Data-Driven Bid Estimates from Different First-Price Auction Models

Ya Shi (Andrew) Zhang
NetID: ysz211
yashi.zhang@nyu.edu

1 Introduction

In economics, auctions can be used to allocate resources efficiently by allowing the free market to determine the price of a good or service. Game theorists define an auction as a strategic game in which players, or agents, bid on an object or resource. The player who bids the highest price wins the auction and gets to keep the object or resource. From this game-theoretic perspective, we can analyze different bidding strategies under different kinds of auctions. Each agent is incentivised to maximize their own utility. We wish to study, model, and therefore predict the behavior of individual agents in these auctions. There are several types of auctions that can be studied this way, some common types are:

1. First-price sealed-bid auction
2. Second-price sealed-bid auction
3. English auction
4. Dutch auction

For our purposes, we will be exclusively delving into first-price sealed-bid auctions (FPSBA), also known as blind auctions. In FPSBA, all bidders simultaneously submit sealed bids (such that no bidder knows the bid of anyone else), and the highest bidder pays the price that was submitted [3]. For a FPSBA with N bidders, bidder i can be characterized by their private valuation of the item v_i (how much the item is worth in their minds). Note that these valuation are only known to themselves (likewise, they cannot know the valuation of any other player).

Our goal is to analyze this game under two different assumptions (risk neutral, risk averse) using a Bayes-Nash lens, propose different models that capture our analysis, and match them to real, experimental data obtained from [2] in order to decide on a model of best fit. This project recreates, using Python, the computational analysis used in [1]. We assume validity of the results in Bajari and Hortaçsu as they have been externally validated by a journal committee and accepted. We will first look at it from the bidder's perspective, where we model bidding given a bidder's private valuation. Then, we move onto the useful case, where we estimate the ground truth value of an item based on the collected bids using our two risk assumptions. As we are modeling with data, we can validate our method by simply observing if our results match the experimental data.

2 Data & Equations

The experimental data was provided by [2]. There were three experiments (denoted 3, 4, and 5) ran with six different subjects in each run. In the experiment, bidder i, $1 \leq i \leq 6$, was assigned i.i.d. valuations $v_i \sim \mathcal{U}[0, 30]$ and asked to bid. If they won the auction, they were paid their valuation minus their bid as incentive. Each subject participated in 28 auctions and, in the analysis of both [2] and [1], the first 5 auctions were excluded for the sake of higher quality data. The number of bidders in the auction would be determined at random ($N = 3$ and $N = 6$ w.p. 1/2 each), subjects submitted two bids, one for each case. A coin toss would determine afterwards the number of bidders and therefore which bid to use for the blind auction. After each auction round, everyone's valuations and corresponding bids were made public. We will approach these models with a Bayesian Mindset, and so we assume a prior uniform distribution over the support, which is $[0, 30]$, for each bidder.

2.1 Bidding

To model how bidder's bid given their own private valuations of the item, we assume that they are risk neutral, and so the Bayes-Nash equilibrium bidding strategy in this structure is given by

$$b(v_i) = \frac{N-1}{N} v_i. \tag{1}$$

The Bayes-Nash equilibrium bidding strategy in the risk averse case is given by

$$b(v_i) = \frac{N-1}{N-\theta} v_i, \tag{2}$$

where $\theta > 0$ is the risk averse constant and will be discussed in more detail in 2.3.

Knowing this, we can then posit a cumulative distribution, which is parameterized by the number of bidders N and, empirically, by the experiment number $e \in \{3, 4, 5\}$, denoted $Q(b; N, e)$. We can estimate this function and denote it $\hat{Q}(b; N, e)$. Now, we define the expected profit given bids b to be

$$\pi(b, v; N, e) = (v - b) \cdot Q(b; N, e)^{N-1} \tag{3}$$

$$\hat{\pi}(b, v; N, e) = (v - b) \cdot \hat{Q}(b; N, e)^{N-1}. \tag{4}$$

This, in turn, gives us an optimization error

$$\omega(b, v; N, e) = \left[\max_{b'} (v - b') \cdot \hat{Q}(b'; N, e)^{N-1} \right] - \hat{\pi}(b, v; N, e). \tag{5}$$

2.2 Risk-Neutral Valuation

Now, given bids, we want to estimate the true value of the item. In an auction with N symmetric bidders, we posit that each bidder's valuation v_i is i.i.d. with cumulative distribution function $F(v)$ and probability density function $f(v)$. Bidder's simultaneously submit private bids in FPSBA, so the bidder's utility is $v_i - b_i$ if bidder i's bid is the highest and is 0 otherwise.
If we assume that the equilibrium function is strictly increasing and differentiable, as is the case with 1,

its inverse $\phi(b)$ exists and will also be strictly increasing and differentiable. Now, we can model bidder i's expected profit from bidding b_i as

$$\pi_i(b_i; v_i) = (v_i - b_i) \cdot F(\phi(b_i))^{N-1}, \tag{6}$$

which is the bidder's expected utility times the probability that bidder i wins the auction. We can maximize π_i with respect to b_i by differentiating and finding the roots of the equation. We end up getting

$$v_i = b_i + \frac{F(\phi(b_i))}{f(\phi(b_i))\phi'(b_i)(N-1)}.$$

We can simplify by letting $G(b) = F(\phi(b))$ and $g(b) = f(\phi(b))\phi'(b)$, which are known as cdf and pdf of the bids, respectively. If we estimate $G(b)$ and $g(b)$, we have an estimate of v_i,

$$\hat{v}_i = b_i + \frac{\hat{G}(b_i)}{\hat{g}(b_i)(N-1)}. \tag{7}$$

2.3 Risk-Averse Valuation

We need a new utility function that captures risk-aversion, and will use a constant relative risk aversion (CRRA) utility function $U(x) = x^\theta$, $\theta \in [0, 1]$. Here, we can see that $\theta = 1$ would correspond to risk-neutrality, and that lowering θ would increase risk aversion. We use the same derivation as above, except replacing $(v_i - b_i)$ with $(v_i - b_i)^\theta$. Now, we have

$$\hat{v}_i = b_i + \theta \cdot \frac{\hat{G}(b_i)}{\hat{g}(b_i)(N-1)}, \tag{8}$$

which makes sense as setting $\theta = 1$ reduces the above to 7.

3 Numerical Method

Now that we have modeled both risk-averse and risk-neutral cases, we need to estimate some of the functions and parameters.

3.1 Risk-Neutral

In this simpler case, we only need to estimate $g(b)$ ($G(b)$ follows). To do so, we apply the method of kernel density estimation, a non-parametric method to estimate the pdf of some random variable using kernels and data. We choose the kernel to be a normal density function $N(x)$ (Gaussian with mean 0 and variance 1) and use Silvermann's rule of thumb for the choice of bandwidth $h = 0.9 \cdot \min\{\hat{\sigma}, \frac{IQR}{1.34}\} n^{-1/5}$, where $\hat{\sigma}$ is the sample standard deviation, n is the number of samples, and IQR is the interquartile range. The distribution is then estimated as

$$\hat{g}_h(b) = \frac{1}{nh} \sum_{i=1}^{n} N(\frac{b - b_i}{h}). \tag{9}$$

Note that b_i above refers to the ith bid from the dataset, and not the ith bidder's bid.

3.2 Risk-Averse

We will still estimate $G(b), g(b)$ as above. However, we must also estimate θ. This can be done easily using a least squares estimate to solve a system of equations. We first compute integer percentiles of the bid data, $\alpha \in \{0, \ldots, 100\}$, and compute the cdf and pdf estimates for both $N = 3$ and $N = 6$, denoting them $G(b_\alpha^{(3)})$ and $G(b_\alpha^{(6)})$ respectively (similar for $g(\cdot)$). The reason why we use integer percentiles rather than each data point for the dimension of the system of equations is so that both $N = 3$ and $N = 6$ systems of equations matrices are compatible with each other for matrix arithmetic. Now, we have two systems of 101 equations, one for $N = 3$ and one for $N = 6$:

$$v_\alpha = b_\alpha^{(3)} + \theta \cdot \frac{G(b_\alpha^{(3);3})}{2g(b_\alpha^{(3)}; 3)}$$
$$v_\alpha = b_\alpha^{(6)} + \theta \cdot \frac{G(b_\alpha^{(6);6})}{2g(b_\alpha^{(6)}; 6)}.$$

Now, we can subtract them from each other and get the following form:

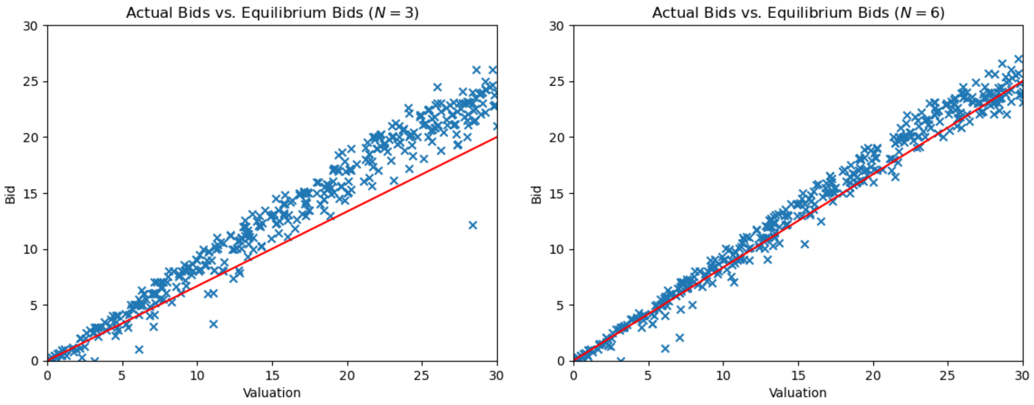
$$b_\alpha^{(3)} - b_\alpha^{(6)} = \theta \cdot \left[\frac{G(b_\alpha^{(3);3})}{2g(b_\alpha^{(3)}; 3)} - \frac{G(b_\alpha^{(6);6})}{2g(b_\alpha^{(6)}; 6)} \right],$$

which is now in the form to be solved for θ.
Note that we later experiment with using only percentiles $\alpha \in \{5, \ldots, 95\}$ and $\alpha \in \{25, \ldots, 75\}$ under the belief that bids are centered around the 50th percentile and edge percentile values are inaccurate.

4 Results, Validation, and Discussion

4.1 Bidding

First, we want to see how close our theoretical risk-neutral equilibrium bidding is to actual experimental data. We do see linear behavior in the experimental bidding. However, in the $N = 3$ case, the equi-



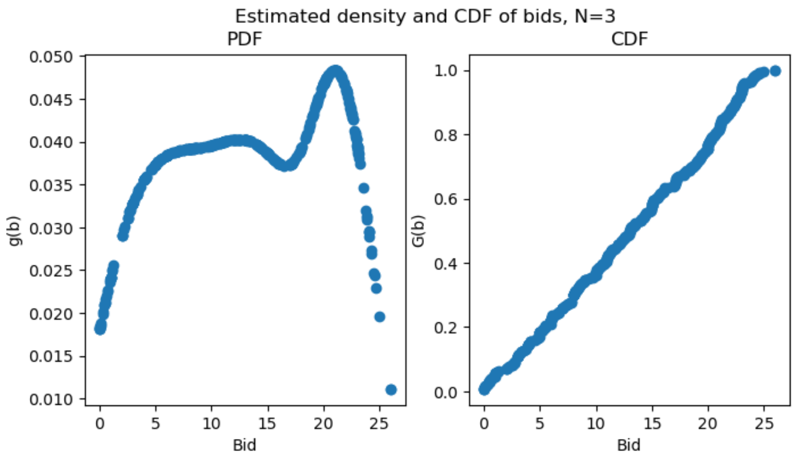
librium bid is lower than the experimental data. This could possibly be explained as the experiment is non-standard in the sense that subjects are asked to submit both bids for $N = 3$ bidders and $N = 6$ bidders at the same time, leading subjects' bid behavior to be similar.

