Empirical Industrial Organisations I: PSet 2

S M Sajid Al Sanai

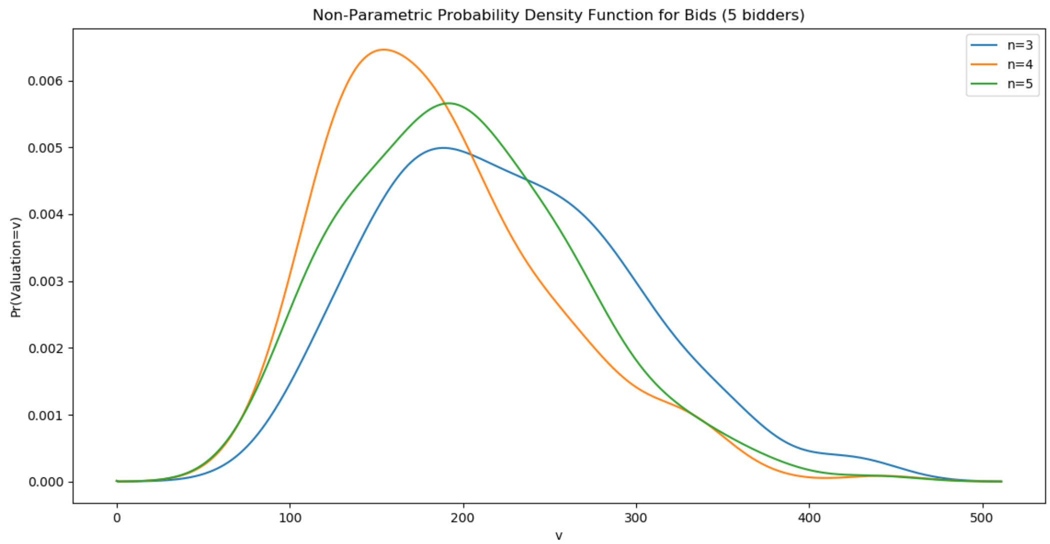
November 18, 2018

Contents

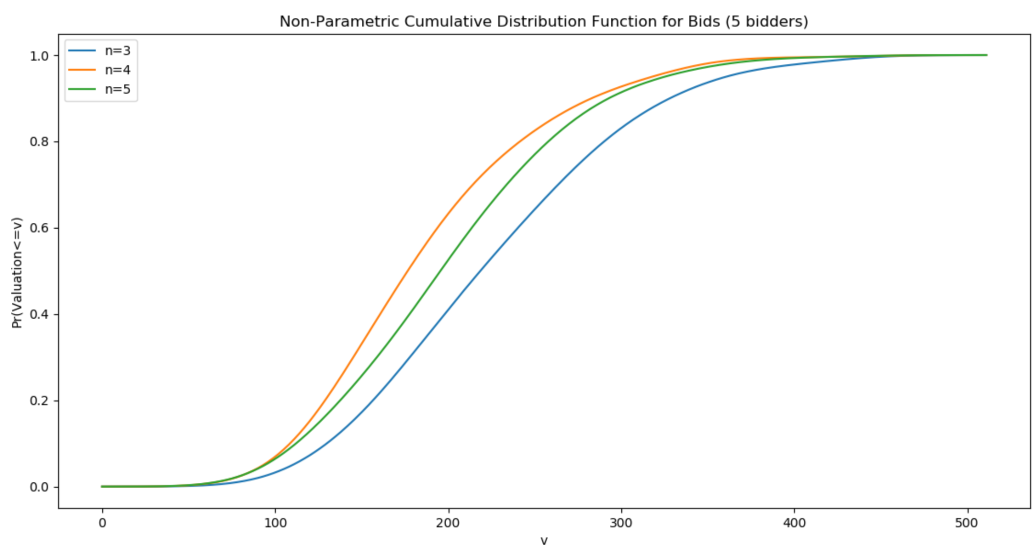
1 Question I

1.1 Distributions

Code is included in Appendix: Source, with Cumulative Distribution Function being calculated using a canned package for Non-Parametric Kernel Density Estimation. This case represents an Ascending Auction under the Symmetric Independent Private Values with no reserve price and the button model under Milgrom and Weber (1982).



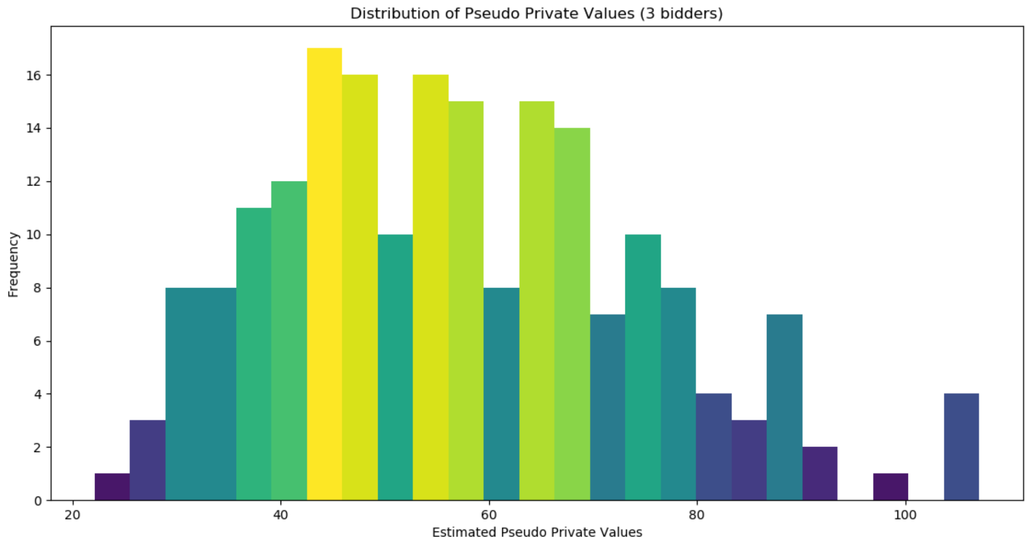
Three individual Probability Density Functions were produced to coincide with each set of simulated auctions of a particular bidder count of n. The number of bidders are described as $n = \{3, 4, 5\}$.

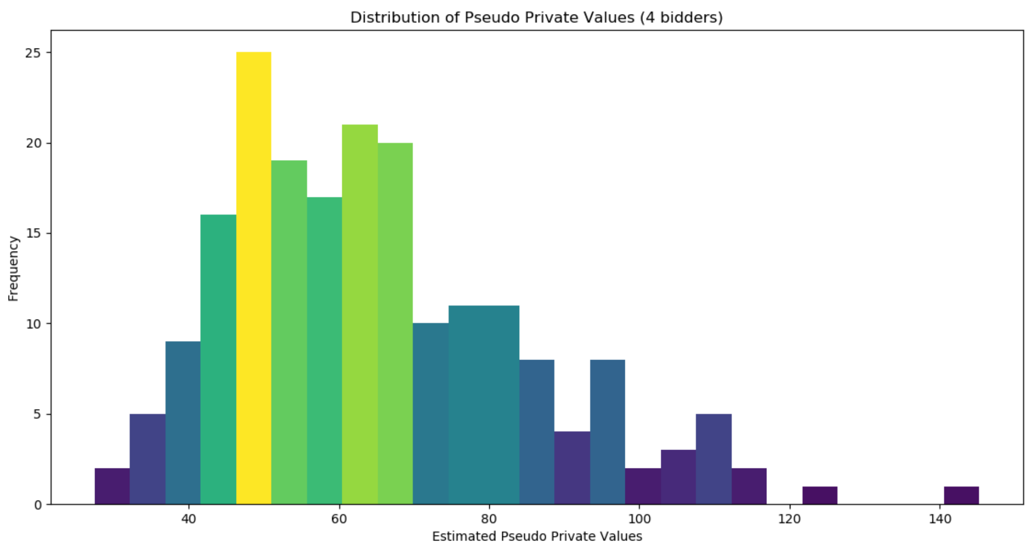


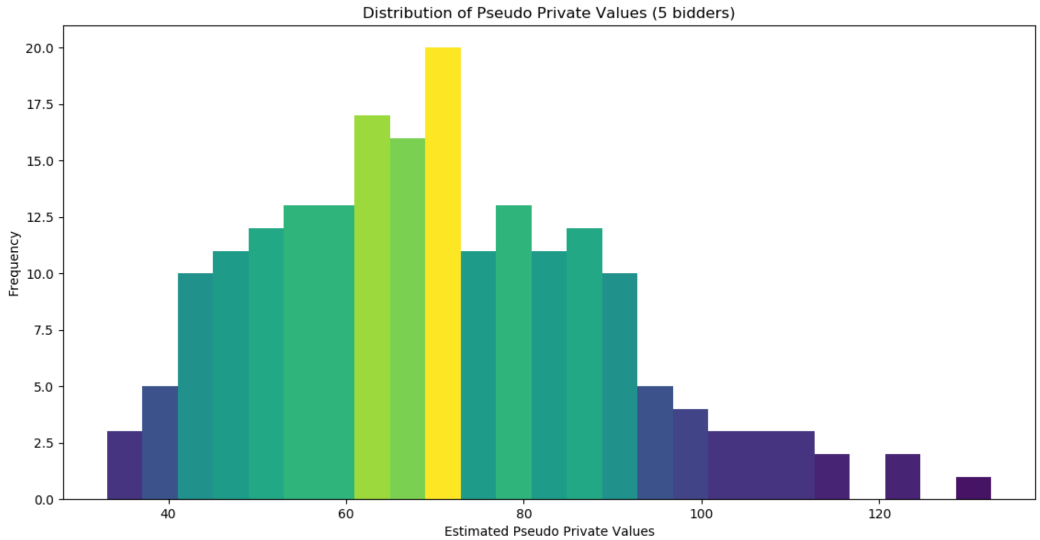
Three individual Cumulative Distribution Functions were produced to coincide with each set of simulated auctions of a particular bidder count of n. Auctions with a bidder count of five appear to first order stochastically dominate auctions with alternate bidder counts.

