In [1]:
```
# ECON 289 Problem set 5:
# Maxmin 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]:
```
# Set up and solve the lower bound program for the max guarantee. For this calculation,
# it is convenient to decouple the distance between actions (Delta) and the number of actions (which
# in class we took to be 1/Delta). A value of Delta=0.1 and 100 actions is about right for this
# calculation, but you can play around with it to see how the value changes. Make the value uniformly
# distributed on a grid on [0,1], with 10 grid points.
```

```
# What do you calculate for the lower bound on the guarantee? How does this compare to the guarantee of t
# first price auction for the uniform distribution?

# Try to increase the number of actions and decrease Delta. What happens to the lower bound?

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

Delta=0.1
numActions=100;
numVals=10
K=range(0,numActions)
L=range(0,numVals)
V={l:l/(numVals-1) for l in L}
mu={l:1/numVals for l in L}

model = gp.Model()

model.Params.Method = 2 # Barrier algorithm
model.Params.Crossover = 0 # Disable crossover
model.Params.OutputFlag = 1 # Enable output
model.Params.InfUnbdInfo = 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)
lam=model.addVars(L,lb=-GRB.INFINITY)

# Feasibility constraints
model.addConstrs(q1[k1,k2]+q2[k1,k2]==1 for k1 in K for k2 in K)

# Participation security
model.addConstrs(V[l]*q1[0,k2]-t1[0,k2]>=0 for l in L for k2 in K)
model.addConstrs(V[l]*q2[k1,0]-t2[k1,0]>=0 for l in L for k1 in K)

# Strategic virtual objective constraints
sigma=model.addConstrs(lam[l]<=t1[k1,k2]+t2[k1,k2]
                +V[l]*1/Delta*((0 if k1==numActions-1 else q1[k1+1,k2]-q1[k1,k2])
                               +(0 if k2==numActions-1 else q2[k1,k2+1]-q2[k1,k2]))
                -1/Delta*((0 if k1==numActions-1 else t1[k1+1,k2]-t1[k1,k2])
                               +(0 if k2==numActions-1 else t2[k1,k2+1]-t2[k1,k2]))
                for l in L for k1 in K for k2 in K)
```

```
model.setObjective(sum(mu[l]*lam[l] for l in L),GRB.MAXIMIZE)
model.optimize()

print('The optimal value is about 0.221. This is significantly higher than the guarantee of the first pri

print('With Delta=0.05 and 200 actions, the lower bound increases to 0.226.')
# ----------------------------------------
```

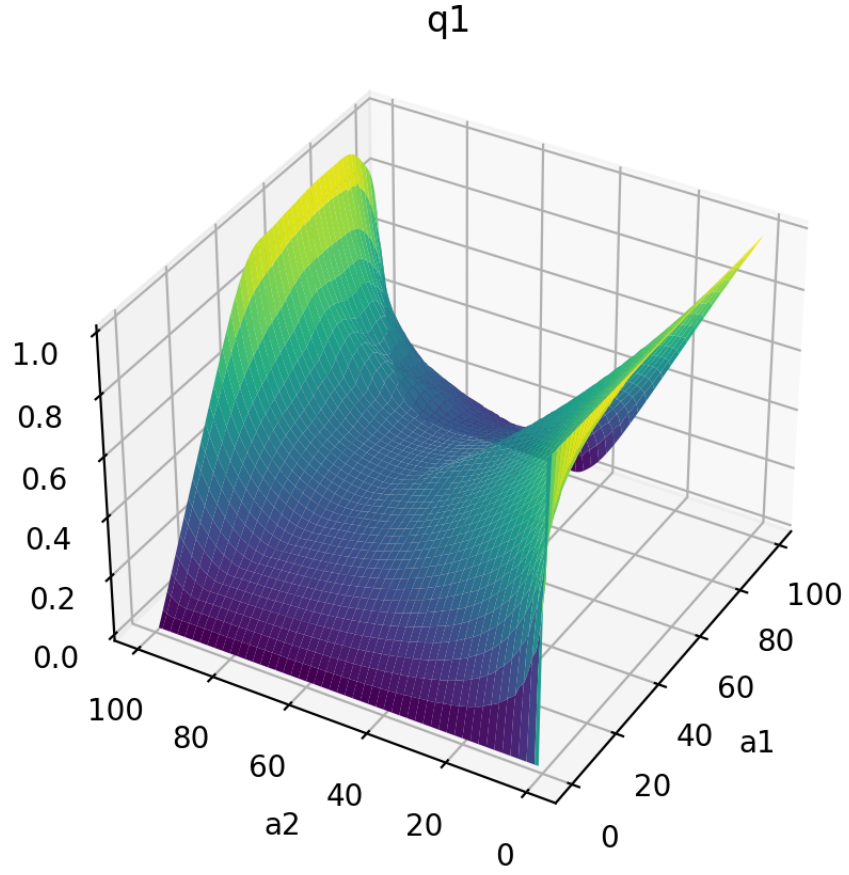
```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-03-14
Set parameter Method to value 2
Set parameter Crossover to value 0
Set parameter InfUnbdInfo to value 1
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 112000 rows, 40010 columns and 878200 nonzeros
Model fingerprint: 0xa0466911
Coefficient statistics:
  Matrix range     [1e-01, 1e+01]
  Objective range  [1e-01, 1e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 20198 rows and 2199 columns
Presolve time: 0.72s
Presolved: 19812 rows, 109801 columns, 488003 nonzeros
Ordering time: 0.02s

