بسم الله الرحمن رحیم

پروژه ۲ آمار و احتمالات مهندسی

دکتر توسلی پور

مهدی وجهی

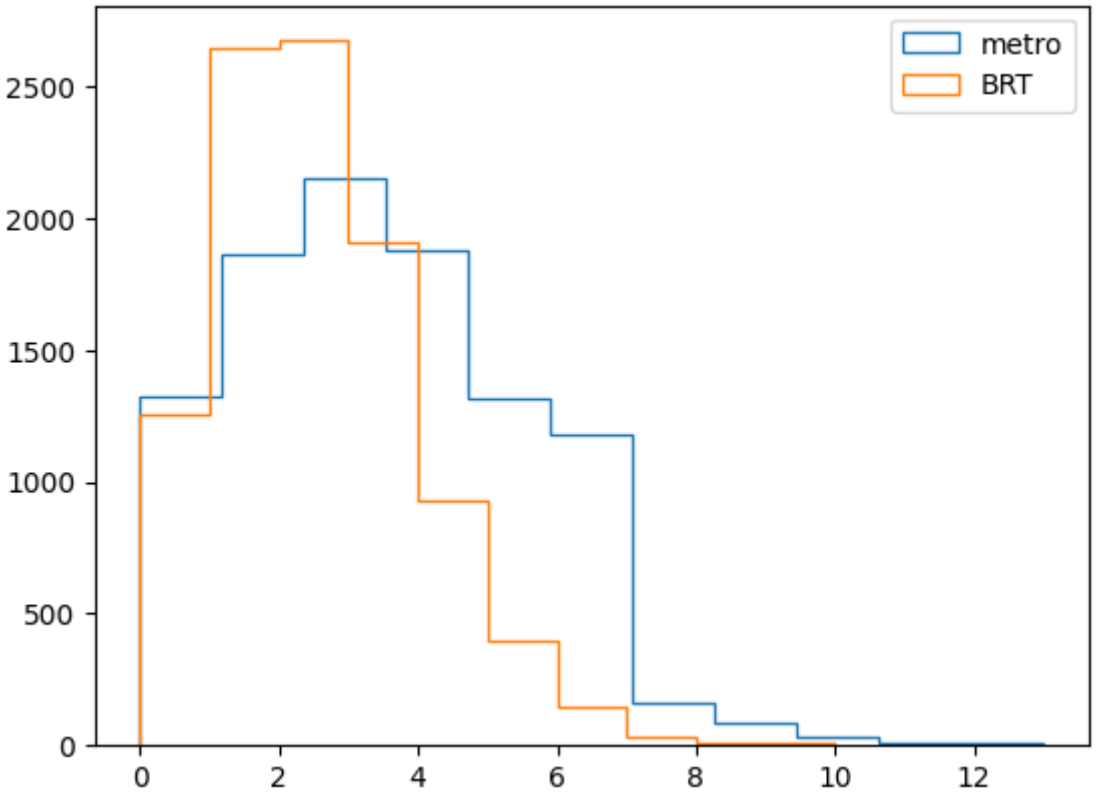
۸۱۰۱۰۱۵۵۸

فهرست

سوال ۱

بخش ۱

در این قسمت خواسته شده که اطلاعات مربوط به اتوبوس و مترو را رسم کنیم.
به آسانی و با استفاده از کتابخانه matplotlib آن را می کشیم