1.2 Estimated Private Values

Distributions for Estimated Private Values subject to bidder count for a set of auction simulations are depicted below.



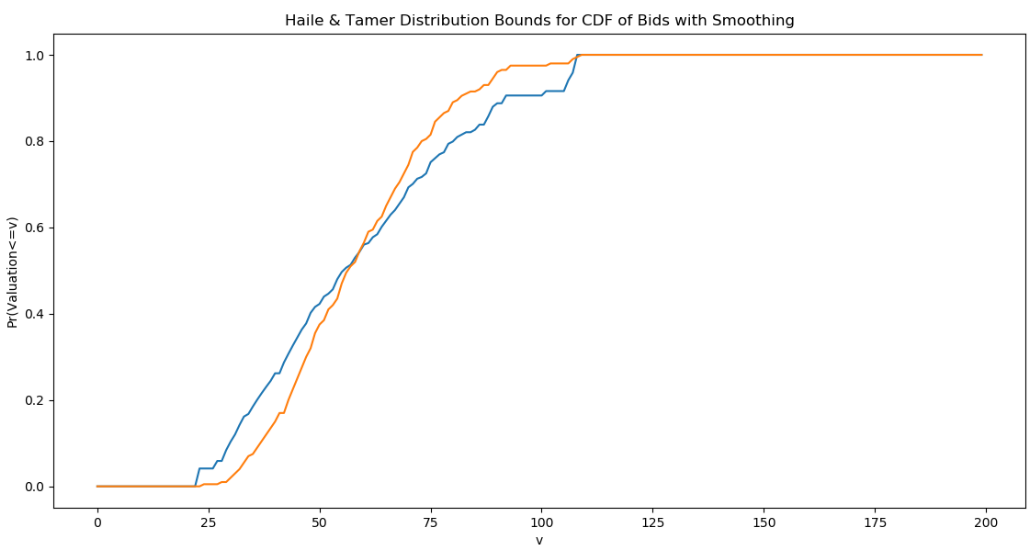




2 Question II

2.1 Bounds on Distributions for Private Values

Using the Haile and Tamer (2003) approach on the prior dataset, I calculated (albeit incorrectly), the upper and lower bounds for the Cumulative Distribution Function of bids using a bid increment of $\Delta = 1.0$. The crossing of the lower bound above the upper bound at a single point highlights an error in the procedure.

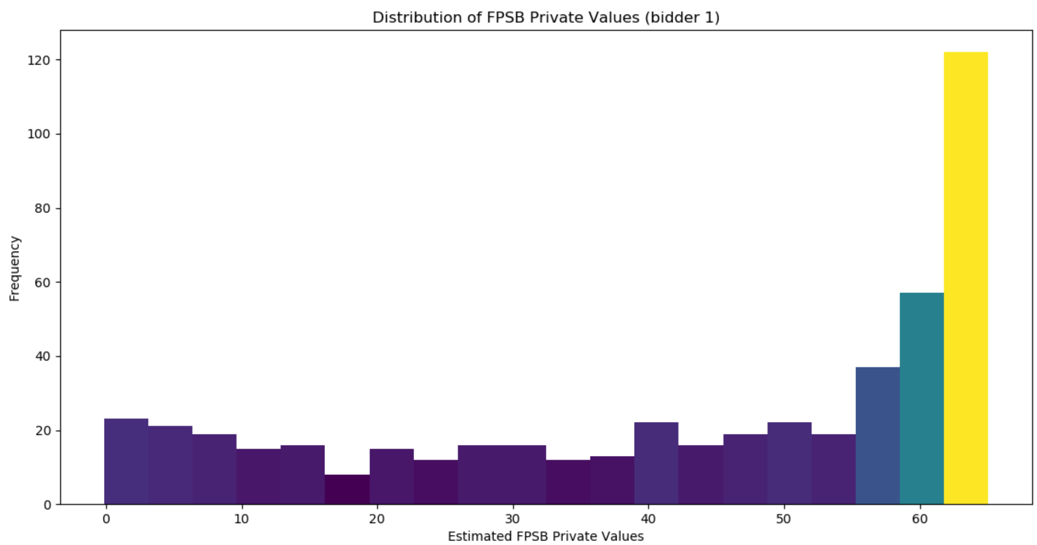


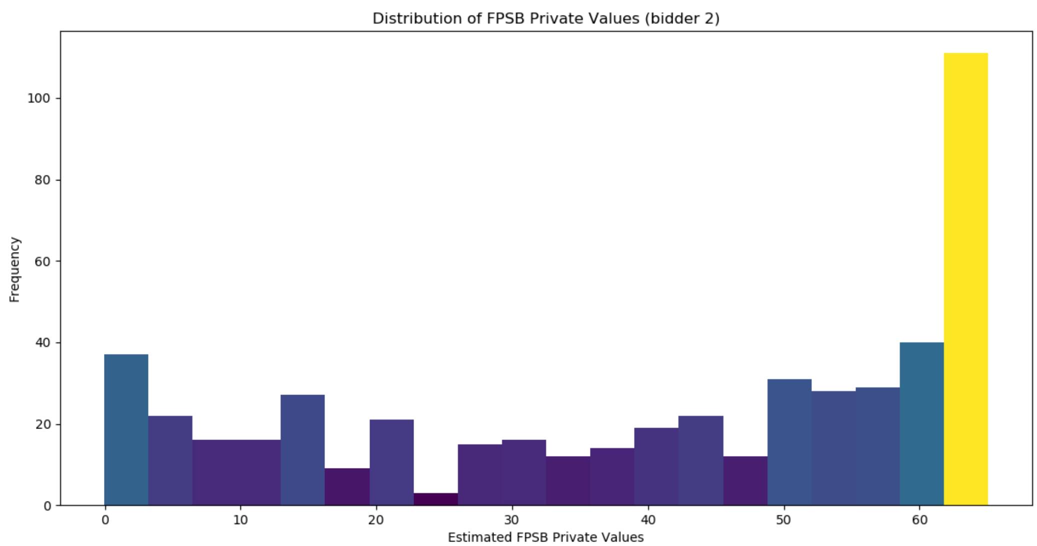
3 Question III

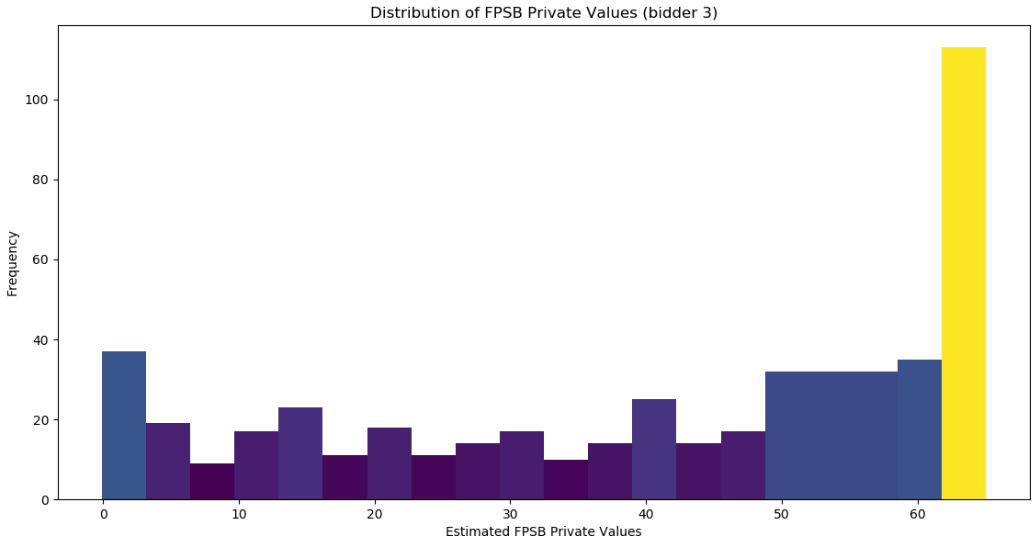
Relevant code may be found in the Appendix: Source.

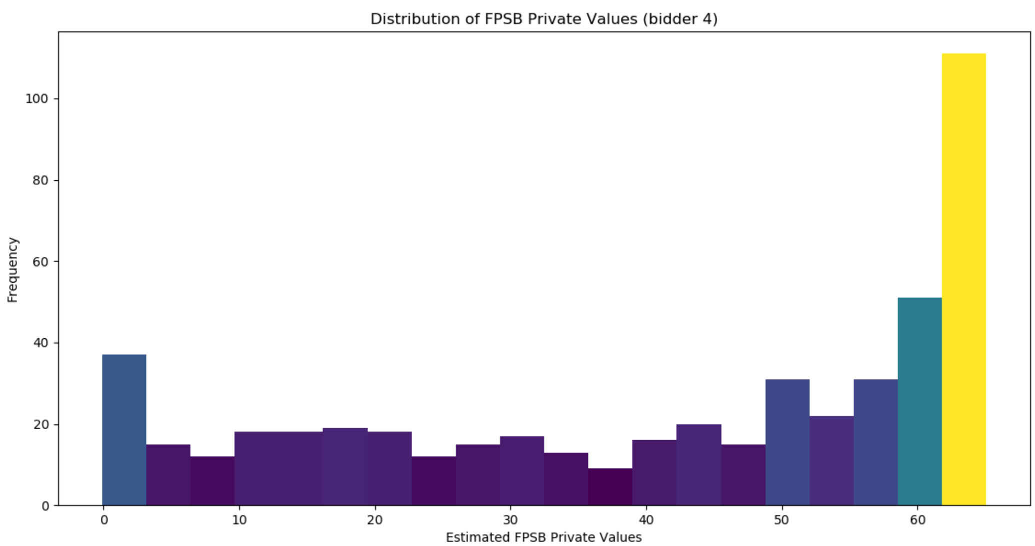
3.1 Private Values and Vector Permutations

Estimation was conducted using the Guerre, Perrigne, and Vuong approach using Kernel Density Estimation (Triweight Kernel Function on support $|u| < 1$) and iteratively calculated Bandwidths h_f, h_g. Estimated distributions and densites were calculated for $\hat{G}, \hat{g}, \hat{F}, \hat{f}$.









