گزارش کار

computer assignment 1

مصطفی کرمانی نیا

شماره دانشجویی : 810101575

سوال اول

❋بخش 1 :

برای این بخش تابعی زدم که سه آرگومان میگیرد: ❋تعداد آزمایش های برنولی انجام شده در طول ساخت هر متغیر دوجمله ای(n) ، ❋تعداد کل متغیرهای دو جمله ای(m) ، ❋احتمال رخ دادن هر کدام از مقادیر 0 و 1 در هر بار آزمایش برنولی در داخل این تابع ابتدا مقادیر ممکن برای یک آزمایش برنولی که 0 و 1 هست را ست کردم، سپس ابتدا به اندازه ی کل برنولی های مورد نیاز در ساخت کل متغیرهای دوجمله ای (n*m) ، از توزیع برنولی نمونه گیری کردم.

سپس با reshape کردن آرایه ی حاصل که دارای یک سطر و n*m ستون بود، به یک آرایه با m سطر و n ستون، به آرایه ی all_samples رسیدم که هر سطر آن رسما یک متغیر دوجمله ای را نشان میدهد، پس با جمع مقادیر هر سطر آن توسط تابع sum به یک آرایه ی یک سطری می رسیم که هر عضو آن نشانگر مجموع یک سطر از آرایه ی قبلی است و یعنی نشانگر یک متغیر دوجمله ای از نوعی که مدنظر ما است، می باشد.

```
import numpy as np
# Part 1
def sample_binomial(bernoulli_trials_count, binomial_trials_count, probabilities):
    samples_count = bernoulli_trials_count * binomial_trials_count
    results = [0, 1]

    all_samples = np.random.choice(results, samples_count, p=probabilities)
            samples_array      =      np.array(all_samples).reshape(binomial_trials_count,
bernoulli_trials_count )

    binomial_variables = np.sum(samples_array, axis=1)
    return binomial_variables
```

مثلا: گرفتن 3 نمونه ی دو جمله ای که هر کدام با 500 بار تکرار برنولی با احتمال های 0.5 و 0.5 ساخته شده اند

```
bernoulli_trials_count = 500
binomial_trials_count = 3
probabilities = [0.5, 0.5]
sample_binomial(bernoulli_trials_count, binomial_trials_count, probabilities)

out : array([251, 267, 268])
```

***بخش 2 :**

در این بخش ابتدا یک آرایه از مقادیر p که قرار است آزمایش شوند(از 0 تا خود عدد 1، با فاصله های 0.01) ساخته و به ازای هر کدام، یک دسته ی 5000 تایی از نمونه های دوجمله ای می سازیم، در نیتجه در نهایت یک آرایه دو بعدی با 101 سطر داریم که هر سطر یک آرایه با 5000 عضو است که هر کدام یک متغیر دوجمله ای است.

```
success_probabilities = np.arange(0, 1+0.01, 0.01)
bin_samples = np.array([build_binomial_sample(
    bernoulli_trials_count, binomial_trials_count, prob) for prob in
success_probabilities])
```

پس حالا میتوان با محاسبه ی واریانس و میانگین(امیدریاضی) هر سطر به دو روش گفته شده در سوال، به لیستی از امید ریاضی و واریانس ها برسیم که آنها را مقایسه کنیم، برای گرفتن امید ریاضی و واریانس به دو روش تئوری و عملی از توابع زیر استفاده میکنیم(منظور از binomial sample یک دسته ی 5000 تایی از متغیر های دو جمله ای است که قرار است واریانس یا میانگین دستی را روی آنها حساب کنیم):

```
def calculate_theory_variance(success_prob, bernoulli_trials_count):
    p = success_prob
    n = bernoulli_trials_count
    return n*p*(1-p)
def calculate_theory_expectation(success_prob, bernoulli_trials_count):
    p = success_prob
    n = bernoulli_trials_count
    return n*p
def calculate_practical_expectation(binomial_samples):
    return np.sum(binomial_samples)/binomial_samples.size
def calculate_practical_variance(binomial_samples):
```

```
    return calculate_practical_expectation(np.power(binomial_samples -
calculate_practical_expectation(binomial_samples), 2))
```

-برای محاسبه ی واریانس و میانگین، توابع از پیش آماده ای در نامپای هم موجود بودند اما من بطور دستی حساب کردن را بهتر میدانستم، پس برای محاسبه ی امید ریاضی، از اعداد موجود در لیست متغیر های دوجمله ای میانگین گرفتم و برای محاسبه ی واریانس هم $(x_i - E_X)^2$ را برای تک تک اعضای آن لیست حساب کرده و سپس امیدریاضی(یا همان میانگین) آنها را گرفتم.

در نهایت این توابع را روی تک تک سطرهای آن آرایه ی شامل متغیرهای دوجمله ای اعمال کردم(در اینجا هم منظور از bin_sample فقط یک متغیر دو جمله ای نیست، بلکه لیستی 5000 تایی از متغیر های دو جمله ای است)

```
practical_expectations = np.array(
    [calculate_practical_expectation(bin_sample) for bin_sample in bin_samples])
theory_expectations = np.array([calculate_theory_expectation(
    prob, bernoulli_trials_count) for prob in success_probabilities])
practical_variances = np.array(
    [calculate_practical_variance(bin_sample) for bin_sample in bin_samples])
theory_variances = np.array([calculate_theory_variance(
    prob, bernoulli_trials_count) for prob in success_probabilities])
```

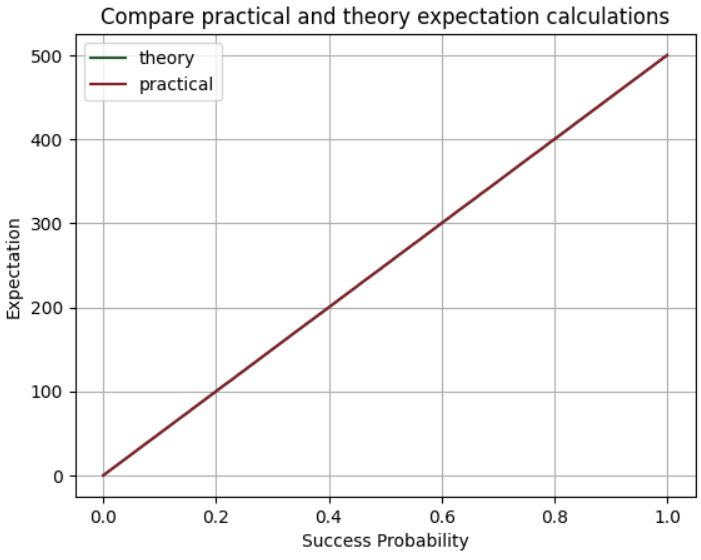
در نهایت برای میانگین ها داریم:

```
List of practical expectations:
 [  0.       4.967    9.9622  14.9272  20.0654  25.1702  29.9462  34.9634
  40.0576  44.9918  50.1128  55.0962  59.8474  65.0394  69.871   75.137
  79.828   85.2884  90.2168  95.051   99.7738 105.0434 110.1442 115.0964
 119.9274 124.6774 130.0208 135.0668 139.8994 144.9668 150.01   155.1738
 160.0826 165.0408 169.762  175.0924 180.079  184.753  189.816  194.7364
 199.9434 204.9266 209.8364 214.735  219.8978 224.9714 229.8692 235.0136
 239.9794 245.2196 249.8944 254.701  260.1324 265.059  269.8142 274.8012
 279.9756 284.937  290.2738 294.8546 300.0142 305.2256 310.1602 315.1278
 319.9524 324.9136 329.9304 334.826  340.1748 344.7302 349.8632 354.947
 360.23   365.0338 369.9896 375.131  380.0508 384.9482 390.0194 395.047
 399.9574 404.923  409.9056 414.928  419.8274 424.9792 429.7632 434.9826
 439.8866 444.859  449.8712 454.9142 459.8732 465.0048 470.0386 475.0202
 480.07   484.9934 490.0146 494.9846 500.    ]

List of theory expectations:
 [  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.  50.  55.  60.  65.
  70.  75.  80.  85.  90.  95. 100. 105. 110. 115. 120. 125. 130. 135.
 140. 145. 150. 155. 160. 165. 170. 175. 180. 185. 190. 195. 200. 205.
 210. 215. 220. 225. 230. 235. 240. 245. 250. 255. 260. 265. 270. 275.
 280. 285. 290. 295. 300. 305. 310. 315. 320. 325. 330. 335. 340. 345.
 350. 355. 360. 365. 370. 375. 380. 385. 390. 395. 400. 405. 410. 415.
 420. 425. 430. 435. 440. 445. 450. 455. 460. 465. 470. 475. 480. 485.
 490. 495. 500.]
```