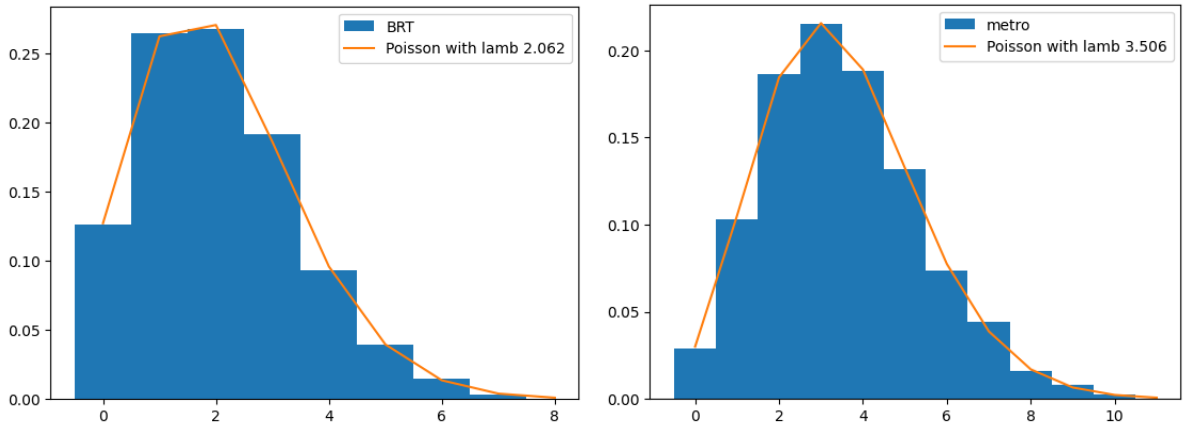


بخش ۲

با بررسی نمودار ها متوجه شباهت آنها با توزیع پواسون می شویم. برای محاسبه پارامتر لامبدا از کتابخانه scipy و تابع curve_fit استفاده می کنیم. این تابع با دریافت نقاط پارامتر تابع داده شده را پیدا می کند.
کد به صورت زیر است:

```
def fit_poisson(data_set : list, name : str, plot_range : int) -> float:
    x_plot = np.arange(plot_range)
    entries, bin_edges, _ = plt.hist(data_set, bins=x_plot-0.5, density=True,
label=name)

    mid_bins = (bin_edges[1:] + bin_edges[:-1]) * 0.5

    poi_lamb, _ = curve_fit(lambda k, lamb: poisson.pmf(k, lamb), mid_bins,
entries)
    poi_lamb = poi_lamb[0]

    plt.plot(poisson.pmf(x_plot, poi_lamb), label="Poisson with lamb " +
str(poi_lamb)[:5])
    plt.legend()
    plt.show()

    return poi_lamb
```

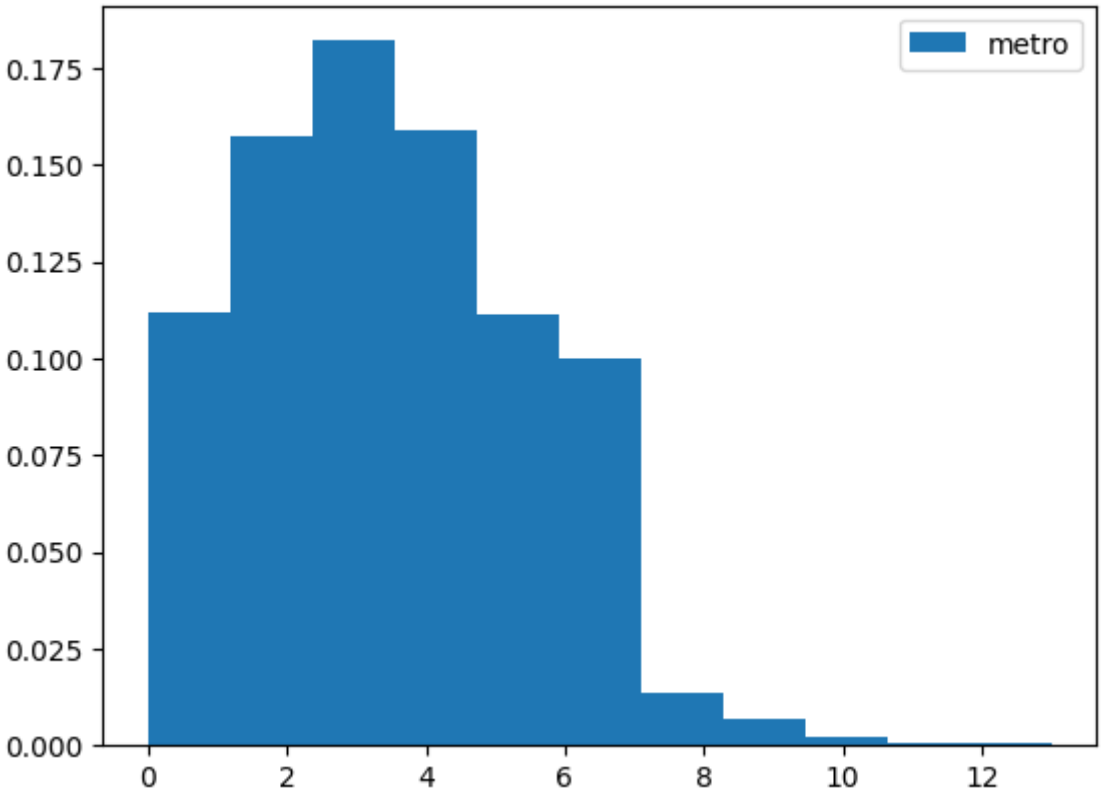
در این تابع ابتدا نمودار ستونی را رسم می کنیم فقط باید به این نکته توجه داشت که برای رسم پواسون نیاز داریم که نمودار خود را با `density=True` درست کنیم که باعث می شود از نمودار درصد بگیرد. خود تابع نمودار ستونی به ما دو مکان لبه های ستون ها (`bin_edges`) و مقدار ستون ها (`entries`) بر می گرداند سپس نقاط وسط نمودار های ستونی را به دست می آوریم. حال تابع `curve_fit` را صدا می زنیم و به آن تابع پواسون و مکان ستون ها و مقدار آن ها را پاس می دهیم تا لامبدا توزیع پواسون را به دست بیاورد. در آخر هم آن را رسم می کنیم.
نتایج به شکل زیر است:



لامبدا برای مترو برابر با ۳.۵ و برای اتوبوس برابر با ۲.۰۶ است.