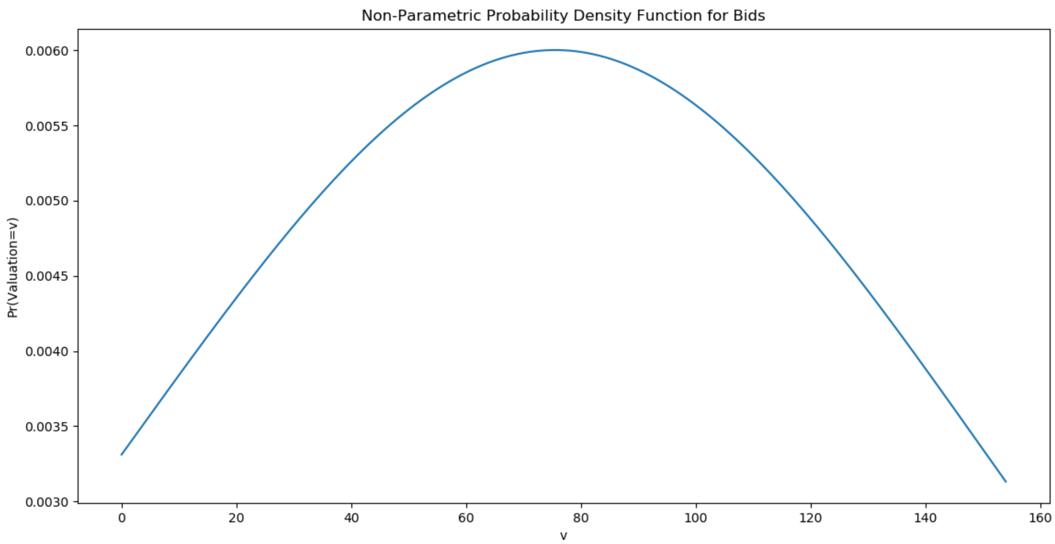
Code snippet below attempts to find $F_U(u_1; u_2; h_3; h_4)$ for 2^4 permutations of the vector (u_1, u_2, u_3, u_4) containing percentiles of the marginal density F_U. It does not appear that the independence assumption necessarily holds.

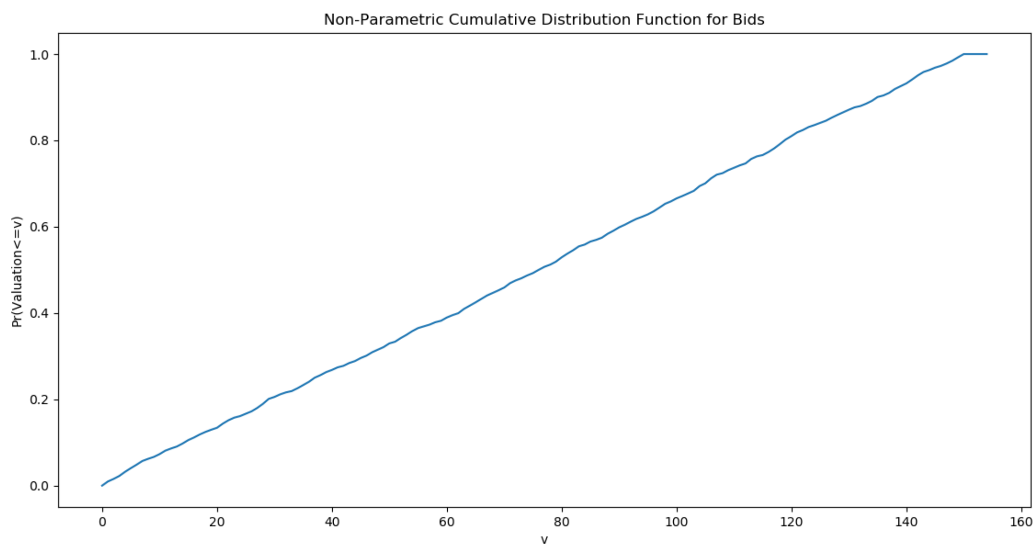
```
[0]
u_pc:  [25.  25.  25.  25.]
u_i:   [34.70032361  34.70032361  34.70032361  34.70032361]
Gu_i:  16.56288984454293
Fu_i:  0.00390625

[1]
u_pc:  [25.  25.  25.  75.]
u_i:   [ 34.70032361   34.70032361   34.70032361  139.11512202]
```

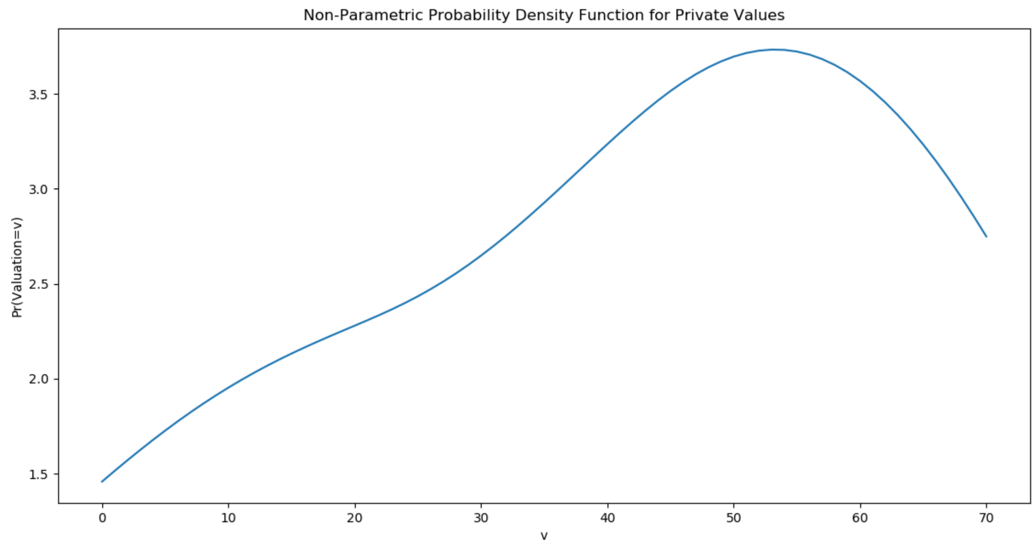
```
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[2]
u_pc: [25. 25. 75. 25.]
u_i:  [ 34.70032361  34.70032361 139.11512202  34.70032361]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[3]
u_pc: [25. 25. 75. 75.]
u_i:  [ 34.70032361  34.70032361 139.11512202 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[4]
u_pc: [25. 75. 25. 25.]
u_i:  [ 34.70032361 139.11512202  34.70032361  34.70032361]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[5]
u_pc: [25. 75. 25. 75.]
u_i:  [ 34.70032361 139.11512202  34.70032361 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[6]
u_pc: [25. 75. 75. 25.]
u_i:  [ 34.70032361 139.11512202 139.11512202  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[7]
u_pc: [25. 75. 75. 75.]
u_i:  [ 34.70032361 139.11512202 139.11512202 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[8]
u_pc: [75. 25. 25. 25.]
u_i:  [139.11512202  34.70032361  34.70032361  34.70032361]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[9]
u_pc: [75. 25. 25. 75.]
u_i:  [139.11512202  34.70032361  34.70032361 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[10]
u_pc: [75. 25. 75. 25.]
u_i:  [139.11512202  34.70032361 139.11512202  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[11]
u_pc: [75. 25. 75. 75.]
```

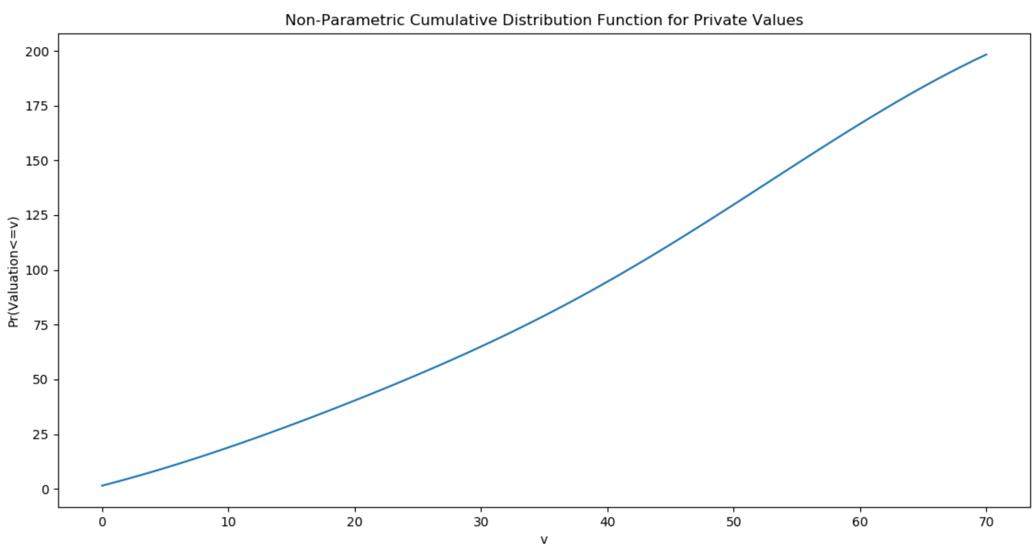
```
u_i:  [139.11512202  34.70032361 139.11512202 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[12]
u_pc: [75. 75. 25. 25.]
u_i:  [139.11512202 139.11512202  34.70032361  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[13]
u_pc: [75. 75. 25. 75.]
u_i:  [139.11512202 139.11512202  34.70032361 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[14]
u_pc: [75. 75. 75. 25.]
u_i:  [139.11512202 139.11512202 139.11512202  34.70032361]
Gu_i: 48.89307452123868
Fu_i: 0.10546875

[15]
u_pc: [75. 75. 75. 75.]
u_i:  [139.11512202 139.11512202 139.11512202 139.11512202]
Gu_i: 59.66980274680394
Fu_i: 0.31640625
% Code Snippet
```