و همچنین واریانس ها:

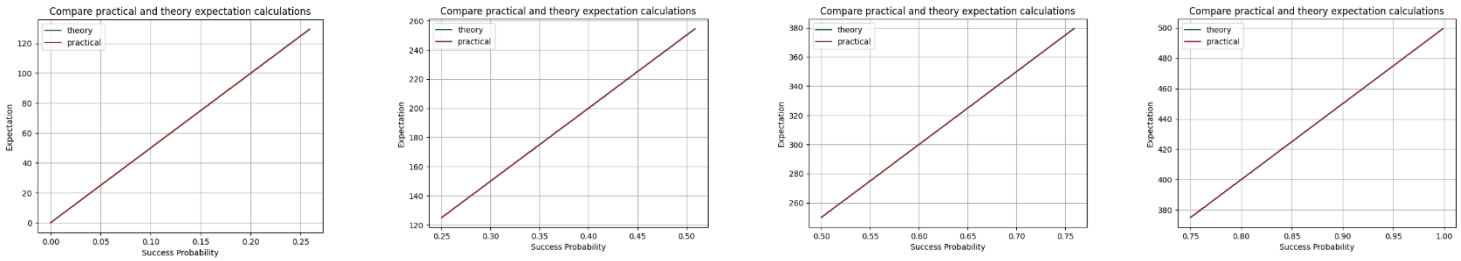
```
List of practical variances:
[  0.          5.03149824   9.55406656  15.167104    18.83520576
  23.76008604  28.71958064  32.94838684  35.94826544  41.36313536
  44.25817904  49.66420224  51.55637696  56.74251504  58.93051056
  63.204719    69.07790044  69.91734396  73.88433584  75.34317504
  81.42716284  81.84202944  86.36787644  88.85970524  92.23005276
  92.08030876  96.03831104  99.29284796 100.70699196 103.15714736
 104.007984   105.30578624 106.9995     115.22998684 110.63316316
 111.69267164 112.98492444 118.17630876 118.56270896 119.83229596
 116.91058236 120.414876   125.29535004 122.70423616 122.89696796
 124.699824   122.324524   121.25625664 123.06629244 121.52380416
 129.11360604 126.26236924 124.94639744 128.48013404 125.94441596
 123.19763456 120.95328956 120.111975   119.80531056 123.95223616
 121.60801884 116.01361404 119.759136   116.57918976 113.88268604
 113.46535344 112.35480316 107.42077936 106.61943036 108.29936896
 107.84910044 105.2084     100.87365436 103.91725564  97.23440316
  91.85489596  91.07300224  88.01470784  83.97016476  84.44352176
  81.79936576  78.24535196  74.49679856  71.89022784  66.95941504
  64.80715696  60.37692544  55.57949184  52.68978044  49.329596
  45.136864    42.502711    36.523079    32.23058556  27.89310876
  24.034599    19.01930716  14.08912444   9.75275036   4.998991
   0.        ]

List of theory variances:
[  0.     4.95   9.8   14.55  19.2   23.75  28.2   32.55  36.8   40.95
  45.    48.95  52.8   56.55  60.2   63.75  67.2   70.55  73.8   76.95
  80.    82.95  85.8   88.55  91.2   93.75  96.2   98.55 100.8  102.95
 105.   106.95 108.8  110.55 112.2  113.75 115.2  116.55 117.8  118.95
 120.   120.95 121.8  122.55 123.2  123.75 124.2  124.55 124.8  124.95
 125.   124.95 124.8  124.55 124.2  123.75 123.2  122.55 121.8  120.95
 120.   118.95 117.8  116.55 115.2  113.75 112.2  110.55 108.8  106.95
 105.   102.95 100.8   98.55  96.2   93.75  91.2   88.55  85.8   82.95
  80.    76.95  73.8   70.55  67.2   63.75  60.2   56.55  52.8   48.95
  45.    40.95  36.8   32.55  28.2   23.75  19.2   14.55   9.8    4.95
   0.  ]
```

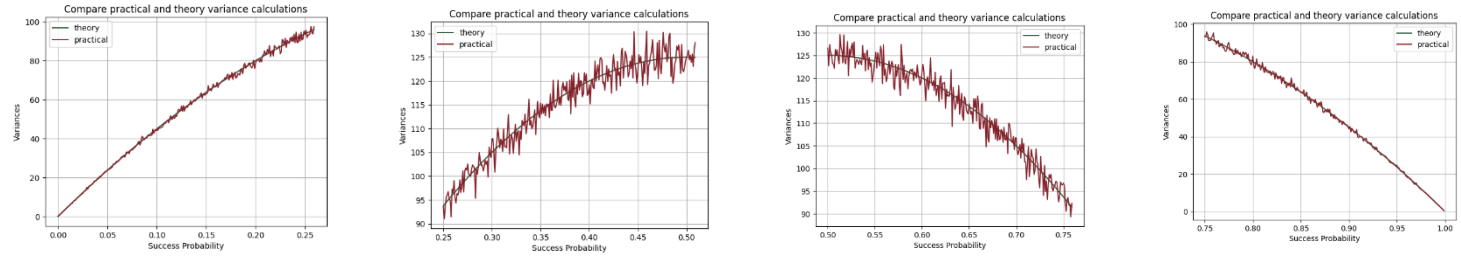
***بخش 3 :**



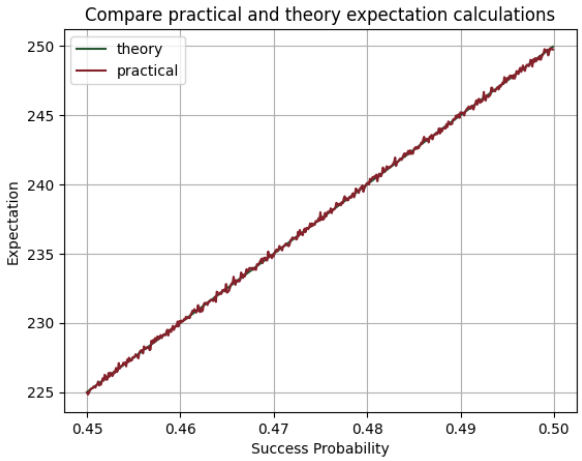
***بخش 4 :**

چون میزان نویز محاسبه ی دستی بسیار کم است، در نمودار های کلی بخوبی مشخص نمیشود(مخصوصا برای امید ریاضی ها) در نتیجه تصمیم گرفتم برای بهتر مشخص شدن تفاوت محاسبه ی دستی و تئوری و تحلیل دقیقتر این مقادیر در بازه های مختلف p، اولا گام های تغییر p را افزایش دهم و بجای تغییر 0.01 در هر مرحله، 0.001 در هر مرحله آنرا زیادتر کنم، همچنین نمودار را بجای نمایش در بازه ی 0 تا 1 برای p ، در زیربازه های 0 تا 0.25 و 0.25 تا 0.5 و 0.5 تا 0.75 و 0.75 تا 1 نمایش دهم:





برای دیدن دقیقتر نویز در میانگین، یکبار با گام های 0.0001، نمودار را بین 0.45 تا 0.5 رسم کردم:

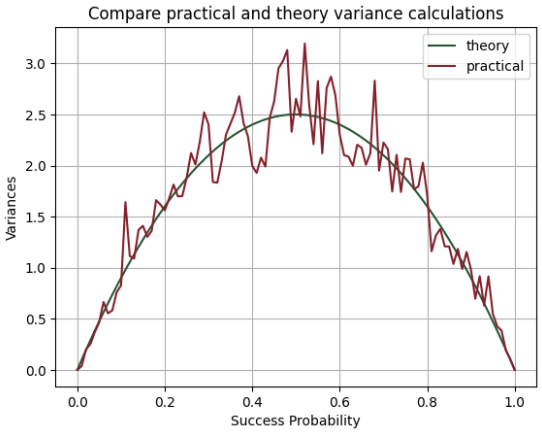


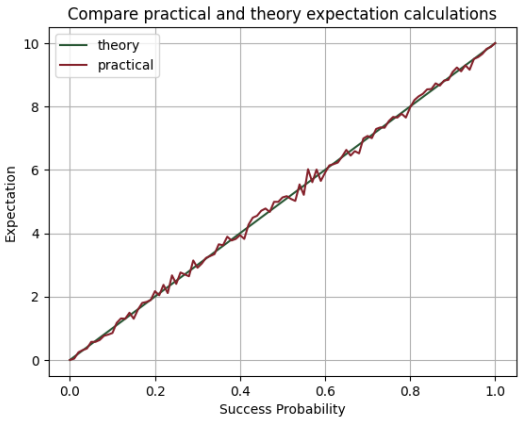
نکته اینجاست که در واریانس، میزان نویز در اطراف $p = 0.5$ بیشتر میشود و در کل هم میزان نویز در واریانس بیشتر است و دلیل افزایش نویز واریانس در اطراف $p = 0.5$ این است که ما به هر حال مقداری خطا داریم،اما با افزایس مقدار خود واریانس در نزدیکی 0.5(زیرا هر چقدر احتمال موفقیت و عدم موفقیت در

برنولی ها نزدیک به هم باشد، پراکندگی مقادیر متغیر دوجمله ای هم بیشتر است و واریانس زیاد میشود) مقدار خطای محاسبه ی دستی آن هم بیشتر میشود که طبیعی است.

از طرفی دلیل زیادتر بودن تفاوت در واریانس ها نسبت به تفاوت میانگین ها هم این است که اولا در محاسبه ی واریانس یکبار داریم امیدریاضی را هم حساب میکنیم، در نتیجه در هر محاسبه همزمان هم خطای محاسبه ی امید ریاضی را داریم و هم موقع محاسبه ی اختلاف مقدار متغیر تصادفی با امیدریاضی آن خطا داریم، همچنین چون نهایتا به توان دو میرسانیم اعداد را و جمع میزنیم، توان دوم خودش عامل بولد کردن این خطا ها میشود.

در نهایت هم بنظرم یکی از دلایل کم بودن تفاوت مقدار دستی و تئوری امید ریاضی، زیاد بودن تعداد نمونه های ما است، مثلا اگر تعداد نمونه های دو جمله ای را از 5000 به 100 برسانیم و هر کدام هم بجای 500 آزمایش برنولی شامل 10 آزمایش برنولی باشند، همزمان هم تعداد مقادیر ممکن برای دو جمله ای کمتر میشود و هم تعداد نمونه های دو جمله ای کم میشود و این باعث میشود خطای عملی و تئوری(ایده آل) بیشتر نمایان شود:





سوال دوم

***بخش 1 :**

در وهله ی اول پارامتر های توزیع پواسون و نرمال معادل با توزیع دو جمله ای گفته شده را می یابیم:

```
Probability = 0.008
Num_of_bernollies = 250
poisson_mu = Probability * Num_of_bernollies
normal_mean = Probability * Num_of_bernollies
normal_var = Probability * (1-Probability) * Num_of_bernollies
```

حالا با مقدار دهی به متغیر samples_count مشخص میکنیم چه تعدادی نمونه ی دوجمله ای میخواهیم بگیریم و بررسی کنیم.

حالا برای بررسی PMF و PDF برای توزیع های دو جمله ای و پواسون و نرمال، دو راه داریم:

-یک روش این است که توسط کلاس numpy.random.(binomial,poisson,normal) بیاییم و به اندازه ی samples_count نمونه ی تصادفی بگیریم، سپس یا بطور ریاضیاتی و با توابع دستی، مقدار PMFیا PDF حدودی را بدست بیاوریم، که در این صورت محاسبات ما کاملا وابسته به مقادیر تصادفی دریافت شده هستند و رسما با هر بار ران کردن کد، به نمودار های متفاوتی میرسیم(البته تفاوت ها قاعدتا کوچک است و با افزایش samples_count تقریبا بی تفاوت میشوند. نام این روند را روش1 میگذاریم که بنظرم روش بهتری است چون رسما داریم PDF یا PMF را با TRAIN کردن نمونه ی تصادفی مان میسازیم

-یک روش دیگر این است که یکسری test_values تعریف کنیم مثلا بوسیله ی تابع np.arrange و سپس توسط توابع موجود در کلاس scipy.stats مثل norm.pdfو poisson.pmf بظور تئوری به توابع توزیع آنها برسیم و همان را هم رسم کنیم.در نتیجه چون این توابع بطور تئوری مقدار PDF و PMF را حساب میکنند و مقادیر ورودی و اولیه ی ما نیز ثابت هستند و همان هایی هستند که توسط np.arrange یا np.linespace(برای حالت پیوسته) تولید شده اند، پس هر چقدر که کد را ران کنیم، نتیجه ثابت است اما باز هم با تغییر دادن samples_count میتوانیم بررسی های خود را مبنی بر اینکه کدام تقریب بهتر است انجام دهیم. که نام این روش را هم روش2 میگذارم

در نهایت من هر دو روش را استفاده کردم و بصورت زیر عمل کردم:

-در ابتدا برای هر سه حالت، 3 تا plot میخواهیم بکشیم. ابتدا برای پواسون و نرمال از روش 2 استفاده میکنم یعنی یکسری مقدار گسسته و (تقریبا) پیوسته در حدودی که مدنظرمان است تولید میکنم و سپس به توابع PMF و PDF میدهم که plot ها را رسم کنند:

```
discrete_test_values = np.arange(
    0, Probability * Num_of_bernollies + 3*normal_standard_deviation)
continuous_test_values = np.linspace(
    0, Probability * Num_of_bernollies + 3*normal_standard_deviation)
```

```
ax.plot(discrete_test_values, poisson.pmf(discrete_test_values, poisson_mu),
        color='#85212b', label='Poisson Distribution')
ax.plot(continuous_test_values, norm.pdf(continuous_test_values, normal_mean,
normal_standard_deviation),
        color='#f2c24b', label='Normal Distribution')
```

(در باب رنجی که test_values در آن قرار میگیرند دستمان باز بود اما طبق دانشی که از حدود مقادیر در توزیع نرمال داریم و میدانیم حدودا بین $\mu \pm 3\sigma$ قرار دارند، ترجیح دادم همین بازه را در plot رسم کنم.و برای کف بازه هم که میدانیم مقدار توزیع دو جمله ای منفی نمیشود اما خب نرمال ممکن است منفی بشود پس کشیدن آن نکته ای نداشت)

اما برای دو جمله ای تابعی زدم که به روش1 و بطور کاملا دستی بیاید و نسبت به سمپل هایی که داریم، PMF را برای توزیع دوجمله ای محاسبه کند:

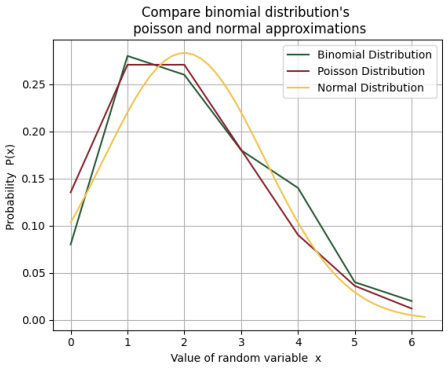
```
def calculate_prob_of_binomial_x(x, binom_sample):
  return sum(binom_sample == x)/samples_count

binom_sample = np.random.binomial(
    Num_of_bernollies, Probability, samples_count)
binomial_probs = np.array([calculate_prob_of_binomial_x(t, binom_sample)
                           for t in discrete_test_values])
```

و سپس همانند قبلی ها آنرا رسم کردم:

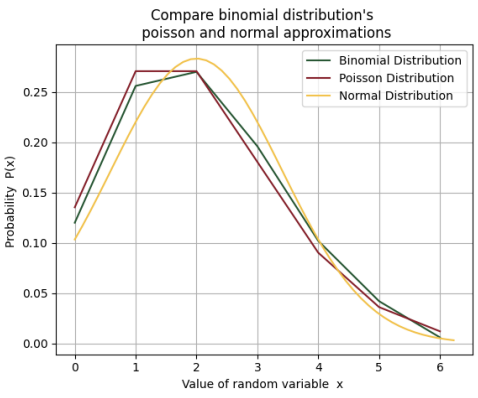
```
ax.plot(discrete_test_values, binomial_probs,
        color='#285734', label="Binomial Distribution")
```

در نهایت توسط این روش برای تعداد نمونه های مختلف به این نتایج میرسیم:

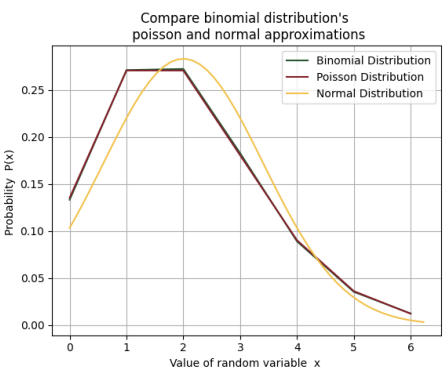
Samples_count = 50



Samples_count = 500



Samples_count = 50000



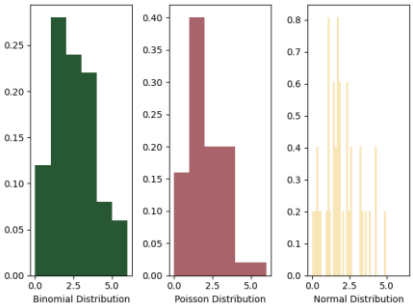
مشخص است که توزیع پواسون مخصوصا با افزایش نمونه ها بسیار تقریب بهترین است.

-چون دقیق نمیدانستم همچین چیزی مدنظر سوال است و یا مدنظرش است که از روش اول برویم برای محاسبه ی احتمال ها، یکبار هم بوسیله ی تابع axis.hist همین چیزها را بصورت هیستوگرام و البته به روش اول و با محاسبه ی کاملا دستی تمام مقادیر احتمال انجام دادم، یعنی هر دفعه که برنامه ران میشود یک دسته نمونه ی جدید گرفته شده و PDF و PMF آنها محاسبه شده و هیستوگرام ها رسم میشوند:

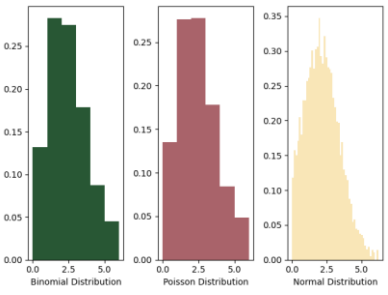
```
fig, ax = plt.subplots(1, 3)
ax[0].hist(np.random.binomial(Num_of_bernollies, Probability,
           samples_count), discrete_test_values, density=True, color='#285734')
ax[1].hist(np.random.poisson(poisson_mu, samples_count),
           discrete_test_values, alpha=0.7, density=True, color='#85212b')
ax[2].hist(np.random.normal(normal_mean, normal_standard_deviation,
           samples_count), continuous_test_values, alpha=0.4, density=True,
color='#f2c24b')
ax[0].set(xlabel="Binomial Distribution")
ax[1].set(xlabel='Poisson Distribution')
ax[2].set(xlabel='Normal Distribution')
```

نتایج این روش نیز همانند قبلی شده که برای مقادیر مختلف samples_count عبارتند از:

Samples_count = 50

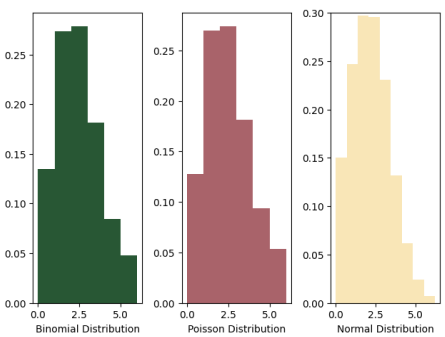


sample_count = 5000

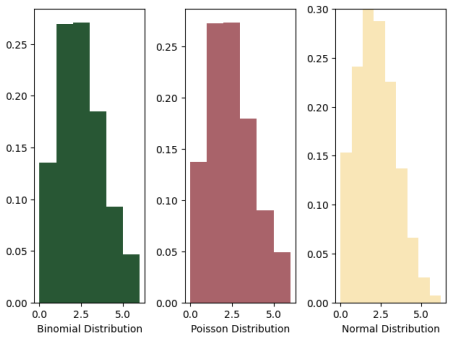


برای بهتر شدن مقایسه میتوان در linespace ای که برای متغیر نرمال تعریف کرده ایم، تعداد متغیر ها را فیکس کنیم روی مثلا 10 تا، تا با محاسبه ی احتمال بین هر دو عددی که رویش یک ستون هیستوگرام کشیده شده است، شکل هیستوگرام نرمال هم به شکل های گسسته نزدیک تر شود و مقایسه راحتتر شود:

sample_count = 5000

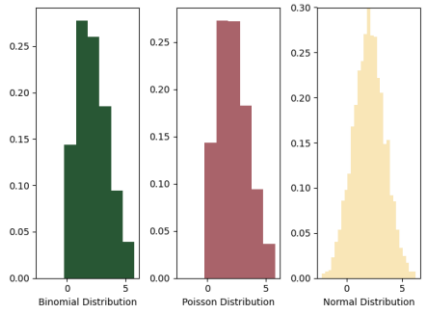


samples_count = 50000



***بخش 2 :**

اکثر تحلیل ها را در بخش قبلی نوشتم اما نتیجه ی کلی این است که در زمان هایی که همانند این مثال، یک توزیع دوجمله ای داریم که در آن همزمان هم p کوچک است و هم n بزرگ است بطوریکه حاصلضرب np یک عدد از بازه ی حدود 1 تا 10 است، در حالت تئوری همیشه و در حالت عملی وقتی تعداد نمونه هایمان زیاد باشند، تقریب زدن توزیع دو جمله ای با توزیع پواسونی با پارامتر np تقریب بسیار خوبی است که همانطور که در plot و هیستوگرام های روش های تئوری و عملی من مشخص بودند، در نمونه های زیاد رسما معادل با توزیع دو جمله ای میشود جوابمان. در تعداد نمونه های کم هم اگر بخواهد نویزی اتفاق بیوفتد، در اطراف میانگین است زیرا کلا مقدار P(x) زیاد است و باز هم درصد نویزی که هست، همان مقدار است که در جاهای دیگر است ولی اینجا بیشتر به چشم می آید. از طرف دیگر توزیع نرمال در نقطه ی پیکش از دو توزیع دیگر بیشتر است و همچنین برخلاف دو توزیع دیگر بوضوح در مقادیر منفی هم ناصفر میشود که اگر بازه ها را جابه جا کنیم مشخص است:

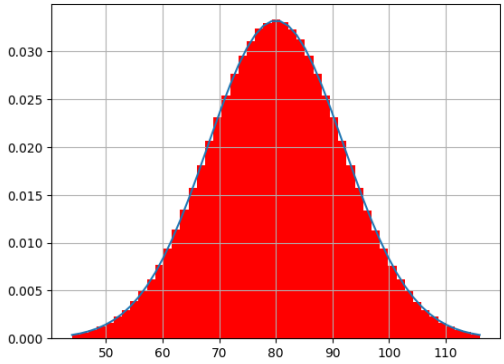


سوال سوم

ابتدا میتوانیم برای شهود بهتر نسبت به موضوع، نمودار هیستوگرام اسکیل شده و plot را برای این توزیع بکشیم:

```
# we can draw plot
fig, ax = plt.subplots()
test_values = np.linspace(normal_mean-
3*normal_standard_deviation,normal_mean+3*normal_standard_deviation)
ax.plot(test_values, normal_distribution.pdf(test_values))
ax.grid()
# or histogram
ax.hist(np.random.normal(normal_mean,normal_standard_deviation,samples_count), test_values,
density = True, color='red')
plt.show()
```

که داریم: (تعداد نمونه ها 5000000 هست تا نتیجه ی هیستوگرام و plot که به دو روش مختلف کشیده شده اند مشابه شوند)



برای این سوال نیز در بین انتخاب دو روش محاسبه ی تئوری و عملی به مشکل خوردم در نتیجه هر دو را نوشتم که در ادامه جزییات آنها را خواهم گفت

***روش تئوری**

برای روش تئوری، ابتدا یک توزیع نرمال با پارامترهای گفته شده میسازیم:

```
normal_distribution = norm(normal_mean, normal_standard_deviation)
```

حالا برای این متغیر، تمام توابع احتمالاتی مانند pdf , cdf, ppf را داریم

ppf همان تابع معکوس cdf است:

```
# F(x) = 0.9 => x = ppf(0.9)
```

در نتیجه پاسخ بخش های مختلف سوال براحتی بدست می ایند:

```
best_students_min_score_theory = normal_distribution.ppf(0.9)
max_score_in_second_square_theory = normal_distribution.ppf(0.5)
max_score_in_third_square_theory = normal_distribution.ppf(0.75)
less_than_80_prob_theory = normal_distribution.cdf(80)
less_than_90_prob_theory = normal_distribution.cdf(90)
between_80_and_90_theory = less_than_90_prob_theory - less_than_80_prob_theory
```

که در آن برای بخش اول سوال داریم:

```
# F(x) = 0.9 => x = ppf(0.9)
```

و برای بخش دوم هم چارک دوم همان صدک 50ام و چارک سوم همان صدک 75ام است پس عددی چارک دوم است که F(x) = 0.5 که یعنی نمره ای که 50 درصد بچه ها نمره ای کمتر از آن گرفته اند و چارک

سوم هم یعنی F(x) = 0.75 که یعنی نمره ای که 75 درصد بچه ها نمره ای کمتر از آن گرفته اند در نتیجه با کد زیر میتوان حوزه ای که نمرات بین چارک دوم و سوم میسازند را یافت:

```
max_score_in_second_square_theory = normal_distribution.ppf(0.5)
max_score_in_third_square_theory = normal_distribution.ppf(0.75)
```

که ابتدای بازه همان max_score_in_second_square_theory است و انتهای بازه همان max_score_in_third_square_theory است. پس خروجی اینگونه است:

```
best_students_min_score_theory:  95.3786187865352
scores_beween_second_and_third_square_theory:
 80.0  to  88.09387700235298
between_80_and_90_theory:  0.29767161903635686
```

و مقادیر دریافت شده با هر بار ران کردن ثابت هستند زیرا از توابع و فرمول های ثابت تئوری بدست آمده اند.

***روش عملی**

در این روش ابتدا یک دسته متغیر تصادفی نرمال تولید میکنیم :

```
test_values = np.random.normal(loc = normal_mean, scale=normal_standard_deviation, size = samples_count)
```

و سپس تمام کارها را روی همان انجام میدهیم در نتیجه با هر بار ران کردن پروژه به نتایج جدیدی میرسیم که حاصل از train شدن همان دسته متغیر تصادفی است، برای دو بخش اول سوال براحتی از تابع صدک استفاده میکنیم :

```
best_students_min_score_practical = np.percentile(test_values, 90)
max_score_in_second_square_practical = np.percentile(test_values, 50)
max_score_in_third_square_practical = np.percentile(test_values, 75)
```

اما برای بخش سوم سوال نیاز داریم که CDF را بیابیم پس:

```
sorted_values = np.sort(test_values)
n = len(sorted_values)
cdf = np.arange(n + 1) / n
less_than_80_prob_practical = cdf[np.searchsorted(sorted_values, 80, side='right')]
less_than_90_prob_practical = cdf[np.searchsorted(sorted_values, 90, side= 'right')]
between_80_and_90_practical = less_than_90_prob_practical - less_than_80_prob_practical
```

ابتدا مقادیر تصادفی را سورت کرده و سپس یک آرایه با اعداد 1/n تا n/n بطور موازی با آن تعریف میکنیم سپس برای یافت CDF هر مقدار، از سمت راست آرایه ی دوم شروع کرده و اولین جایی که در آرایه ی سورت

شده، به آن مقدار رسیدیم را از آرایه یCDF میخوانیم که آنجا همان مقدار CDF مدنظرمان است. پس خروجی یکی از دفعاتی که ران کردم اینگونه است:(فرض کردم samples_count =5000000 است)

```
best_students_min_score_practical:  95.37995586818414
scores_beween_second_and_third_square_practical:  79.99521479295157  to  88.0854637738718
between_80_and_90_practical:  0.2976278
```

پس چون تعداد نمونه ها زیاد بود خروجی بخش تئوری با عملی تفاوت زیادی نمیکرد.

