پروژه Bonus درس ارزیابی کارایی سیستم های کامپیوتری

نام استاد: دکتر احمد خونساری
تاریخ: ۲۵ اردیبهشت ۱۴۰۱

نویسندگان:

امیرحسین مهرورز: ۸۱۰۱۰۰۴۸۴
آرمان داوری: ۸۱۰۱۹۹۱۵۳
مجتبی مژگان فر: ۸۱۰۱۰۰۴۷۱
دانیال خلج: ۸۱۰۱۰۰۵۱۵
نیما شکری: ۸۱۰۱۹۹۲۰۰
یگانه کردی: ۸۱۰۱۰۰۵۵۱
محمدجواد دیانت: ۸۱۰۱۰۰۳۴۸
هادی قوامی نژاد: ۸۱۰۱۰۰۴۳۸
مینا فریدی: ۸۱۰۱۰۰۴۳۰
فاروق شایسته رودی: ۸۱۰۱۹۹۱۹۸
محمدمهدی یادگاری فرد: ۸۱۰۹۹۳۰۸

مقدمه:

در این پروژه تمرین های فصل دوم و سوم کتاب Introduction-to-Probability به وسیله ی زبان پایتون حل شده است. با اجرای فایل simulation_project_group_۱.py برنامه ی اصلی شروع به کار کردن میکند و با پارامتر های مناسب میتوان شماره ی فصل و شماره ی هر سوال را وارد کرد تا خروجی جواب آن سوال به نمایش در بیاید.

کتابخانه های استفاده شده:

در طول برنامه نویسی این تمارین، کتابخانه های متعددی مورد استفاده قرار گرفته است که در زیر نام آنهارا بیان میکنیم:

matplotlib
numpy
gmpy2
scipy

همچنین برخی از کتابخانه های خود پایتون نیز مورد استفاده واقع شد:

math
random
time

کد اصلی برنامه:

کد زیر مربوط به قسمت اصلی برنامه میباشد که کار مدیریت کل تمارین را بر عهده دارد. قبل از اجرای کد های حل تمرین ها، این کد اجرا میشود:

```
if __name__ == '__main__':

  while True:
  chapter_number = int(input('Enter Chapter (2 or 3):'))

  if chapter_number not in [2,3]:
    print('Please enter a valid chapter (2 or 3)')
  else:
    if chapter_number == 2:
      while True:
        question_number = int(input('Enter Question (1,5,9,13,17,21,25,41):'))
        if question_number not in [1,5,9,13,17,21,25,41]:
          print('Please enter a valid question')
        else:
          if question_number == 1:
            solving_chapter2_problem_c2_q1()
          elif question_number == 5:
             solving_chapter2_problem_c2_q5()
          elif question_number == 9:
             solving_chapter2_problem_c2_q9()
          elif question_number == 13:
             solving_chapter2_problem_c2_q13()
          elif question_number == 17:
```

```
            solving_chapter2_problem_c2_q17()
        elif question_number == 21:
            solving_chapter2_problem_c2_q21()
        elif question_number == 25:
            solving_chapter2_problem_c2_q25()
        elif question_number == 41:
            solving_chapter2_problem_c2_q41()
        break

  elif chapter_number == 3:
    while True:
      question_number = int(input('Enter Question (1,5,13,21):'))
      if question_number not in [1,5,13,21]:
        print('Please enter a valid question')
      else:
        if question_number == 1:
          solving_chapter2_problem_c3_q1()
        elif question_number == 5:
          solving_chapter2_problem_c3_q5()
        elif question_number == 13:
          solving_chapter2_problem_c3_q13()
        elif question_number == 21:
          solving_chapter2_problem_c3_q21()
        break
  resume = input('Do you want to continue ?
    (1 for yes and anything else for no):')
  if resume == '1':
    print('\n\n')
    continue
  else:
    break
```

کد های تمرین ها:

در این قسمت از گزارش، کد های مربوط به هر تمرین و توضیحات مربوط به آن کد قرار خواهد گرفت.

فصل دوم

جواب سوال اول:

```
def calculate_pmf_of_2_games(nl_first,l_first,nl_second,
  l_second,p_tie,p_win):
  result = [0 for i in range(5)]
  # x -> 0 : losing , losing
  result[0] = l_first * l_second
  # x -> 1 : not losing -> tie , losing + losing + not tising -> tie
  result[1] = (nl_first * p_tie * l_second) + (l_first * nl_second * p_tie)
  # x -> 2 : not losing -> tie , not losing -> tie + losing , not losing ->
    win + not losing -> win , losing
  result[2] = round((nl_first * p_tie * nl_second * p_tie) +
    (nl_first * p_win * l_second) + (l_first * nl_second * p_win),2)
  # x -> 3 : not losing -> win , not losing -> tie + not losing ->
    tie , not losing -> win
  result[3] = round((nl_first * p_win * nl_second * p_tie) * 2,2)
  # x -> 4 : not losing -> win , not losing -> win
  result[4] = round(nl_first * p_win * nl_second * p_win,2)
  return result
```

```
def display_pmf(input_array):
  # displaying the probability of each item in the input
  for i in range(len(input_array)):
    print(f'P(x = {i}) : {input_array[i]}')
  # displaying otherwise probability
  print(f'P(X > {len(input_array)}) : 0')


def solving_chapter2_problem_c2_q1():
  # probabilities:
  nl_first  = 0.4 # not losing in the first game probability
  nl_second = 0.7 # not losing in the second game probability
  l_first   = 0.6 # losing in the first game probability
  l_second  = 0.3 # losing in the second game probability

  p_tie     = 0.5 # probability of tie
  p_win     = 0.5 # probability of win

  # calling the caculte pmf with our probability definition
  result = calculate_pmf_of_2_games(nl_first,
    l_first,nl_second,l_second,p_tie,p_win)
  display_pmf(result)
```