بخش ۳

برای این بخش تنها کافیست `density=True` را در آرگومان ها قرار دهیم.
نمودار به شکل زیر است:



بخش ۴

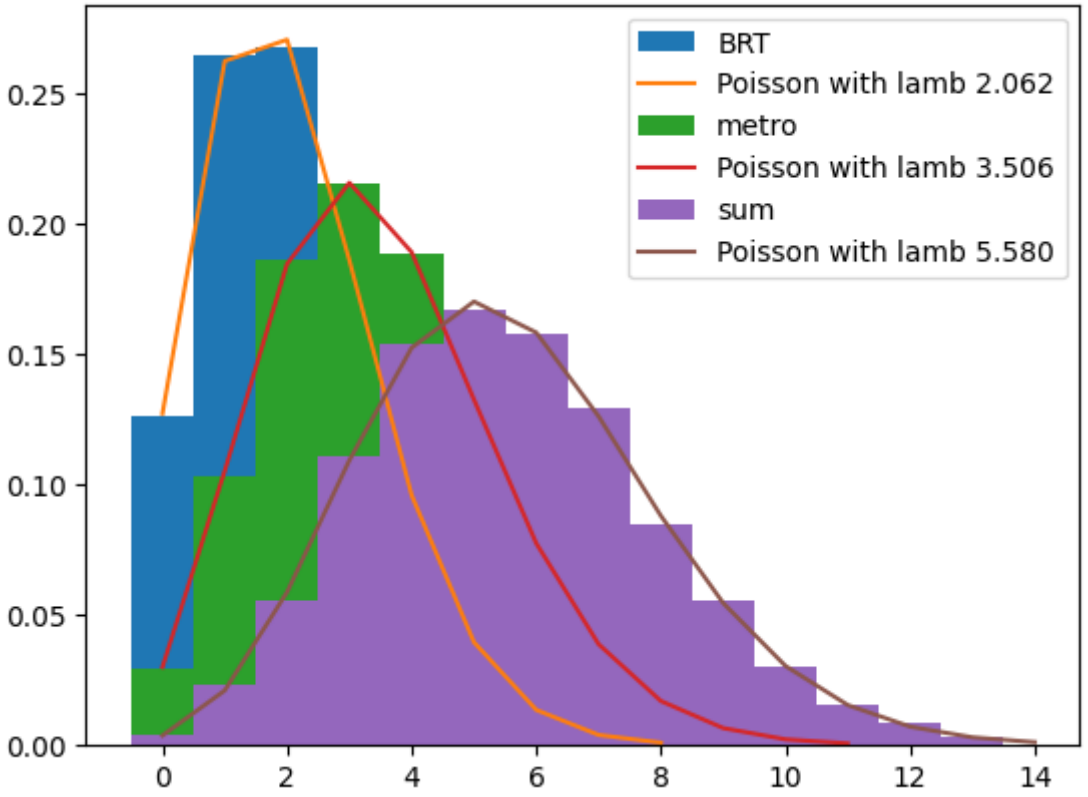
این کار در **بخش ۲** انجام شده بود.

بخش ۵

می دانیم که مجموع ۲ توزیع پواسون خود نیز توزیعی پواسون با پارامتر مجموع ۲ توزیع قبلی است. با استفاده از تابع نوشته شده در بخش ۲ آن را حساب می کنیم.

```
lamb["BRT"] = fit_poisson(data["BRT"], "BRT", 9)
lamb["metro"] = fit_poisson(data["metro"], "metro", 12)
lamb["sum"] = fit_poisson([sum(values) for values in zip(*data.values())],
"sum", 15)
plt.legend()
plt.show()
```

خروجی نموداری به شکل زیر است و پارامتر Z می شود ۵.۵۸ که مجموع پارامتر های قبلی است.



بخش ۶

کافیست توزیع شرطی را باز کنیم و آن را ساده کنیم.