Aside from the bid-distribution being different empirically, we do see non-negligible differences in the statistics of experimental bidding behavior when subjects are bidding in a 3-player auction versus a 6-player:

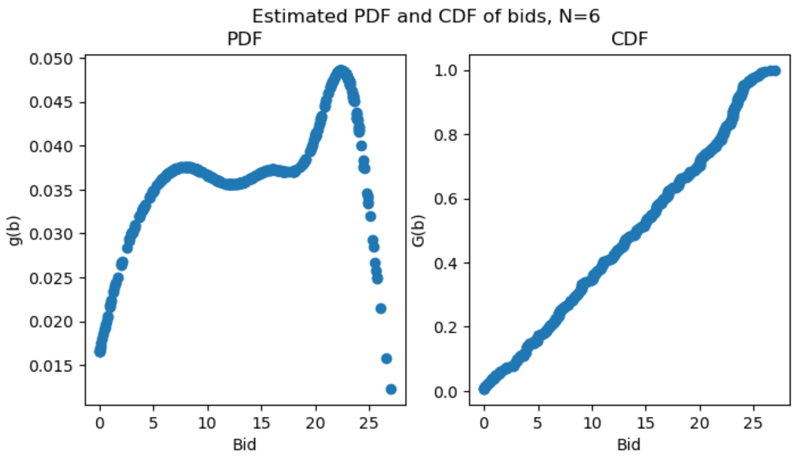
Variable	Mean	Std Dev	25%tile	50%tile	75%tile
Observed Nash Bid:					
N=3	2.44	1.86	1.06	2.46	3.83
N=6	0.65	1.05	0.03	0.56	1.41
Expected Profit:					
N=3	1.32	1.59	0.08	0.62	1.99
N=6	0.57	1.02	0.00	0.04	0.55
Optimization Error:					
N=3	0.34	0.44	0.02	0.14	0.54
N=6	0.11	0.25	0.00	0.01	0.10

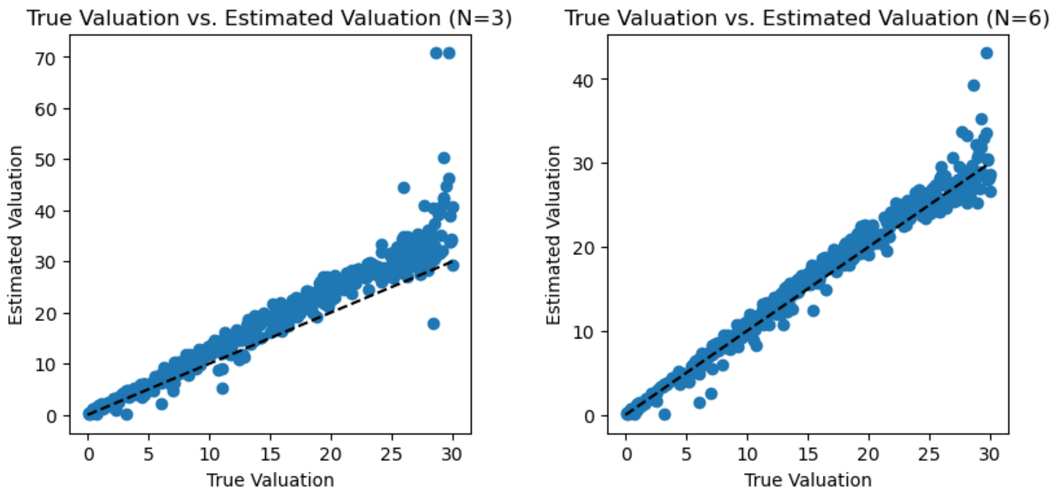
4.2 Risk-Neutral Valuation-Prediction Model

First, let us plot the kernel density estimated distribution of bids, one for $N = 3$ and one for $N = 6$:



Although there are small differences around $b \in [10, 17]$, there are nothing statistically significant that can be said on the differences between the two distributions. Now, using the estimated distributions and densities, we can produce estimated valuations and compare them with true valuations (note that





deviation from the dashed line indicates inaccuracy of our prediction): Our model is good at estimating the true valuation for high amount of auction participants, but overestimates the true value for small auctions. This, again, could potentially be attributed to a flaw in the experiment procedures as outlined in 4.1. We have also computed the L_1 and L_2 distances from these predictions to the ground truth, one with $N = 3$ and one with $N = 6$:

Case	L_1	L_2
$N = 3$	3.86	5.49
$N = 6$	1.15	1.68

4.3 Risk-Averse Valuation-Prediction Model

Using the method outlined in 3, we compute three different values for θ: $\alpha \in \{0, \ldots, 100\}$, $\alpha \in \{5, \ldots, 95\}$, and $\alpha \in \{25, \ldots, 75\}$. Now, for each choice of 'trimming', we will use to estimate the true valuation,

$\theta_{\{0,\ldots,100\}}$	$\theta_{\{5,\ldots,95\}}$	$\theta_{\{25,\ldots,75\}}$
0.1142	0.1136	0.1167

plot it, and display the L_2 and L_1 distances for the $N = 3$ and $N = 6$ cases.

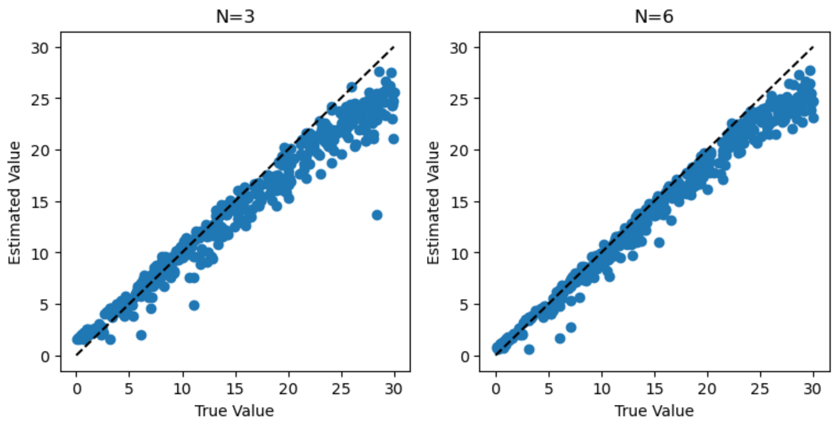


Figure 3: $\alpha \in \{0, \ldots, 100\}$

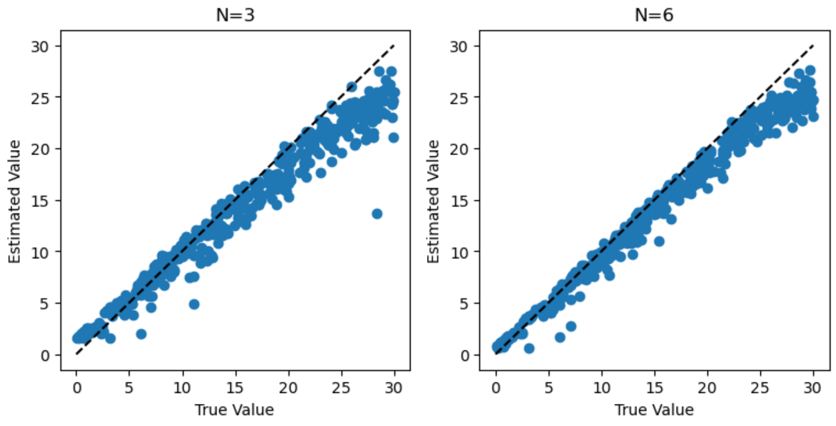


Figure 4: $\alpha \in \{5, \ldots, 95\}$

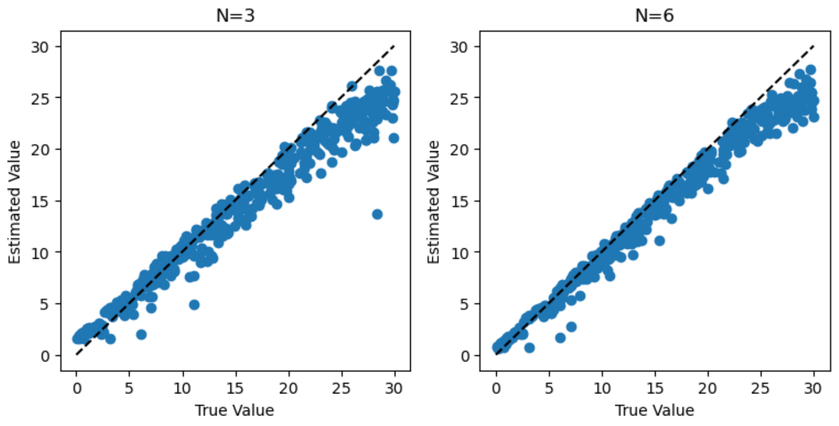


Figure 5: $\alpha \in \{25, \ldots, 75\}$

Cases	L_1	L_2
$\alpha \in \{0, \ldots, 100\}$:		
N=3	1.832	2.492
N=6	1.493	2.051
$\alpha \in \{5, \ldots, 95\}$:		
N=3	1.834	2.496
N=6	1.495	2.053
$\alpha \in \{25, \ldots, 75\}$:		
N=3	1.819	2.476
N=6	1.485	2.042

We see here that varying the size of the percentile set does not significantly impact the quality of valuation estimations, though the case where $\alpha \in \{25, \ldots, 75\}$ does perform the best with respect to the L_2 norm.

5 Summary and Conclusions

We have studied, computed results, and validated the bid-estimating and valuation-estimating Bayes-Nash models. Surprisingly, our experimental data has indicated a better fit (in the L_1 and L_2 sense) for the risk-neutral model. I believe this is because of the percentile method that we used to equalize the dimensions of the systems of equations matrix. Otherwise, the risk-neutral bid estimating function in 2.1 performed well.

References

[1] Patrick Bajari and Ali Hortaçsu. Are structural estimates of auction models reasonable? evidence from experimental data. *Journal of Political Economy*, 113(4):703–741, 2005.

[2] Douglas Dyer, John Kagel, and Dan Levin. Resolving uncertainty about the number of bidders in independent private-value auctions: An experimental analysis. *RAND Journal of Economics*, 20(2):268–279, 1989.

[3] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

A Code

The code (Jupyter Notebook) and data can be found on GitHub at `https://github.com/yashizhang/blindauction`. A raw PDF of the code has been included below.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.stats import norm
```

Data-Driven Bid and Valuation Estimates from Different First-Price Auction Models

Author: Ya Shi Zhang | Email: yashi.zhang@nyu.edu

In [ ]:
```
data = pd.read_csv('auctiondata.csv')
data.head()
```

Out[ ]:

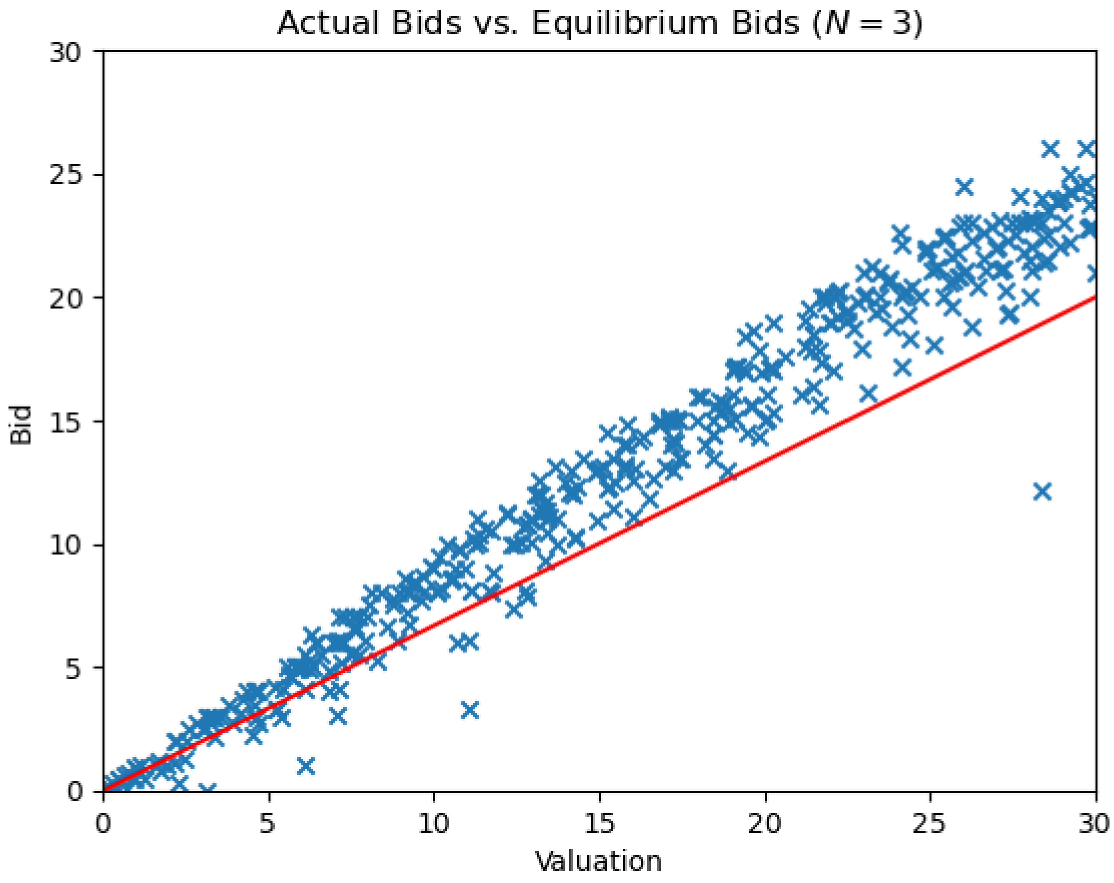
	Experiment	Period	Subject	Value	BidC3_o	Unnamed: 5	BidNC	Market	BidC3	BidC6	Ncont	Si
0	3	5	1	9.21	8.00	.	.	1	8.00	8.50	0	
1	3	5	2	4.38	4.00	.	.	0	4.00	4.25	0	
2	3	5	3	28.05	21.05	.	.	0	21.05	22.05	0	
3	3	5	4	26.44	20.44	.	.	1	20.44	21.44	0	
4	3	5	5	21.49	18.49	.	.	0	18.49	19.49	0	

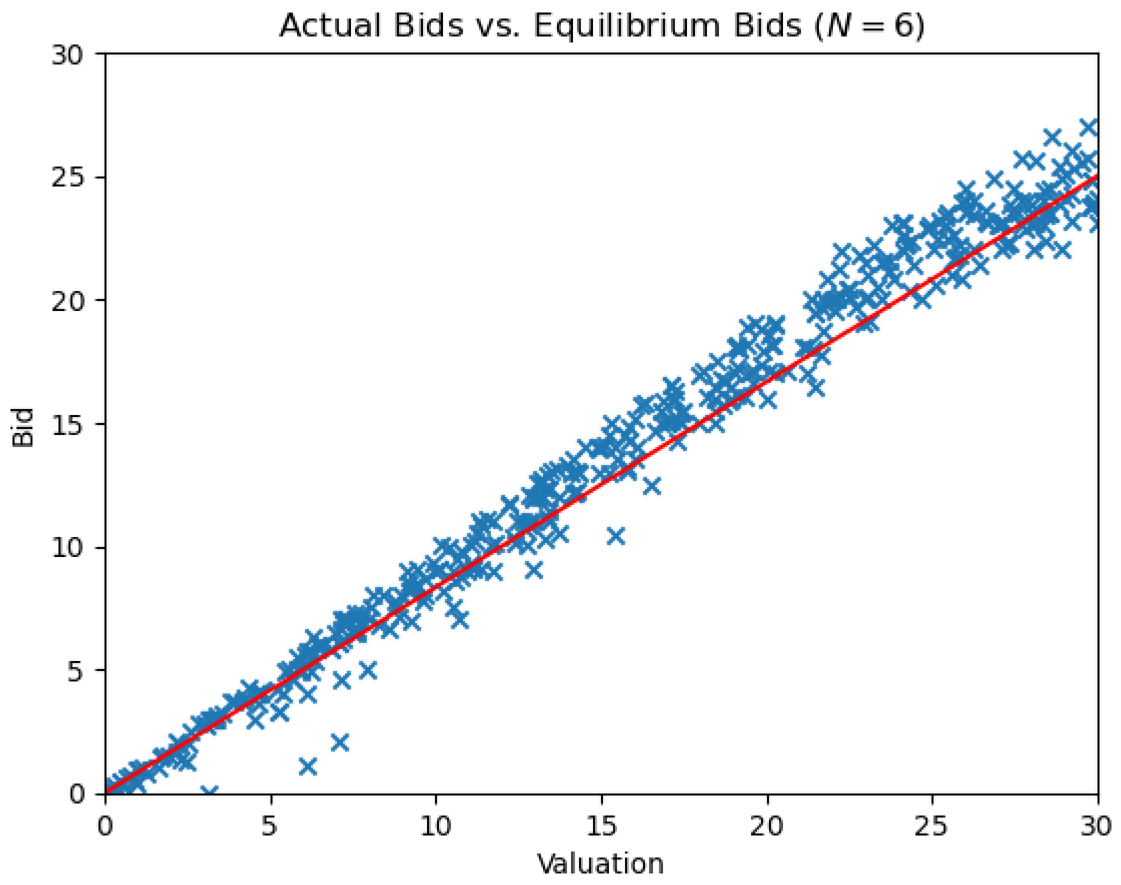
Risk-Neutral Equilibrium Model for Bid-Prediction

Theoretical Bids vs. Actual Bids

In [ ]:
```
# Plotting N=3
plt.figure()
plt.scatter(data['Value'].values, data['BidC3'].values, marker='x')
plt.plot(np.linspace(0,30,5), (2/3)*np.linspace(0,30,5), 'r')
plt.xlim(0,30)
plt.ylim(0,30)
plt.title('Actual Bids vs. Equilibrium Bids ($N=3$)')
plt.xlabel('Valuation')
plt.ylabel('Bid')
plt.show()
```