در این سوال ۲ بازی مطرح شده است که احتمال باختن، بردن و مساوی کردن را در هر دو بازی به ما داده است و برای هر پیشامد یک امتیازی در نظر گرفته است. این سوال از ما PMF این ۲ بازی را میخواهد. برای بدست آوردن جواب این سوال میبایست تمامی حالت هارا در نظر بگیریم و تمامی احتمالات ممکن را بر اساس پیشامد ها بدست بیاوریم.

خروجی این کد که در واقع نمایانگر PMF است به صورت زیر میباشد:

```
P(x = 0) : 0.18
P(x = 1) : 0.27
P(x = 2) : 0.34
P(x = 3) : 0.14
P(x = 4) : 0.07
P(X > 5) : 0
```

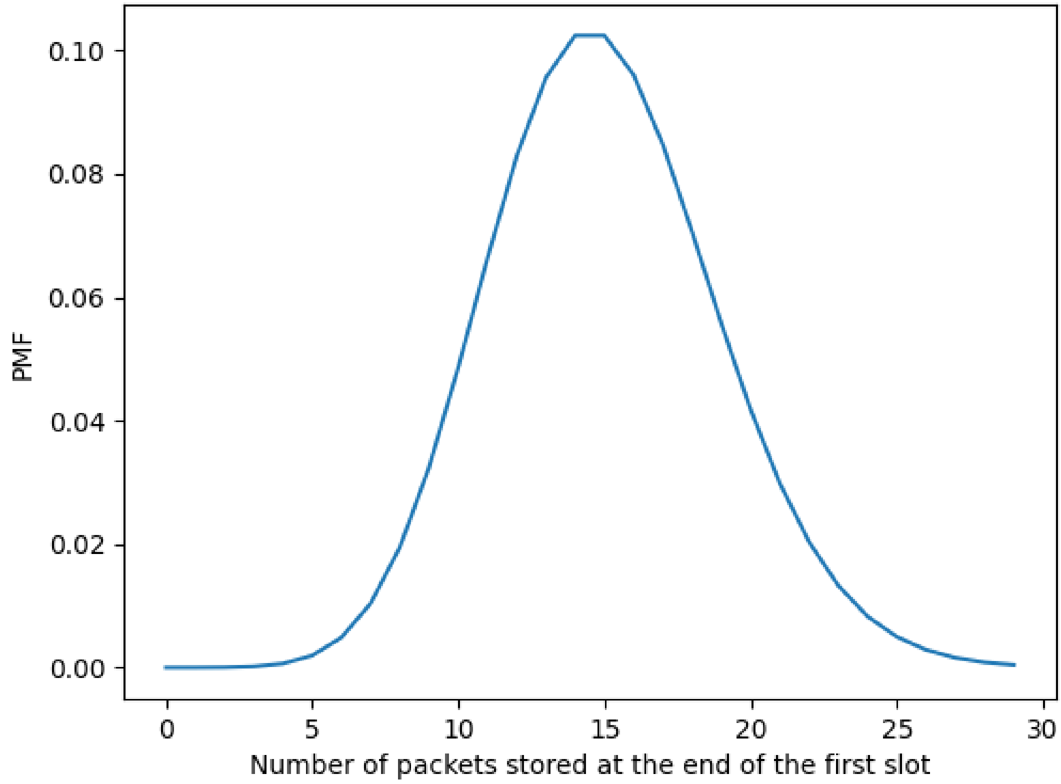
جواب سوال پنجم:

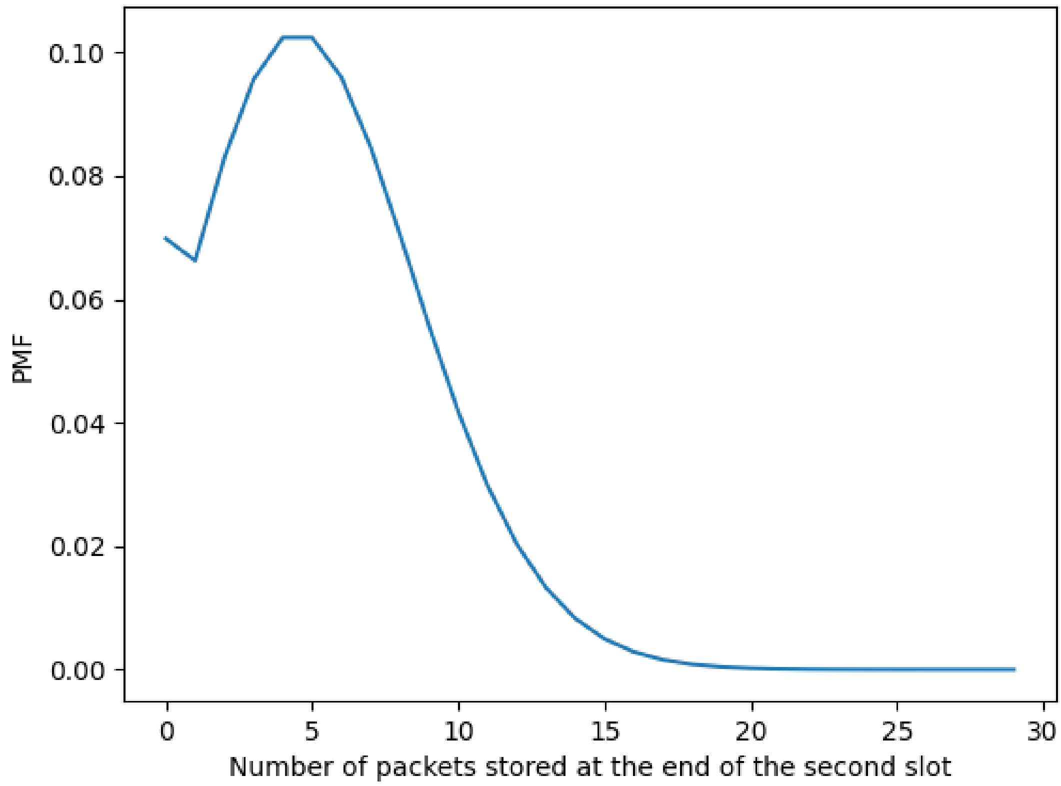
```
b = 30
c = 10
l = 15 # lambda = 15

def fact(n):
  fact = 1
  for i in range(1,n+1):
    fact = fact * i
  return fact

def poisson(k):
  return exp(-1 * l) * ((l**k)/fact(k))

def x_pmf(k):
  return poisson(k)
```

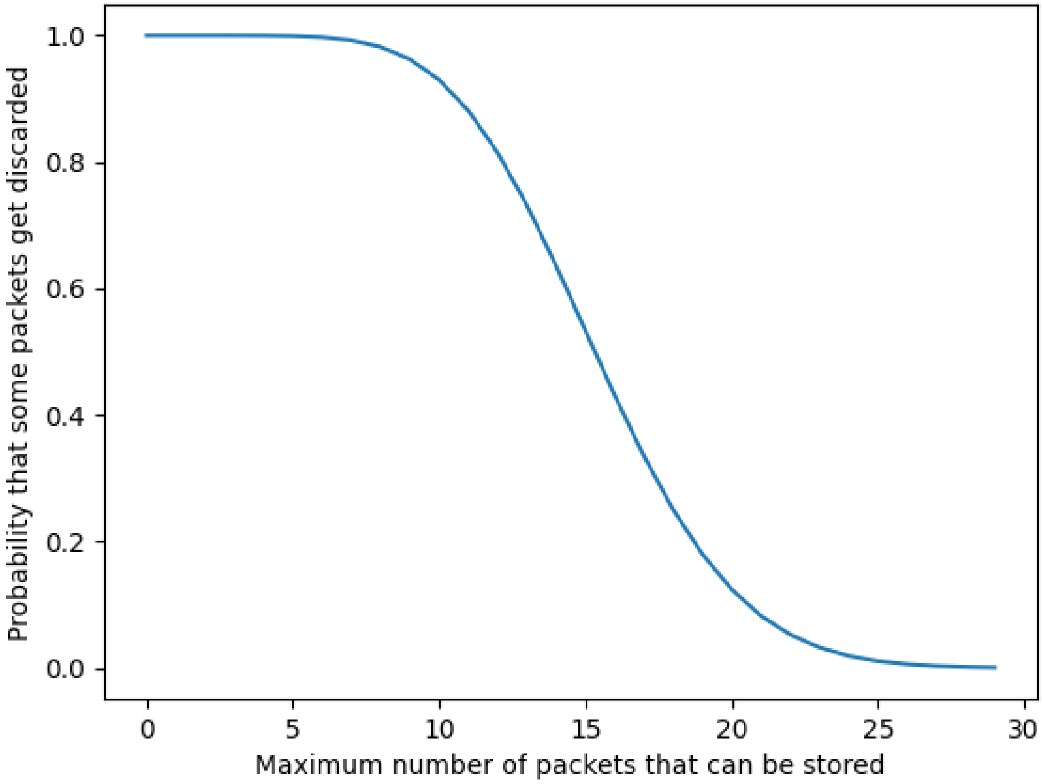
```
def y_pmf(k):
  if k == 0:
    prob = []
    for i in range (0,c):
    prob.append(poisson(i))
    return sum(prob)

  if k == (b-c):
    return x_pmf(b)

  return poisson(k+c)

def discard(max_number):
  prob = []
  for i in range (0,max_number):
    prob.append(poisson(i))
  return 1 - sum(prob)

def solving_chapter2_problem_c2_q5():
  x = []
  for i in range(0,30):
    x.append(x_pmf(i))

  y = []
  for i in range(0,30):
    y.append(y_pmf(i))

  d = []
  for i in range(0,30):
    d.append(discard(i))

  plt.plot(range(0,30),x)
  plt.xlabel("Number of packets stored at the end of the first slot")
  plt.ylabel("PMF")
  plt.show()

  plt.plot(range(0,30),y)
  plt.xlabel("Number of packets stored at the end of the second slot")
  plt.ylabel("PMF")
  plt.show()

  plt.plot(range(0,30),d)
  plt.xlabel("Maximum number of packets that can be stored")
  plt.ylabel("Probability that some packets get discarded")
  plt.show()
```