Barrier statistics:
 AA' NZ     : 2.483e+05
 Factor NZ  : 8.319e+05 (roughly 60 MB of memory)
 Factor Ops : 4.801e+07 (less than 1 second per iteration)
 Threads    : 4

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.29197219e+03  0.00000000e+00  5.21e+02 0.00e+00  1.07e-01     1s
   1   5.25888404e+02 -2.61334246e+00  7.97e+01 1.19e-13  1.76e-02     1s
   2   1.06604912e+01 -2.45945425e+00  1.50e+00 2.82e-11  3.58e-04     1s
   3   1.56055680e+00 -8.69769191e-01  1.66e-01 1.71e-10  4.77e-05     1s
   4   7.02649708e-01 -8.49055727e-02  5.59e-02 3.57e-10  1.71e-05     1s
   5   4.64453285e-01  6.99257458e-02  2.58e-02 3.09e-10  7.91e-06     1s
   6   3.14635108e-01  1.35462987e-01  8.41e-03 2.69e-10  2.92e-06     1s
```

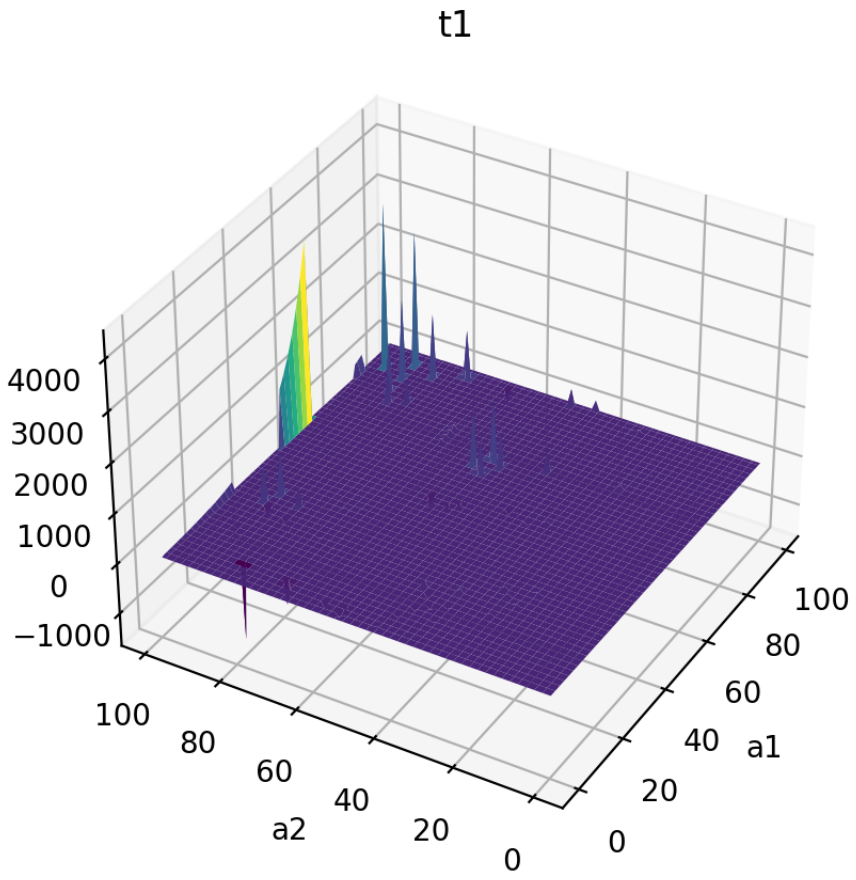
```
  7   2.73400880e-01  1.69610054e-01  4.10e-03 1.90e-10  1.52e-06     1s
  8   2.54781021e-01  1.91601216e-01  2.39e-03 1.68e-10  8.84e-07     1s
  9   2.46211978e-01  2.02460057e-01  1.68e-03 1.92e-10  5.99e-07     2s
 10   2.38093479e-01  2.08892555e-01  1.04e-03 1.31e-10  3.81e-07     2s
 11   2.30909479e-01  2.13499568e-01  5.24e-04 1.00e-10  2.10e-07     2s
 12   2.26830247e-01  2.17670007e-01  2.67e-04 5.21e-11  1.06e-07     2s
 13   2.25183856e-01  2.18774337e-01  1.79e-04 4.54e-11  7.20e-08     2s
 14   2.24255278e-01  2.19451868e-01  1.32e-04 3.32e-11  5.30e-08     2s
 15   2.23064306e-01  2.20261058e-01  7.69e-05 5.94e-11  3.00e-08     2s
 16   2.22309865e-01  2.20579365e-01  4.36e-05 3.91e-11  1.79e-08     2s
 17   2.21848305e-01  2.20788041e-01  2.60e-05 4.48e-11  1.07e-08     2s
 18   2.21464014e-01  2.20862442e-01  1.21e-05 2.74e-11  5.85e-09     2s
 19   2.21437040e-01  2.20949536e-01  1.12e-05 3.10e-11  4.75e-09     2s
 20   2.21281513e-01  2.21017925e-01  6.35e-06 3.16e-11  2.51e-09     2s
 21   2.21174015e-01  2.21033769e-01  3.19e-06 3.55e-11  1.32e-09     2s
 22   2.21167991e-01  2.21039153e-01  3.04e-06 2.85e-11  1.21e-09     2s
 23   2.21142872e-01  2.21040329e-01  2.33e-06 2.28e-11  9.55e-10     2s
 24   2.21123495e-01  2.21044659e-01  1.80e-06 2.31e-11  7.30e-10     2s
 25   2.21096871e-01  2.21046444e-01  1.11e-06 2.88e-11  4.63e-10     2s
 26   2.21068042e-01  2.21049799e-01  3.85e-07 3.13e-11  1.65e-10     2s
 27   2.21055444e-01  2.21051196e-01  8.09e-08 2.19e-11  3.78e-11     2s
 28   2.21052535e-01  2.21051462e-01  1.90e-08 4.29e-11  9.49e-12     3s
 29   2.21051750e-01  2.21051509e-01  3.85e-09 3.47e-11  2.12e-12     3s
 30   2.21051547e-01  2.21051535e-01  1.80e-10 2.64e-11  1.08e-13     3s
 31   2.21051536e-01  2.21051536e-01  2.44e-12 4.31e-11  1.57e-15     3s