```
# Plotting N=6
plt.figure()
plt.scatter(data['Value'].values, data['BidC6'].values, marker='x')
plt.plot(np.linspace(0,30,5), (5/6)*np.linspace(0,30,5), 'r')
plt.xlim(0,30)
plt.ylim(0,30)
plt.title('Actual Bids vs. Equilibrium Bids ($N=6$)')
plt.xlabel('Valuation')
plt.ylabel('Bid')
plt.show()
```

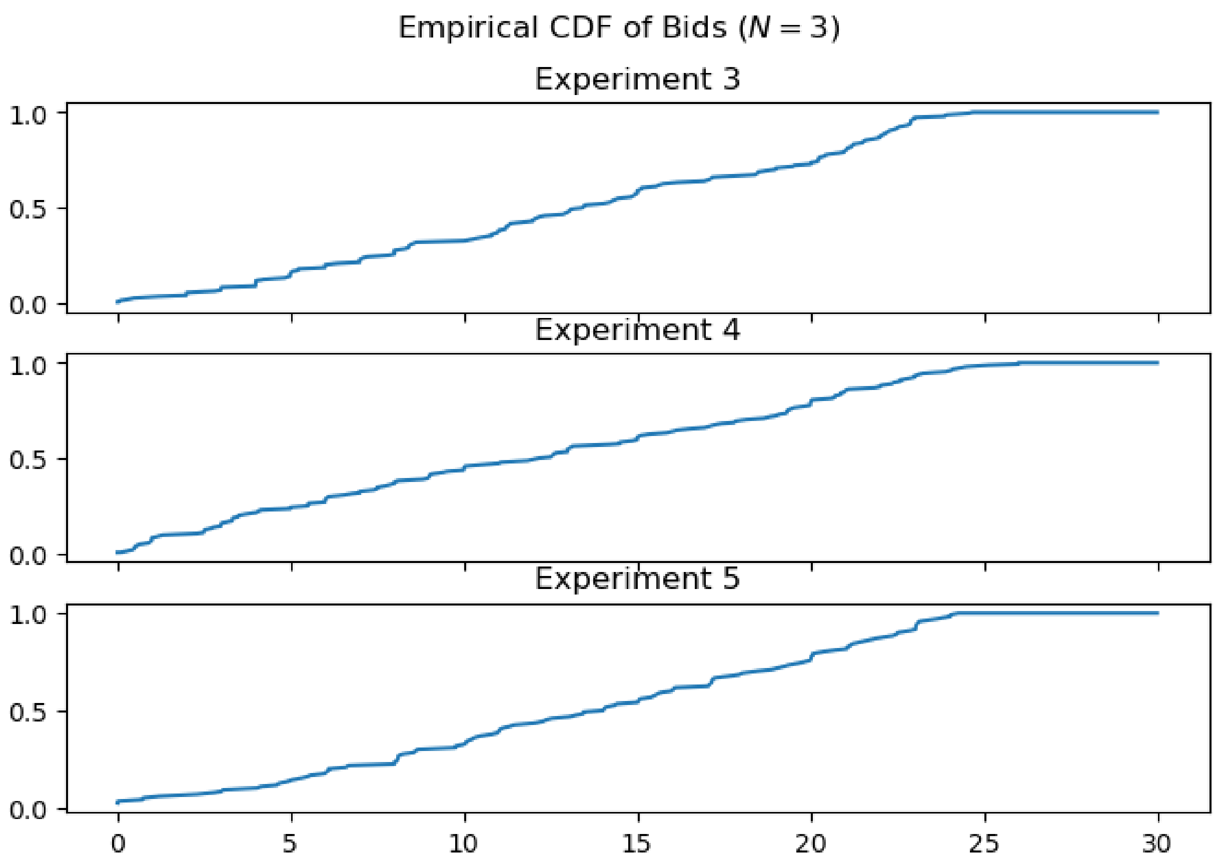


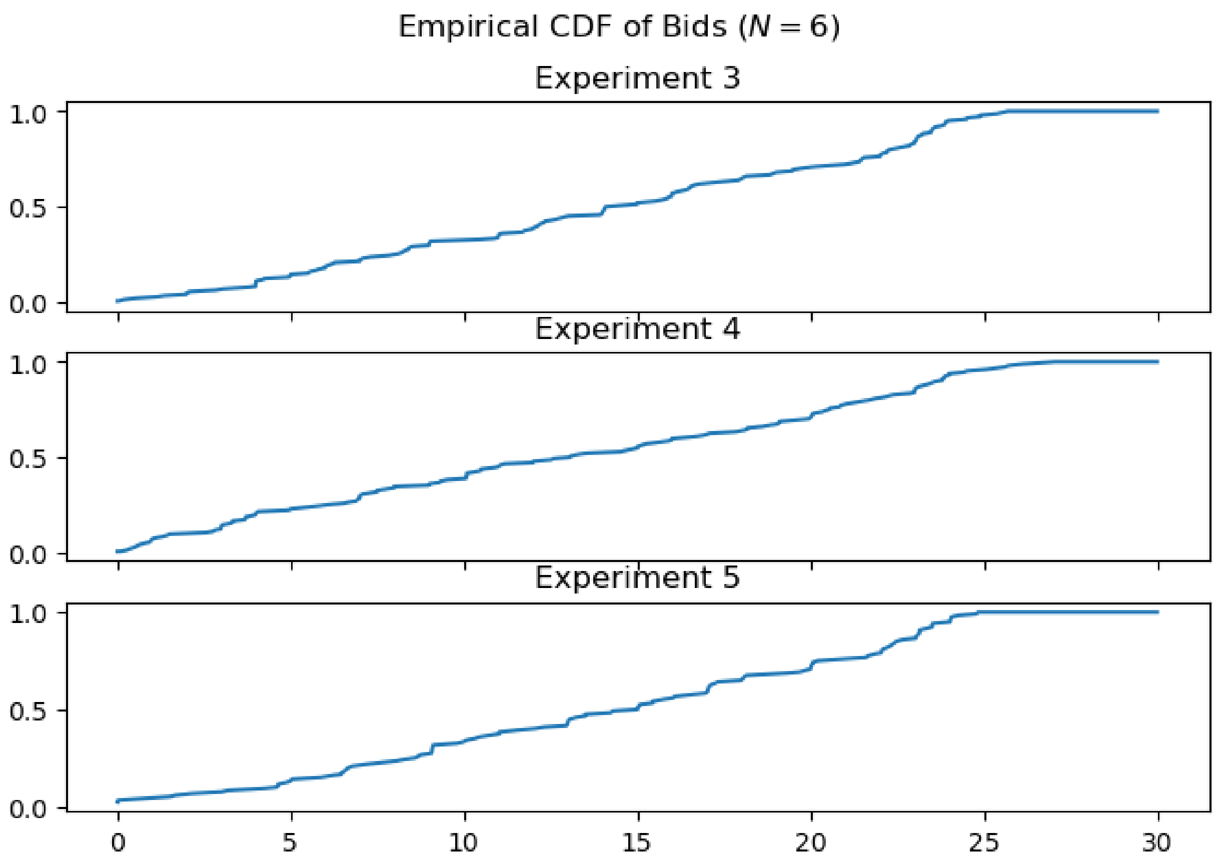


Computing the Empirical Cumulative Distribution Function

```
In [ ]:
# Given a list of values, returns the empirical distribution of values
def ecdf(bid_data):
    x = np.sort(bid_data)
    n = x.size
    y = np.arange(1, n+1) / n
    return x, y

# Given a list of bids and input, returns the empirical distribution of
bids evaluated at input
def empirical_distribution(bid_data, input):
    x, y = ecdf(bid_data)
    return np.interp(input, x, y)

# Splitting data by experiment number
data_e3 = data[data['Experiment'] == 3]
data_e4 = data[data['Experiment'] == 4]
data_e5 = data[data['Experiment'] == 5]
```

```
# For each auction size, plot the empirical CDF of the 3 experiments for
N=3
grid = np.linspace(0.01,30,3000)

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharex=True, figsize=(8,5));
ax1.plot(grid, empirical_distribution(data_e3['BidC3'].values, grid),
 label='Experiment 3');
ax1.set_title('Experiment 3');
ax2.plot(grid, empirical_distribution(data_e4['BidC3'].values, grid),
 label='Experiment 4');
ax2.set_title('Experiment 4');
ax3.plot(grid, empirical_distribution(data_e5['BidC3'].values, grid),
 label='Experiment 5');
ax3.set_title('Experiment 5');
fig.suptitle('Empirical CDF of Bids ($N=3$)');

# Now do the same for N=6
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharex=True, figsize=(8,5));
ax1.plot(grid, empirical_distribution(data_e3['BidC6'].values, grid),
 label='Experiment 3');
ax1.set_title('Experiment 3');
ax2.plot(grid, empirical_distribution(data_e4['BidC6'].values, grid),
 label='Experiment 4');
ax2.set_title('Experiment 4');
ax3.plot(grid, empirical_distribution(data_e5['BidC6'].values, grid),
 label='Experiment 5');
ax3.set_title('Experiment 5');
fig.suptitle('Empirical CDF of Bids ($N=6$)');
```





The empirical cumulative distributions do not seem very different from each other across experiments.

Computing Expected Profit and Optimization Error

In [ ]:
```
# Given a true valuation, bid input, bid data, and number of bidders,
# returns the expected profit of the bidder
def expected_profit(input, bid_data, valuation, N):
    cdf = empirical_distribution(bid_data, input)
    return (valuation - input) * (cdf ** (N-1))
# We define the negative of expected_profit for scipy's minimize to
maximize
def neg_expected_profit(input, bid_data, valuation, N):
    return -1 * expected_profit(input, bid_data, valuation, N)

# Given a true valuation, bid input, bid data, and number of bidders,
# returns the optimization error of the bidder
def optimization_error(input, bid_data, valuation, N):
    # if valuation is a scalar
    if np.isscalar(valuation) and np.isscalar(input):
        best_bid = minimize(neg_expected_profit, input, args=(bid_data,
valuation, N),
        method='Nelder-Mead', tol=1e-4).x
        return expected_profit(best_bid, bid_data, valuation, N) - \
            expected_profit(input, bid_data, valuation, N)
    # else valuation is a vector
    else:
        best_bid = np.zeros(valuation.size)
        for i in range(valuation.size):
            best_bid[i] = minimize(neg_expected_profit, input[i], args=
(bid_data, valuation[i], N),
            method='Nelder-Mead', tol=1e-4).x
        return expected_profit(best_bid, bid_data, valuation, N) - \
            expected_profit(input, bid_data, valuation, N)

# Compute the expected profit and optimization error for N=3 and N=6
nashbid_N3 = data['BidC3'].values - 2/3 * data['Value'].values
nashbid_N6 = data['BidC6'].values - 5/6 * data['Value'].values
exprofit_n3 = expected_profit(data['BidC3'].values, data['BidC3'].values,
data['Value'].values, 3)
exprofit_n6 = expected_profit(data['BidC6'].values, data['BidC6'].values,
data['Value'].values, 6)
```