$$W = P(X|X + Y = n) = \frac{P(X=m,X+Y=n)}{P(X+Y=n)}$$

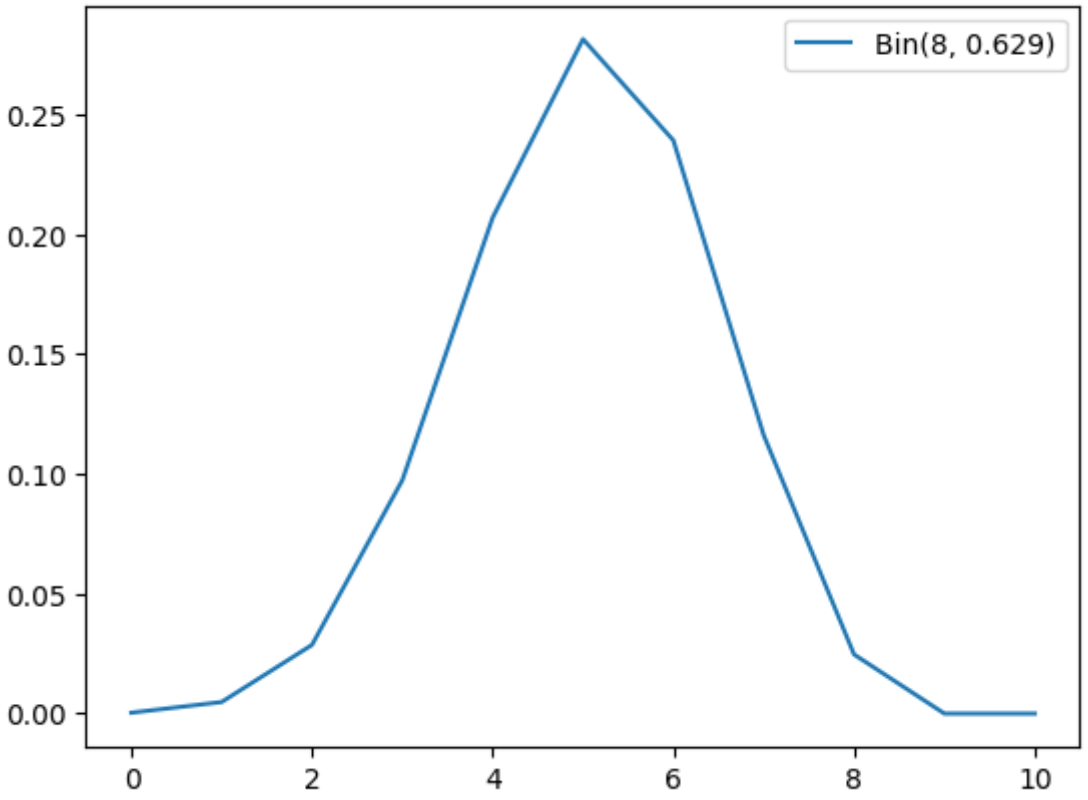
$$= \frac{P(X=n,Y=n-m)}{P(X+Y=n)} = \frac{\frac{e^{-\lambda_x}\lambda_x^m}{m!} \times \frac{e^{-\lambda_y}\lambda_y^{n-m}}{(n-m)!}}{\frac{e^{-(\lambda_x+\lambda_y)}(\lambda_x+\lambda_y)^n}{n!}}$$

$$= \binom{n}{m} \left(\frac{\lambda_x}{\lambda_x+\lambda_y}\right)^m \left(\frac{\lambda_y}{\lambda_x+\lambda_y}\right)^{n-m} = Bin\left(n, \frac{\lambda_x}{\lambda_x+\lambda_y}\right)$$

بخش ۷

طبق بخش ۶ توزیع دوجمله ای را با پارامتر های داده شده می کشیم.

```
n = 8
p = lamb["metro"] / (lamb["metro"] + lamb["BRT"])
x_plot = np.arange(11)
plt.plot(binom.pmf(x_plot, n, p), label="Bin(" + str(n) + ", " + str(p)[:5]
+ ")")
```