3.2 Symmetric Independent Private Value Assumption






Non-Parametric Probability Density Function for Private Values
Pr(Valuation=v)
v
3.5
3.0
2.5
2.0
1.5
0
10
20
30
40
50
60
70


Non-Parametric Cumulative Distribution Function for Private Values
Pr(Valuation<=v)
v
200
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70

4 Appendix: Source Code

4.1 Output

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Dell\Documents\Graduate - Economics\Empirical Industrial Organisation I\
    pset2_working\pset2_SMSajidAlSanai.py

Question 1:

T  = 600
Tn = [3. 4. 5.]
     [200 200 200]

Outputting pdf of Bids ...
(512, 3)
[[8.28417691e-06 1.09694037e-05 1.24888043e-05]
 [1.13442873e-06 1.74005966e-06 1.92487070e-06]
 [1.27767885e-06 1.99282896e-06 2.19829780e-06]
 ...
 [2.84301712e-06 1.31576181e-06 1.22845582e-06]
 [2.53536149e-06 1.16024315e-06 1.09005998e-06]
 [2.25789807e-06 1.02120094e-06 9.65634803e-07]]

Generating Plot of Distribution for pdf of Bids ...
Graph [3]

Outputting CDF of Bids ...
(512, 3)
[[8.28417691e-06 1.09694037e-05 1.24888043e-05]
 [9.41860565e-06 1.27094634e-05 1.44136750e-05]
 [1.06962845e-05 1.47022923e-05 1.66119728e-05]
 ...
 [9.99978496e-01 9.99991069e-01 9.99991195e-01]
 [9.99981031e-01 9.99992229e-01 9.99992285e-01]
 [9.99983289e-01 9.99993251e-01 9.99993250e-01]]

Generating Plot of Distribution for CDF of Bids ...
Graph [3]

Outputting Pseudo Private Values ...
[[ 57.89238034 104.43838192  87.25874964]
 [ 37.78273212  52.3086597   81.06134497]
 [ 52.75195193  76.44402781  49.32285624]
 [ 57.41133248  58.73302322  83.71018525]
 [ 45.66640759  62.61320167  70.9184098 ]
 [ 63.04878363  86.4870121   92.39247511]
 [ 59.21375671  73.24788403  46.67644676]
 [ 85.21782192  67.30714195  83.58223426]
 [ 30.71777227  45.12185483  81.07888688]
 [ 41.60057586  53.56047755  89.07540914]
 [ 28.89576421  61.65412945 122.18239814]
 [ 46.94006277  66.34728864  49.82593209]
 [ 65.64302569  58.6195803  109.14792255]
 [ 30.2389162   94.67539543  56.54510649]
 [ 82.94169251  66.22863092  90.72383652]
 [ 62.11680239  59.80260158  82.66437694]
```

```
[ 67.34551554  48.33295947  56.06763865]
[ 35.21230928  54.64054645  53.95107415]
[ 43.99148135  87.9530864   48.75234939]
[ 39.77389067  64.00388312  70.46862059]
[ 69.13454632  68.37892038  67.26676024]
[ 47.85249789  68.74425561  41.18131352]
[ 31.1070219   97.33960255  62.41595926]
[ 63.33870413  68.49356396  70.41792217]
[ 35.45917187  53.9196028   76.34293494]
[ 87.77200033 113.92608623  46.02333521]
[ 69.12182903  46.34325904  75.30342048]
[ 61.06645487  47.31230412  81.6132006 ]
[ 53.36277043  67.53849931  60.40925891]
[ 60.23538875  51.02610363  46.01074639]
[ 71.36237263  63.25483546  66.23279104]
[ 28.01154076  68.18448069  70.25786575]
[ 49.90795518 110.91142484  78.86139447]
[ 68.77744359  76.27954952  68.86413664]
[ 28.99118014  81.37482644  67.12348592]
[106.3183872   53.92107105  57.45276796]
[ 53.57999619 106.28941985 114.8063389 ]
[ 88.46128853  40.71327373  62.54489262]
[ 45.62292601  69.93192384  63.52590458]
[ 70.45784015  36.13167308  78.83556909]
[ 74.01357071  56.65331998 105.77005294]
[ 41.34463156  53.59575481  57.73939884]
[ 34.0484885   65.941971    48.31108781]
[ 84.10438028  81.40039859  83.46775692]
[ 50.43310494  69.66865866  84.46414854]
[ 66.26612304  39.93663758  64.13233699]
[ 69.37162386  58.91211728 106.42477283]
[ 67.50957687  93.47138719  79.30308329]
[ 71.50236864  94.43542523  53.43358721]
[ 74.41054316  36.67785008 109.45961786]
[ 88.98048117  95.59651424  64.3532586 ]
[ 47.97101483  46.67196744  43.61037103]
[ 66.96462139  95.30083415  65.96862264]
[ 44.62066778  48.96780195 112.33444635]
[ 45.60353017  69.42509502  62.02246435]
[ 47.69466464  56.02211151  98.44095994]
[ 69.27096764  76.83256932  58.13095378]
[104.57482342  82.14907738  70.2355565 ]
[ 58.04997376  68.07154993  92.30793908]
[ 51.10640882  48.8645627   90.16370195]
[ 41.32282364  88.43174151  86.45348346]
[ 79.44104056  76.5820803   51.95685741]
[ 38.4147099   27.44842961  49.9009317 ]
[ 56.93195175  75.57853982  65.44241942]
[ 80.86356966  42.75350384  76.26547539]
[ 44.2988799   46.49227971  68.22582973]
[ 71.68420298  71.23337266  41.02538765]
[ 53.84648164  57.51800452  68.20961098]
[ 49.95542124  39.08991101  44.8662464 ]
[ 45.55651152  56.13220132  60.34495923]
[ 99.66972597  60.51961056  95.51490682]
[ 58.94033093  47.97024025 132.60346458]
[ 74.61110739  60.47793374  62.52653461]
[ 37.36250202 102.57902518  43.18479569]
[ 91.03410123 102.52269581  71.86592445]
```

```
[ 42.39915871  77.60627181  64.51976949]
[ 53.79416254  49.65050757  96.21781305]
[ 62.28684144  82.29380461  65.48526232]
[ 68.38441981  45.11854828  38.77339207]
[ 87.86375351  53.60257391  73.17549088]
[ 51.16746658  60.50386342  54.48316934]
[ 46.88520128  55.83851497  65.95109343]
[ 54.90115303  62.59708505  95.25348247]
[ 57.216663    57.49241279  61.6579749 ]
[ 25.52873395  45.48959287  64.30742793]
[ 47.93064394  46.71134938  69.11205159]
[ 47.05946929  39.0601871   61.25150372]
[ 88.42902518  43.21953992  86.40396181]
[ 44.6808564   71.85674789 106.36262554]
[ 87.44354702  89.255397    53.78923943]
[ 66.40857714  72.23216137  73.6309593 ]
[ 67.76559505  57.27317218  75.96920056]
[ 64.06462411  61.41229698  97.94271123]
[ 47.45031928  61.98767448  46.427516  ]
[ 69.39501782  56.69010592  97.50048294]
[ 52.92008924  37.97192799  71.95030678]
[ 48.23505633  56.46348138  77.73082003]
[ 66.00127921  46.14635476  44.45854472]
[ 33.76270912  36.63988125 104.71224828]
[ 43.0488031   63.62309533  57.44087932]
[ 78.41297636  53.26466202  89.13916927]
[ 39.43925887  40.85783594  38.55356433]
[ 42.1312154   44.85218294  83.80331703]
[ 62.97859567  81.23114573  59.65868967]
[ 73.87580101  71.18569805  79.99611054]
[ 36.71639092  65.14692217  51.32177583]
[ 43.965286    34.00558466  95.37681199]
[ 42.5970299   74.77110891  56.74611191]
[ 64.0708302   54.18775912  70.48866685]
[ 61.37122796  48.12828039  62.16580336]
[ 54.59996793  88.758642    40.53604443]
[ 78.46743817  66.43873969  69.75520319]
[ 68.2041054   53.13686362  50.47671828]
[ 59.30939878  42.30647654  48.5135871 ]
[ 37.61374441  93.36292257  37.82805404]
[ 50.19692766 111.86905856  61.69259968]
[ 61.34597426  70.29013441  68.6054717 ]
[ 58.92011284  48.69649062  51.01026851]
[ 36.4298007   90.2609917   42.08674387]
[ 43.20288131  37.90996344  72.66418271]
[ 64.95107585  43.19079231  50.12532651]
[ 90.96068855  45.49511349  55.57257877]
[ 80.8885089   59.92346798 114.90604904]
[ 32.28480469  60.97404929  68.6780493 ]
[ 52.57280531  59.79060921  78.85336173]
[ 85.87273451  60.49587004  44.9938431 ]
[ 29.07161577  49.10162305  79.58073657]
[ 38.92397987  63.25418112  88.35388086]
[ 75.43475475  69.38850347  41.83765027]
[ 50.44028875  86.46263082  85.21209884]
[ 52.42703723  29.57821243  69.08166054]
[ 35.72310226 111.17553282  94.08079333]
[ 41.2711842   42.46988508  68.84143861]
[ 58.36286228  50.52989634 101.38337469]
```