```
opterror_n3 = optimization_error(data['BidC3'].values,
data['BidC3'].values, data['Value'].values, 3)
opterror_n6 = optimization_error(data['BidC6'].values,
data['BidC6'].values, data['Value'].values, 6)
```

Printing Statistics

In [ ]:
```
# Calculate mean, standard deviation, and percentiles of nashbid,
exprofit, and opterror
# and print them into a pretty table
print(f"{'Variable': <19} {'Mean' : ^10} {'Std Dev' : ^10} {'25%tile' :
^10} {'50%tile' : ^10} {'75%tile' : ^10}")
print('-'*72)
print('Observed Nash bid:')
print(f"{'  N=3' : <19} {np.mean(nashbid_N3) : ^10.2f} {np.std(nashbid_N3)
: ^10.2f} \
{np.percentile(nashbid_N3, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(nashbid_N3, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(nashbid_N3, [25, 50, 75])[2] : ^10.2f}")
print(f"{  '  N=6' : <19} {np.mean(nashbid_N6) : ^10.2f}
{np.std(nashbid_N6) : ^10.2f} \
{np.percentile(nashbid_N6, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(nashbid_N6, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(nashbid_N6, [25, 50, 75])[2] : ^10.2f}")
print('Expected profit:')
print(f"{'  N=3' : <19} {np.mean(exprofit_n3) : ^10.2f}
{np.std(exprofit_n3) : ^10.2f} \
{np.percentile(exprofit_n3, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(exprofit_n3, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(exprofit_n3, [25, 50, 75])[2] : ^10.2f}")
print(f"{  '  N=6' : <19} {np.mean(exprofit_n6) : ^10.2f}
{np.std(exprofit_n6) : ^10.2f} \
{np.percentile(exprofit_n6, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(exprofit_n6, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(exprofit_n6, [25, 50, 75])[2] : ^10.2f}")
print('Optimization error:')
print(f"{'  N=3' : <19} {np.mean(opterror_n3) : ^10.2f}
{np.std(opterror_n3) : ^10.2f} \
{np.percentile(opterror_n3, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(opterror_n3, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(opterror_n3, [25, 50, 75])[2] : ^10.2f}")
print(f"{  '  N=6' : <19} {np.mean(opterror_n6) : ^10.2f}
{np.std(opterror_n6) : ^10.2f} \
{np.percentile(opterror_n6, [25, 50, 75])[0] : ^10.2f} \
{np.percentile(opterror_n6, [25, 50, 75])[1] : ^10.2f} \
{np.percentile(opterror_n6, [25, 50, 75])[2] : ^10.2f}")
```

```
Variable                 Mean      Std Dev     25%tile     50%tile     75%tile
──────────────────────────────────────────────────────────────────────────────
Observed Nash bid:
  N=3                    2.44       1.86        1.06        2.46        3.83
  N=6                    0.65       1.05        0.03        0.56        1.41
Expected profit:
  N=3                    1.32       1.59        0.08        0.62        1.99
  N=6                    0.57       1.02        0.00        0.04        0.55
Optimization error:
  N=3                    0.34       0.44        0.02        0.14        0.54
  N=6                    0.11       0.25        0.00        0.01        0.10
```

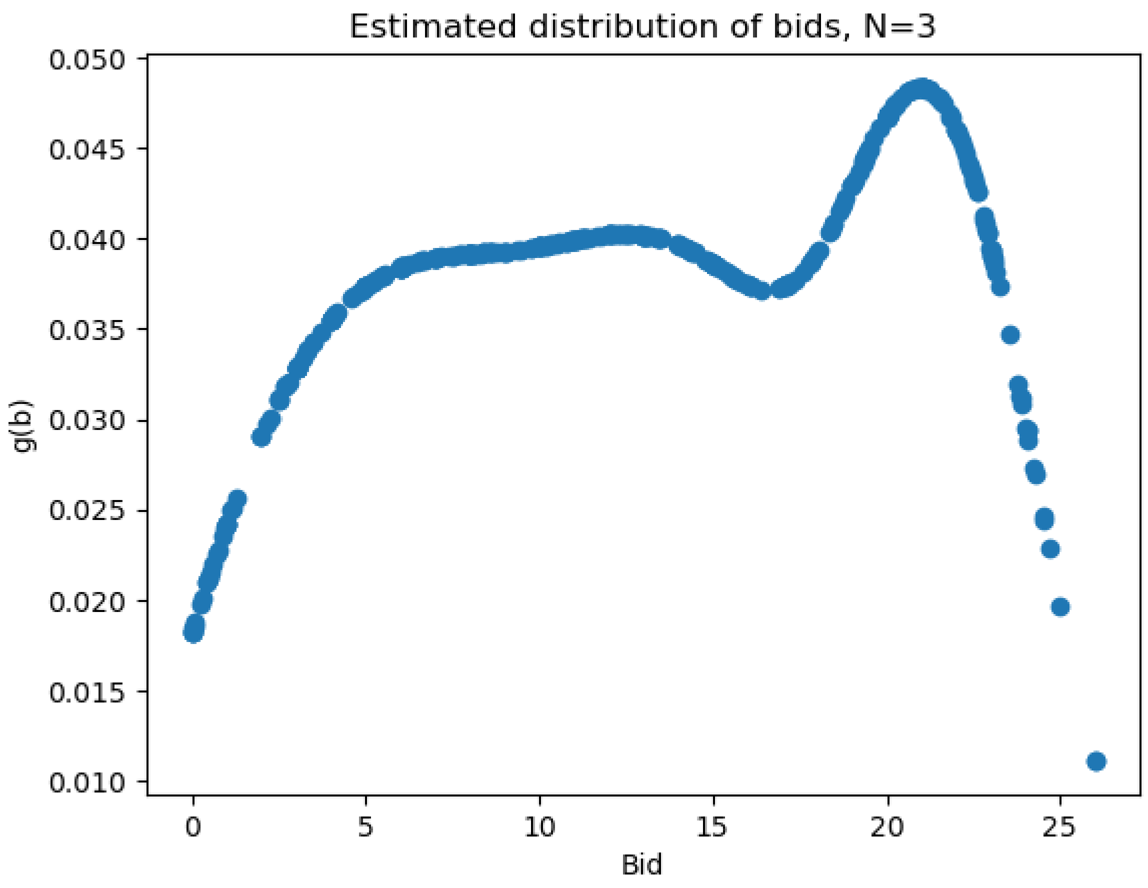
Risk Neutral Structural Model for Valuation-Prediction

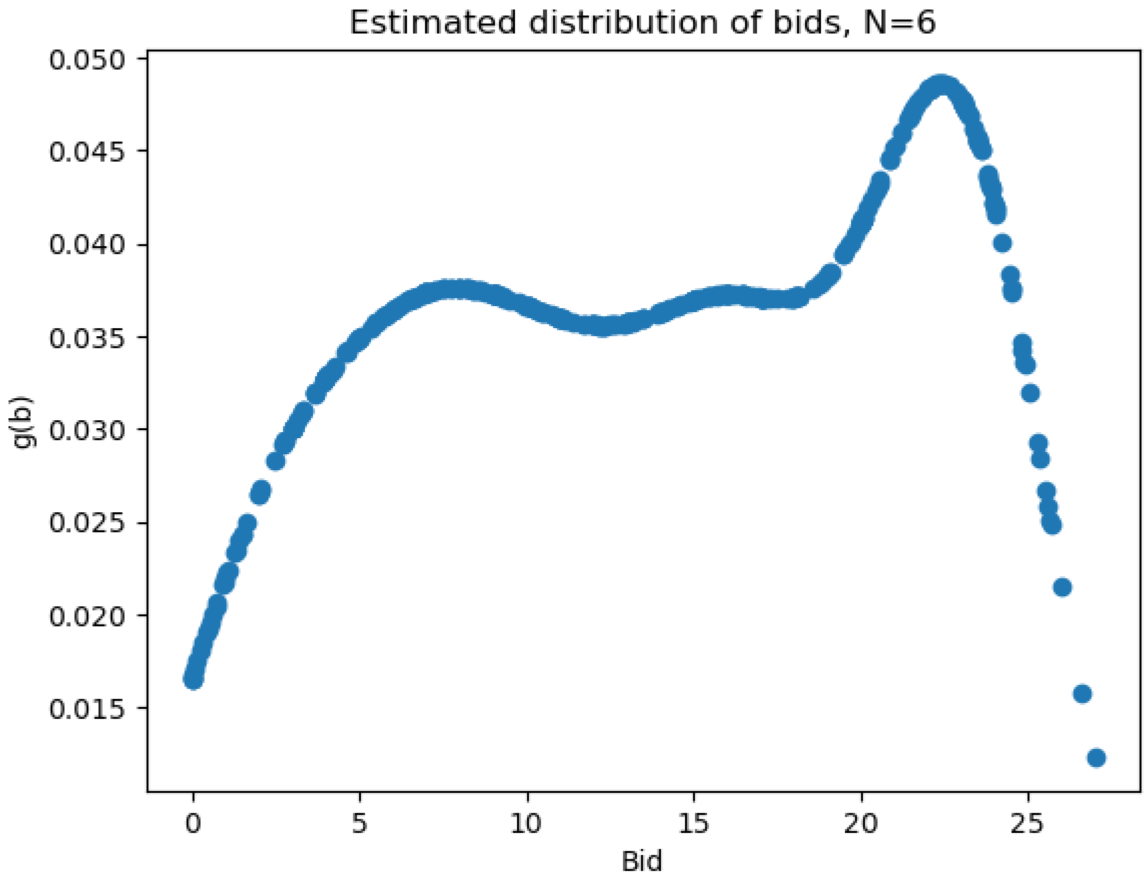
Empirical Probability Distribution Function of Bids

In [ ]:

```
# Compute an estimate of g(b) for each bid in the data using Silverman's
rule of thumb for the bandwidth
# and plot the results

# Estimate the distribution of the bids using the normal kernel
def estimate_g(bid_data, bandwidth):
    """Estimate the distribution of the bids using the normal kernel."""
    g = np.zeros(len(bid_data))
    for i in range(len(bid_data)):
        g[i] = np.mean(norm.pdf(bid_data, bid_data[i], bandwidth))
    return g

# Compute the bandwidths using Silverman's rule of thumb
# compute iqr
iqr = np.percentile(data['BidC3'].values, 75) -
np.percentile(data['BidC3'].values, 25)
h3 = 0.9 * np.min([np.std(data['BidC3'].values), iqr/1.34]) *
len(data['BidC3'].values)**(-1/5)
# Compute the g(b) values
```

```
g3 = estimate_g(data['BidC3'].values, h3)
# Plot the results
plt.plot(data['BidC3'].values, g3, 'o')
plt.xlabel('Bid')
plt.ylabel('g(b)')
plt.title('Estimated distribution of bids, N=3')
plt.show()

# Compute the bandwidths
iqr = np.percentile(data['BidC6'].values, 75) -
np.percentile(data['BidC6'].values, 25)
h6 = 0.9 * np.min([np.std(data['BidC6'].values), iqr/1.34]) *
len(data['BidC6'].values)**(-1/5)
# Compute the g(b) values
g6 = estimate_g(data['BidC6'].values, h6)
# Plot the results
plt.plot(data['BidC6'].values, g6, 'o')
plt.xlabel('Bid')
plt.ylabel('g(b)')
plt.title('Estimated distribution of bids, N=6')
plt.show()
```





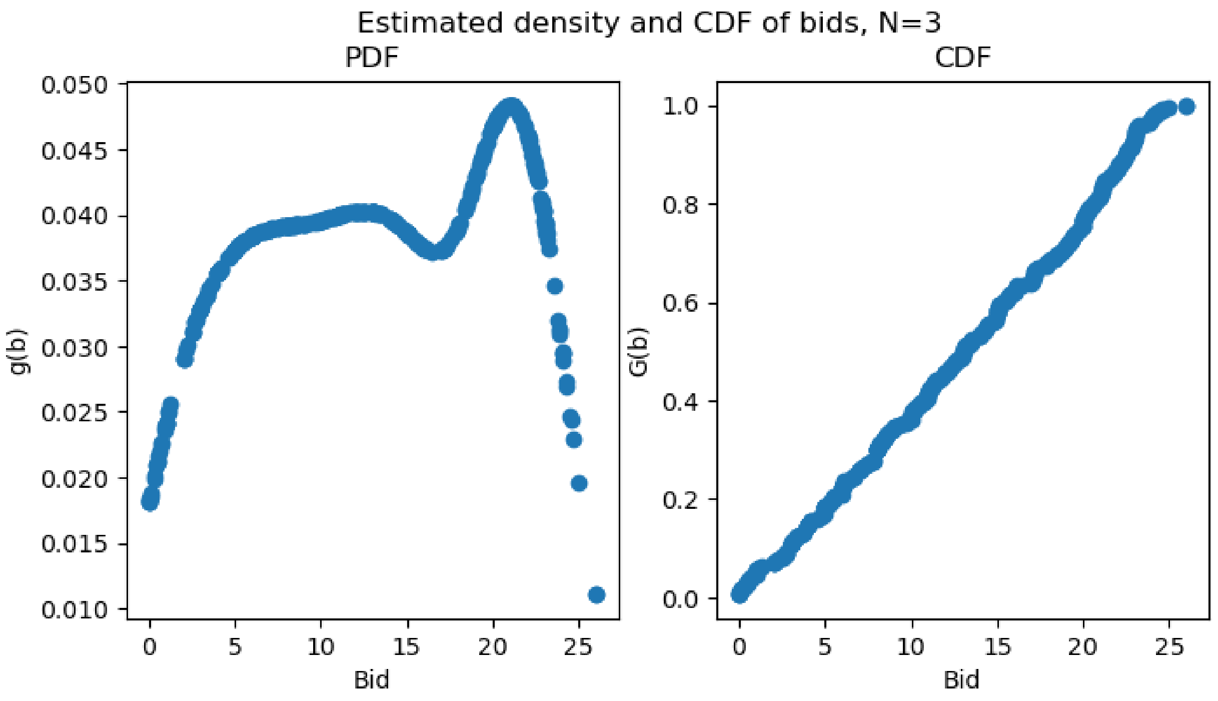
CDF Estimation 2 Ways: Within Experiment & Across Experiment

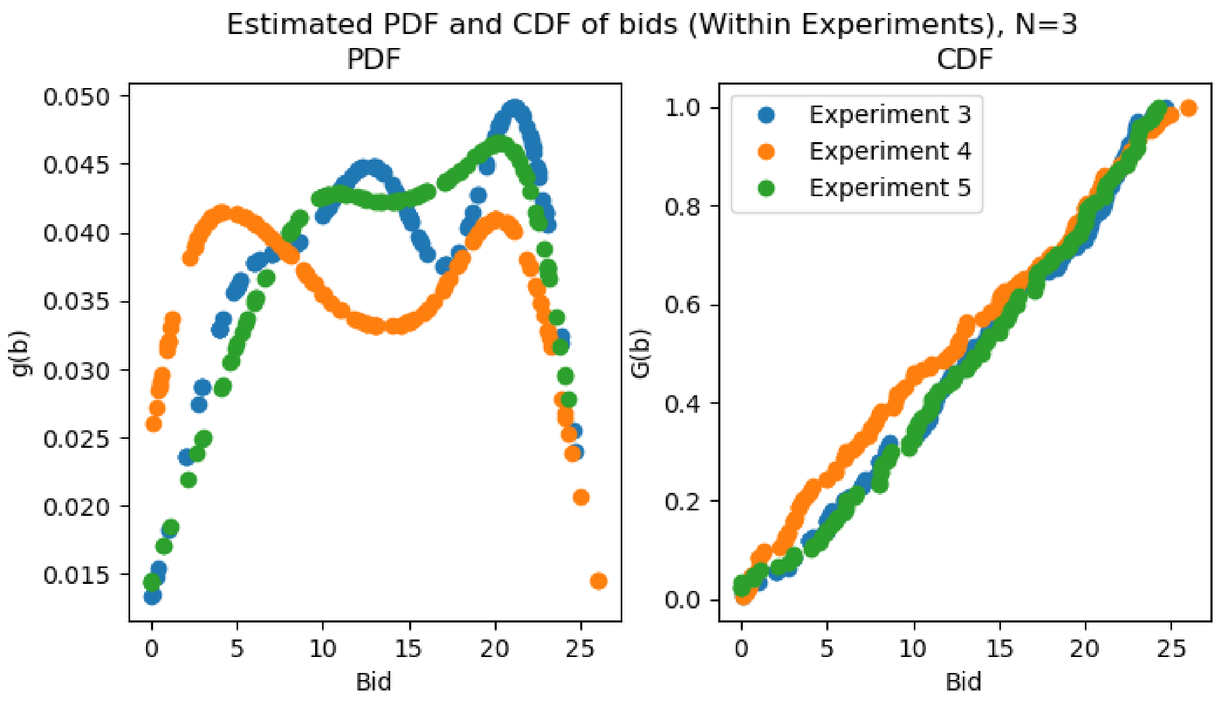
```
In [ ]: # N=3 case
        # Estimate density and CDF of bids using data across experiments 4, 5, and
        6
        # Estimate the bandwidth
        iqr = np.percentile(data['BidC3'].values, 75) -
        np.percentile(data['BidC3'].values, 25)
        h3 = 0.9 * np.min([np.std(data['BidC3'].values), iqr/1.34]) *
        len(data['BidC3'].values)**(-1/5)
        # Estimate the density
        g3 = estimate_g(data['BidC3'].values, h3)
        # Estimate the CDF
        G3 = np.zeros(len(data['BidC3'].values))
        for i in range(len(data['BidC3'].values)):
            G3[i] = np.mean(data['BidC3'].values <= data['BidC3'].values[i])
        # Plot the results
        fig, (ax1, ax2)  = plt.subplots(1, 2, figsize=(8, 4))
        ax1.plot(data['BidC3'].values, g3, 'o', label='Density')
        ax1.set_xlabel('Bid')
        ax1.set_ylabel('g(b)')
        ax1.set_title('PDF')
        ax2.plot(data['BidC3'].values, G3, 'o', label='CDF')
        ax2.set_xlabel('Bid')
        ax2.set_ylabel('G(b)')
        ax2.set_title('CDF')
        fig.suptitle('Estimated density and CDF of bids, N=3')
        plt.show()
        # Save this result for later use
        g_N3 = g3
        G_N3 = G3