براي محاسبه تعداد پکت هايي که در دو اسلات مي آيند سوال پارامتري است. چيزي که مي توانيم نشان دهيم شکل خروجي با استفاده از پارامترهايي است که بدست مي آيد. لذا نمودار تعداد پکت هاي اسلات اول و تعداد پکت هاي اسلات دوم رسم شده است. نمودارها بيانگر تاثير تغيير پارامترها مي باشد.

خروجی این سوال به صورت زیر میباشد:


0.10
0.08
0.06
0.04
0.02
0.00
PMF
0
5
10
15
20
25
30
Number of packets stored at the end of the first slot


0.10
0.08
0.06
0.04
0.02
0.00
PMF
0
5
10
15
20
25
30
Number of packets stored at the end of the second slot



جواب سوال نهم:

```
n = 99
p = 0.4
k = np.linspace(10,60,num=50,endpoint=False)
ans = []


def facto (n):
  fact = 1
  for i in range(1,n+1):
    fact = fact * i

  return fact

def combo(n, k):
  temp1 = facto(n)
  temp2 = facto(k)
  temp3 = facto(n-k)
  cmb = temp1/(temp2*temp3)
  return cmb

def q9(n,k,p):
  temp1 = combo(n,k)*pow(p,k)*pow(1-p,n-k)
  temp2 = combo(n,k-1)*pow(p,k-1)*pow(1-p,n-k+1)
  return temp1/temp2

def solving_chapter2_problem_c2_q9():
```

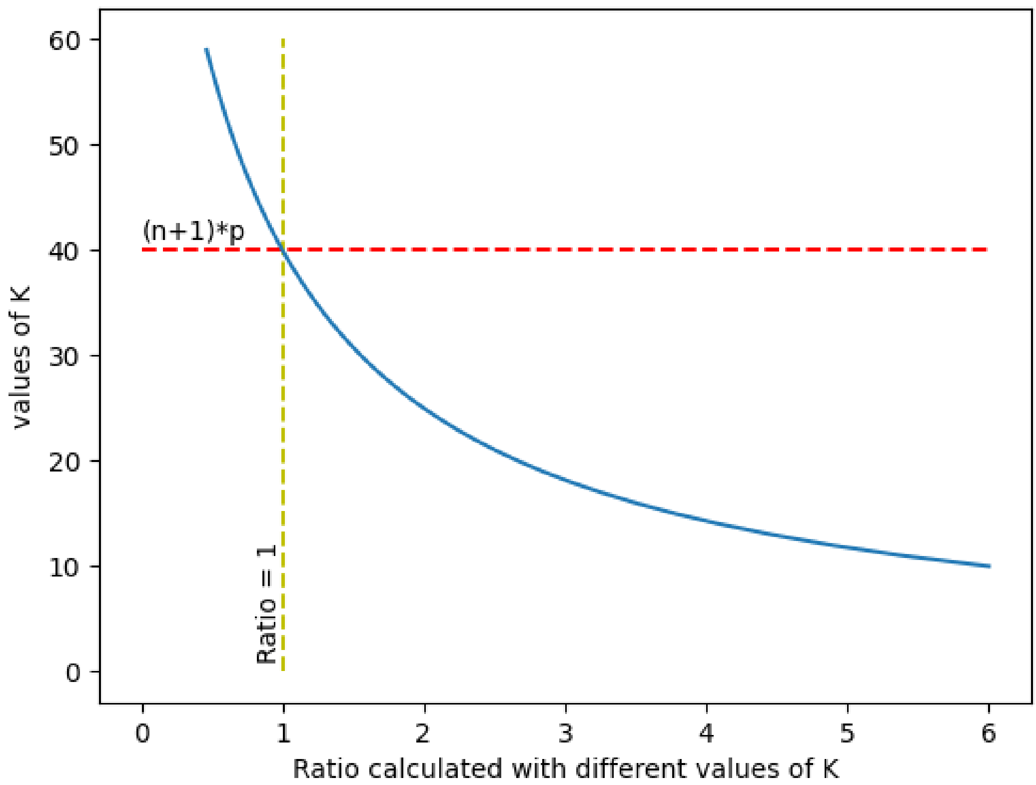
```
print(k)

for ks in k:
  ans.append(q9(n,int(ks),p))

plt.plot(ans,k)
plt.hlines(y = (n+1)*p, xmin = 0, xmax = 6, color ='r', linestyle='--')
plt.vlines(x = 1, ymin=0, ymax = 60, color = 'y', linestyle = '--')
plt.text(0, ((n+1)*p)+0.5, '(n+1)*p', ha ='left', va ='bottom')
plt.text(1, 0.8, 'Ratio = 1',rotation ='vertical', ha ='right', va ='bottom')
plt.xlabel("Ratio calculated with different values of K")
plt.ylabel("values of K")

plt.show()
```