سوال سوم(امتیازی)

ابتدا فرضیاتم را بیان و توجیح میکنم:

اولا که نمرات بچه ها حدودا از 20 نمره است اما ممکن است برخی ها با نمرات امتیازی به بالای 20 هم برسند. از طرفی در هر کلاسی معمولا بالای 70و80 درصد بچه ها پاس میشوند که یعنی نمره ی بالای 12 دارند، پس بعنوان یک حدس حدودی میانگین نمرات بچه ها در هر سه درس را 16 گرفتم و میزان انحراف معیار این نمرات را هم 1.5 گرفتم تا اگر توزیع کل را نمایی بگیریم، اکثر نمرات در بازه ی $16-3*1.5=11.5$ تا $16+3*1.5=20.5$ بیوفتند، و طبق همین تقریب، پارامتر های سه توزیع را تعیین کردم:

برای توزیع پواسون: میانگین نمرات را 16 گرفتیم پس پارامتر لاندا در پواسون که مساوی با میانگین و واریانس مقادیر هم هست را 16 میگیریم

برای یکنواخت: طبق محاسبات قبلی، نمرات بین 11.5 تا 20.5 هستند پس سر و ته توزیع را 11.5 و 20.5 گرفتم

برای نمایی: تنها پارامتر موجود لامبدا است که 1/لامبدا مساوی میانگین است در نتیجه اگر قرار بود مستقیما لامبدا را بگذاریم، 1/16انتخاب منطقی ای بود اما در آرگومان متودی که برای تابع نمایی داریم، همان 1/lambda گذاشته میشود پس همان 16 را میگذاریم.

در نتیجه داریم:

```
scores_mean = 16
scores_standard_deviation = 1.5 #scores between 16-4.5 and 16+4.5
```

ابتدا تعداد سمپل هایی از هر متغیر تصادفی که قرار است دریافت شده و از آنها میانگین بگیریم را تعیین میکنیم:

```
samples_count = 10000
```

سپس تعداد میانگین هایی که قرار است حساب کنیم و ثابت کنیم اجماعا دارای توزیع نمایی هستند را تعیین میکنیم:

```
means_count = 10000
```

پس برای هر توزیع قرار است به تعداد 10000 بار هر بار 10000 تا نمونه بگیریم و میانگین آنها را حساب کنیم و نهایتا توزیع بین میانگین ها را حساب کنیم:

برای گسسته با توزیع پواسون داریم:

```
means_sample_poisson = np.array([sum(np.random.poisson(
    lam=scores_mean, size=samples_count))/samples_count for i in range(means_count)])
```

برای فیزیک با توزیع یکنواخت داریم:

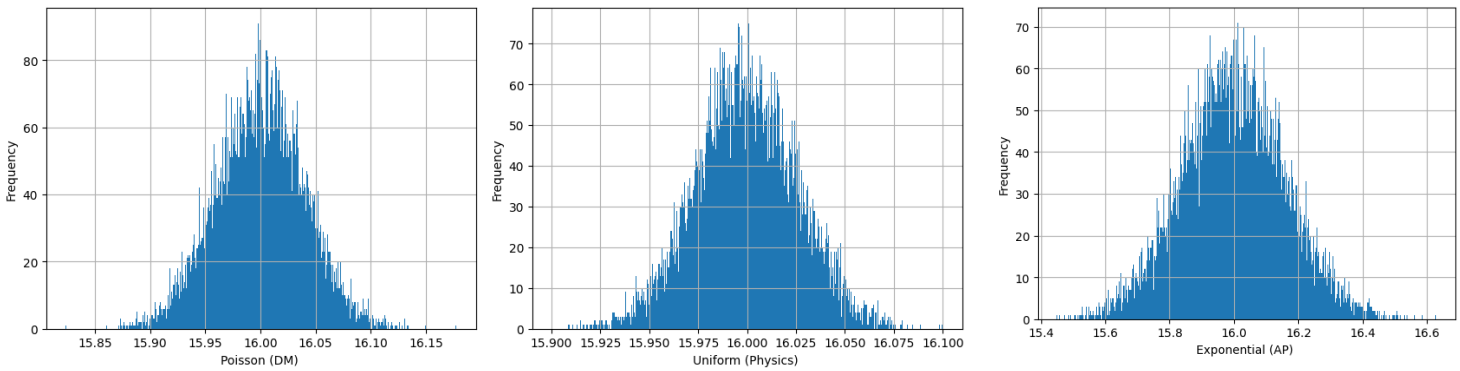
```
means_sample_uniform = np.array([sum(np.random.uniform(
    low=scores_mean-3*scores_standard_deviation,
high=scores_mean+3*scores_standard_deviation, size=samples_count))/samples_count for i in
range(means_count)])
```

و برای APبا توزیع نمایی داریم:

```
means_sample_exp = np.array([sum(np.random.exponential(
    scale=scores_mean, size=samples_count))/samples_count for i in range(means_count)])
```

پس الان 3 توزیع از میانگین ها داریم که لازم است رسم کنیم و شباهتشان با توزیع نمایی را بررسی کنیم:

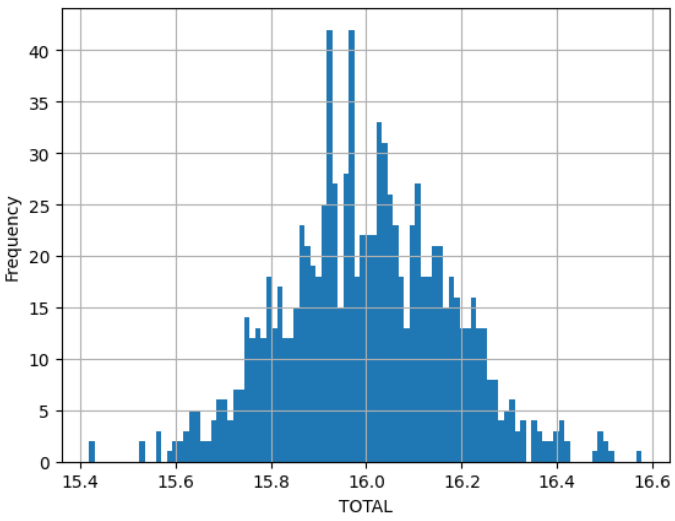
*مقادیر رسم شده با samples_count = 10000 و means_count = 10000 و bins = 500 در هر هیستوگرام میباشند و هیستوگرام ها اسکیل شده اند تا مجموعشان 1 شود:

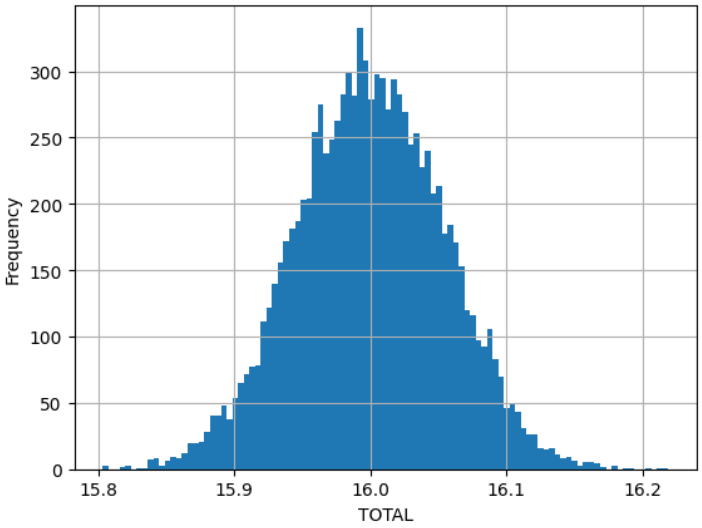


در نتیجه همانطور که انتظار می رفت تک تک توزیع ها به سمت یک توزیع نمایی میل کردند که حالا چون پارامتر ها را از قصد طوری تنظیم کردم که میانگین نهایی مساوی 16 شود، همگی به توزیع نرمالی با میانگین 16 میل کرده اند.

