

Closest answer = choice - answer that is closest to zero

1.

The screenshot shows a code editor window with a dark theme. On the left is a sidebar with various icons. The main area contains Python code for generating coin flips and calculating statistics. The code includes imports for random, definitions for flip_coin, thousand, morecoins, and a print statement. It uses nested loops to simulate 1000 coin flips and calculate heads, tails, and variance. The morecoins function then repeats this process 10,000 times to calculate average heads, tails, and variance. The terminal tab at the bottom shows the command run and the output of the script.

```
1 import random
2 def flip_coin():
3     return random.choice([0,1]) #0 = heads; 1 = tails.
4 def thousand(num):
5     crand = random.randint(0, 1000)
6     v1 = 0
7     vrnd = 0
8     vmin = 1
9     for i in range(num):
10         heads = 0
11         for j in range(10):
12             result = flip_coin()
13             if result == 0:
14                 heads += 1
15             if i == 0:
16                 v1 = (heads / 10)
17             if i == crand:
18                 vrnd = heads/10
19             if(heads / 10) < vmin:
20                 vmin = heads / 10
21     return(v1, vrnd, vmin)
22 def morecoins():
23     vminav = 0
24     v1 = 0
25     vrnd = 0
26     for i in range(100000):
27         res = thousand(1000)
28         v1 += res[0]
29         vrnd += res[1]
30         vminav += res[2]
31     return (vminav/100000, v1/100000, vrnd/100000)
32 print(morecoins())
33
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/ml2.py
(0.03763499999997673, 0.499600000000021, 0.49917299999999626)

Ln 22, Col 17 Spaces: 4 UTF-8 LF ↵ Python 3.10.6 64-bit

- a. b. Closest answer = choice - answer closest to zero thus:
 - c. [a] $0 - 0.0377 = -0.0376$
 - d. [b] $0.01 - 0.0377 = -0.0276$
 - e. [c] $0.1 - 0.0377 = 0.0624$
 - f. [d] $0.5 - 0.0377 = 0.4624$
 - g. [e] $0.67 - 0.0377 = 0.6324$
- h. Hence the correct answer is **b** as it's closest to zero.
2. We know that we have a fair coin thus mu should be 0.5 for the single bin Hoeffding Inequality hence looking at the probability distributions above from question one's code, we see that C1 and the random coin are closest approximations of mu hence they satisfy the single bin Hoeffding Inequality. As we have a fair coin the probability of getting heads should be 0.50 for all coins but only C1 and the random coin distributions approximate that as they are about 0.5 each thus both satisfying the inequality.
 - a. [a] Wrong because it does not match the description above.
 - b. [b] Wrong because it does not match the description above.
 - c. [c] Wrong because it does not match the description above.
 - d. [d] Correct because it matches the description above.
 - e. [e] Wrong because it does not match the description above.

3. For notation purposes let Lambda = L in this question. We know that
3 and 4:

HW2:

(3)

$$\begin{aligned} P(h \neq y) &= P(h \neq y | y=1) \cdot P(y=1) + P(h \neq y | y \neq 1) \cdot P(y \neq 1) \\ &= M \cdot \lambda + (1-M)(1-\lambda) \\ &= (1-\lambda)(1-M) + M \cdot \lambda \end{aligned}$$

Hence the answer is c because it's the
only one that matches the work above.

(4) $P(h \neq 1) = (1-\lambda)(1-M) + \lambda \cdot M$

$$= 1 - M - \lambda + M\lambda + \lambda M$$

$$= M(2\lambda - 1) + 1 - 1$$

For $M \neq 0$ solve $0 = M(2\lambda - 1)$ to

thus: $2\lambda - 1 = 0$

$$\lambda = \frac{1}{2}$$

find when
independent
of M as
the M factor
will be
zero

Hence the answer is

b as it is the only
value that matches
the above.

5:

```
ml2.py
Users > thierno > Desktop > ml2.py ...
36 def get_slope(x1, y1, x2, y2):
37     slope = (y2 - y1)/(x2 - x1)
38     intercept = y1 - (slope * x1)
39     return (slope, intercept)
40 def algo(N):
41     point1 = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
42     point2 = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
43     slope = get_slope(point1[1], point1[2], point2[1], point2[2])
44     scores = []
45     data = []
46     def classify(point):
47         if (slope[0] * point[1]) + slope[1] > point[2]:
48             scores.append(-1)
49         else:
50             scores.append(1)
51     for i in range(N):
52         curr = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
53         data.append(curr)
54         classify(curr)
55     psuedo_inv = numpy.linalg.pinv(data)
56     w = numpy.dot(psuedo_inv, numpy.array(scores).T)
57     def Ein():
58         summation = 0
59         for d, s in zip(data, scores):
60             if s != numpy.sign(numpy.dot(w, d)):
61                 summation += 1
62     return summation/N
63
64 return Ein()
65
pscl.py
Users > thierno > Desktop > ml2.py ...
63     return Ein()
64
65 def do():
66     errin = []
67     N = 300
68     runs = 1000
69     for i in range(runs):
70         errin.append(algo(N))
71     return (sum(errin) / len(errin))
72
73 print(do())
74
75
76
77
78
79
```

We get an $\text{Ein} = 0.0392$

- [a] $0 - \text{Ein} = -0.0392$
- [b] $0.001 - \text{Ein} = -0.0382$
- [c] $0.01 - \text{Ein} = -0.0292$
- [d] $0.1 - \text{Ein} = 0.0608$
- [e] $0.5 - \text{Ein} = 0.4608$

Hence the answer is **C** because it is closest to zero.

6.

```
m2.py
Users > thierno > Desktop > m2.py > ...
36 def get_slope(x1, y1, x2, y2):
37     slope = (y2 - y1) / (x2 - x1)
38     intercept = y1 - (slope * x1)
39     return slope, intercept
40
41 def genData(N):
42     point1 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
43     point2 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
44     slope = get_slope(point1[1], point1[2], point2[1], point2[2])
45     scores = []
46     data = []
47
48     def classify(point, score):
49         if ((slope[0] * point[1]) + slope[1]) > point[2]:
50             scores.append(-1)
51             curr = -1
52         else:
53             scores.append(1)
54             curr = 1
55
56         for i in range(N):
57             curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
58             data.append(curr)
59             classify(curr, scores)
60
61     pseudoinv = numpy.linalg.pinv(data)
62     w = numpy.dot(pseudoinv, numpy.array(scores).T)
63
64     def Ein():
65         summation = 0
66         for d, s in zip(data, scores):
67             if s != numpy.sign(numpy.dot(w, d)):
68                 summation += 1
69         return summation/N
70
71     def Eout():
72         nscores = []
73         test_points = 1000
74         outofsample = 0
75         for i in range(test_points):
76             curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
77             score = classify(curr, nscores)
78             weight = numpy.sign(numpy.dot(w, curr))
79             if score != weight:
80                 outofsample += 1
81
82         return outofsample/test_points
83
84     return Eout()
85
86 # def q5():
87 #     errin = []
88 #     N = 1000
89 #     runs = 1000
90 #     for i in range(runs):
91 #         errin.append(algoN(i))
92 #     return sum(errin) / len(errin)
93
94 def q6():
95     error = []
96     N = 100
97     runs = 1000
98     for i in range(runs):
99         error.append(algoN(i))
100    return sum(error) / len(error)
101
102 print(q6())
103
104
105
106
```

PROBLEMS DEBUG CONSOLE TERMINAL JUPYTER

thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/m2.py

0.048525999999999965

thierno@Thiernos-MacBook-Pro ~ %

Python - thierno + - ☰ ^ X

Ln 40, Col 5 Spaces: 4 UTF-8 LF Python 3.10.6 64-bit

$$E_{out} = 0.0485$$

- [a] $0 - E_{out} = -0.0485$
- [b] $0.001 - E_{out} = -0.0475$
- [c] $0.01 - E_{out} = -0.0385$
- [d] $0.1 - E_{out} = 0.0515$
- [e] $0.5 - E_{out} = 0.4515$

Hence the correct answer according to the above code is c as it is closest.

7.

Iteration av = 5,752

- [a] $1 - av = -4.752$
 - [b] $15 - av = 9.248$
 - [c] $300 - av = 294.248$
 - [d] $5000 - av = 4994.248$
 - [e] $10000 - av = 9994.248$

Hence based on the code then **a** is the correct answer because it is closest to zero.

The screenshot shows a Jupyter Notebook interface with several files open in the sidebar: ml2.py, ml8.py, psct1.py, and m2.java. The main area displays Python code for calculating Ein . The code uses numpy to calculate the sign of the sum of $x_1 \cdot x_2$ and $x_2 \cdot x_3$, and then performs a weighted average over multiple runs. The code is as follows:

```
1 import random
2 import numpy
3
4 def func(x1, x2):
5     return numpy.sign((x1*x2) + (x2*x3) - 0.6)
6
7 def alg0(N):
8     data = []
9     scores = []
10    for i in range(N):
11        curr = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
12        scores.append(func(curr[0], curr[1]))
13        data.append(curr)
14
15    for k in range(int(N * .10)):
16        score = random.randint(0, N-1)
17        hold = scores[score]
18        scores[score] = -1 * hold
19
20    psudo_inverse = numpy.linalg.pinv(data)
21    wv = numpy.dot(psudo_inverse, numpy.array(scores).T)
22
23    def Ein():
24        error = 0
25        for d, s in zip(data, scores):
26            curr = numpy.sign(numpy.dot(wv, d))
27            if s != curr:
28                error += 1
29        return error/N
30
31    N = 1000
32    runs = 1000
33    summation = 0
34    for i in range(runs):
35        summation += alg0(N)
36
37    return summation / runs
38
39
```

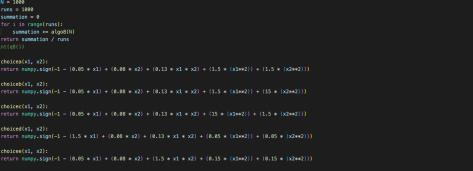
8.

$$Ein = 0.508$$

- [a] $0 - Ein = -0.508$
- [b] $0.1 - Ein = -0.408$
- [c] $0.3 - Ein = -0.208$
- [d] $0.5 - Ein = -0.008$
- [e] $0.8 - Ein = 0.292$

Hence according to the code **d** is the correct answer because it is closest to zero.

9.



The screenshot shows a Jupyter Notebook interface with several cells of Python code. The code defines a function `sum_mansions` that calculates the sum of mansions based on input parameters. It includes logic to handle different cases of `m` and `n`, and uses nested loops and conditionals to calculate the sum. The notebook also contains a cell for testing the function with `m=10000` and `n=10000`. The output cell shows the result of the calculation.

```
User: > Homeo > Desktop > miltopy > algof > bin
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
```



The screenshot shows a Jupyter Notebook interface with the following details:

- Code Cell:** Contains Python code for generating a scatter plot. The code uses nested loops to calculate points based on parameters like n , m , and α , and then plots them using `plt.scatter`.
- Output Cell:** Shows the resulting scatter plot with a grid of points.
- TERMINAL:** Displays command-line history related to the current session.
- PROBLEMS:** Shows a list of potential issues or errors.

Max frequency is at index 0 according to the code above hence the answer choice is **a** according to frequency sort in the code of which $g(x)$ best matches.

10.