در ابتدا به ازای مقادیر صفر تا n نسبتی که از طریق تقسیم $p_x(k)$ به $p_x(k-1)$ بدست می آید را محاسبه می کنیم. مشاهداتی که به شکل گرافیکی در نمودار آورده شده نشان می دهد به ازای k کوچکتر مساوی k^* مقدار $k-kp$ کوچکتر مساوی $(n+1)p-kp$ می شود.
در نتیجه همانطور که در نمودار ترسیم شده مشهود است نسبت محاسبه شده فوق بزرگتر و یا مساوی ۱ می شود که نشان می دهد احتمال اکیدا غیرنزولی است و در صورتی که k بزرگتر از k^* باشد، این نسبت محاسبه شده با توجه به نمودار مقداری کمتر از ۱ خواهد داشت که تابع احتمال نیز اکیدا نزولی خواهد بود.

نتیجه ی این کد به صورت زیر میباشد:

```
[10. 11. 12. 13. 14. 15. 16. 17. 18. 19.
 20. 21. 22. 23. 24. 25. 26. 27. 28. 29.
 30. 31. 32. 33. 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.]
```

همچنین نمودار حاصل از اجرای این کد به صورت زیر رسم میشود:



جواب سوال سیزدهم:

```
def pmf_girls (g, naturals, adoptedgirls):
  all = naturals + adoptedgirls
  if (adoptedgirls>g or all <g):
    return (0)
  else:
    return math.comb(naturals,g-2) * math.pow(0.5,naturals)

def solving_chapter2_problem_c2_q13():
  # print("insert number of the girls to find PMF")
  # g = int(input()) #pmf variable (num of girls)
  g = 3 #for example

  naturals = 5  # number of natural children
  adoptedgirls = 2  # number of adopted girls

  print(pmf_girls (g, naturals, adoptedgirls))
```

در این سوال pmf تعداد دخترهای خانواده خواسته شده است. این خانواده ۵ فرزند طبیعی دارد که جنسیت شان را نمی دانیم همچنین ۲ دختر دارد که به سرپرستی گرفته شده اند.

به عنوان مثال میخواهیم بدانیم احتمال اینکه تعداد کل دخترها ۳ باشد چقدر است. در تابع تعریف شده ابتدا g مورد تقاضا را بررسی می کنیم که در بازه مورد نظر باشد. g بایستی حداقل ۲ باشد و حداکثر نیز بایستی از تعداد کل بچه ها بیشتر باشد در غیراینصورت مقدارش ۰ می شود. انتخاب $g-2$ از فرزندان طبیعی را محاسبه می کنیم و $return$ می کنیم.

خروجی این کد به صورت زیر است:

0.15625

جواب سوال هفدهم:

```
def cel_to_fahr(cel_mu: float, cel_sigma: float) -> float:
    """
    since we know transform is linear, we can use the inverse transform
    fahr  = (cel * 9 / 5) + 32

    and new mean and sigma are:

    E[fahr] = (E[cel] * 9 / 5) + 32;

    var(fahr) = (9/5)^2 * var(cel)
    => sigma(fahr) = (9/5) * sigma(cel)

    """
    fahr_mu    = cel_mu * 9 / 5 + 32
    fahr_sigma = cel_sigma * 9 / 5

    return fahr_mu, fahr_sigma

def normal_pdf(x, mu:float, sigma:float) -> float:
    """
    normal distribution pdf function
    """
    return (1.0 / np.sqrt(2 * np.pi * sigma ** 2)) *
    np.exp(-0.5 * (x - mu) ** 2 / sigma ** 2)

def make_plot(axs, title: str, mu: float, sigma: float) -> None:
    """
    axs: matplotlib axes
    title: string
    mu: float
    sigma: float
    ----------------------------
    We want to plot the distribution of the temperature
    and since we know it is normal, we can use the normal distribution function
    from scipy.stats.norm to plot it.
    for range of x, we use mu-3*sigma to mu+3*sigma
    to make sure we get a full range of the distribution.

    The normal range includes the Mu +- Sigma range, so we separate this area.
    """
    x     = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
    y     = normal_pdf(x, mu, sigma)
    left  = mu - sigma
    right = mu + sigma

    axs.plot(x, y, "k")
    axs.set_title(title, fontsize=14, loc="left")
    axs.vlines(left, 0,
    normal_pdf(left, mu, sigma),
    colors="r", linestyles="dashed")
    axs.text(
    left + (0.5 if left == 0 else 2),
```

```
    normal_pdf(left, mu, sigma) / 2,
    int(left),
    fontsize=10,
    )
    axs.vlines(right, 0,
    normal_pdf(right, mu, sigma),
    colors="r", linestyles="dashed")
    axs.text(
    right - (5 if right % 10 == 0 else 9),
    normal_pdf(right, mu, sigma) / 2,
    int(right),
    fontsize=10,
    )
    axs.spines["top"  ].set_visible(False)
    axs.spines["right"].set_visible(False)
    axs.spines["left" ].set_visible(False)
    axs.tick_params(left=False, labelleft=False)


def solving_chapter2_problem_c2_q17():
    fig, axs = plt.subplots(1, 2, figsize=(10, 6))

    celsius_mu    = 10
    celsius_sigma = 10

    fahrenheit_mu, fahrenheit_sigma = cel_to_fahr(celsius_mu, celsius_sigma)

    make_plot(
    axs[0], r"$\mathbf{Temperature \; in}$" +"\nCelsius",
    celsius_mu,
    celsius_sigma
    )
    make_plot(
    axs[1],
    r"$\mathbf{Temperature \; in}$" + "\nFahrenheit",
    fahrenheit_mu,
    fahrenheit_sigma,
    )
    plt.show()
```

این کد شامل سه فانکشن عملیاتی به صورت زیر است:

- Cel_to fahr

  این فانکشن با توجه به فرمول تبدیل سلسیوس به فارنهایت μ و σ مربوط به تابع توزیع فارنهایت را محاسبه می کند.