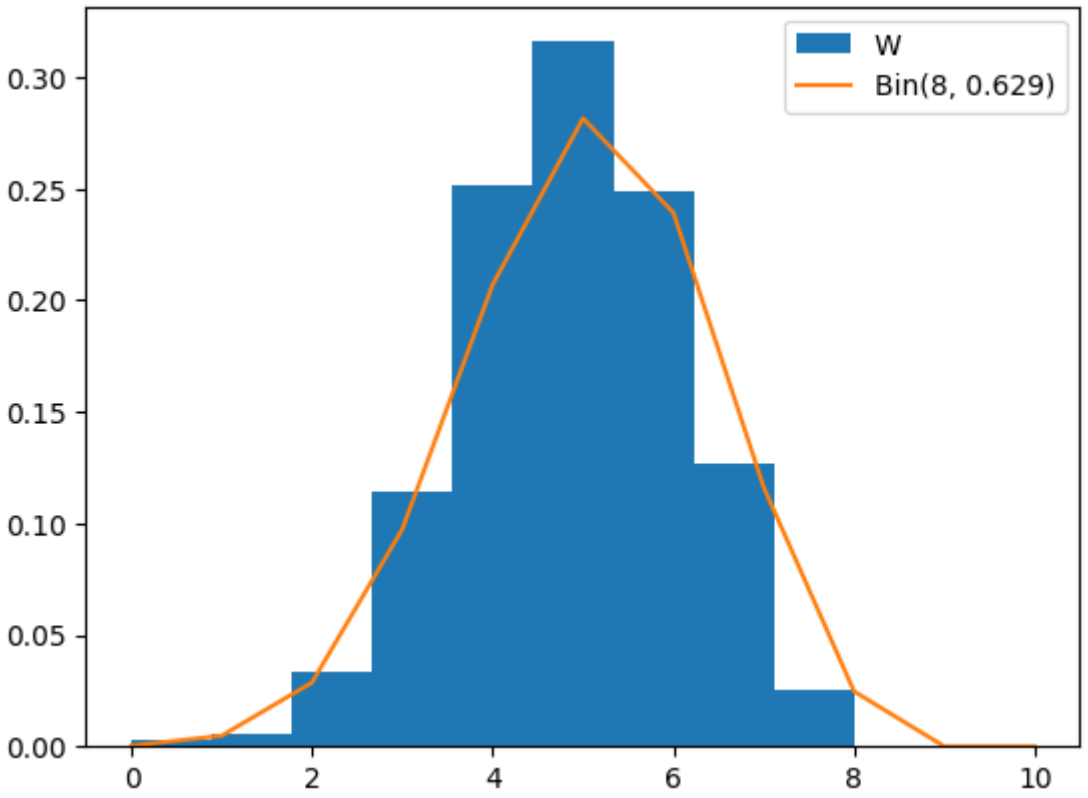


بخش ۸

توزیع شرطی را به شکل زیر به دست می آوریم.

```
for i in range(len(data["metro"])):
   if data["metro"][i] + data["BRT"][i] == n:
       W.append(data["metro"][i])
```

سپس آن را رسم می کنیم.



همانطور که مشاهده می شود توزیع دوجمله ای به صورت تقریبی بر توزیع اصلی منطبق است و همخوانی دارد.

سوال ۲

بخش ۱

ابتدا یک بار مسئله را شبیه سازی می کنیم به این صورت که یک فهرست از کالابرگ ها درست می کنیم و با استفاده از اعداد شانسی نمونه برداری می کنیم و در فهرست کالابرگ ها را علامت می زنیم. این کار را تا وقتی که همه ی کالا برگ ها علامت بخورند ادامه می دهیم. (تعداد علامت خورده ها را با استفاده از از شمارنده اندازه می گیریم.) در آخر هم تعداد تلاش هایی که انجام داده ایم را بر می گردانیم.

```
def simulate_coupon(n : int) -> int:
    check_list = [0] * n
    check_list_counter = 0
    try_counter = 0
    while (check_list_counter < n):
        try_counter += 1
        new_coupon = np.random.randint(0, n)
        if check_list[new_coupon] == 0:
            check_list[new_coupon] = 1
            check_list_counter += 1
    return try_counter
```

در ادامه باید مسئله را برای k بار شبیه سازی کنیم. برای این کار یک حلقه می گذاریم و شبیه سازی را k بار تکرار می کنیم و نتایج را جمع می زنیم در آخر هم بر تعداد تقسیم می کنیم و نتیجه را بر می گردانیم.

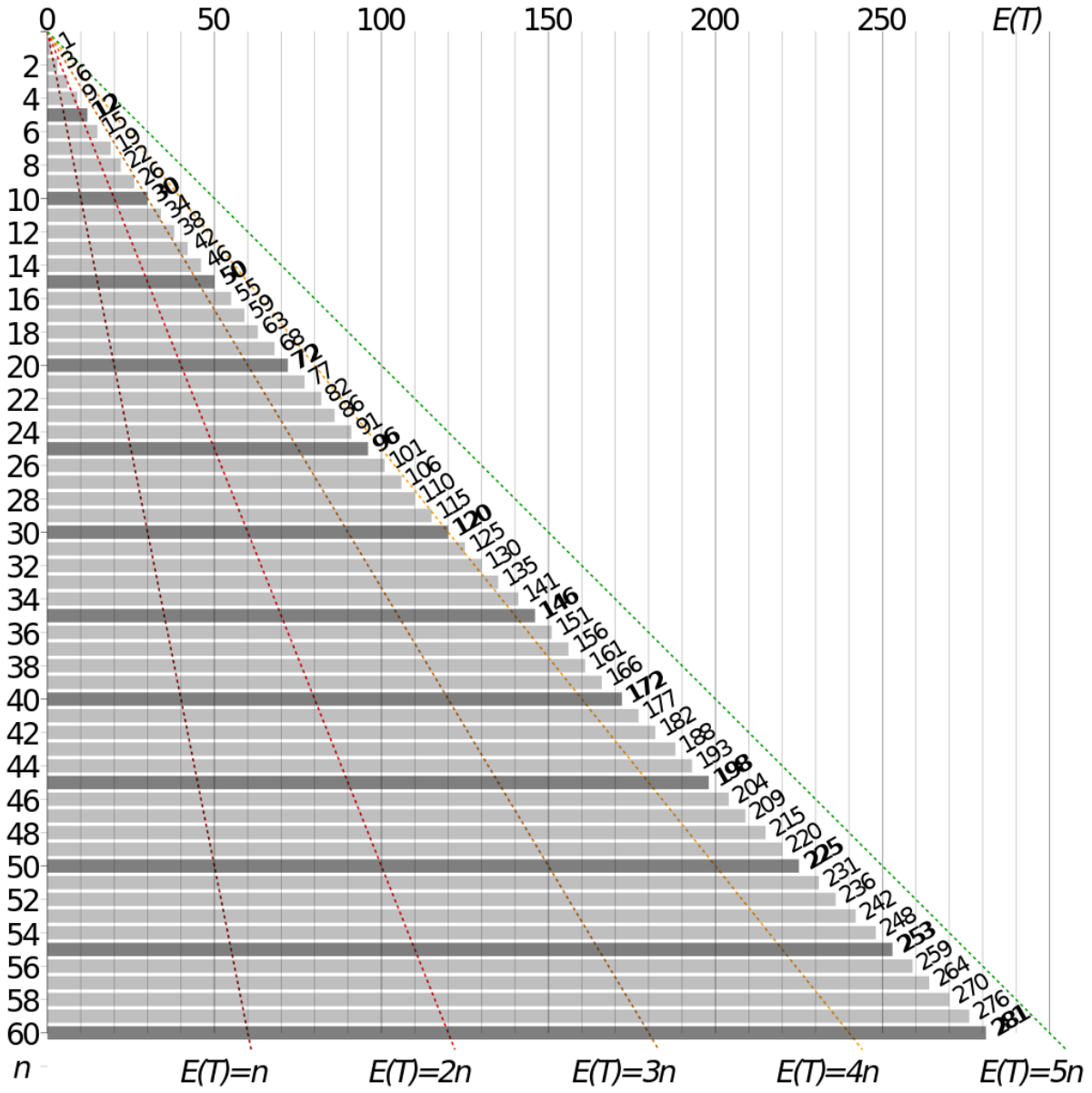
```
def solve_coupon_M_C(n : int, k : int) -> float:
    sum_of_try = 0
    for i in range(k):
        sum_of_try += simulate_coupon(n)
    return sum_of_try / k
```