Barrier solved model in 31 iterations and 2.65 seconds (1.40 work units)
Optimal objective 2.21051536e-01
```

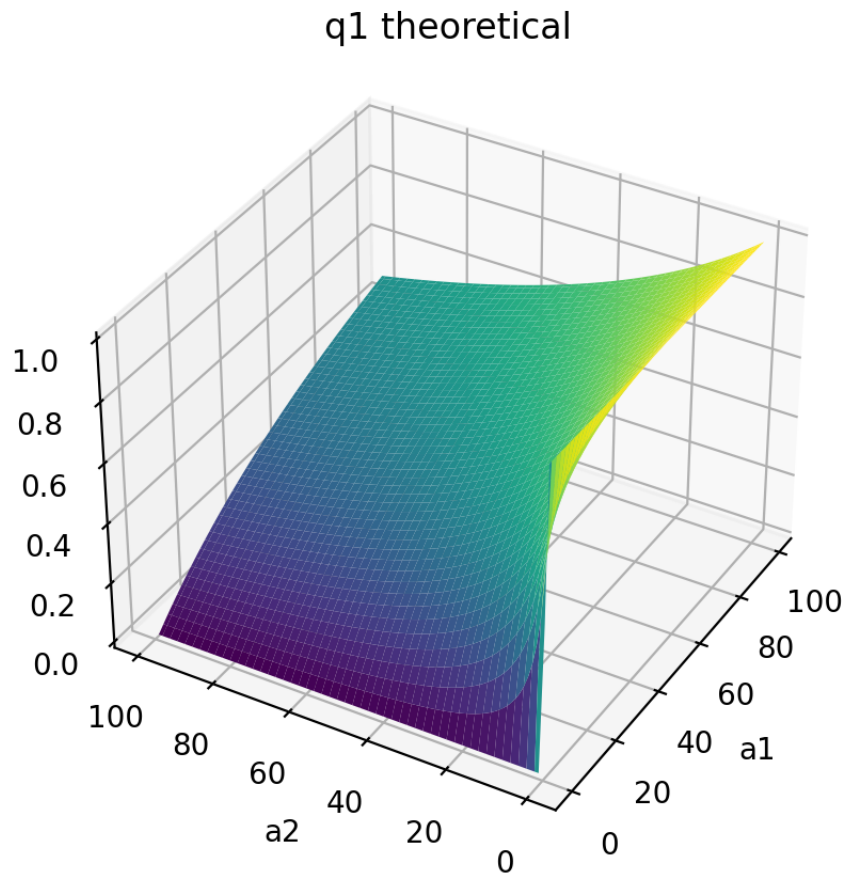
The optimal value is about 0.221. This is significantly higher than the guarantee of the first price auction, which is 1/6~0.167 for the uniform value and two bidders.
With Delta=0.05 and 200 actions, the lower bound increases to 0.226.

In [4]:
```
# Plot the optimal allocation and transfers. Also plot the proportional allocation.
# How does this compare to the numerical solution? Can you make sense of the transfers?
# What can you conclude about the multiplicity of transfers?