- Normal_pdf

  محاسبه تابع توزیع نرمال با استفاده از ورودی های لازم

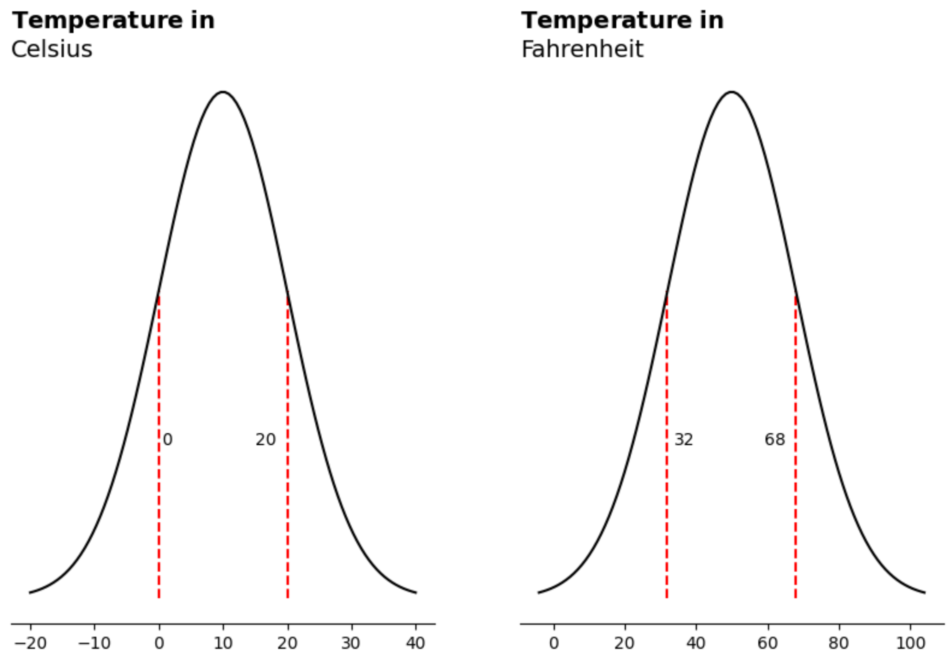
  X,μ,σ

- Make_plot

  برای رسم توزیع دما استفاده می شود

از آنجایی که می دانیم تابع توزیع نرمال است، می توانیم از تابع Normal_pdf برای رسم آن استفاده کنیم. برای محدوده x، ما از $\mu - 3 * \sigma$ تا $\mu + *\sigma$ استفاده می کنیم تا مطمئن شویم

که طیف کاملی از توزیع را دریافت می کنیم.
محدوده نرمال شامل محدوده $\mu \mp \sigma$ می شود، بنابراین این ناحیه را نیز به صورت مشخص جدا می کنیم.
نتیجه ی خروجی این کد به صورت زیر است:



جواب سوال بیست و یکم

```
def tossCoin():
  resualt=None
  if random.randint(0,1)==0:
    resualt='head'
  else:
    resualt='tail'
  return resualt


# Press the green button in the gutter to run the script.

def solving_chapter2_problem_c2_q21():
  deposit = 0
  while True:
    numberOfTosses=0
    result=tossCoin()
    while result=='head':
      numberOfTosses+=1
      result=tossCoin()
      if numberOfTosses>0:
        deposit += pow(2, numberOfTosses)
        print('Congrats!! You have earned ' + str(pow(2, numberOfTosses)) +
          '$ In this round!')
      else:
```

```
        print('Sorry :( You did not win in this round')
    print('Total Deposit: '+str(deposit)+'$ ')
    print()
    print('————————————————————————————————')
    print()
    time.sleep(1.5)
```

در این سوال سکه ای را پرتاب می کنیم اگر شیر بیاید می شماریم. مادامی که شیر بیاید می شماریم و هنگامی که خط آمد، ۲ به توان آن مقدار پول می دهد.
قطعه کد ما شبیه سازی پرتاب سکه است. مکرراً سکه را می اندازد و وابسته به اینکه چه تعدادی شیر آمده اعلام می کند در آن راند چقدر پیروز شده ایم.
نشان داده می شود که به سمت بی نهایت رشد پیدا می کند.