        # Estimate density and CDF of bids using data within experiments 3, 4, and
        5
        g3dict = {}
        G3dict = {}
        for experiment in [3, 4, 5]:
            iqr = np.percentile(data[data['Experiment'] == experiment]
        ['BidC3'].values, 75) - \
                    np.percentile(data[data['Experiment'] == experiment]
        ['BidC3'].values, 25)
            h3 = 0.9 * np.min([np.std(data[data['Experiment'] == experiment]
        ['BidC3'].values), iqr/1.34]) * \
                    len(data[data['Experiment'] == experiment]['BidC3'].values)**
        (-1/5)
            g3dict[experiment] = estimate_g(data[data['Experiment'] == experiment]
        ['BidC3'].values, h3)
            G3 = np.zeros(len(data[data['Experiment'] == experiment]
        ['BidC3'].values))
            for i in range(len(data[data['Experiment'] == experiment]
        ['BidC3'].values)):
                G3[i] = np.mean(data[data['Experiment'] == experiment]
        ['BidC3'].values <= \
                                data[data['Experiment'] == experiment]
        ['BidC3'].values[i])
            G3dict[experiment] = G3
        # Plot the results
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
        plotx = np.concatenate([data[data['Experiment'] == 3]['BidC3'].values, \
                                data[data['Experiment'] == 4]['BidC3'].values, \
                                data[data['Experiment'] == 5]['BidC3'].values])
        ax1.plot(data[data['Experiment'] == 3]['BidC3'].values, g3dict.get(3),
        'o', label='Experiment 3')
        ax1.plot(data[data['Experiment'] == 4]['BidC3'].values, g3dict.get(4),
        'o', label='Experiment 4')
        ax1.plot(data[data['Experiment'] == 5]['BidC3'].values, g3dict.get(5),
        'o', label='Experiment 5')
        ax1.set_xlabel('Bid')
        ax1.set_ylabel('g(b)')
        ax1.set_title('PDF')
        ax2.plot(data[data['Experiment'] == 3]['BidC3'].values, G3dict.get(3),
        'o', label='Experiment 3')
        ax2.plot(data[data['Experiment'] == 4]['BidC3'].values, G3dict.get(4),
        'o', label='Experiment 4')
        ax2.plot(data[data['Experiment'] == 5]['BidC3'].values, G3dict.get(5),
        'o', label='Experiment 5')
        ax2.set_xlabel('Bid')
        ax2.set_ylabel('G(b)')
        ax2.set_title('CDF')
```

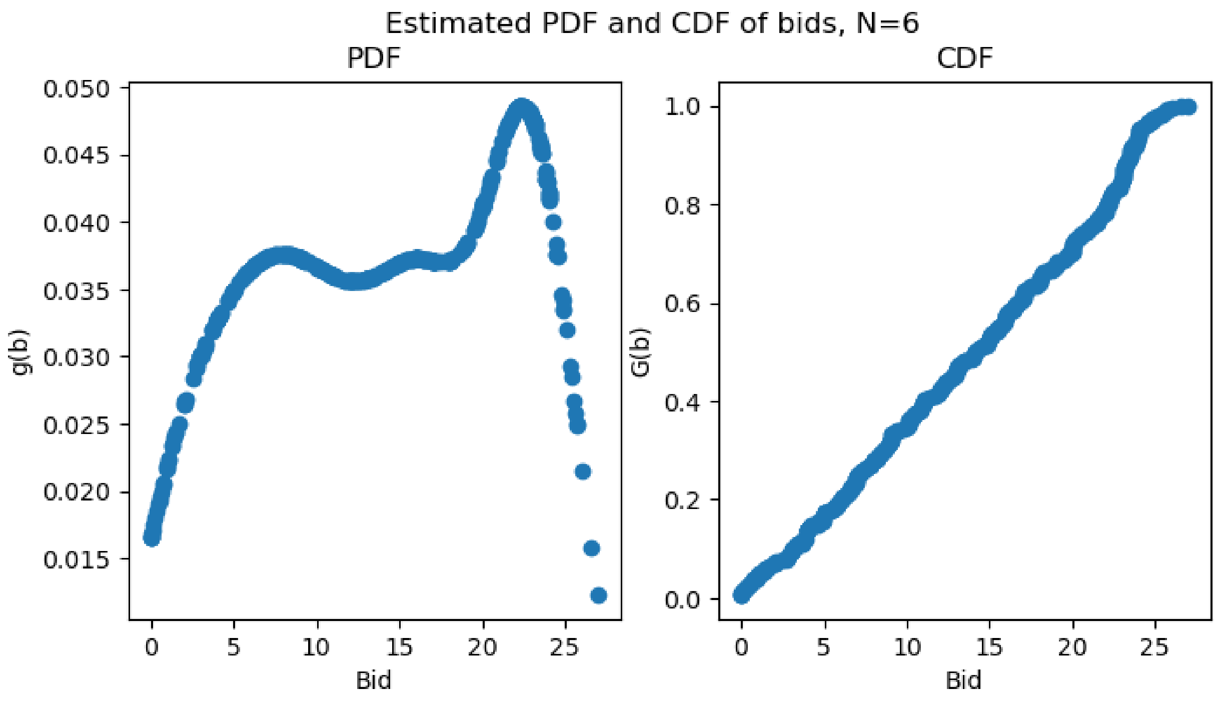
```
ax2.legend()
fig.suptitle('Estimated PDF and CDF of bids (Within Experiments), N=3')
plt.show()



# N=6 case
# Estimate density and CDF of bids using data across experiments 4, 5, and
6
# Estimate the bandwidth
iqr = np.percentile(data['BidC6'].values, 75) -
np.percentile(data['BidC6'].values, 25)
h6 = 0.9 * np.min([np.std(data['BidC6'].values), iqr/1.34]) *
len(data['BidC6'].values)**(-1/5)
# Estimate the density
g6 = estimate_g(data['BidC6'].values, h6)
# Estimate the CDF
G6 = np.zeros(len(data['BidC6'].values))
for i in range(len(data['BidC6'].values)):
    G6[i] = np.mean(data['BidC6'].values <= data['BidC6'].values[i])
# Plot the results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
ax1.plot(data['BidC6'].values, g6, 'o', label='Density')
ax1.set_xlabel('Bid')
ax1.set_ylabel('g(b)')
ax1.set_title('PDF')
ax2.plot(data['BidC6'].values, G6, 'o', label='CDF')
ax2.set_xlabel('Bid')
ax2.set_ylabel('G(b)')
ax2.set_title('CDF')
fig.suptitle('Estimated PDF and CDF of bids, N=6')
plt.show()
# Save this result for later use
g_N6 = g6
G_N6 = G6

# Estimate density and CDF of bids using data within experiments 3, 4, and
5
g6dict = {}
G6dict = {}
```

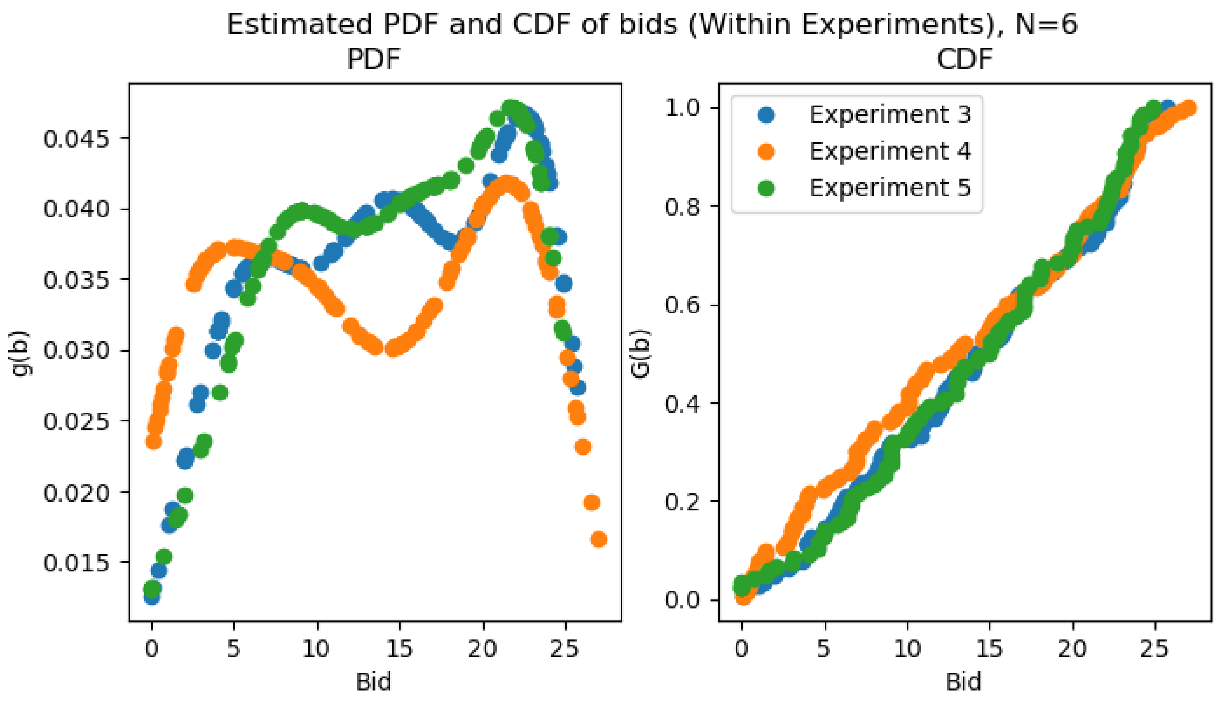
```
for experiment in [3, 4, 5]:
    iqr = np.percentile(data[data['Experiment'] == experiment]
['BidC6'].values, 75) - \
            np.percentile(data[data['Experiment'] == experiment]
['BidC6'].values, 25)
    h6 = 0.9 * np.min([np.std(data[data['Experiment'] == experiment]
['BidC6'].values), iqr/1.34]) * \
            len(data[data['Experiment'] == experiment]['BidC6'].values)**
(-1/5)
    g6dict[experiment] = estimate_g(data[data['Experiment'] == experiment]
['BidC6'].values, h6)
    G6 = np.zeros(len(data[data['Experiment'] == experiment]
['BidC6'].values))
    for i in range(len(data[data['Experiment'] == experiment]
['BidC6'].values)):
        G6[i] = np.mean(data[data['Experiment'] == experiment]
['BidC6'].values <= \
                        data[data['Experiment'] == experiment]
['BidC6'].values[i])
    G6dict[experiment] = G6
# Concatenate results across experiments
g6 = np.concatenate([g6dict[3], g6dict[4], g6dict[5]])
G6 = np.concatenate([G6dict[3], G6dict[4], G6dict[5]])
# Plot the results
plotx = np.concatenate([data[data['Experiment'] == 3]['BidC6'].values, \
                        data[data['Experiment'] == 4]['BidC6'].values, \
                        data[data['Experiment'] == 5]['BidC6'].values])
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
ax1.plot(data[data['Experiment'] == 3]['BidC6'].values, g6dict.get(3),
'o', label='Experiment 3')
ax1.plot(data[data['Experiment'] == 4]['BidC6'].values, g6dict.get(4),
'o', label='Experiment 4')
ax1.plot(data[data['Experiment'] == 5]['BidC6'].values, g6dict.get(5),
'o', label='Experiment 5')
ax1.set_xlabel('Bid')
ax1.set_ylabel('g(b)')
ax1.set_title('PDF')
ax2.plot(data[data['Experiment'] == 3]['BidC6'].values, G6dict.get(3),
'o', label='Experiment 3')
```