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

Q1 = np.array([[q1[k1,k2].X for k1 in K] for k2 in K])
T1 = np.array([[t1[k1,k2].X for k1 in K] for k2 in K])
T2 = np.array([[t2[k1,k2].X for k1 in K] for k2 in K])
T=T1+T2
```

```
Qprop = np.array([[(k1/(k1+k2) if (k1+k2>0) else 0.5) for k1 in K] for k2 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('a1')
ax.set_ylabel('a2')
ax.set_title('q1');
ax.view_init(35,210)


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

ax.set_xlabel('a1')
ax.set_ylabel('a2')
ax.set_title('t1');
ax.view_init(35,210)

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

ax.set_xlabel('a1')
ax.set_ylabel('a2')
ax.set_title('q1 theoretical');
ax.view_init(35,210)

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


q1
1.0
0.8
0.6
0.4
0.2
0.0
100
80
60
40
20
0
a2
100
80
60
40
20
0
a1


t1
4000
3000
2000
1000
0
−1000
100
80
60
40
20
0
a2
100
80
60
40
20
0
a1



In [5]:
```
# Now compute our upper bound on the min potential. Use the same values of Delta=0.1 and 100 types.
# Be careful in how you define the discrete upward derivative for the informational virtual objective!

# What do you compute for the upper bound on the min potential? How does this compare to the lower bound
# on the guarantee?

# What happens if you increase the number of types and decrease Delta? Do you think we are on the right
# track with the local multipliers?

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


```

```
Delta=0.1
numActions=100;
numVals=10
K=range(0,numActions)
L=range(0,numVals)
V={l:l/(numVals-1) for l in L}
mu={l:1/numVals for l in L}

model = gp.Model()

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

pi=model.addVars(L,K,K)
gamma=model.addVars(K,K,lb=-GRB.INFINITY)

# Probability constraints
model.addConstrs(sum(pi[l,k1,k2] for k1 in K for k2 in K) == mu[l] for l in L)

# Informational virtual objective constraints
t1=model.addConstrs(sum(pi[l,k1,k2]
                        +((0 if k1==numActions-1 else (1 if k1==numActions-2 else 1/Delta)*pi[l,k1+1,k2])
                          -(1 if k1==numActions-1 else 1/Delta)*pi[l,k1,k2]) for l in L)==0
                    for k1 in K for k2 in K)
t2=model.addConstrs(sum(pi[l,k1,k2]
                        +((0 if k2==numActions-1 else (1 if k2==numActions-2 else 1/Delta)*pi[l,k1,k2+1])
                          -(1 if k2==numActions-1 else 1/Delta)*pi[l,k1,k2]) for l in L)==0
                    for k1 in K for k2 in K)
q1=model.addConstrs(gamma[k1,k2]>=
                    -sum(V[l]*((0 if k1==numActions-1 else (1 if k1==numActions-2 else 1/Delta)*pi[l,k1+1
                          -(1 if k1==numActions-1 else 1/Delta)*pi[l,k1,k2]) for l in L)
                    for k1 in K for k2 in K)
q2=model.addConstrs(gamma[k1,k2]>=
                    -sum(V[l]*((0 if k2==numActions-1 else (1 if k2==numActions-2 else 1/Delta)*pi[l,k1,k
                          -(1 if k2==numActions-1 else 1/Delta)*pi[l,k1,k2]) for l in L)
                    for k1 in K for k2 in K)

model.setObjective(sum(gamma[k1,k2] for k1 in K for k2 in K),GRB.MINIMIZE)
model.optimize()

print('The upper bound on min potential is ~0.244. This is about an 8 percent gap.')
```