```
[  34.14264562   47.82864459   54.08134346]
[  56.78279891  110.77312489   79.16823032]
[105.63682006   77.02782849   49.99665495]
[  78.70160931   53.72390576   69.86435461]
[  46.96735692   59.66465235   77.48786657]
[  63.63998959   46.05299749   71.07388022]
[  74.34607295   74.6693367    69.4151991 ]
[  39.2566261    53.4633695    50.32455143]
[  38.14602956  107.19292607   45.51491026]
[107.12461166   66.18059938   71.62925909]
[  58.91009315   46.93586865   60.53268643]
[  54.31069843   72.51867108   60.4862078 ]
[  55.49044382   88.10250777   77.09031202]
[  44.21951188   37.51099781   62.55896977]
[  74.64954987  122.19854912   53.99700619]
[  72.20527256   43.25159254   56.38898315]
[  31.24642036   51.35900512   36.03147801]
[  32.7514827    41.81372447   92.58811601]
[  73.42469813   69.77856128   58.0368053 ]
[  50.0060028    86.36401437   81.45195836]
[  46.22012484   61.60400103   86.86957073]
[  32.58402283   83.66512183   46.85427907]
[  65.04417619   73.76281521   35.46798719]
[  59.96289504   79.53471427   70.30841203]
[  57.51942308   49.29415577   49.03184086]
[  36.60706427   60.15752229   76.81282927]
[  87.91898288   61.91403357   65.70338103]
[  33.30719801   75.68489404   64.4890813 ]
[  22.11646989   71.97081994   68.76289412]
[  38.96937635   52.63924997   45.86121691]
[  45.69551091   96.53748924   89.19226479]
[  46.27736893   47.16816882   77.36776206]
[  48.99160852  108.03708793   74.33373198]
[  41.85421917   65.3578509    90.97736652]
[  48.62131474   87.99117561   58.88973076]
[  56.86455606   40.13667284   54.28885488]
[  78.20396084   53.42681941   75.87814683]
[  64.63868407   68.4918896    86.00430228]
[  77.12572141   49.54894706   70.05620975]
[  43.24021726   42.67838688   77.77935676]
[  73.57277194   35.05517973   62.52920445]
[  65.46998923   46.70572686   88.5363626 ]
[  64.33666815   96.08961869   72.53544743]
[  69.93050718   50.09302398   67.89213177]
[  70.22103401   66.36089809   81.95955121]
[  53.54691923   63.38229678   33.10721628]
[  43.26099325   49.45219432   99.68913251]
[  53.1031396    51.21358607   60.70034759]
[  75.80745754   86.16000363   80.00039172]
[  46.98029892  145.28287893   87.74567758]
[  55.07276886   81.78691542   43.21646231]
[  80.9379042    45.35894901   50.77772593]
[  76.70746249   62.26988145   62.93947108]
[  54.30505966   54.94407436  102.33394998]
[  42.43904544   69.18430222   87.35730181]
[  42.7147434    81.92609604   43.29687303]
[  54.91997993   49.73594583   57.5862397 ]
[  59.48324182   63.55660489   72.3427526 ]
[  47.5988528    76.31479663  122.39784079]
```

```
 [ 41.92773468  49.63687561  75.89179222]
 [ 68.26121723 112.99949518  89.36355887]
 [ 63.44907976  55.62072613  88.01919233]
 [ 31.34471971  81.44481054  85.06743083]
 [ 61.16266931  83.66672252  75.91566107]
 [ 54.02490562  50.00606994  47.63787853]
 [ 76.86569435  61.07947678  55.43128371]]

Generating Histogram of Distribution for Pseudo Private Values ...
Graph [1]
Graph [2]
Graph [3]


Question 2:

[G^_i=2:n=3] (200,)
[G^_i=2:n=4] (200,)
[G^_i=2:n=5] (200,)

[G^_i=2:n=3,4,5]
[[0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.     0.     0.    ]
 [0.005  0.     0.    ]
 [0.005  0.     0.    ]
 [0.005  0.     0.    ]
 [0.005  0.     0.    ]
 [0.01   0.     0.    ]
 [0.01   0.     0.    ]
 [0.02   0.005  0.    ]
 [0.03   0.01   0.    ]
 [0.04   0.01   0.    ]
 [0.055  0.01   0.    ]
 [0.07   0.01   0.    ]
 [0.075  0.01   0.005]
 [0.09   0.015  0.005]
 [0.105  0.02   0.01 ]
 [0.12   0.035  0.015]
```

```
[0.135 0.045 0.02 ]
[0.15  0.05  0.03 ]
[0.17  0.06  0.03 ]
[0.17  0.08  0.035]
[0.2   0.085 0.05 ]
[0.225 0.105 0.055]
[0.25  0.12  0.075]
[0.275 0.125 0.085]
[0.3   0.15  0.1  ]
[0.32  0.19  0.125]
[0.355 0.205 0.13 ]
[0.375 0.23  0.145]
[0.385 0.27  0.165]
[0.41  0.285 0.19 ]
[0.42  0.3   0.2  ]
[0.435 0.31  0.205]
[0.47  0.36  0.22 ]
[0.495 0.375 0.24 ]
[0.51  0.385 0.25 ]
[0.52  0.41  0.27 ]
[0.545 0.425 0.29 ]
[0.565 0.44  0.305]
[0.59  0.46  0.31 ]
[0.595 0.49  0.335]
[0.615 0.515 0.35 ]
[0.625 0.535 0.39 ]
[0.65  0.56  0.395]
[0.67  0.565 0.42 ]
[0.69  0.575 0.435]
[0.705 0.605 0.45 ]
[0.725 0.615 0.465]
[0.745 0.645 0.5  ]
[0.775 0.675 0.525]
[0.785 0.68  0.565]
[0.8   0.7   0.58 ]
[0.805 0.71  0.6  ]
[0.815 0.72  0.61 ]
[0.845 0.73  0.615]
[0.855 0.74  0.635]
[0.865 0.765 0.655]
[0.87  0.775 0.68 ]
[0.89  0.775 0.695]
[0.895 0.78  0.71 ]
[0.905 0.78  0.72 ]
[0.91  0.81  0.745]
[0.915 0.82  0.75 ]
[0.915 0.83  0.77 ]
[0.92  0.83  0.775]
[0.93  0.83  0.785]
[0.93  0.85  0.805]
[0.945 0.86  0.82 ]
[0.96  0.875 0.835]
[0.965 0.88  0.855]
[0.965 0.885 0.87 ]
[0.975 0.885 0.87 ]
[0.975 0.885 0.885]
[0.975 0.895 0.885]
[0.975 0.905 0.89 ]
[0.975 0.915 0.905]
```

```
[0.975 0.925 0.91 ]
[0.975 0.93  0.915]
[0.975 0.93  0.925]
[0.975 0.93  0.93 ]
[0.98  0.93  0.93 ]
[0.98  0.93  0.935]
[0.98  0.93  0.94 ]
[0.98  0.94  0.94 ]
[0.98  0.945 0.94 ]
[0.99  0.945 0.95 ]
[0.995 0.95  0.96 ]
[1.    0.955 0.96 ]
[1.    0.96  0.96 ]
[1.    0.96  0.97 ]
[1.    0.965 0.97 ]
[1.    0.975 0.97 ]
[1.    0.98  0.975]
[1.    0.985 0.975]
[1.    0.99  0.975]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
```

```
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]
 [1.        1.        1.       ]]

[G^_i=3:n=3:del=1] (200,)
[G^_i=4:n=4:del=1] (200,)
[G^_i=5:n=5:del=1] (200,)

[G^_i=n:n=3,4,5:del=1]
[[0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
 [0.        0.        0.       ]
```

```
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.    0.    0.   ]
[0.005 0.    0.   ]
[0.005 0.    0.   ]
[0.005 0.    0.   ]
[0.005 0.    0.   ]
[0.01  0.    0.   ]
[0.01  0.    0.   ]
[0.02  0.005 0.   ]
[0.03  0.01  0.   ]
[0.04  0.01  0.   ]
[0.055 0.01  0.   ]
[0.07  0.01  0.   ]
[0.075 0.01  0.005]
[0.09  0.015 0.005]
[0.105 0.02  0.01 ]
[0.12  0.035 0.015]
[0.135 0.045 0.02 ]
[0.15  0.05  0.03 ]
[0.17  0.06  0.03 ]
[0.17  0.08  0.035]
[0.2   0.085 0.05 ]
[0.225 0.105 0.055]
[0.25  0.12  0.075]
[0.275 0.125 0.085]
[0.3   0.15  0.1  ]
[0.32  0.19  0.125]
[0.355 0.205 0.13 ]
[0.375 0.23  0.145]
[0.385 0.27  0.165]
[0.41  0.285 0.19 ]
[0.42  0.3   0.2  ]
[0.435 0.31  0.205]
[0.47  0.36  0.22 ]
[0.495 0.375 0.24 ]
[0.51  0.385 0.25 ]
[0.52  0.41  0.27 ]
[0.545 0.425 0.29 ]
[0.565 0.44  0.305]
[0.59  0.46  0.31 ]
[0.595 0.49  0.335]
[0.615 0.515 0.35 ]
[0.625 0.535 0.39 ]
[0.65  0.56  0.395]
[0.67  0.565 0.42 ]
[0.69  0.575 0.435]
```

```
[0.705 0.605 0.45 ]
[0.725 0.615 0.465]
[0.745 0.645 0.5  ]
[0.775 0.675 0.525]
[0.785 0.68  0.565]
[0.8   0.7   0.58 ]
[0.805 0.71  0.6  ]
[0.815 0.72  0.61 ]
[0.845 0.73  0.615]
[0.855 0.74  0.635]
[0.865 0.765 0.655]
[0.87  0.775 0.68 ]
[0.89  0.775 0.695]
[0.895 0.78  0.71 ]
[0.905 0.78  0.72 ]
[0.91  0.81  0.745]
[0.915 0.82  0.75 ]
[0.915 0.83  0.77 ]
[0.92  0.83  0.775]
[0.93  0.83  0.785]
[0.93  0.85  0.805]
[0.945 0.86  0.82 ]
[0.96  0.875 0.835]
[0.965 0.88  0.855]
[0.965 0.885 0.87 ]
[0.975 0.885 0.87 ]
[0.975 0.885 0.885]
[0.975 0.895 0.885]
[0.975 0.905 0.89 ]
[0.975 0.915 0.905]
[0.975 0.925 0.91 ]
[0.975 0.93  0.915]
[0.975 0.93  0.925]
[0.975 0.93  0.93 ]
[0.98  0.93  0.93 ]
[0.98  0.93  0.935]
[0.98  0.93  0.94 ]
[0.98  0.94  0.94 ]
[0.98  0.945 0.94 ]
[0.99  0.945 0.95 ]
[0.995 0.95  0.96 ]
[1.    0.955 0.96 ]
[1.    0.96  0.96 ]
[1.    0.96  0.97 ]
[1.    0.965 0.97 ]
[1.    0.975 0.97 ]
[1.    0.98  0.975]
[1.    0.985 0.975]
[1.    0.99  0.975]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.99  0.985]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
```

```
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 0.995]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    0.995 1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
[1.    1.    1.   ]
```

```
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]
 [1.     1.     1.    ]]

Calculating Estimated Distributions forming Upper and Lower Bounds ...
Generating Plot of Bounds on Distribution for CDF of Bids with Smoothing ...