```
m2.py    m8.py    alg08    m2.java 1 *  
Users : iThemo : Desktop > m2.py > alg08 > Eout  
import random  
import numpy  
  
def func(x1, x2):  
    return numpy.sign((x1*x2) + (x2*x2) - 0.6)  
  
def algo8():  
    data = []  
    scores = []  
    for i in range(N):  
        x1 = random.uniform(-1, 1)  
        x2 = random.uniform(0, N-1)  
        curr = [1, x1, x2, x1*x2, x1*x2, x2*x2]  
        scores.append(func(x1, x2))  
        data.append(curr)  
  
    for k in range(int(N*.10)):  
        scores.sort()  
        hold = scores[0]  
        scores[0] = -1 * hold  
  
    pseudo_inverse = numpy.linalg.pinv(data)  
    w = numpy.dot(pseudo_inverse, numpy.array(scores)).T  
  
def Eout():  
    error = 0  
    for d, s in zip(data, scores):  
        curr = numpy.linalg.norm(numpy.dot(w, d))  
        if s < curr:  
            error += 1  
    return error/N  
  
def Eout1():  
    error = 0  
    points = 1000  
    count = 0  
    for i in range(points):  
        currx1 = random.uniform(-1, 1)  
        currx2 = random.uniform(0, N-1)  
        curr = [1, currx1, currx2, currx1*x2, currx1*x2, currx2*x2]  
        insample = numpy.sign(numpy.dot(w, curr))  
        outsample = func(currx1, currx2)  
        if count > 0:  
            if insample == random.choice([0, 1]):  
                if insample == outsample:  
                    error += 1  
        count += 1  
    return error/N  
  
def alg08():  
    N = 1000  
    runs = 1000  
    summation = 0  
    for i in range(runs):  
        summation += alg08()  
    return summation / runs  
#print(alg08())  
  
> def choice(x1, x2):=br/>64 > def choiceb(x1, x2):=br/>65 > def choiced(x1, x2):=br/>66 > def choicecc(x1, x2):=br/>67 > def choicec(x1, x2):=br/>68 > def choicee(x1, x2):=br/>69 > def choiceee(x1, x2):=br/>70  
  
m8.py    m2.py    alg08    m2.java 1 *  
Users : iThemo : Desktop > m8.py > alg08 > Eout  
N = 1000  
runs = 1000  
sum = 0  
for i in range(runs):  
    sum += alg08()  
return sum / runs  
#print(sum)  
  
> def q0():=br/>72 > def q1():=br/>73 > def q2():=br/>74 > def q3():=br/>75 > def q4():=br/>76 > def q5():=br/>77 > def q6():=br/>78 > def q7():=br/>79 > def q8():=br/>80 > def q9():=br/>81 > def q10():=br/>82 > def q11():=br/>83 > def q12():=br/>84 > def q13():=br/>85 > def q14():=br/>86 > def q15():=br/>87 > def q16():=br/>88 > def q17():=br/>89 > def q18():=br/>90 > def q19():=br/>91 > def q20():=br/>92 > def q21():=br/>93 > def q22():=br/>94 > def q23():=br/>95 > def q24():=br/>96 > def q25():=br/>97 > def q26():=br/>98 > def q27():=br/>99 > def q28():=br/>100 > def q29():=br/>101 > def q30():=br/>102 > def q31():=br/>103 > def q32():=br/>104 > def q33():=br/>105 > def q34():=br/>106 > def q35():=br/>107 > def q36():=br/>108 > def q37():=br/>109 > def q38():=br/>110 > def q39():=br/>111 > def q40():=br/>112 > def q41():=br/>113 > def q42():=br/>114 > def q43():=br/>115 > def q44():=br/>116 > def q45():=br/>117  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER  
iThemo:Desktop-iThemo-MacBook-Pro ~ % /usr/bin/python -c "users:iThemo:Desktop:m2.py"  
0.07899999999999999  
iThemo:Desktop-iThemo-MacBook-Pro ~ %  
Ls 44 Col 44 Spaces: 4 UTF-8 LF Python 3.10.6 64-bit
```

Eout from the above code is 0.0789

- [a] $0 - \text{Eout} = -0.0789$
- [b] $0.1 - \text{Eout} = 0.0211$
- [c] $0.3 - \text{Eout} = 0.2211$
- [d] $0.5 - \text{Eout} = 0.4211$
- [e] $0.8 - \text{Eout} = 0.7211$

Hence the answer is **b** as it is closest to zero