از طرفی اگر توزیع مجموع این سه تا را هم رسم کنیم و آنرا اسکیل کنیم دوباره، داریم:

```
plt.hist((means_sample_poisson+ means_sample_uniform+means_sample_exp)/3, bins=500)
```

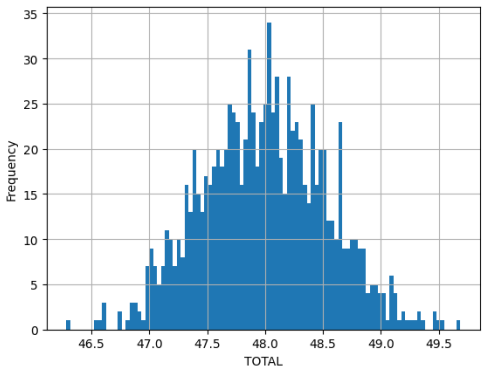


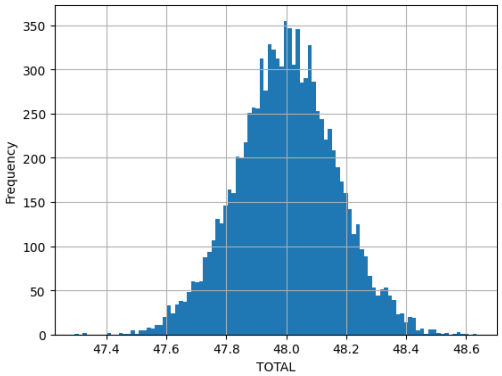


در هر دو تصویر مقدار bins = 100 است اما در سمت راست مقدار samples_count = 10000 و means_count = 10000 است و در سمت چپ samples_count = 1000 و means_count = 1000 است پس با افزایش تعداد نمونه گیری ها، توزیع حاصل از تک تک مجموع میانگین ها و همچنین توزیع کلی حاصل از حاصلجمع کلی تمام میانگین ها، بیشتر به نرمال میل میکند

اگر هم با تقسیم بر 3 کردن اسکیل نکنیم، به همچین حالتی برای مجموع سه توزیع داریم(سمت راست با تعداد سمپل های بیشتری کشیده شده)

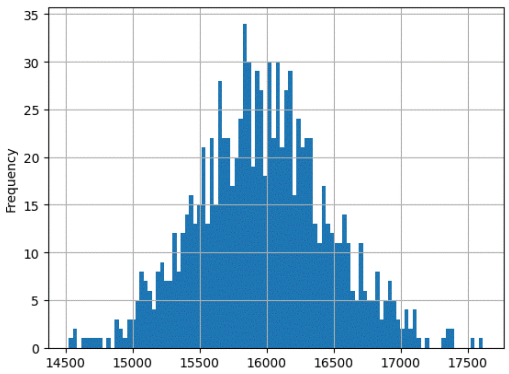
پس نهایتا به حالتی شبیه نرمال رسیدیم با میانگین 3*16 = 48





البته این قضیه ی حد مرکزی درباره ی متغیر هایی از فقط یک توزیع که دقیقا شبیه هم هستند صادقتر است، و اگر قرار باشد از توزیع های مختلف در کنار هم استفاده کنیم باید تعداد بسیار بالا برود یا همانند این کاری که الان کردم، پارامتر های آنها را حدودا شبیه به هم انتخاب کنیم.

در کل هم اگر از توزیع ها میانگین نمیگرفتیم و همینطوری جمع میزدیم، به ازای هر n متغیر تصادفی از نوع X با میانگین m و انحراف معیار s ، نهایتا حاصلجمع آنها به یک Y از توزیع نرمال با میانگین nm و انحراف معیار sn^0.5 میرسید



سوال چهارم

این سوال بسیار شبیه به سوال دوم است با این تفاوت که در آنجا بدلیل کم بودم مقدار P در نهایت توزیع پواسون تقریب بهتری بود اما اینجا چون مقدار pنزدیک به 0.5 است و فقط مقدار n است که زیاد است، تقریب نرمال بهتر است.

❋بخش 1 :

دقیقا مثل سوال 2 ابتدا پارامتر ها را معادل سازی میکنیم و یک تعدادی برای samples_count در نظر میگیریم:

```
Probability = 0.45
Num_of_bernollies = 7072
poisson_mu = Probability * Num_of_bernollies
normal_mean = Probability * Num_of_bernollies
normal_standard_deviation = np.sqrt(Probability * (1-Probability) * Num_of_bernollies)
samples_count = 500000
```

حالا باز دو روش برای محاسبه ی احتمال ها داریم: دستی و تئوری.

در نتیجه ابتدا به روش دستی عمل کرده و PMF را برای متغیر دوجمله ای و پواسون می یابیم:

```
def calculate_prob_of_binomial_x(x):
    return sum(np.random.binomial(Num_of_bernollies, Probability, samples_count) ==
x)/samples_count
def calculate_prob_of_poisson_x(x):
    return sum(np.random.poisson(poisson_mu, samples_count) == x)/samples_count

discrete_test_values = np.arange(np.floor(normal_mean-3*normal_standard_deviation),
np.floor(normal_mean+3*normal_standard_deviation))
continuous_test_values = np.linspace(normal_mean-3*normal_standard_deviation,
normal_mean+3*normal_standard_deviation)

binomial_probs = np.array([calculate_prob_of_binomial_x(t)
                           for t in discrete_test_values])
poisson_probs = np.array([calculate_prob_of_poisson_x(t)
                           for t in discrete_test_values])
```

*مقادیر بازه ی تغییر x را از normal_mean-3*normal_standard_deviation تا normal_mean+3*normal_standard_deviation گرفتم چون طبق قواعد ریاضیاتی میدانیم اغلب مقادیر توزیع نرمال و در نتیجه حدودا توزیع دوجمله ای و پواسون، در این حوزه قرار دارند.

