicating that the players can indeed be used as a money pump. But this is beca
use that characterization does not impose IC. If we add interim IC back into the model, then the players
cannot be used as a money pump:
Set parameter Method to value 2
Set parameter Crossover to value 1
Set parameter InfUnbdInfo to value 1
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 5304 rows, 5202 columns and 525402 nonzeros
Model fingerprint: 0x3e566825
Coefficient statistics:
  Matrix range     [1e-02, 3e-02]
  Objective range  [2e-04, 5e-04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve removed 102 rows and 0 columns
Presolve time: 0.28s
Presolved: 5202 rows, 5202 columns, 525402 nonzeros

Ordering time: 0.23s

Barrier statistics:
 Free vars  : 2601
 AA' NZ     : 2.563e+05
 Factor NZ  : 2.263e+06 (roughly 20 MB of memory)
 Factor Ops : 2.531e+09 (less than 1 second per iteration)
 Threads    : 4

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -0.00000000e+00 -0.00000000e+00  0.00e+00 1.19e-01  1.00e-02     1s
   1  -2.50383132e-01 -0.00000000e+00  8.50e-02 1.05e-01  9.84e-03     1s
```

```
   2  -8.19285127e+00 -0.00000000e+00  6.36e-02 7.90e-02  3.60e-02     1s
   3  -3.47658147e+01 -0.00000000e+00  3.71e-02 4.41e-02  2.60e-01     1s
   4  -1.66027423e+01 -0.00000000e+00  5.44e-04 3.78e-03  1.98e-01     1s
   5  -5.90704018e-02 -0.00000000e+00  0.00e+00 1.54e-03  8.60e-04     2s
   6  -1.87249112e-04 -0.00000000e+00  3.11e-04 1.08e-05  2.54e-06     2s
   7   4.97879709e-07 -0.00000000e+00  3.20e-06 1.15e-08  2.68e-09     2s
   8   4.50363059e-10 -0.00000000e+00  1.96e-09 1.33e-12  1.68e-12     2s
   9  -8.23709156e-13 -0.00000000e+00  6.44e-12 4.98e-15  1.61e-15     2s

Barrier solved model in 9 iterations and 1.85 seconds (1.12 work units)
Optimal objective -8.23709156e-13

Crossover log...

       0 DPushes remaining with DInf 0.0000000e+00                 2s

    2550 PPushes remaining with PInf 0.0000000e+00                 2s
       0 PPushes remaining with PInf 0.0000000e+00                 4s

  Push phase complete: Pinf 0.0000000e+00, Dinf 8.8153351e-12      4s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    2553   -0.0000000e+00   0.000000e+00   0.000000e+00      4s

Use crossover to convert LP symmetric solution to basic solution...
Crossover log...

    5100 PPushes remaining with PInf 0.0000000e+00                 4s
    2575 PPushes remaining with PInf 0.0000000e+00                 6s
       0 PPushes remaining with PInf 0.0000000e+00                 9s

  Push phase complete: Pinf 0.0000000e+00, Dinf 1.5013941e-11      9s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    7656   -0.0000000e+00   0.000000e+00   0.000000e+00      9s