بخش ۲

با شبیه سازی مسئله با مقادیر n=۱۰ و k=۱۰،۱۰۰،۱۰۰۰ نتایج به صورت زیر می شود:

نتیجه	n	k
۲۹.۱	۱۰	۱۰
۳۱.۱۲	۱۰	۱۰۰
۲۹.۲۲۴	۱۰	۱۰۰۰

همانطور که مشاهده می شود نتیجه به ۲۹.۲ همگراست. با جست و جویی در اینترنت می توان به تصویر زیر رسید که نتیجه ی ما را تایید می کند.



بخش ۳

ابتدا متغیر های خود را تعریف می کنیم.

```
n = sp.symbols('n')
i = sp.symbols('i')
s = sp.symbols('s')
```

سپس تابع مولد را تعریف می کنیم. می دانیم که این تابع برای توزیع هندسی به شکل زیر تعریف می شود.

$$p_i = \frac{n-(i-1)}{n} \qquad \phi_{X_i}(x_i) = \frac{p_i e^s}{1-(1-p_i)e^s}$$

که به صورت زیر پیاده می کنیم:

```
p_i = ((n - (i - 1)) / n)
phi_x = ((sp.E ** s) * p_i) / (1 - ((1 - p_i) * sp.E ** s))
```

بخش ۴

می دانیم که اگر چند متغیر تصادفی مستقل برابر با حاصل ضرب آنهاست.

پس کافیست عبارت زیر را حساب کنیم:

$$\phi_X(x) = \prod_{i=1}^{n} \phi_{X_i}(x_i)$$

که کد آن به صورت زیر است:

```
phi_X = sp.Product(phi_x, (i, 1, n))
```

بخش ۵

برای محاسبه پاسخ لازم است یکبار مشتق بگیریم و سپس مقادیر را جایگذاری کنیم.

```
E_X = sp.diff(phi_X, s).subs({n : 10, s : 0}).evalf()
```

حاصل عبارت برابر است با ۲۹.۲۸۹۶ که با قسمت ۲ تطابق دارد.

سوال ۳

بخش ۱

داده ها را از پرونده مربوطه خوانده و به شکل زیر تبدیل می کنیم:

[label, [px0,px1,...]]

سپس دو عنصر آخر را در جایی دیگر ذخیره کرده و حذف می کنیم.

```
two_last_line = list()
two_last_line.append(data.pop())
two_last_line.append(data.pop())
```

بخش ۲

برای این کار کافیست به جای مقدار هر پیکسل نتیجه مقایسه آن با ۱۲۸ را بگذاریم:

```
bin_data = [[i[0], [int(j >= 128) for j in i[1]]] for i in data]
```

بخش ۳

ابتدا یک نمونه تصادفی انتخاب می کنیم:

```
random_case = bin_data[np.random.randint(0, len(bin_data))]
```