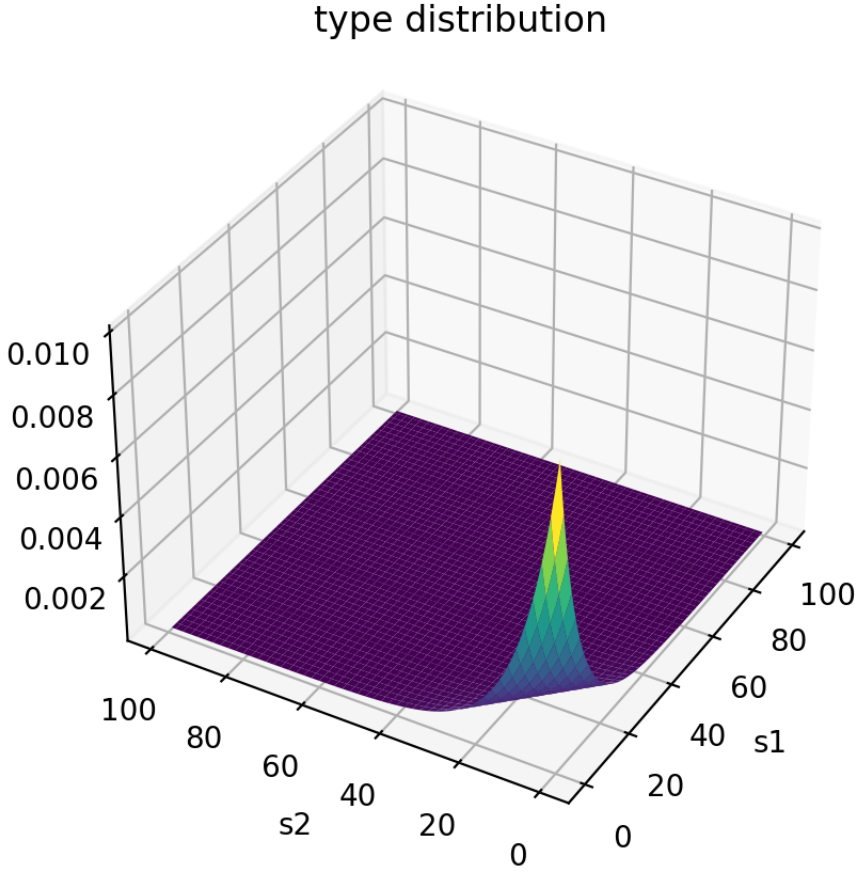
```python
print('The upper bound decreases to 0.238 with Delta=0.05 and 200 types. The gap reduces to 5%.')

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

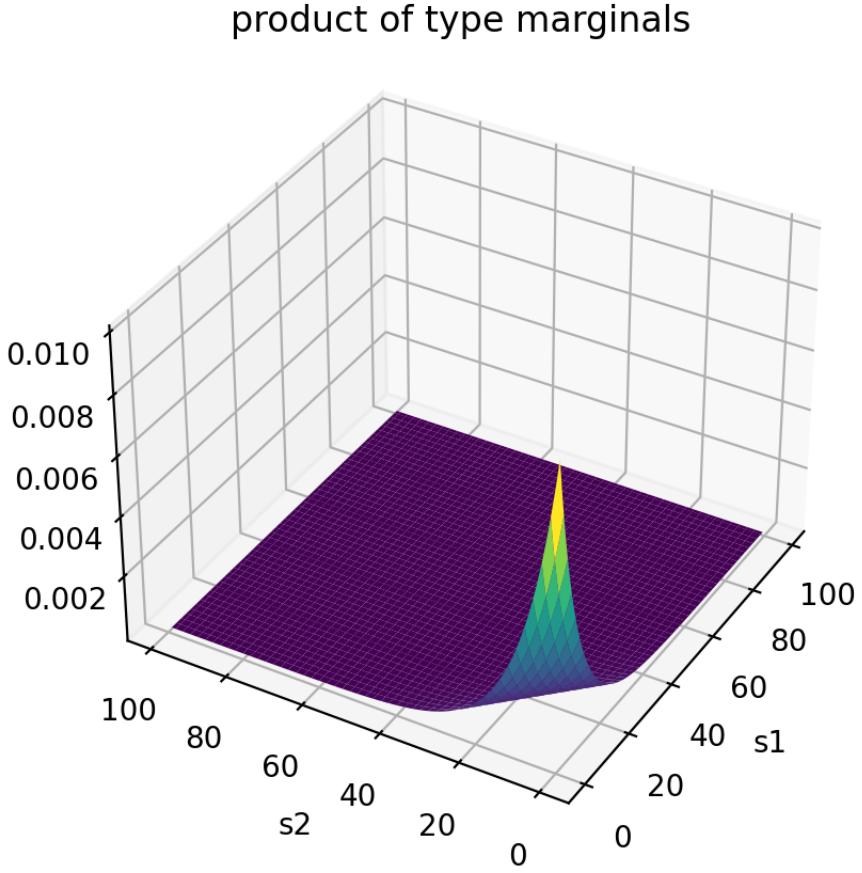
```
Set parameter Method to value 2
Set parameter Crossover to value 0
Set parameter InfUnbdInfo to value 1
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 40010 rows, 110000 columns and 874200 nonzeros
Model fingerprint: 0x60eceda1
Coefficient statistics:
  Matrix range     [1e-01, 1e+01]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e-01, 1e-01]
Presolve removed 20002 rows and 1 columns
Presolve time: 0.61s
Presolved: 20008 rows, 109999 columns, 489961 nonzeros
Ordering time: 0.02s