```
ax2.plot(data[data['Experiment'] == 4]['BidC6'].values, G6dict.get(4),
'o', label='Experiment 4')
ax2.plot(data[data['Experiment'] == 5]['BidC6'].values, G6dict.get(5),
'o', label='Experiment 5')
ax2.set_xlabel('Bid')
ax2.set_ylabel('G(b)')
ax2.set_title('CDF')
ax2.legend()
fig.suptitle('Estimated PDF and CDF of bids (Within Experiments), N=6')
plt.show()
```




Estimated PDF and CDF of bids (Within Experiments), N=3
PDF
CDF
g(b)
G(b)
Bid
Bid
Experiment 3
Experiment 4
Experiment 5

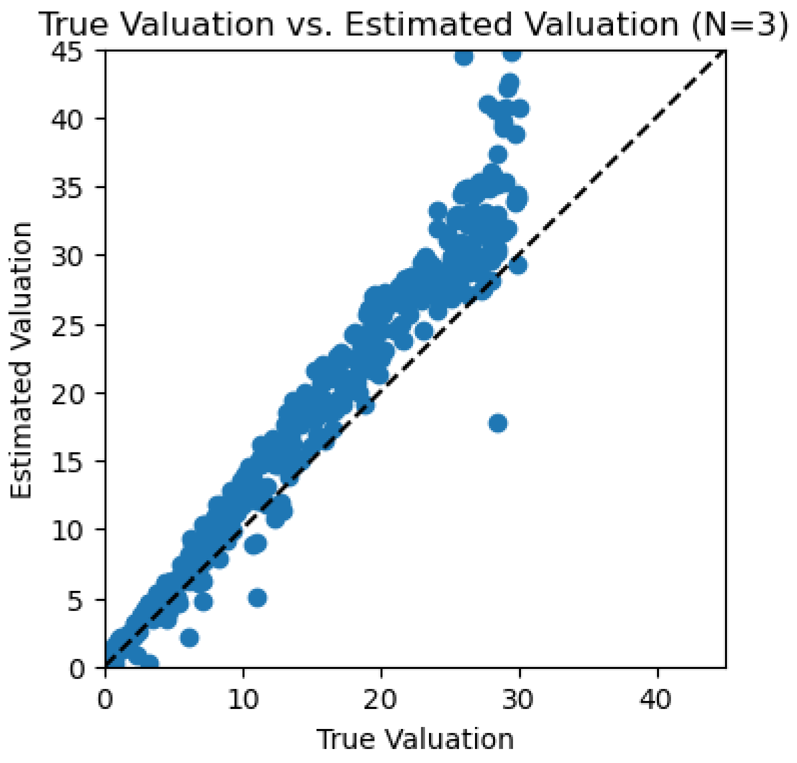

Estimated PDF and CDF of bids, N=6
PDF
CDF
g(b)
G(b)
Bid
Bid

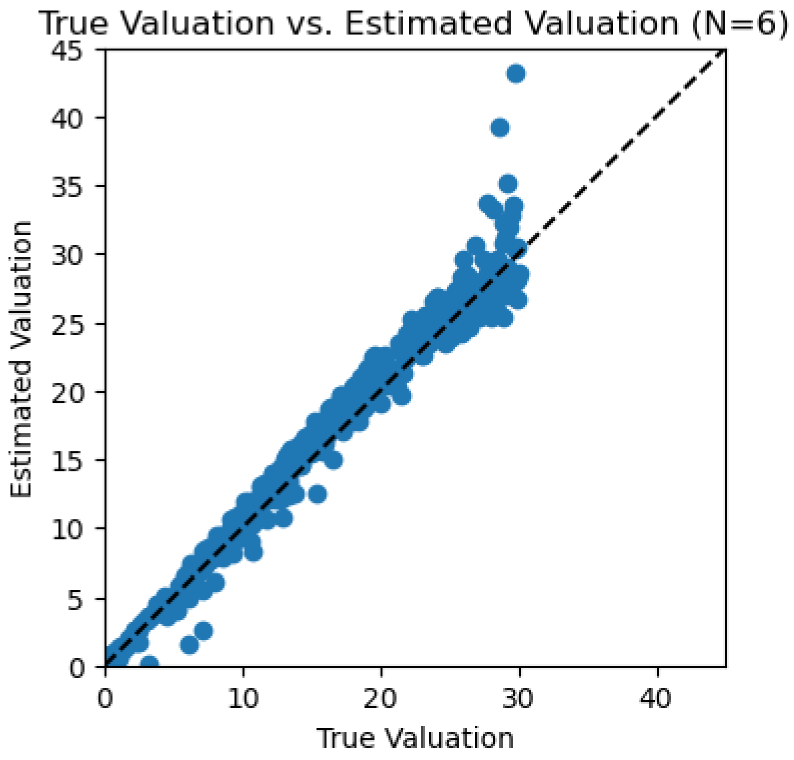


True Valuation vs. Model Estimated Valuations

```
In [ ]:
# Part (a)
# Use equation (9) and previous calculations to compute two estimates for
each valuation (N=3 and N=6)
# N=3 case
def v_risk_neutral(b, g, G, N):
    return b + G/(g*(N-1))
v3 = np.zeros(len(data['BidC3'].values))
for i in range(len(data['BidC3'].values)):
    v3[i] = v_risk_neutral(data['BidC3'].values[i], g_N3[i], G_N3[i], 3)
# Plot the results
fig, ax = plt.subplots(1, 1, figsize=(4, 4))
ax.scatter(data['Value'].values, v3)
ax.plot([0, 45], [0, 45], 'k--')
ax.set_xlim([0, 45])
ax.set_ylim([0, 45])
ax.set_xlabel('True Valuation')
ax.set_ylabel('Estimated Valuation')
ax.set_title('True Valuation vs. Estimated Valuation (N=3)')
plt.show()

# N=6 case
v6 = np.zeros(len(data['BidC6'].values))
```

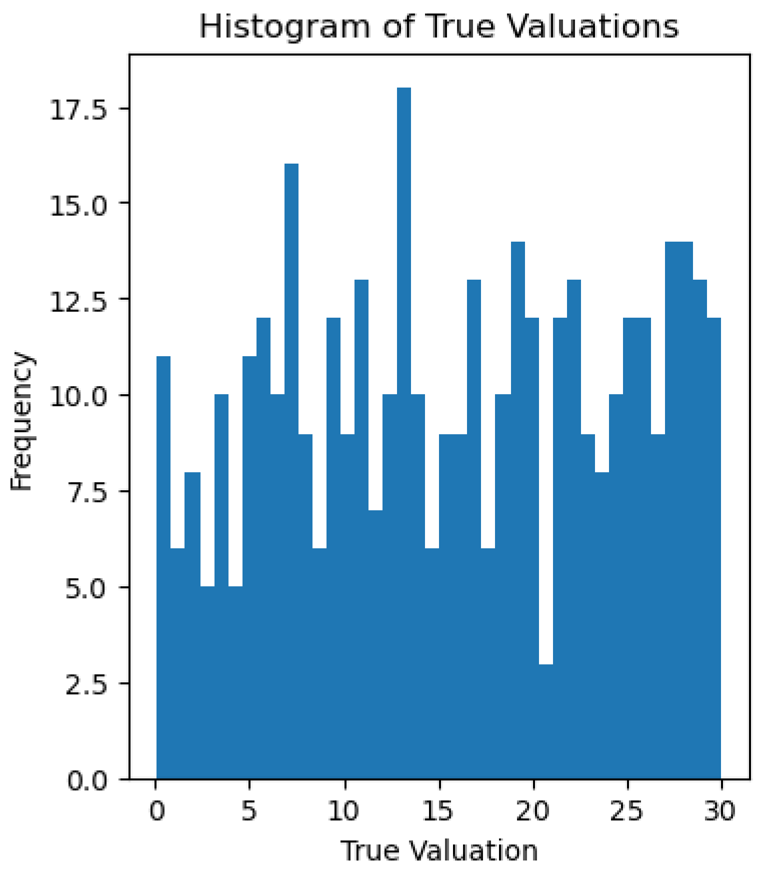
```
for i in range(len(data['BidC6'].values)):
    v6[i] = v_risk_neutral(data['BidC6'].values[i], g_N6[i], G_N6[i], 6)
# Plot the results
fig, ax = plt.subplots(1, 1, figsize=(4, 4))
ax.scatter(data['Value'].values, v6)
ax.plot([0, 45], [0, 45], 'k--')
ax.set_xlim([0, 45])
ax.set_ylim([0, 45])
ax.set_xlabel('True Valuation')
ax.set_ylabel('Estimated Valuation')
ax.set_title('True Valuation vs. Estimated Valuation (N=6)')
plt.show()
```

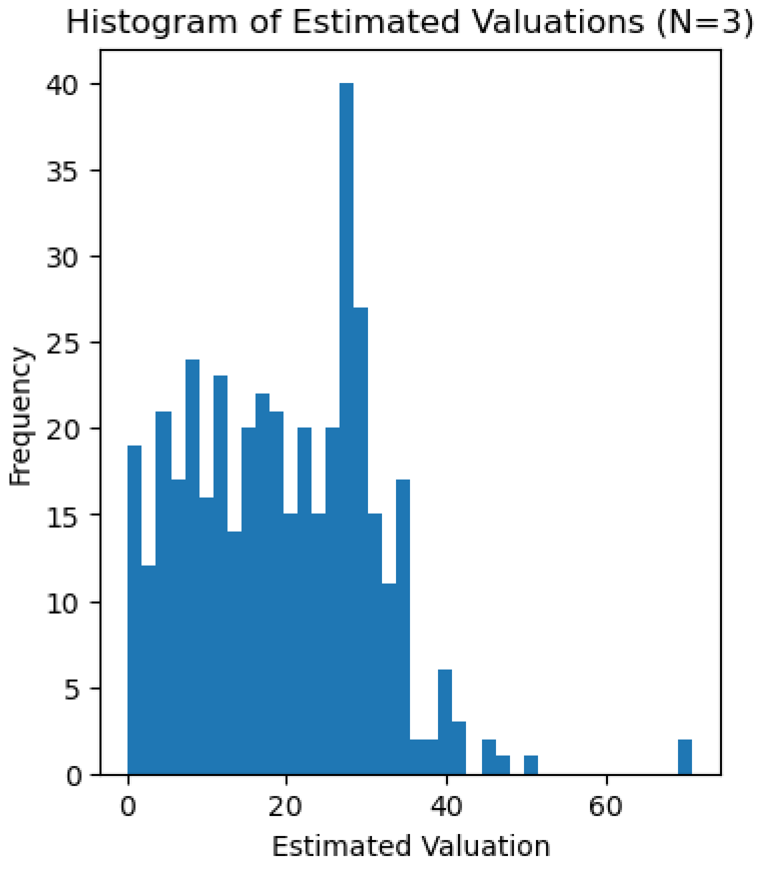


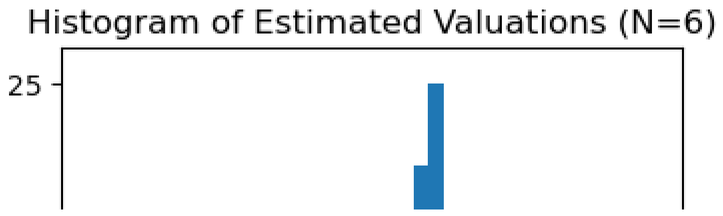


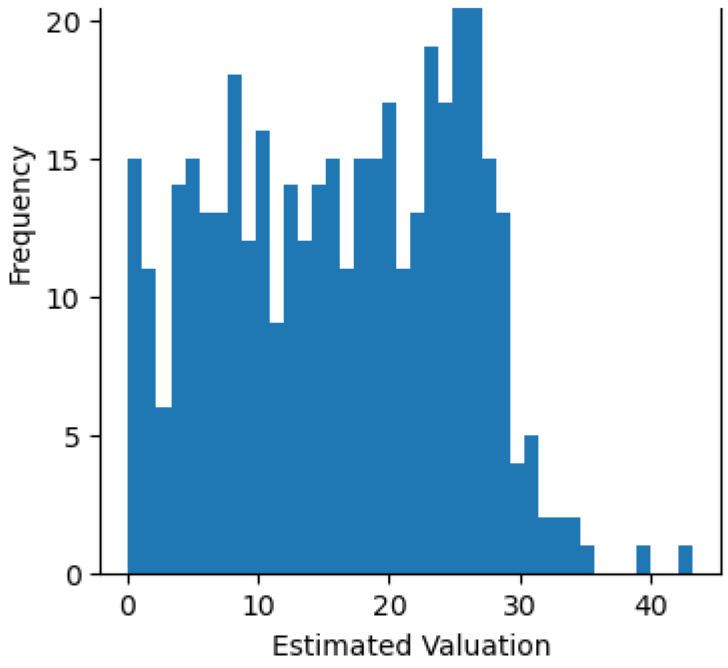
In [ ]:

```
# Part (b)
# Plot histograms of true valuations and estimated valuations (N=3 and
N=6)
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(4, 16))
ax1.hist(data['Value'].values, bins=40)
ax1.set_xlabel('True Valuation')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of True Valuations')
ax2.hist(v3, bins=40)
ax2.set_xlabel('Estimated Valuation')
ax2.set_ylabel('Frequency')
ax2.set_title('Histogram of Estimated Valuations (N=3)')
ax3.hist(v6, bins=40)
ax3.set_xlabel('Estimated Valuation')
ax3.set_ylabel('Frequency')
ax3.set_title('Histogram of Estimated Valuations (N=6)')
plt.show()
```


Histogram of True Valuations
Frequency
True Valuation
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
20
25
30


Histogram of Estimated Valuations (N=3)
Frequency
Estimated Valuation
0
5
10
15
20
25
30
35
40
0
20
40
60


Histogram of Estimated Valuations (N=6)
25



Computing Distance between Estimates and Truths

```
In [ ]:
# Compute L1 and L2 distances for the two ways of estimating the CDF and
PDF in CDF Estimation
# Across Experiment (N=6)
L1_across = np.zeros(len(data['Value'].values))
L2_across = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1_across[i] = np.abs(data['Value'].values[i] - v6[i])
    L2_across[i] = (data['Value'].values[i] - v6[i])**2
L1_across = np.mean(L1_across)
L2_across = np.sqrt(np.mean(L2_across))
print('L1 distance (across experiment):', L1_across)
print('L2 distance (across experiment):', L2_across)
# Across Experiment (N=3)
L1_across = np.zeros(len(data['Value'].values))
L2_across = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1_across[i] = np.abs(data['Value'].values[i] - v3[i])
    L2_across[i] = (data['Value'].values[i] - v3[i])**2
L1_across = np.mean(L1_across)
L2_across = np.sqrt(np.mean(L2_across))
print('L1 distance (across experiment):', L1_across)
print('L2 distance (across experiment):', L2_across)
```

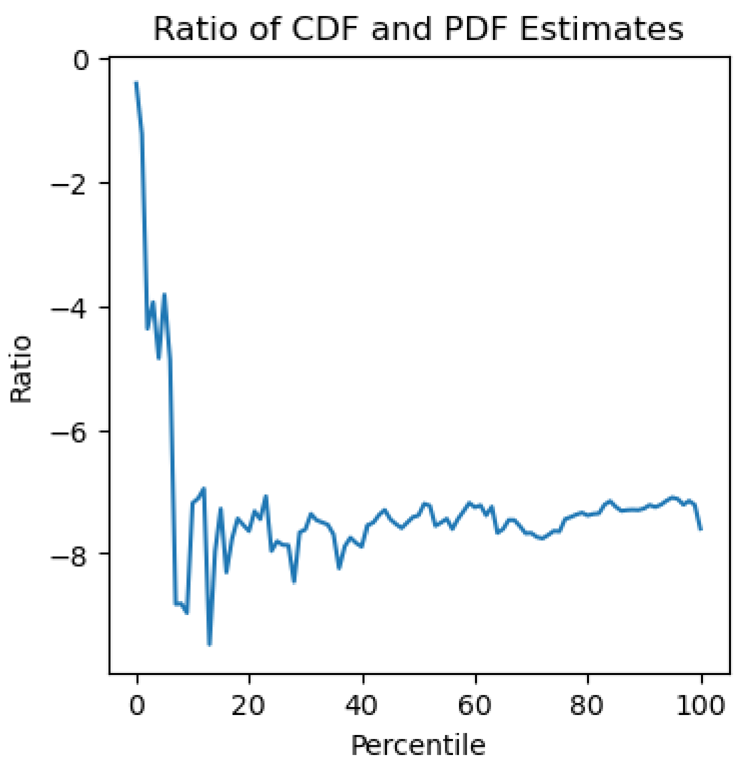
```
L1 distance (across experiment): 1.1498789494094162
L2 distance (across experiment): 1.675483656671308
L1 distance (across experiment): 3.8603540756253176
L2 distance (across experiment): 5.492224258244711
```

Risk Averse Structural Model for Valuation-Prediction

Computing Integer Percentiles of Bid Data

In [ ]:

```
# Construct and store 0 through 100 integer percentiles of the bid data.
# At each percentile, compute CDF and PDF estimates.
# Use these esimates to compute G(b;N=6)/(5*g(b;N=6)) -
G(b;N=3)/(2*g(b;N=3)) for each percentile.

# N=6
percentiles = np.linspace(0, 100, 101)
G6 = np.zeros(len(percentiles))
g6 = np.zeros(len(percentiles))
for i in range(len(percentiles)):
    G6[i] = np.percentile(data['BidC6'].values, percentiles[i])
    g6[i] = np.sum(data['BidC6'].values <=
G6[i])/len(data['BidC6'].values)
# N=3
G3 = np.zeros(len(percentiles))
g3 = np.zeros(len(percentiles))
for i in range(len(percentiles)):
    G3[i] = np.percentile(data['BidC3'].values, percentiles[i])
    g3[i] = np.sum(data['BidC3'].values <=
G3[i])/len(data['BidC3'].values)
# Compute the ratio
ratio = np.zeros(len(percentiles))
for i in range(len(percentiles)):
    ratio[i] = G6[i]/(5*g6[i]) - G3[i]/(2*g3[i])

# Plot the ratio
fig, ax = plt.subplots(1, 1, figsize=(4, 4))
ax.plot(percentiles, ratio)
ax.set_xlabel('Percentile')
ax.set_ylabel('Ratio')
ax.set_title('Ratio of CDF and PDF Estimates')
plt.show()
```



Estimating Parameters of Linear Model

In [ ]:
```
# Estimate theta = (b_{N=3} - b_{N=6})/ratio using OLS in three ways:
# (1) Using the full sample of percentiles
# (2) Using the percentiles 5 to 95
# (3) Using the percentiles 25 to 75
# Report point estimates and standard errors for each of the three
estimates

# (1) Using the full sample of percentiles
X = np.ones((len(percentiles), 1))
X[:, 0] = ratio
y = G3 - G6
theta1 = np.linalg.inv(X.T @ X) @ X.T @ y
print('theta1:', theta1)
# (2) Using the percentiles 5 to 95
X = np.ones((len(percentiles)-10, 1))
X[:, 0] = ratio[5:-5]
y = G3[5:-5] - G6[5:-5]
theta2 = np.linalg.inv(X.T @ X) @ X.T @ y
print('theta2:', theta2)
# (3) Using the percentiles 25 to 75
```

```
X = np.ones((len(percentiles)-50, 1))
X[:, 0] = ratio[25:-25]
y = G3[25:-25] - G6[25:-25]
theta3 = np.linalg.inv(X.T @ X) @ X.T @ y
print('theta3:', theta3)
```

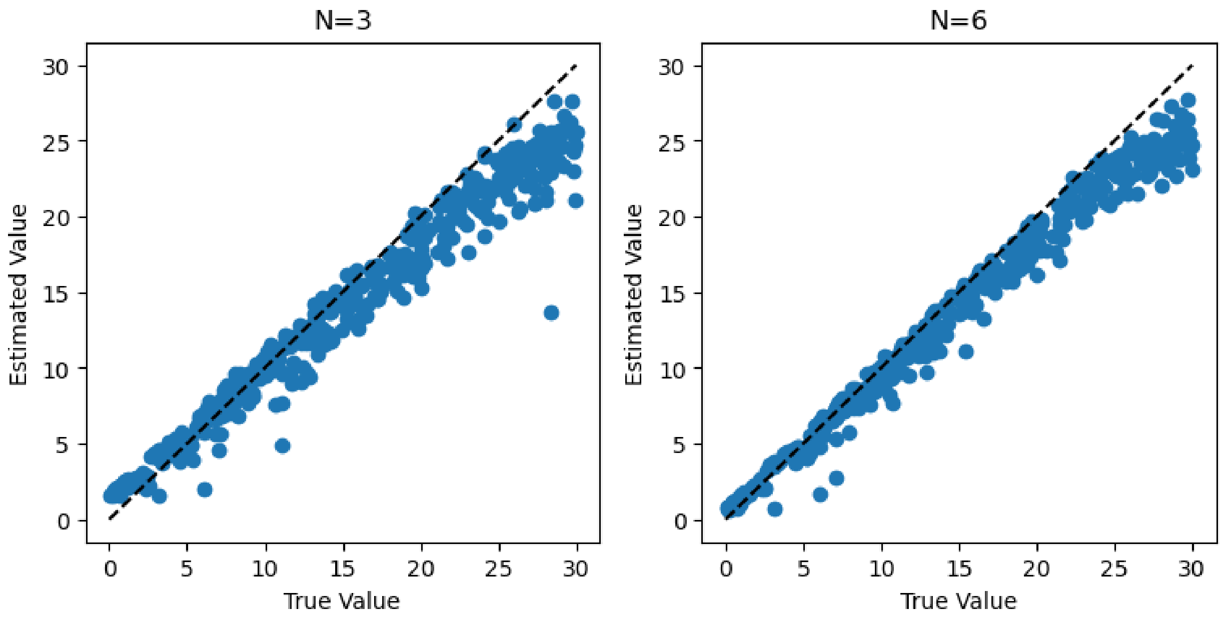
```
theta1: [0.11422194]
theta2: [0.11366252]
theta3: [0.11673834]
```

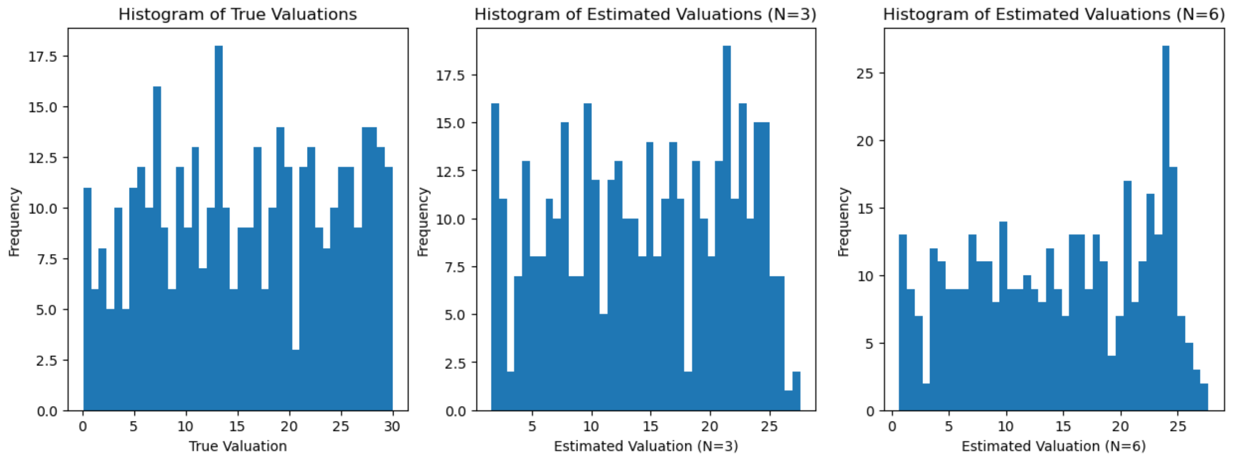
True Valuation vs. Model Estimated Valuations

In [ ]:

```
# First stretch g3 g6 G3 G6 to the same length as data['Value'].values
g3_stretch = np.zeros(len(data['Value'].values))
g6_stretch = np.zeros(len(data['Value'].values))
G3_stretch = np.zeros(len(data['Value'].values))
G6_stretch = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    g3_stretch[i] = g3[i//10]
    g6_stretch[i] = g6[i//10]
    G3_stretch[i] = G3[i//10]
    G6_stretch[i] = G6[i//10]
g3, g6, G3, G6 = g3_stretch, g6_stretch, G3_stretch, G6_stretch

# Compute estimates of the true value for N=3 and N=6 using theta3
def v_risk_averse(b, g, G, N, theta):
    return b + theta*(G/(g*(N-1)))
v3_risk_averse = np.zeros(len(data['BidC3'].values))
v6_risk_averse = np.zeros(len(data['BidC3'].values))
for i in range(len(data['BidC3'].values)):
    v3_risk_averse[i] = v_risk_averse(data['BidC3'].values[i], g3[i],
G3[i], 3, theta3)
    v6_risk_averse[i] = v_risk_averse(data['BidC6'].values[i], g6[i],
G6[i], 6, theta3)

# Also draw a scatter plot
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
ax[0].scatter(data['Value'].values, v3_risk_averse)
ax[0].plot([0, 30], [0, 30], 'k--')
ax[0].set_xlabel('True Value')
ax[0].set_ylabel('Estimated Value')
```

```
ax[0].set_title('N=3')
ax[1].scatter(data['Value'].values, v6_risk_averse)
ax[1].plot([0, 30], [0, 30], 'k--')
ax[1].set_xlabel('True Value')
ax[1].set_ylabel('Estimated Value')
ax[1].set_title('N=6')
plt.show()

# Plot histograms of the true valuations and the estimated valuations for
N=3 and N=6
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.hist(data['Value'].values, bins=40)
ax1.set_xlabel('True Valuation')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of True Valuations')
ax2.hist(v3_risk_averse, bins=40)
ax2.set_xlabel('Estimated Valuation (N=3)')
ax2.set_ylabel('Frequency')
ax2.set_title('Histogram of Estimated Valuations (N=3)')
ax3.hist(v6_risk_averse, bins=40)
ax3.set_xlabel('Estimated Valuation (N=6)')
ax3.set_ylabel('Frequency')
ax3.set_title('Histogram of Estimated Valuations (N=6)')
plt.show()