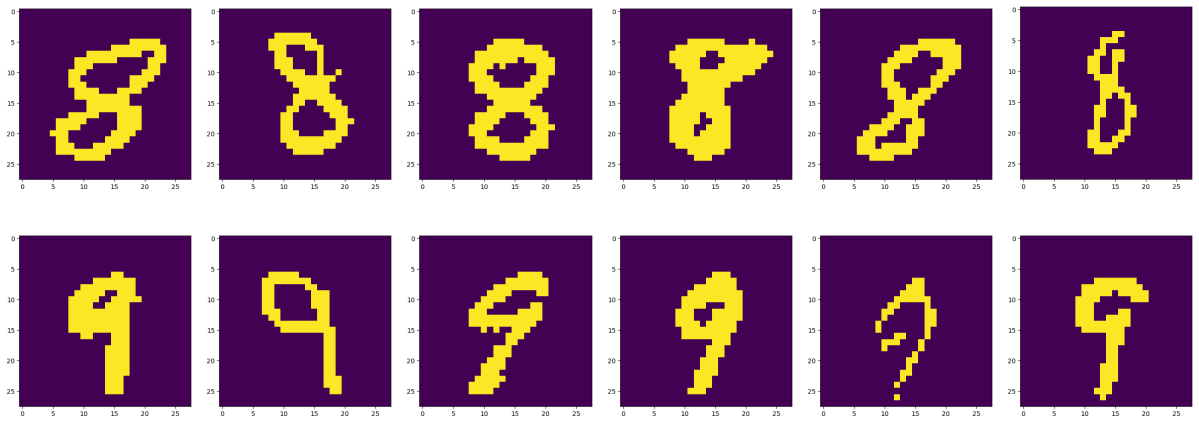
سپس فهرست پیکسل ها را به آرایه ۲ بعدی تبدیل می کنیم:

```
img_m = np.reshape(random_case[1], (28,28))
```

در پایان این بخش نیز خروجی را نمایش می دهیم:

```
print(random_case[0])
_ = plt.imshow(img_m)
```

که نمونه های آن به صورت زیر است:



بخش ۴

در این بخش لازم است بخش های خالی کد را طبق فرمول های داده شده پر کنیم:

```
def update(fy: np.array, n:bool) -> np.array:
    p = np.linspace(0,1,t)
    # calculate P(N = n| Y = p) which is a bernouli distribution
    # calculate integral(0 -> 1) fy * pny
    pny = stats.bernoulli.pmf(n, p)
    integral = np.sum(pny * fy / t)
    post = pny * fy / integral
    return post
```

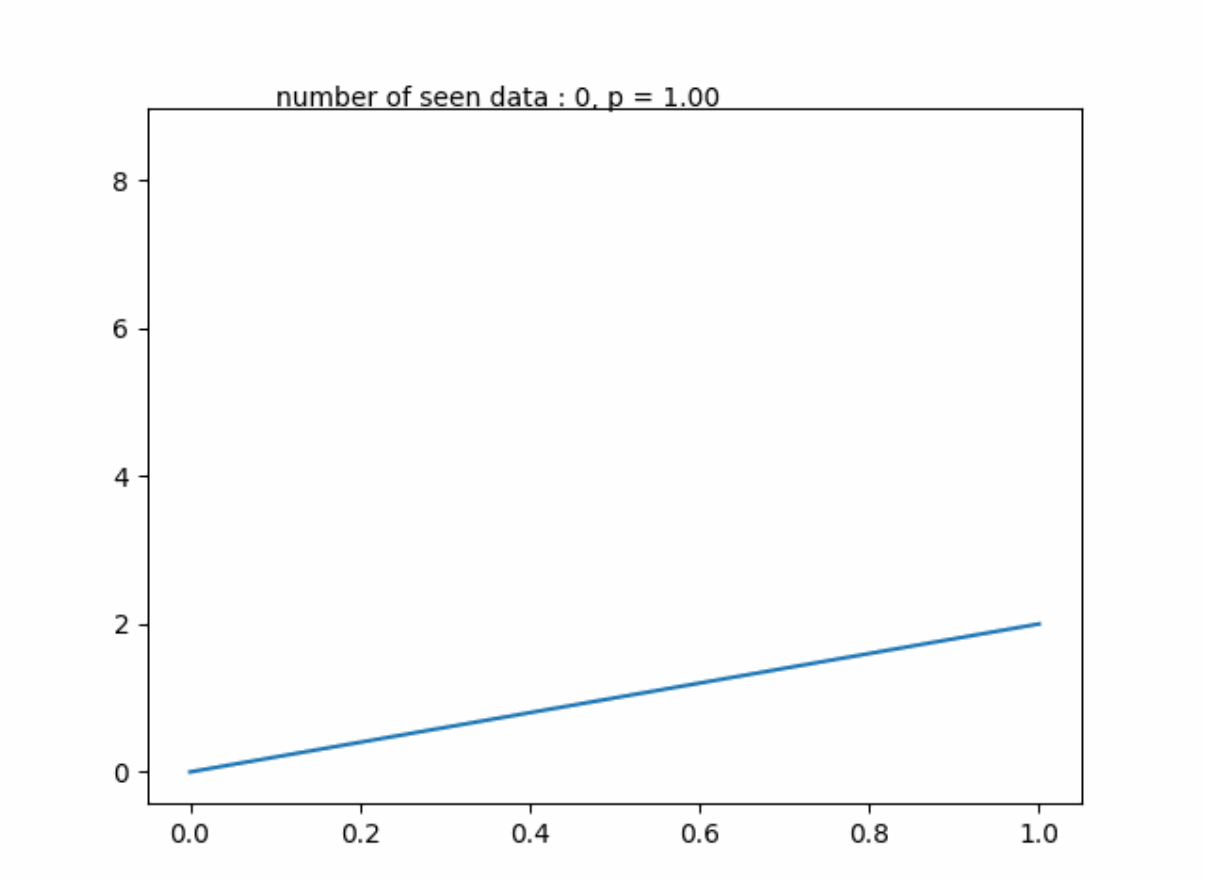
فقط چون از ساختار داده های من با قطعه کد داده شده همخوانی نداشت لازم بود یکسری تغییرات کوچک در آن بدهیم:

```
counter = 0
selected_pixel = 404
for i in bin_data:
    if i[0] != 8:
      continue
    counter += 1
    # replace 'df' with your dataframe's name, this is just a suggestion,
you do not have to code exactly like this
    n = i[1][selected_pixel]
    fy = update(fy, n)
```