Solved in 7656 iterations and 9.29 seconds (6.23 work units)
Optimal objective -0.000000000e+00
```

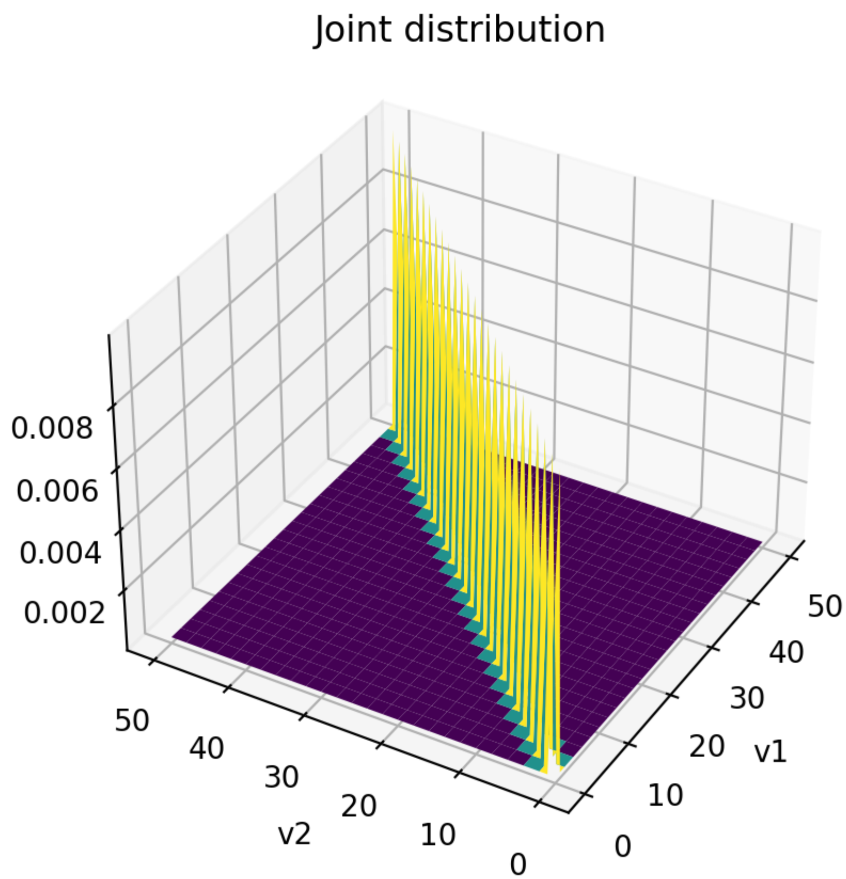
In [11]:

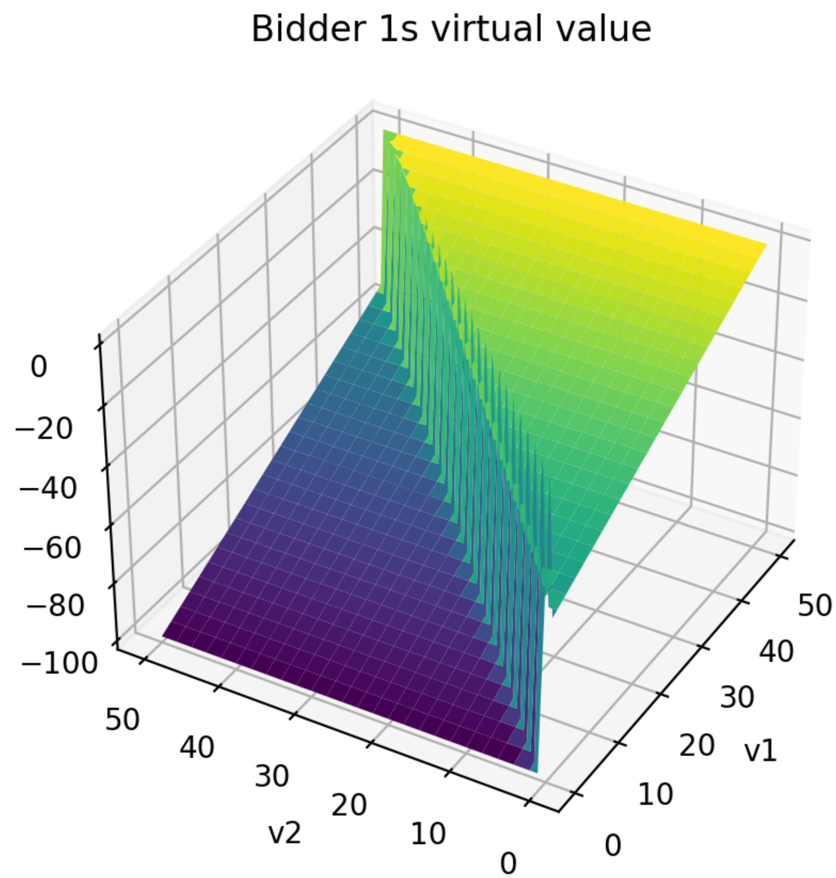
```
# Consider the following value distribution: with probability 1/2, the
# two bidders have the same value, which is uniformly distributed on [0,1].
# With probability 1/2, they are iid uniform on [0,1].
```

```python
# Check visually if this distribution satisfies Chung-Ely regularity.

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

numVals=51
K=range(0,numVals)

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

f={(v1,v2):0.5/(numVals*numVals)+0.5/numVals*(1 if v1==v2 else 0) for v1 in K for v2 in K};

F = np.array([[f[v1,v2] for v1 in K] for v2 in K])

vv1 = np.array([[V[v1]-sum(f[v,v2] for v in K if v > v1)/f[v1,v2] 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, F, cmap='viridis')

ax.set_xlabel('v1')
ax.set_ylabel('v2')
ax.set_title('Joint distribution');
ax.view_init(35,210)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, vv1, cmap='viridis')

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

print('The virtual value is no longer non-decreasing. It decreases when v1==v2.')