Question 3:

No. of Auctions: 500
No. of Bidders:  4

Outputting pdf of Bids ...
(155,)
[0.00331097 0.00336405 0.00341713 0.00347021 0.00352326 0.00357626
 0.00362921 0.00368207 0.00373484 0.0037875  0.00384004 0.00389242
 0.00394465 0.00399669 0.00404854 0.00410018 0.00415158 0.00420274
 0.00425362 0.00430422 0.00435451 0.00440446 0.00445407 0.00450331
 0.00455215 0.00460058 0.00464856 0.00469608 0.00474312 0.00478965
 0.00483564 0.00488108 0.00492593 0.00497018 0.0050138  0.00505677
 0.00509906 0.00514064 0.00518151 0.00522162 0.00526096 0.00529951
 0.00533724 0.00537413 0.00541016 0.0054453  0.00547954 0.00551286
 0.00554522 0.00557662 0.00560703 0.00563644 0.00566483 0.00569217
 0.00571845 0.00574366 0.00576777 0.00579078 0.00581267 0.00583342
 0.00585302 0.00587146 0.00588872 0.0059048  0.00591969 0.00593336
 0.00594583 0.00595707 0.00596708 0.00597585 0.00598338 0.00598967
 0.0059947  0.00599848 0.006001   0.00600226 0.00600226 0.006001
 0.00599848 0.0059947  0.00598967 0.00598339 0.00597586 0.00596708
 0.00595707 0.00594583 0.00593337 0.00591969 0.0059048  0.00588871
 0.00587144 0.005853   0.00583338 0.00581262 0.00579072 0.0057677
 0.00574356 0.00571834 0.00569203 0.00566466 0.00563625 0.00560681
 0.00557636 0.00554492 0.00551251 0.00547915 0.00544485 0.00540965
 0.00537355 0.00533659 0.00529878 0.00526015 0.00522071 0.00518049
 0.00513952 0.00509781 0.00505539 0.00501229 0.00496852 0.0049241
 0.00487907 0.00483345 0.00478725 0.00474051 0.00469324 0.00464547
 0.00459722 0.00454851 0.00449936 0.0044498  0.00439985 0.00434953
 0.00429885 0.00424784 0.00419652 0.0041449  0.00409301 0.00404085
 0.00398845 0.00393583 0.003883   0.00382997 0.00377677 0.00372342
 0.00366992 0.00361631 0.00356259 0.00350878 0.00345491 0.00340098
 0.00334703 0.00329306 0.0032391  0.00318516 0.00313126]

Generating Plot of Distribution for pdf of Bids ...

Outputting CDF of Bids ...
(155,)
[0.     0.0095 0.0155 0.0225 0.032  0.0405 0.0485 0.057  0.062  0.0665
 0.073  0.081  0.086  0.0905 0.0975 0.1055 0.1115 0.1185 0.1245 0.1295
```

```
 0.134   0.1435 0.1515 0.1575 0.161   0.1665 0.172   0.18    0.1895 0.201
 0.2055 0.2115 0.216   0.219   0.2255 0.233   0.2405 0.25    0.256   0.263
 0.268   0.274   0.2775 0.284   0.2885 0.2955 0.301   0.309   0.315   0.321
 0.3295 0.3335 0.342   0.3495 0.358   0.365   0.369   0.373   0.3785 0.382
 0.3895 0.395   0.3995 0.4095 0.417   0.4245 0.4325 0.4405 0.4465 0.4525
 0.459   0.469   0.4755 0.4805 0.487   0.4925 0.5     0.507   0.512   0.519
 0.529   0.5375 0.5455 0.5545 0.5585 0.5655 0.5695 0.5745 0.5835 0.5905
 0.5985 0.6045 0.6115 0.618   0.623   0.6285 0.6355 0.644   0.653   0.6585
 0.6655 0.671   0.677   0.683   0.694   0.7005 0.712   0.7205 0.724   0.731
 0.7365 0.742   0.7465 0.757   0.763   0.766   0.773   0.7815 0.7915 0.802
 0.81    0.8185 0.824   0.831   0.8355 0.8405 0.8455 0.8525 0.859   0.865
 0.871   0.8765 0.8795 0.885   0.8915 0.9005 0.904   0.91    0.919   0.9255
 0.932   0.941   0.9505 0.9585 0.963   0.9685 0.9725 0.978   0.9845 0.9925
 1.      1.      1.      1.      1.    ]

Generating Plot of Distribution for CDF of Bids ...

Outputting FPSB Private Values ...
[[63.39634469 47.33367471 39.35760182 62.435337  ]
 [27.05139294 20.929711    3.2330617  61.4927359 ]
 [64.61940672 56.66778458 56.06738446 60.99606182]
 ...
 [ 1.572058   53.0154454  27.06744229 31.59649118]
 [33.52282657 58.4615557   9.73519983 57.60495064]
 [64.4089873   6.72552346 63.72368918 64.72023724]]

Generating Histogram of Distribution for FPSB Private Values ...
Graph [1]
Graph [2]
Graph [3]
Graph [4]

Outputting pdf of Private Values  ...
[1.4588881  1.51530007 1.57049691 1.62433874 1.67669072 1.72742335
 1.77641508 1.82356081 1.86879822 1.91210338 1.95347367 1.99293041
 2.03051858 2.06630973 2.10040442 2.13294011 2.16409351 2.19407504
 2.22312554 2.25154328 2.27968497 2.30799152 2.337042   2.36745764
 2.39976109 2.43435425 2.47152821 2.51147315 2.55428747 2.59998645
 2.64851026 2.69973142 2.75346161 2.80945795 2.86742866 2.92703813
 2.98791137 3.04963796 3.11177528 3.17385127 3.23536651 3.29579575
 3.35458887 3.41117116 3.46494329 3.51528802 3.56160075 3.60332985
 3.63998755 3.67113358 3.69637015 3.7153434  3.72774476 3.73331608
 3.73184311 3.72314752 3.707102   3.68362979 3.65269574 3.6143042
 3.56850948 3.5154305  3.45523735 3.38813305 3.31434725 3.23414829
 3.14785302 3.05582275 2.95846443 2.85622306 2.74957246]

Generating Plot of Distribution for pdf of Private Values ...

Outputting CDF of Private Values...
[   1.4588881    2.97418817    4.54468508    6.16902382    7.84571455
    9.5731379   11.34955298   13.17311378   15.041912     16.95401538
   18.90748905   20.90041946   22.93093804   24.99724776   27.09765218
   29.23059229   31.3946858    33.58876084   35.81188638   38.06342966
   40.34311464   42.65110616   44.98814816   47.3556058    49.75536689
   52.18972114   54.66124935   57.17272251   59.72700998   62.32699643
   64.9755067    67.67523811   70.42869972   73.23815767   76.10558633
   79.03262445   82.02053582   85.07017378   88.18194906   91.35580033
   94.59116684   97.8869626   101.24155146  104.65272262  108.11766591
  111.63295393  115.19455468  118.79788453  122.43787208  126.10900565
```

```
 129.80537581 133.52071921 137.24846397 140.98178006 144.71362317
 148.43677069 152.1438727  155.82750249 159.48019823 163.09450243
 166.66301191 170.1784424  173.63367975 177.0218128  180.33616005
 183.57030834 186.71816136 189.77398412 192.73244855 195.58867161
 198.33824406]

Generating Plot of Distribution for CDF of Private Values ...

[0]
u_pc: [25. 25. 25. 25.]
u_i:  [34.70032361 34.70032361 34.70032361 34.70032361]
Gu_i: 16.56288984454293
Fu_i: 0.00390625

[1]
u_pc: [25. 25. 25. 75.]
u_i:  [ 34.70032361  34.70032361  34.70032361 139.11512202]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[2]
u_pc: [25. 25. 75. 25.]
u_i:  [ 34.70032361  34.70032361 139.11512202  34.70032361]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[3]
u_pc: [25. 25. 75. 75.]
u_i:  [ 34.70032361  34.70032361 139.11512202 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[4]
u_pc: [25. 75. 25. 25.]
u_i:  [ 34.70032361 139.11512202  34.70032361  34.70032361]
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[5]
u_pc: [25. 75. 25. 75.]
u_i:  [ 34.70032361 139.11512202  34.70032361 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[6]
u_pc: [25. 75. 75. 25.]
u_i:  [ 34.70032361 139.11512202 139.11512202  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[7]
u_pc: [25. 75. 75. 75.]
u_i:  [ 34.70032361 139.11512202 139.11512202 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[8]
u_pc: [75. 25. 25. 25.]
u_i:  [139.11512202  34.70032361  34.70032361  34.70032361]
```

```
Gu_i: 27.33961807010818
Fu_i: 0.01171875

[9]
u_pc: [75. 25. 25. 75.]
u_i:  [139.11512202  34.70032361  34.70032361 139.11512202]
Gu_i: 38.11634629567344
Fu_i: 0.03515625

[10]
u_pc: [75. 25. 75. 25.]
u_i:  [139.11512202  34.70032361 139.11512202  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[11]
u_pc: [75. 25. 75. 75.]
u_i:  [139.11512202  34.70032361 139.11512202 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[12]
u_pc: [75. 75. 25. 25.]
u_i:  [139.11512202 139.11512202  34.70032361  34.70032361]
Gu_i: 38.11634629567343
Fu_i: 0.03515625