# Compute L1 and L2 distances between the true valuations and the
estimated valuations for N=3 and N=6
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v3_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v3_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=3):', L1)
print('L2 distance (N=3):', L2)
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v6_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v6_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=6):', L1)
print('L2 distance (N=6):', L2)
```



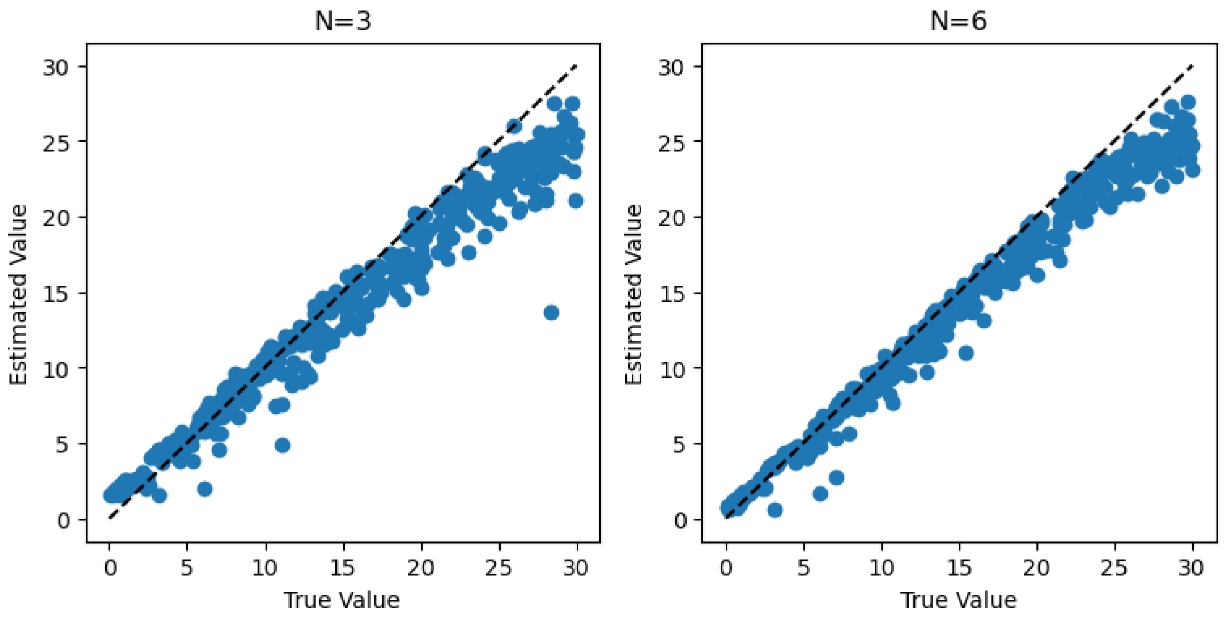


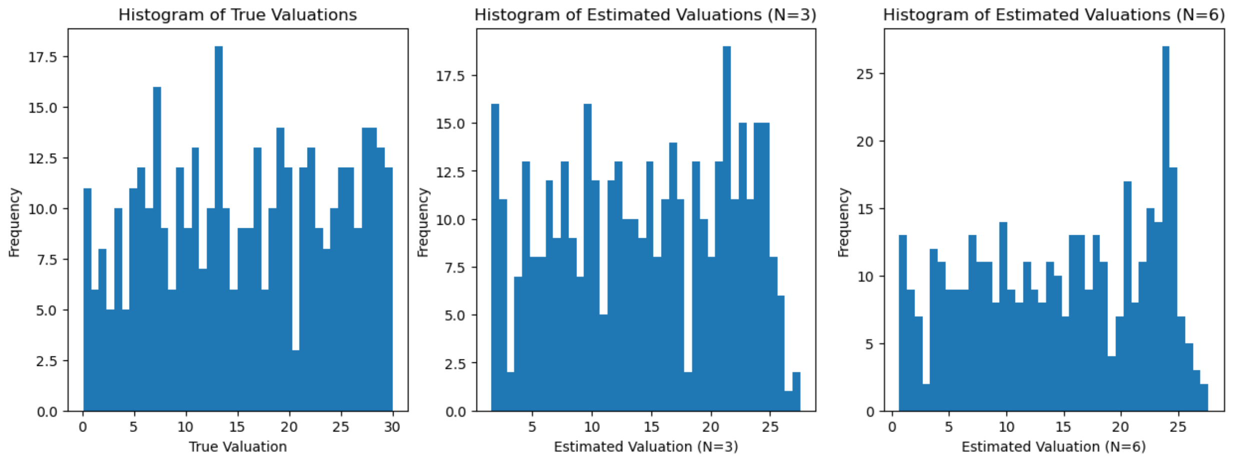
```
L1 distance (N=3): 1.8188910842957358
L2 distance (N=3): 2.4757687517537543
L1 distance (N=6): 1.4854413295904578
L2 distance (N=6): 2.042485947705061
```

In [ ]:

```
# Now we do the same for theta2
v3_risk_averse = np.zeros(len(data['BidC3'].values))
v6_risk_averse = np.zeros(len(data['BidC3'].values))
for i in range(len(data['BidC3'].values)):
    v3_risk_averse[i] = v_risk_averse(data['BidC3'].values[i], g3[i],
G3[i], 3, theta2)
```

```
    v6_risk_averse[i] = v_risk_averse(data['BidC6'].values[i], g6[i],
G6[i], 6, theta2)

# Also draw a scatter plot
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
ax[0].scatter(data['Value'].values, v3_risk_averse)
ax[0].plot([0, 30], [0, 30], 'k--')
ax[0].set_xlabel('True Value')
ax[0].set_ylabel('Estimated Value')
ax[0].set_title('N=3')
ax[1].scatter(data['Value'].values, v6_risk_averse)
ax[1].plot([0, 30], [0, 30], 'k--')
ax[1].set_xlabel('True Value')
ax[1].set_ylabel('Estimated Value')
ax[1].set_title('N=6')
plt.show()

# Plot histograms of the true valuations and the estimated valuations for
N=3 and N=6
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.hist(data['Value'].values, bins=40)
ax1.set_xlabel('True Valuation')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of True Valuations')
ax2.hist(v3_risk_averse, bins=40)
ax2.set_xlabel('Estimated Valuation (N=3)')
ax2.set_ylabel('Frequency')
ax2.set_title('Histogram of Estimated Valuations (N=3)')
ax3.hist(v6_risk_averse, bins=40)
ax3.set_xlabel('Estimated Valuation (N=6)')
ax3.set_ylabel('Frequency')
ax3.set_title('Histogram of Estimated Valuations (N=6)')
plt.show()

# Compute L1 and L2 distances between the true valuations and the
estimated valuations for N=3 and N=6
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v3_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v3_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=3):', L1)
print('L2 distance (N=3):', L2)
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v6_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v6_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=6):', L1)
print('L2 distance (N=6):', L2)
```



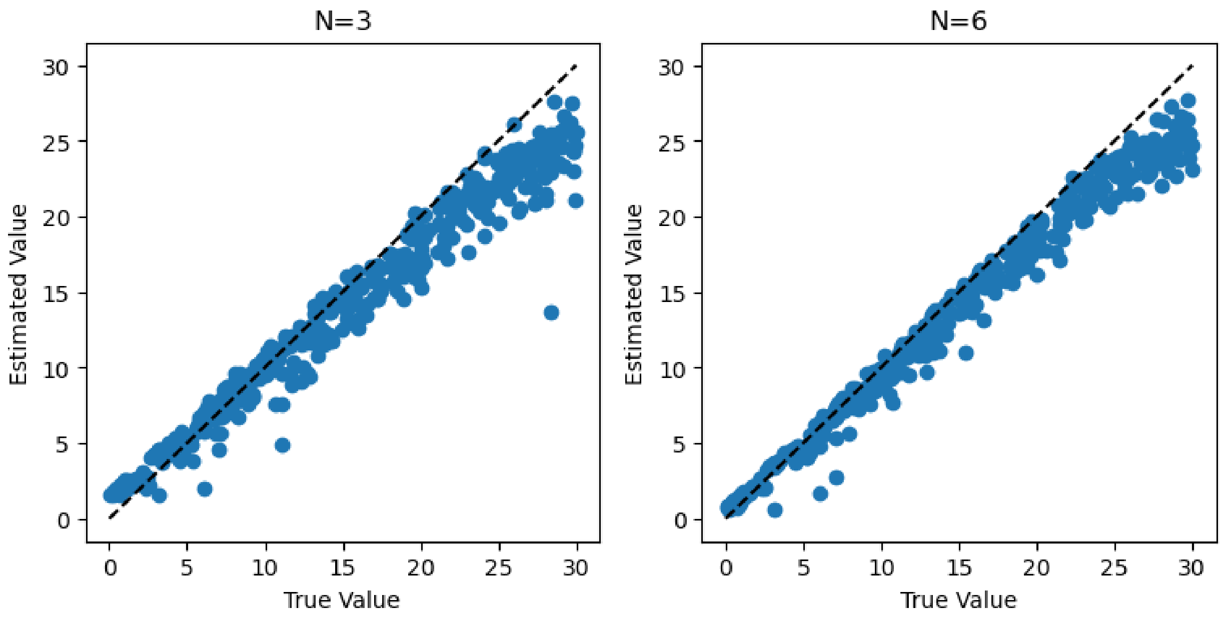


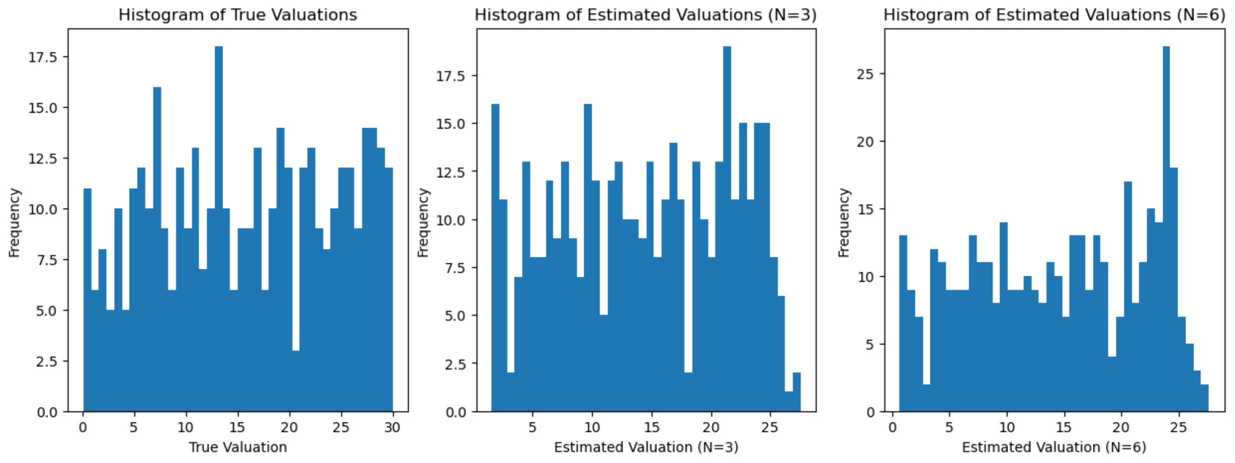
```
L1 distance (N=3): 1.8344837133706948
L2 distance (N=3): 2.495590283569006
L1 distance (N=6): 1.494878520919001
L2 distance (N=6): 2.0532185511334604
```

In [ ]:

```
# And now theta1
v3_risk_averse = np.zeros(len(data['BidC3'].values))
v6_risk_averse = np.zeros(len(data['BidC3'].values))
for i in range(len(data['BidC3'].values)):
    v3_risk_averse[i] = v_risk_averse(data['BidC3'].values[i], g3[i],
G3[i], 3, theta1)
    v6_risk_averse[i] = v_risk_averse(data['BidC6'].values[i], g6[i],
G6[i], 6, theta1)

# Also draw a scatter plot
fig, ax = plt.subplots(1, 2, figsize=(9, 4))
ax[0].scatter(data['Value'].values, v3_risk_averse)
ax[0].plot([0, 30], [0, 30], 'k--')
ax[0].set_xlabel('True Value')
ax[0].set_ylabel('Estimated Value')
ax[0].set_title('N=3')
ax[1].scatter(data['Value'].values, v6_risk_averse)
ax[1].plot([0, 30], [0, 30], 'k--')
ax[1].set_xlabel('True Value')
ax[1].set_ylabel('Estimated Value')
ax[1].set_title('N=6')
plt.show()

# Plot histograms of the true valuations and the estimated valuations for
N=3 and N=6
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.hist(data['Value'].values, bins=40)
ax1.set_xlabel('True Valuation')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of True Valuations')
ax2.hist(v3_risk_averse, bins=40)
ax2.set_xlabel('Estimated Valuation (N=3)')
ax2.set_ylabel('Frequency')
ax2.set_title('Histogram of Estimated Valuations (N=3)')
ax3.hist(v6_risk_averse, bins=40)
```

```
ax3.set_xlabel('Estimated Valuation (N=6)')
ax3.set_ylabel('Frequency')
ax3.set_title('Histogram of Estimated Valuations (N=6)')
plt.show()

# Compute L1 and L2 distances between the true valuations and the
estimated valuations for N=3 and N=6
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v3_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v3_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=3):', L1)
print('L2 distance (N=3):', L2)
L1 = np.zeros(len(data['Value'].values))
L2 = np.zeros(len(data['Value'].values))
for i in range(len(data['Value'].values)):
    L1[i] = np.abs(data['Value'].values[i] - v6_risk_averse[i])
    L2[i] = (data['Value'].values[i] - v6_risk_averse[i])**2
L1 = np.mean(L1)
L2 = np.sqrt(np.mean(L2))
print('L1 distance (N=6):', L1)
print('L2 distance (N=6):', L2)
```





```
L1 distance (N=3): 1.8316417292893707
L2 distance (N=3): 2.491949955501836
L1 distance (N=6): 1.4931505891529544
L2 distance (N=6): 2.051259835598088
```