Barrier statistics:
 AA' NZ     : 2.508e+05
 Factor NZ  : 8.704e+05 (roughly 60 MB of memory)
 Factor Ops : 5.454e+07 (less than 1 second per iteration)
 Threads    : 4

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.21360252e+03  0.00000000e+00  4.92e+02 0.00e+00  7.99e-02     1s
   1   6.07435584e+02 -1.77674931e+00  1.04e+02 2.22e-15  1.77e-02     1s
   2   1.53231120e+01 -1.67934655e+00  2.49e+00 7.09e-11  4.42e-04     1s
   3   2.90496417e+00 -8.50891617e-01  4.07e-01 8.50e-10  8.07e-05     1s
   4   1.07539890e+00 -1.65928445e-01  1.29e-01 7.38e-10  2.62e-05     1s
   5   7.50424927e-01  8.87361850e-02  7.85e-02 7.21e-10  1.59e-05     1s
   6   4.93207412e-01  1.24757472e-01  3.58e-02 1.01e-09  7.74e-06     1s
   7   3.44181050e-01  1.98933092e-01  1.25e-02 7.67e-10  2.79e-06     1s
   8   2.77477604e-01  2.20944016e-01  3.25e-03 5.51e-10  8.63e-07     2s
   9   2.55313630e-01  2.38636926e-01  6.87e-04 4.49e-10  2.15e-07     2s
  10   2.49854006e-01  2.42503116e-01  2.93e-04 2.00e-10  9.00e-08     2s
  11   2.47352326e-01  2.43466012e-01  1.43e-04 1.77e-10  4.52e-08     2s
  12   2.45911653e-01  2.43985892e-01  6.24e-05 1.55e-10  2.11e-08     2s
```