[13]
u_pc: [75. 75. 25. 75.]
u_i:  [139.11512202 139.11512202  34.70032361 139.11512202]
Gu_i: 48.89307452123869
Fu_i: 0.10546875

[14]
u_pc: [75. 75. 75. 25.]
u_i:  [139.11512202 139.11512202 139.11512202  34.70032361]
Gu_i: 48.89307452123868
Fu_i: 0.10546875

[15]
u_pc: [75. 75. 75. 75.]
u_i:  [139.11512202 139.11512202 139.11512202 139.11512202]
Gu_i: 59.66980274680394
Fu_i: 0.31640625


***
[EOF] Output Terminates
***
>>>
```

4.2 Source

```
# Import Libraries
import math
import numpy as np
import scipy as sp
from scipy import optimize
```

```
from scipy import io
import statsmodels.api as sm
import matplotlib.pyplot as plt
from matplotlib import colors

# Question 1:
print('Question 1:')
print('')

# Import Dataset
dataset_file = 'ascending_data.dat'
dataset_raw  = open( dataset_file , 'rt' )
dataset_data = np.genfromtxt( dataset_raw , dtype=(float , float), delimiter=None, names='num_bidder

# Dataset Characteristics
T = dataset_data.size

# Generating New ndarray Variables from Dataset
num_bidders = np.reshape( dataset_data['num_bidders'], (T, 1) )
price_paid  = np.reshape( dataset_data['price_paid'],  (T, 1) )

# Dataset Characteristics
Tn_value , Tn_num = np.unique( num_bidders , return_counts=True )

print( 'T  = ' + str( T ) )
print( 'Tn = ' + str( Tn_value ) )
print( '     ' + str( Tn_num ) )
print('')

# GPV Non-Parametric Estimation Method
def ObtainPseudoPrivateValue( bids , i , cdf , pdf ):
    out = np.zeros( (Tn_num[0],) )
    for bi in range( Tn_num[i] ):
        Gb = ObtainKernelCDF( cdf , bi / Tn_num[i] )
        gb = ObtainKernelpdf( bids[Tn_num[i] * i + bi] )
        out[bi] = bids[Tn_num[i] * i + bi] - ( 1 / ( Tn_value[i] - 1 ) ) * ( Gb / gb )
    return out

def ObtainKernelpdf( bid ):
    density = sm.nonparametric.KDEUnivariate( price_paid )
    density.fit()
    return density.evaluate( bid )

def ObtainKernelCDF( cdf , bid ):
    i = int( np.round( bid * cdf.shape[0] ) )
    return cdf[i]

def EstimateKernelpdf( arr ):
    density = sm.nonparametric.KDEUnivariate( arr )
    density.fit()
    t_cdf = density.cdf
    out   = np.zeros( t_cdf.shape )
    out[0] = t_cdf[0]
    for i in range( (density.cdf).shape[0] - 1 ):
        out[i + 1] = t_cdf[i + 1] - t_cdf[i]
    return out

def EstimateKernelCDF( arr ):
    density = sm.nonparametric.KDEUnivariate( arr )
```

```
    density.fit()
    return density.cdf

# Conduct Kernel Density Distribution
bids_pdf = np.zeros( (512, Tn_value.size) )
bids_cdf = np.zeros( (512, Tn_value.size) )
for i in range( Tn_value.size ):
    bids_pdf[:, i] = EstimateKernelpdf( price_paid[Tn_num[i] * i : Tn_num[i] * (i + 1)] )
    bids_cdf[:, i] = EstimateKernelCDF( price_paid[Tn_num[i] * i : Tn_num[i] * (i + 1)] )

# Draw Graph of Cumulative Distribution
print( 'Outputting pdf of Bids ...' )
print( bids_pdf.shape )
print( bids_pdf )
print('')

print( 'Generating Plot of Distribution for pdf of Bids ...' )
for i in range( Tn_value.size ):
    plt.plot( bids_pdf[:, i], label='n=' + str(3+i) )
print( 'Graph [' + str(i+1) + ']' )
plt.title('Non-Parametric Probability Density Function for Bids (' + str(3+i) + ' bidders)')
plt.xlabel('v')
plt.ylabel('Pr(Valuation=v)')
plt.legend()
plt.show()
print('')

# Draw Graph of Cumulative Distribution
print( 'Outputting CDF of Bids ...' )
print( bids_cdf.shape )
print( bids_cdf )
print('')

print( 'Generating Plot of Distribution for CDF of Bids ...' )
for i in range( Tn_value.size ):
    plt.plot( bids_cdf[:, i], label='n=' + str(3+i) )
print( 'Graph [' + str(i+1) + ']' )
plt.title('Non-Parametric Cumulative Distribution Function for Bids (' + str(3+i) + ' bidders)')
plt.xlabel('v')
plt.ylabel('Pr(Valuation<=v)')
plt.legend()
plt.show()
print('')

# Estimate Pseudo Private Values
pseudo_private_values = np.zeros( (Tn_num[0], Tn_value.size) )
for i in range( Tn_value.size ):
    pseudo_private_values[:, i] = ObtainPseudoPrivateValue( price_paid, i, bids_pdf[:, i], bids_cd
print( 'Outputting Pseudo Private Values ...' )
print( pseudo_private_values )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Histogram of Distribution for Pseudo Private Values ...' )
num_bins = 25
for i in range( Tn_value.size ):
    print( 'Graph [' + str(i+1) + ']' )
    N, bins, patches = plt.hist(pseudo_private_values[:, i], bins=num_bins)
    fracs = N / N.max()
```

```python
    norm  = colors.Normalize( fracs.min(), fracs.max() )
    for ifrac, ipatch in zip( fracs, patches ):
        color = plt.cm.viridis( norm( ifrac ) )
        ipatch.set_facecolor( color )
    plt.title('Distribution of Pseudo Private Values (' + str(3+i) + ' bidders)')
    plt.xlabel('Estimated Pseudo Private Values')
    plt.ylabel('Frequency')
    plt.show()
print('')
print('')


# Question 2:
print('Question 2:')
print('')

# Haile & Tamer Estimation Method
def EstimateG( n, v, delta ):
    out   = 0
    t_sum = 0
    condition_one = False
    condition_two = False
    for t in range( Tn_num[0] ):
        t_index = (np.where(Tn_value == n))[0] * Tn_num[0] + t
        if num_bidders[ t_index ] == n:
            condition_one = True
        if price_paid[ t_index ] + delta <= v:
            condition_two = True
        if condition_one == True and condition_two == True:
            t_sum += 1
        condition_one = False
        condition_two = False
    out = ( 1 / Tn_num[ (np.where(Tn_value == n))[0] ] ) * t_sum
    return out

def MonotoneOperation( cdf, i ):
    out         = np.zeros( (cdf.shape[0], cdf.shape[1]) )
    coefficient = np.zeros( (Tn_value.size,) )
    for k in range( Tn_value.size ):
        for j in range( Tn_num[0] ):
            t_polynom = np.zeros( (int(Tn_value[k]) + 1,) )
            if Tn_value[k] == 3:
                t_polynom = [-1/3, 1/2, 0, ( math.factorial( 3 - i ) * math.factorial( i - 1 ) ) /
            if Tn_value[k] == 4:
                t_polynom = [1/4, -2/3, 1/2, 0, ( math.factorial( 4 - i ) * math.factorial( i - 1
            if Tn_value[k] == 5:
                t_polynom = [-1/5, 3/4, -1, 1/2, 0, ( math.factorial( 5 - i ) * math.factorial( i
            t_polynom[ int(Tn_value[k]) ] *= -cdf[j, k]
            t_root = np.roots( t_polynom )
            t_root = np.unique( np.real( t_root[ (t_root >= 0) & (t_root <= 1) ] ) )
            out[j, k] = np.max( t_root )
    return out

def WeightedAverage( y, rho ):
    t_result_vector1 = np.zeros( (Tn_num[0], Tn_value.size) )
    t_result_vector2 = np.zeros( (Tn_num[0], Tn_value.size) )
    t_result_vector3 = np.zeros( (Tn_num[0], Tn_value.size) )
    t_denominator    = np.zeros( (Tn_num[0],) )
    out              = np.zeros( (Tn_num[0],) )
```

```
    for k in range( Tn_num[0] ):
        for j in range( Tn_value[0].size ):
            t_result_vector1[k, j] = np.exp( y[k, j] * rho )
            t_result_vector2[k, j] = np.exp( y[k, j] * rho ) * y[k, j]
        t_denominator[k] = np.sum( t_result_vector1[k, :] )
        t_result_vector3[k, j] = t_result_vector2[k, j] / t_denominator[k]
        out[k] = np.sum( t_result_vector3[k, :] )
    return out

# Estimate our distributions over differing values v for each n
DELTA = 1.0
CDFU  = np.zeros( (Tn_num[0], Tn_value.size) )
CDFL  = np.zeros( (Tn_num[0], Tn_value.size) )

for i in range( Tn_value.size ):
    for v in range( Tn_num[0] ):
        CDFU[v, i] = EstimateG( Tn_value[i], v, 0 )
        CDFL[v, i] = EstimateG( Tn_value[i], v, DELTA )

print( '[G^_i=2:n=3] ' + str( CDFU[:, 0].shape ) )
print( '[G^_i=2:n=4] ' + str( CDFU[:, 1].shape ) )
print( '[G^_i=2:n=5] ' + str( CDFU[:, 2].shape ) )
print('')

print( '[G^_i=2:n=3,4,5]' )
print(CDFU)
print('')

print( '[G^_i=3:n=3:del=1] ' + str( CDFL[:, 0].shape ) )
print( '[G^_i=4:n=4:del=1] ' + str( CDFL[:, 1].shape ) )
print( '[G^_i=5:n=5:del=1] ' + str( CDFL[:, 2].shape ) )
print('')

print( '[G^_i=n:n=3,4,5:del=1]' )
print(CDFL)
print('')

print('Calculating Estimated Distributions forming Upper and Lower Bounds ...')
FU = WeightedAverage( MonotoneOperation( CDFU, 2 ), -700 )
FL = WeightedAverage( CDFL, 700 )

print( 'Generating Plot of Bounds on Distribution for CDF of Bids with Smoothing ...' )
plt.plot(FU)   # Blue   # Issue lay here previously
plt.plot(FL)   # Orange
plt.title('Haile & Tamer Distribution Bounds for CDF of Bids with Smoothing')
plt.xlabel('v')
plt.ylabel('Pr(Valuation<=v)')
plt.show()

print('')
print('')