با توجه به شانسی بودن احتمال شیر یا خط، نتیجه ی خروجی متفاوت است. در اینجا چند بار آزمایش را نمایش میدهیم:

```
Congrats!! You have earned 2$ In this round!
Total Deposit: 2$

————————————————————————————————

Congrats!! You have earned 2$ In this round!
Total Deposit: 4$

————————————————————————————————

Congrats!! You have earned 2$ In this round!
Total Deposit: 6$

————————————————————————————————

Congrats!! You have earned 2$ In this round!
Total Deposit: 8$

————————————————————————————————

Sorry :( You did not win in this round
Total Deposit: 8$

————————————————————————————————

Sorry :( You did not win in this round
Total Deposit: 8$

————————————————————————————————
```

```
Sorry :( You did not win in this round
Total Deposit: 8$
```

```
Sorry :( You did not win in this round
Total Deposit: 8$
```

```
Sorry :( You did not win in this round
Total Deposit: 8$
```

```
Congrats!! You have earned 2$ In this round!
Total Deposit: 10$
```

```
Congrats!! You have earned 4$ In this round!
Total Deposit: 6$
```

```
Sorry :( You did not win in this roundTotal Deposit: 6$
```

جواب سوال بیست و پنجم

```
def M_i_randGenerator (m,M,n):
  for i in range(0,n):
    M[i] = rn.randint(0,m)

#M = [0,1,2]

def calculate_L(m,M,n,L):
  for x in range(0,n):
    for y in range(0,m):
      if y+1 <= M[x]:
        L[y] = L[y] + 1

def sigma_Mk(M,n):
  sum = 0
  for k in range (0 , n):
    sum = sum + M[k]
  return sum

def calculate_Joint_PMF (n,M,m,Prob_IJ,sum):
  for i in range(0,n):
```

```
    for j in range(0,m):
      if j+1 <= M[i] :
        Prob_IJ[i][j] = 1 / sum
      else:
        Prob_IJ[i][j] = 0.0
      print("Joint PMF of (", i+1 , ",", j+1 , ") is: " , Prob_IJ[i][j])


def calculate_Marginal_PMF_I (n , m,Prob_I,M,sum):
  for i in range(0,n):
    Prob_I[i] = M[i] / sum
    print( "Marginal PMF of I (", i+1 , ") is: ", Prob_I[i] )

def calculate_Marginal_PMF_J (n , m,L,Prob_J,sum):
  for j in range(0,m):
    Prob_J[j] = L[j] / sum
    print( "Marginal PMF of J (", j+1 , ") is: ", Prob_J[j] )

def probabilty_ij(n,m,p_ij):
  for i in range(0,n):
    for j in range(0,m):
      p_ij[i][j] = rn.random()
      print("probabilty of (",i+1, ",",j+1,")" , p_ij[i][j])

def expected_value(n,M,p_ij,a,b):
  sigma = 0
  for i in range(0,n):
    for j in range(0,M[i]):
      sigma = sigma + ( p_ij[i][j] * a ) + ( (1 - p_ij[i][j]) * b )
  print("Exptected Value is: " , sigma)

def solving_chapter2_problem_c2_q25():

  n = int( input("enter integer value for n (4 in the question): " ) )
  m = int( input("enter integer value for m (3 in the question): ") )
  a = int( input("enter integer value for a (1 in the question): " ) )
  b = int( input("enter integer value for b (-1 in the question): " ) )
  print("-" * 60)

  """
  n=3
  m=4
  a = 1
  b = -1
  """

  M = [0] * n
  L = [0] * m
  sum = 0

  Prob_IJ = [[0]*m]*n
  Prob_I = [0] * n
  Prob_J = [0] * m
  p_ij = [[0]*m]*n

  M_i_randGenerator (m,M,n)
  print("M array is:" , M)
  print("-" * 60)
```

```
calculate_L(m,M,n,L)
print("L array is:" , L)
print("-" * 60)

sum = sigma_Mk(M,n)
print("sum is:" , sum)
print("-" * 60)

calculate_Joint_PMF (n,M,m,Prob_IJ,sum)
print("-" * 60)

count = 0
for i in range(0,n):
  for j in range(0,m):
    count = count + Prob_IJ[i][j]
print("count is: ", count)
print("-" * 60)

calculate_Marginal_PMF_I (n , m,Prob_I,M,sum)
print("-" * 60)

calculate_Marginal_PMF_J (n , m,L,Prob_J,sum)
print("-" * 60)

probabilty_ij(n,m,p_ij)
print("-" * 60)

expected_value(n,M,p_ij,a,b)
print("-" * 60)
```

در این سوال n دانش آموز دریم و m سوال. دانش آموز i ام m_i تا سوال اول را جواب می دهد و بقیه سوالات را جواب نداده است. پس به تعداد m_i ها جواب داریم. joint pmf آن ۱ تقسیم بر سیگمای مخرج می باشد اگر j ها کمتر از m_i ها باشد.
تابعی داریم که به صورت رندوم m_i ها را مشخص می‌کنیم. تابعی برای محاسبه سیگمای مخرج داریم. تابعی برای محاسبه اینکه چند نفر به سوال j ام جواب داده اند را محاسبه می کند و در نهایت تابعی داریم که برای i و j تابع pmf حاشیه ای را محاسبه می‌کند.
در قسمت ب، فرض می‌کنیم جواب تحویل داده شده با احتمال p_{ij} درست است و با احتمال $1 - P_{ij}$ نادرست است. اگر درست باشد a پوینت و اگر غلط باشد b پوینت می گیرد در نهایت مقدار Expected Value در تابع آن محاسبه می‌گردد.