و برای رسم آنها هم داریم:

```
ax.plot(discrete_test_values, binomial_probs,
        color='#285734', label="Binomial Distribution", marker = 'o')
ax.plot(discrete_test_values, poisson_probs,
        color='#85212b',label='Poisson Distribution', marker = 'o')
```

*بدیهتا میتوانسیم به روش تئوری و با تابع PMF هم مثلا برای پواسون یا دوجمله ای را حساب کنیم، اما در آن صورت همواره به پاسخ ثابتی میرسیدیم که از فرمول تئوری ناشی میشود:

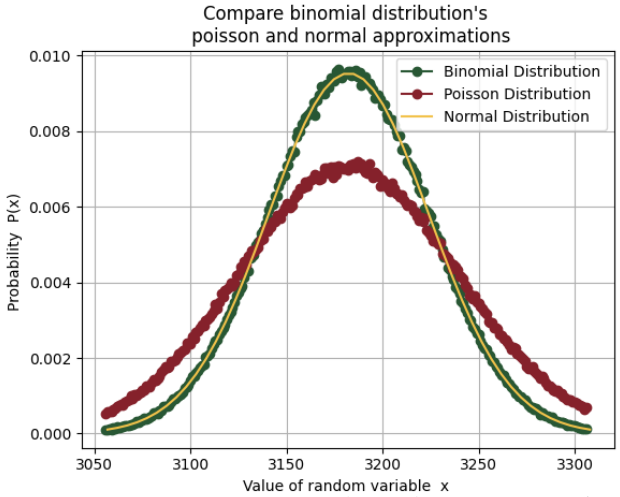
```
ax.plot(discrete_test_values, poisson.pmf(discrete_test_values, poisson_mu),
        color='#85212b',label='Poisson Distribution', marker = 'o')
ax.plot(discrete_test_values, binom.pmf(discrete_test_values,Num_of_bernollies,
Probability),
        color='#285734',label='Binomial Distribution', marker = 'o')
```

و برای نرمال از همین روش تئوری پیش میرویم:

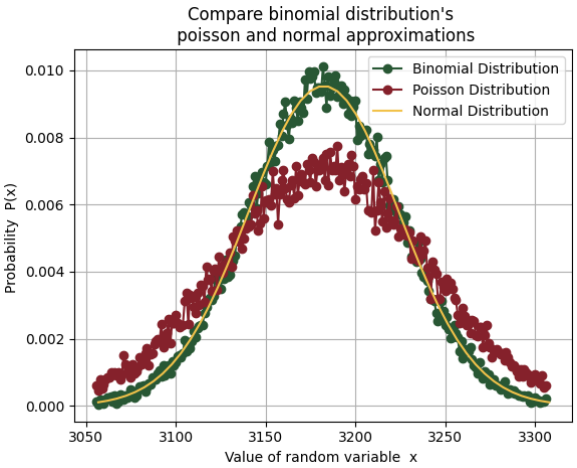
```
ax.plot(continuous_test_values, norm.pdf(continuous_test_values, normal_mean,
normal_standard_deviation),
        color='#f2c24b', label='Normal Distribution')
```

نهایتا برای این بخش داریم:

samples_count = 1000000



samples_count = 50000



نمودار ها را نقطه ای کشیدم جهت تنوع دادن نسبت به سوال2، و اینکه نویز ها بطور جالتبری مشخص میشوند اینگونه.

***بخش 2 :**

همانطور که گفتم، برای نرمال از محاسبات تئوری حدودی استفاده کردم به همین دلیل نویز ندارد در نمودارش، اما به هر حال به نتیجه ای که منتظرش بودیم رسیدیم، یعنی در این مسئله در در نتیجه کلا در مسائلی که در توزیع دو جمله ای مقدار n زیاد میشود اما مقدار p همچنان کم نیست و در حدود 0.5 است، تقریب نرمال تقریب بسیار بهتری نسبت به پواسون است. و درست است که توزیع نرمال توزیع پیوسته است اما میتوان با **اعمال تصحیح پیوستگی** در حین محاسبه ی احتمال ها، به نتایجی همانند نمودار های بالا رسید و نتیجه گرفت که تقریب نرمال بهتر است، از طرفی اگر هم تفاوتی بین دوجمله ای و نرمال باشد، بیشتر در حول میانگین آن است که اتفاقا مقدار p(x) هم بیشتر است در نتیجه نهایتا درصد نویز نسبت به مقدار P خیلی زیاد نبوده و این تقریب مخصوصا در ابعاد بزرگ نمونه ها تقریب خوبیست در حالی که پواسون هم پیک کوتاه تری دارد و هم انحراف معیار بیشتری دارد و سریع تر پخش میشود نسبت به دو توزیع دیگر

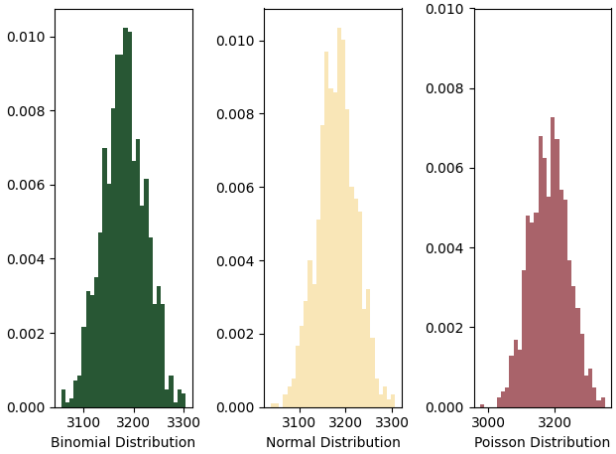
در حالت بعدی همه ی سه توزیع را بطور دستی حساب کرده و هیستوگرام میکشم:

```
fig, ax = plt.subplots(1, 3)
ax[0].hist(np.random.binomial(Num_of_bernollies, Probability,
           samples_count), density=True, color='#285734', bins= 30)
ax[1].hist(np.random.normal(normal_mean, normal_standard_deviation,
```

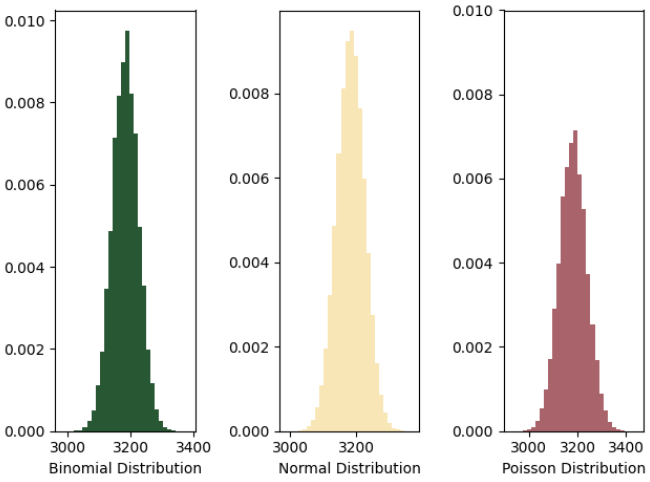
```
        samples_count), alpha=0.4, density=True, color='#f2c24b', bins= 30)
ax[2].hist(np.random.poisson(poisson_mu, samples_count), alpha=0.7, density=True,
color='#85212b', bins= 30)
```

در هر دفعه یک نمونه ی تصادفی به اندازه ی مشخص شده ساخته می شود و سپس هیستوگرامش رسم شده و سپس با ارگومان density=true آنرا اسکیل میکنیم تا به حالت احتمالاتی برسد. و داریم:

samples_count = 1000



samples_count = 1000000



با توجه به نمودار ها کاملا مشخص است توزیع پواسون هم پیک کوتاه تری دارد و هم از لحاظ پخش شدگی، در نزدیک 3000 و 3400 مقدار بیشتری نسبت به دو توزیع دیگر دارد که یعنی انحراف معیارش بیشتر است و در نهایت نرمال تقریب بهتریست.