```
  13   2.45064569e-01  2.44260293e-01  2.06e-05 1.96e-10  8.24e-09     2s
  14   2.44747937e-01  2.44448296e-01  7.61e-06 1.16e-10  2.97e-09     2s
  15   2.44642288e-01  2.44470117e-01  3.92e-06 1.38e-10  1.67e-09     2s
  16   2.44587603e-01  2.44490460e-01  2.24e-06 2.11e-10  9.29e-10     2s
  17   2.44555545e-01  2.44500510e-01  1.30e-06 1.50e-10  5.21e-10     2s
  18   2.44545164e-01  2.44502863e-01  1.01e-06 1.95e-10  3.99e-10     2s
  19   2.44543296e-01  2.44503926e-01  9.59e-07 1.55e-10  3.71e-10     2s
  20   2.44525535e-01  2.44504719e-01  4.59e-07 1.36e-10  1.93e-10     2s
  21   2.44518287e-01  2.44506561e-01  2.65e-07 1.44e-10  1.08e-10     2s
  22   2.44512548e-01  2.44507221e-01  1.18e-07 2.49e-10  4.86e-11     2s
  23   2.44510107e-01  2.44507428e-01  5.87e-08 1.44e-10  2.43e-11     2s
  24   2.44508663e-01  2.44507512e-01  2.46e-08 2.36e-10  1.04e-11     2s
  25   2.44508050e-01  2.44507537e-01  1.05e-08 1.98e-10  4.61e-12     2s
  26   2.44507839e-01  2.44507546e-01  6.00e-09 1.81e-10  2.63e-12     2s
  27   2.44507634e-01  2.44507551e-01  1.69e-09 2.84e-10  7.28e-13     2s
  28   2.44507555e-01  2.44507551e-01  5.75e-11 1.42e-10  2.80e-14     3s

Barrier solved model in 28 iterations and 2.52 seconds (1.29 work units)
Optimal objective 2.44507555e-01

The upper bound on min potential is ~0.244. This is about an 8 percent gap.
The upper bound decreases to 0.238 with Delta=0.05 and 200 types. The gap reduces to 5%.
```

In [6]:
```
# Plot the marginal distribution of types. What does the density appear to be a function of?
# Compute the marginal of each bidder's type and then compute and plot the product of the marginals.
# What do you conclude about the correlation structure between the types?

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

rho = np.array([[sum(pi[l,k1,k2].X for l in L) for k1 in K] for k2 in K])

marg1 = np.array([sum(rho[k1,k2] for k2 in K) for k1 in K])
marg2 = np.array([sum(rho[k1,k2] for k1 in K) for k2 in K])

margProd = np.array([[marg1[k1]*marg2[k2] for k1 in K] for k2 in K])

print(f'Total prob is {sum(rho[k1,k2] for k1 in K for k2 in K)}')

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

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

ax.set_xlabel('s1')
ax.set_ylabel('s2')
ax.set_title('type distribution');
ax.view_init(35,210)

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

ax.set_xlabel('s1')
ax.set_ylabel('s2')
ax.set_title('product of type marginals');
ax.view_init(35,210)

print('The types are obviously independent, and the density is a function of the sum of the types, so it

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

```
Total prob is 1.000000000471609
```