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

# Compute the revenue-maximizing ex post IC and IR mechanism, and then
# recompute if we drop all of the ex post constraints except local downward IC
# and IR for the lowest type. Do the two programs coincide? Can we interpret the
# optimal multipliers for the full ex post problem as beliefs? Why or why not?
```

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

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,v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

# Drop non-local IC and IR except at the bottom
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(V[0]*q1[0,v2]-t1[0,v2]>=0 for v2 in K)
ir2 = model.addConstrs(V[0]*q2[v1,0]-t2[v1,0]>=0 for v1 in K)
ic1 = model.addConstrs((V[v1]*(q1[v1,v2]-q1[v1-1,v2])-(t1[v1,v2]-t1[v1-1,v2]))>=0 for v2 in K for v1 in K
ic2 = model.addConstrs((V[v2]*(q2[v1,v2]-q2[v1,v2-1])-(t2[v1,v2]-t2[v1,v2-1]))>=0 for v1 in K for v2 in K

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,v2] for v1 in K for v2 in K),GRB.MAXIMIZE)
model.optimize()

print('The relaxed program has a strictly higher value. This means that for the unrelaxed program, there

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






```
The virtual value is no longer non-decreasing. It decreases when v1==v2.
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 273105 rows, 10404 columns and 1045704 nonzeros
Model fingerprint: 0x434997fc
Coefficient statistics:
  Matrix range     [2e-02, 1e+00]
  Objective range  [2e-04, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
```

```
Presolve removed 102 rows and 5305 columns
Presolve time: 0.66s
Presolved: 10302 rows, 267800 columns, 1040400 nonzeros
Ordering time: 0.10s

Barrier statistics:
 AA' NZ     : 2.588e+05
 Factor NZ  : 8.069e+05 (roughly 60 MB of memory)
 Factor Ops : 3.032e+08 (less than 1 second per iteration)
 Threads    : 4

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   3.25401616e+01  0.00000000e+00  3.43e+00 0.00e+00  1.01e-02     2s
   1   2.30769242e+01  1.25921552e-02  1.38e+00 3.73e-02  4.02e-03     2s
   2   2.18902649e+01  8.00888544e-02  3.28e-01 1.26e-02  1.29e-03     2s
   3   1.31913028e+01  1.24416383e-01  2.77e-02 2.76e-03  2.27e-04     2s
   4   4.08349754e+00  2.63054232e-01  4.14e-03 1.39e-04  3.81e-05     2s
   5   1.64249465e+00  2.77560789e-01  1.32e-03 4.66e-05  1.29e-05     2s
   6   9.94685821e-01  2.88441503e-01  6.48e-04 2.87e-05  6.54e-06     2s
   7   7.91705513e-01  2.97765321e-01  4.40e-04 2.23e-05  4.54e-06     2s
   8   6.43533521e-01  3.08669524e-01  2.83e-04 1.81e-05  3.06e-06     2s
   9   5.08886400e-01  3.25643677e-01  1.21e-04 1.37e-05  1.61e-06     2s
  10   4.84214230e-01  3.41013434e-01  8.33e-05 1.10e-05  1.24e-06     2s
  11   4.67880894e-01  3.66213770e-01  5.31e-05 7.87e-06  8.74e-07     2s
  12   4.56446096e-01  3.79967452e-01  3.08e-05 6.35e-06  6.41e-07     2s
  13   4.52679233e-01  4.00200768e-01  1.47e-05 4.31e-06  4.27e-07     2s
  14   4.51414298e-01  4.19740210e-01  6.85e-06 2.39e-06  2.54e-07     2s
  15   4.50395613e-01  4.43246948e-01  2.37e-06 1.25e-07  5.94e-08     3s
  16   4.49759259e-01  4.47713812e-01  2.99e-07 2.82e-08  1.60e-08     3s
  17   4.49543254e-01  4.49409989e-01  6.38e-09 1.75e-09  1.01e-09     3s
  18   4.49534800e-01  4.49534686e-01  3.61e-12 1.50e-12  8.57e-13     3s
  19   4.49534794e-01  4.49534794e-01  1.62e-13 6.66e-16  2.33e-18     3s

Barrier solved model in 19 iterations and 2.71 seconds (1.35 work units)
Optimal objective 4.49534794e-01

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 7803 rows, 10404 columns and 25704 nonzeros
Model fingerprint: 0xd4c01725
```

```
Coefficient statistics:
  Matrix range     [2e-02, 1e+00]
  Objective range  [2e-04, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 7803 rows and 10404 columns
Presolve time: 0.04s
Presolve: All rows and columns removed

Barrier solved model in 0 iterations and 0.05 seconds (0.00 work units)
Optimal objective 4.60111496e-01
The relaxed program has a strictly higher value. This means that for the unrelaxed program, there are so
me non-local downard IC multipliers and/or other IR constraints that bind. So, the "weight" that player
i assigns to v_j depends on which bid player i is considering deviating to, and we cannot find a single
set of weights that work for all deviations.
```