# Question 3:
print('Question 3:')
print('')

# Import Dataset
dataset_file = 'fpa.dat'
```

```python
dataset_raw  = open( dataset_file , 'rt' )
dataset_data = np.genfromtxt( dataset_raw , dtype=(float , float ), delimiter=None, names='bids1 , bid

# Dataset Characteristics
num_auctions = dataset_data.shape[0]
num_bidders  = 4

print( 'No. of Auctions: ' + str( num_auctions ) )
print( 'No. of Bidders:  ' + str( num_bidders  ) )
print('')

# Generating New ndarray Variables from Dataset
bids = np.zeros( shape=(num_auctions , num_bidders) )
bids[:, 0] = np.reshape( dataset_data['bids1'], (num_auctions ,) )
bids[:, 1] = np.reshape( dataset_data['bids2'], (num_auctions ,) )
bids[:, 2] = np.reshape( dataset_data['bids3'], (num_auctions ,) )
bids[:, 3] = np.reshape( dataset_data['bids4'], (num_auctions ,) )

def EstimateG( v ):
    out   = 0
    t_sum = 0
    for l in range( num_auctions ):
        for p in range( num_bidders ):
            if bids[l, p] <= v:
                t_sum += 1
    out = ( 1 / ( num_auctions * num_bidders ) ) * t_sum
    return out

def Estimateg( v ):
    out   = 0
    t_sum = 0
    for l in range( num_auctions ):
        for p in range( num_bidders ):
            #h_g
            bandwidth = 2.978 * 1.06 * np.std( bids[:, p] )
            normalise = ( v - bids[l, p] ) / bandwidth
            t_sum += ( 1 / ( num_auctions * num_bidders * bandwidth ) ) * KernelFunction( normalis
    out = t_sum
    return out

def ObtainFPSBPrivateValue():
    out = np.zeros( shape=(num_auctions , num_bidders) )
    for j in range( num_auctions ):
        for i in range( num_bidders ):
            out[j, i] = bids[j, i] - ( 1 / (num_bidders - 1) ) * ( EstimateG( bids[j, i] ) / Estim
    return out

def ObtainFPSBf( v ):
    out   = 0
    t_sum = 0
    for j in range( num_auctions ):
        for i in range( num_bidders ):
            R = 2
            #h_f
            bandwidth = 2.978 * 1.06 * np.std( fpsb_private_values[:, i] ) * ( np.log( num_auction
            normalise = (v - fpsb_private_values[j, i]) / bandwidth
            t_sum += ( 1 / bandwidth ) * KernelFunction( normalise )
        out += ( 1 / ( num_auctions * num_bidders ) ) * t_sum
    return out
```

```
def KernelFunction( value ):
    # Triweight
    out = ( 35 / 32 ) * ( 1 - value**2 )**3 * ( np.abs( value ) <= 1 )
    # Epanechikov
    #out = 0.75 * ( 1 - value**2 ) * ( np.abs( value ) <= 1 )
    # Standard Normal
    #out = ( 2 * np.pi )**(-0.5) * np.exp( -0.5 * value**2 )
    return out

# Estimate our Distributions over differing values for v for each n
maximum_bid = int(np.ceil(np.max(bids))) + 5
Gv = np.zeros( shape=(maximum_bid,) )
gv = np.zeros( shape=(maximum_bid,) )

# Conduct Kernel Density Distribution
for v in range( maximum_bid ):
    Gv[v] = EstimateG( v )
    gv[v] = Estimateg( v )

# Draw Graph of Cumulative Distribution
print( 'Outputting pdf of Bids ...' )
print( gv.shape )
print( gv )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Plot of Distribution for pdf of Bids ...' )
plt.plot( gv )
plt.title('Non-Parametric Probability Density Function for Bids')
plt.xlabel('v')
plt.ylabel('Pr(Valuation=v)')
plt.show()
print('')

# Draw Graph of Cumulative Distribution
print( 'Outputting CDF of Bids ...' )
print( Gv.shape )
print( Gv )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Plot of Distribution for CDF of Bids ...' )
plt.plot( Gv )
plt.title('Non-Parametric Cumulative Distribution Function for Bids')
plt.xlabel('v')
plt.ylabel('Pr(Valuation<=v)')
plt.show()
print('')

# Estimate FPSB Private Values
fpsb_private_values = ObtainFPSBPrivateValue()
print( 'Outputting FPSB Private Values ...' )
print( fpsb_private_values )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Histogram of Distribution for FPSB Private Values ...' )
num_bins = 20
```

```
for i in range( num_bidders ):
    print( 'Graph [' + str(i+1) + ']' )
    N, bins, patches = plt.hist(fpsb_private_values[:, i], bins=num_bins)
    fracs = N / N.max()
    norm  = colors.Normalize( fracs.min(), fracs.max() )
    for ifrac, ipatch in zip( fracs, patches ):
        color = plt.cm.viridis( norm( ifrac ) )
        ipatch.set_facecolor( color )
    plt.title('Distribution of FPSB Private Values (bidder ' + str(i+1) + ')')
    plt.xlabel('Estimated FPSB Private Values')
    plt.ylabel('Frequency')
    plt.show()
print('')

# Conduct Kernel Density Distribution
maximum_value = int( np.ceil( np.max( fpsb_private_values ) ) ) + 5
fpsb_fu = np.zeros( (maximum_value,) )
fpsb_Fu = np.zeros( (maximum_value,) )
for v in range( maximum_value ):
    fpsb_fu[v] = ObtainFPSBf( v )
#normalise_min = np.min(fpsb_fu)
#normalise_max = np.max(fpsb_fu)
#for v in range( maximum_value ):
#    fpsb_fu[v] = ( fpsb_fu[v] - normalise_min ) / ( normalise_max - normalise_min )
fpsb_Fu[0] = fpsb_fu[0]
for v in range( maximum_value - 1 ):
    fpsb_Fu[v + 1] = fpsb_fu[v + 1] + fpsb_Fu[v]

print( 'Outputting pdf of Private Values  ...' )
print( fpsb_fu )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Plot of Distribution for pdf of Private Values ...' )
plt.plot( fpsb_fu )
plt.title('Non-Parametric Probability Density Function for Private Values')
plt.xlabel('v')
plt.ylabel('Pr(Valuation=v)')
plt.show()
print('')

print( 'Outputting CDF of Private Values...' )
print( fpsb_Fu )
print('')

# Draw Graph of Cumulative Distribution
print( 'Generating Plot of Distribution for CDF of Private Values ...' )
plt.plot( fpsb_Fu )
plt.title('Non-Parametric Cumulative Distribution Function for Private Values')
plt.xlabel('v')
plt.ylabel('Pr(Valuation<=v)')
plt.show()
print('')

# First and Third Quartiles of Marginal Distribution F of our Private Values
pv_quantile = np.zeros( (2,) )
pv_quantile[0] = np.percentile( fpsb_Fu, 25 )
pv_quantile[1] = np.percentile( fpsb_Fu, 75 )
pv_label   = np.zeros( (2**4, 4) )
```

```
pv_matrix  = np.zeros( (2**4, 4) )
pv_results = np.zeros( (2**4, 4) )
pv_final   = np.zeros( (2**4,) )
m = 0

for i in range( 2 ):
    for j in range( 2 ):
        for k in range( 2 ):
            for l in range( 2 ):
                pv_label[m, 0] = 25 + 50 * i
                pv_label[m, 1] = 25 + 50 * j
                pv_label[m, 2] = 25 + 50 * k
                pv_label[m, 3] = 25 + 50 * l
                pv_matrix[m, :] = np.array( [pv_quantile[i], pv_quantile[j], pv_quantile[k], pv_qu
                for n in range( num_bidders ):
                    pv_results[m, n] = np.percentile( fpsb_Fu, pv_label[m, n] )
                a = np.reshape( np.array( fpsb_Fu[0] ), (1, 1) )
                b, c, d = a, a, a
                n = 1
                while fpsb_Fu[n] <= pv_results[m, 0]:
                    a = np.concatenate( (a, np.reshape( np.array( fpsb_Fu[n] ), (1, 1) ) ), axis=0
                    n += 1
                a = np.mean( a )
                n = 1
                while fpsb_Fu[n] <= pv_results[m, 1]:
                    b = np.concatenate( (b, np.reshape( np.array( fpsb_Fu[n] ), (1, 1) ) ), axis=0
                    n += 1
                b = np.mean(b)
                n = 1
                while fpsb_Fu[n] <= pv_results[m, 2]:
                    c = np.concatenate( (c, np.reshape( np.array( fpsb_Fu[n] ), (1, 1) ) ), axis=0
                    n += 1
                c = np.mean(c)
                n = 1
                while fpsb_Fu[n] <= pv_results[m, 3]:
                    d = np.concatenate( (d, np.reshape( np.array( fpsb_Fu[n] ), (1, 1) ) ), axis=0
                    n += 1
                d = np.mean(d)
                n = 1
                pv_final[m] = np.mean( [a, b, c, d] )
                print( '[' + str(m) +']' )
                print( 'u_pc: ' + str( pv_label[m, :] ) )
                print( 'u_i:  ' + str( pv_results[m, :] ) )
                print( 'Gu_i: ' + str( pv_final[m] ) )
                print( 'Fu_i: ' + str( pv_label[m, 0] * pv_label[m, 1] * pv_label[m, 2] * pv_label
                print('')
                m += 1


print('')
print('***')
print('[EOF] Output Terminates')
print('***')
# EOF
```