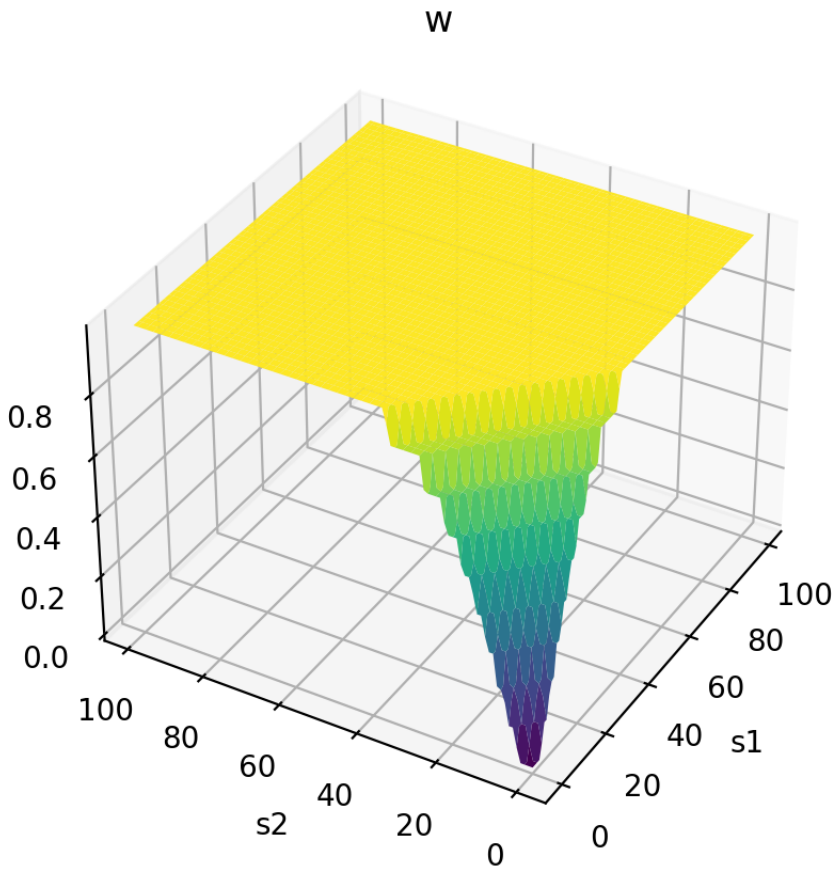

type distribution
0.010
0.008
0.006
0.004
0.002
100
80
60
40
20
0
s2
100
80
60
40
20
0
s1



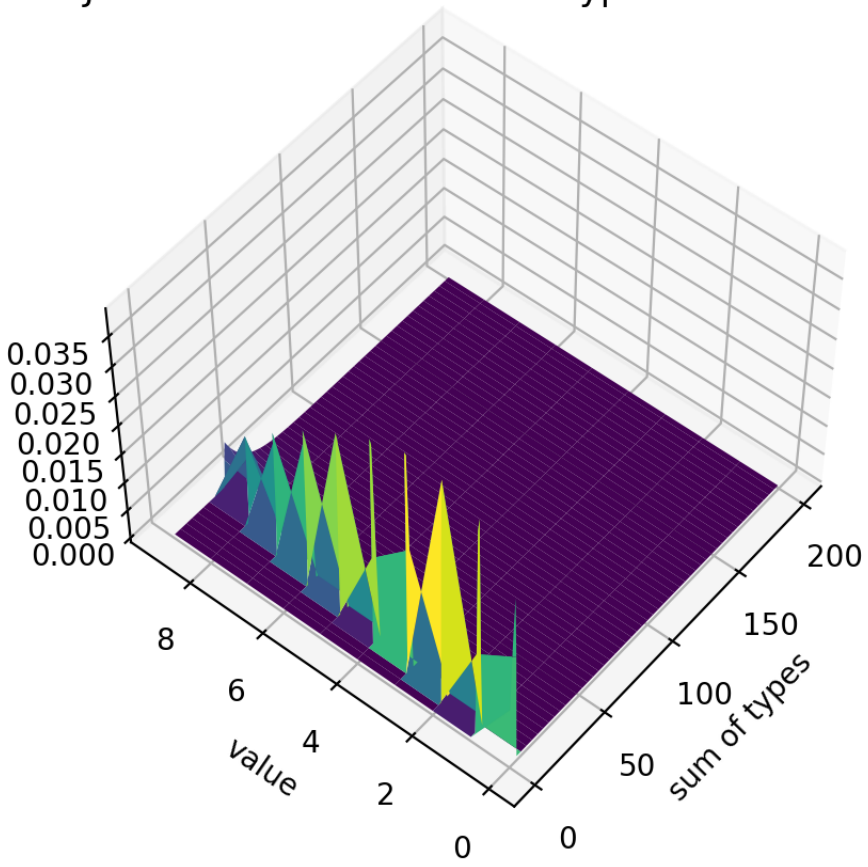
The types are obviously independent, and the density is a function of the sum of the types, so it must b
e iid exponential.

In [7]:
```
# Calculate and plot the interim expectation of the value given the signals. What structure does it have?

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

w= np.array([[sum(V[l]*pi[l,k1,k2].X for l in L)/rho[k1,k2] for k1 in K] for k2 in K])

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

fig = plt.figure()
```

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

ax.set_xlabel('s1')
ax.set_ylabel('s2')
ax.set_title('w');
ax.view_init(35,210)

print('The value is non-decreasing and is a function of just the sum of the signals.')

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

# Finally, plot the joint distribution of the sum of the types and the value. What is the
# statistically relationship between these two variables?

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

M = range(0,2*numActions)

sumValMarg = np.array([[sum(pi[v,k1,k2].X for k1 in K for k2 in K if k1+k2==m) for m in M] for v in L])

X, Y = np.meshgrid(M,L)

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

ax.set_xlabel('sum of types')
ax.set_ylabel('value')
ax.set_title('Joint distribution of sum of types and value');
ax.view_init(35,210)

print('The value and the sum of the types are perfectly comonotonic.')

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



The value is non-decreasing and is a function of just the sum of the signals.



The value and the sum of the types are perfectly comonotonic.