می توان از خروجی تصویر متحرک نیز تولید کرد:

```
fig, ax = plt.subplots()
x_values = list(range(0, 1))
y_values = history
line, = ax.plot(history[99][0], history[99][1])
time_text = ax.text(0.1,9, '')


def update_2(frame):
    line.set_data(history[frame][0], history[frame][1])
    time_text.set_text(f'number of seen data : {frame}, p =
{history[frame][1].argmax() / t :.2f}')
    return line,

ani = animation.FuncAnimation(fig, update_2, frames=len(history),
interval=500, blit=True)

ani.save('function_animation.gif', writer='pillow')
plt.show()
```

نتیجه: (در صورتی که تصویر متحرک نیست **اینجا** کلیک کنید.)



بخش ۵ (امتیازی)

فقط لازم است کد قسمت قبل را ابتدا برای تمام پیکسل ها و سپس برای برچسب ها گسترش دهیم.

برای این کار ابتدا کد بخش قبل را تابع می کنیم:

```
def calculate_pixel(label: int, selected_pixel : int):
  t = 1000
  p = np.linspace(0,1,t)
  fy = stats.beta.pdf(p, a=1, b=1)

  counter = 0
  for i in bin_data:
      if i[0] != label:
        continue
      counter += 1
      n = i[1][selected_pixel]
      fy = update(fy, n)

  return fy.argmax() / t
```

سپس یک تابع برای تمام پیکسل ها می نویسیم.

```
def calculate_f_for_all_pixel(label : int):
  out = list()
  for i in range(len(bin_data[0][1])):
    out.append(calculate_pixel(label, i))
  return out
```

در آخر هم برای ۲ برچسب ۸ و ۹ آن را صدا می زنیم:

```
px_label = dict()
for i in {8, 9}:
  px_label[i] = calculate_f_for_all_pixel(i)
```

در آخر جواب محاسبه می شود و طول آن هم ۲ آرایه به اندازه ۷۸۴ است و پیکسل ۴۰۴ برچسب ۸ با بخش های قبل همخوانی دارد.

بخش ۶ (امتیازی)

مطابق دستورالعمل احتمال P(label|X) را برای مقادیر داده شده حساب می کنیم:

```
p_label_X = dict()
for label in {8, 9}:
  p_label_X[label] = dict()
  for X in range(len(two_last_line)):
    p_label_X[label][X] = cal_p_label_X(label, two_last_line[X][1])
```

برای محاسبه مقادیر مربوطه را محاسبه و در فرمول جایگذاری می کنیم:

```
def cal_p_label_X(label : int, img):
  p_X_label = cal_p_X_label(img)
  p_label = {i: 1 / 2 for i in {8,9}}

  ans_sum = 0
  for i in {8, 9}:
    ans_sum += p_X_label[i] + p[i]

  p_label_X = p_X_label[label] * p[label] / ans_sum
  return p_label_X
```

برای محاسبه P(X|label) هم مطابق فرمول عمل می کنیم و در فرمول جایگذاری می کنیم:

```
def cal_p_X_label(img):
  p_X_label = dict()
  for label in {8, 9}:
    p_X_label[label] = 1
    for i in range(len(px_label[label])):
      p_X_label[label] *= (px_label[label][i] ** img[i]) * ((1 -
px_label[label][i]) ** (1 - img[i]))
  return p_X_label
```

در انتها نتیجه به صورت زیر است:

```
{8: {0: 6.382202468734408e-70, 1: 1.6310256117046204e-69},
 9: {0: 1.8839824851423493e-73, 1: 2.8874908532820055e-51}}

for 201: guess: 8, ans: 8, is_True: True
for 202: guess: 9, ans: 9, is_True: True
```