خروجی این کد به ازای n=۴ و m=۳ و a=۱ و b=-۱ به صورت زیر میباشد:

```
------------------------------------------------------------
M array is: [0, 1, 2, 0]
------------------------------------------------------------
L array is: [2, 1, 0]
------------------------------------------------------------
sum is: 3
------------------------------------------------------------
```

```
Joint PMF of ( 1 , 1 ) is: 0.0
Joint PMF of ( 1 , 2 ) is: 0.0
Joint PMF of ( 1 , 3 ) is: 0.0
Joint PMF of ( 2 , 1 ) is: 0.3333333333333333
Joint PMF of ( 2 , 2 ) is: 0.0
Joint PMF of ( 2 , 3 ) is: 0.0
Joint PMF of ( 3 , 1 ) is: 0.3333333333333333
Joint PMF of ( 3 , 2 ) is: 0.3333333333333333
Joint PMF of ( 3 , 3 ) is: 0.0
Joint PMF of ( 4 , 1 ) is: 0.0
Joint PMF of ( 4 , 2 ) is: 0.0
Joint PMF of ( 4 , 3 ) is: 0.0
```

```
count is: 0.0
```

```
Marginal PMF of I ( 1 ) is: 0.0
Marginal PMF of I ( 2 ) is: 0.3333333333333333
Marginal PMF of I ( 3 ) is: 0.6666666666666666
Marginal PMF of I ( 4 ) is: 0.0
```

```
Marginal PMF of J ( 1 ) is: 0.6666666666666666
Marginal PMF of J ( 2 ) is: 0.3333333333333333
Marginal PMF of J ( 3 ) is: 0.0
```

```
probabilty of ( 1 , 1 ) 0.6086080476233853
probabilty of ( 1 , 2 ) 0.4557177120051581
probabilty of ( 1 , 3 ) 0.2898182856630127
probabilty of ( 2 , 1 ) 0.2515811634212529
probabilty of ( 2 , 2 ) 0.747979635611269
probabilty of ( 2 , 3 ) 0.37566283360759256
probabilty of ( 3 , 1 ) 0.009347532561182 37
probabilty of ( 3 , 2 ) 0.8965086235710871
probabilty of ( 3 , 3 ) 0.26113698759359616
probabilty of ( 4 , 1 ) 0.7549944764423108
probabilty of ( 4 , 2 ) 0.06808941759715448
probabilty of ( 4 , 3 ) 0.767159373012873
```

```
Exptected Value is: 0.15615674096355203
```

جواب سوال چهل و یکم:

```
def solving_chapter2_problem_c2_q41():
  w = 50 # week
  d = 5 #day
  p = 0.02
  n = d * w
```

```
k = 5
lambdaa = n * p # for part b

pay_t = [10,20,50] # Ticket prices for part c
p_pay_t = [0.5,0.3,0.2] # respective probabilities for part c

pk= comb(n,k) * p**k * (1-p)**(n-k)
print("P(X = k) = " , pk)

# defining list of r values
r_values = list(range(16))
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# plotting the graph
plt.bar(r_values, dist)
plt.show()

approximation = e**(-lambdaa) * (lambdaa ** k / fac(k))
print("Poisson  approximation of part a : " , approximation)


tp_pay_t = dot(p, p_pay_t)
p_mean = sum(dot(pay_t,tp_pay_t))
mean = n * p_mean
p_var = sum(dot(power(pay_t,2),tp_pay_t)) - p_mean**2
var = n * p_var

print("mean: " , mean )
print("Variance: " , var )
```

در پارت نخست آزمایش برنولی می باشد که از قطعه کد $pk = comb(n,k) * p * *k * (1 - p) * *(n - k)$ استفاده می نماییم. به جهت افزایش قدرت تغییر و انعطاف در برنامه، Expected Value را محاسبه نکردیم بلکه می توان به ازای k هر مقداری قرار داد و احتمال آن را محاسبه نمود. در صورتی که Expected Value را بخواهیم منطبق بر صورت سوال به k مقدار ۵ می دهیم و از لاندا که در پارت دوم استفاده شده است، می توانیم استفاده کنیم. قسمتی اضافه تر از خواسته صورت سوال نیز انجام شده است که آن محاسبه PMF است.

برای پارت دوم از Poisson Approximation منطبق در ذیل استفاده می کنیم.

Poisson approximations

The Bin(n, p) can be thought of as the distribution of a sum of independent indicator random variables $X_1 + \ldots + X_n$, with $\{X_i = 1\}$ denoting a head on the ith toss of a coin. The normal approximation to the Binomial works best when the variance $np(1-p)$ is large, for then each of the standardized summands $(X_i - p)/\sqrt{np(1-p)}$ makes a relatively small contribution to the standardized sum. When n is large but p is small, in such a way that np is not large, a different type of approximation to the Binomial is better.

<8.1> on **Definition.** *A random variable Y is said to have a* Poisson distribution *with parameter λ, abbreviated to* Poisson(λ), *if it can take values* $0, 1, 2, \ldots$ *with probabilities*

$$\mathbb{P}\{Y = k\} = \frac{e^{-\lambda}\lambda^k}{k!} \qquad \text{for } k = 0, 1, 2, \ldots$$

The parameter λ must be positive. □

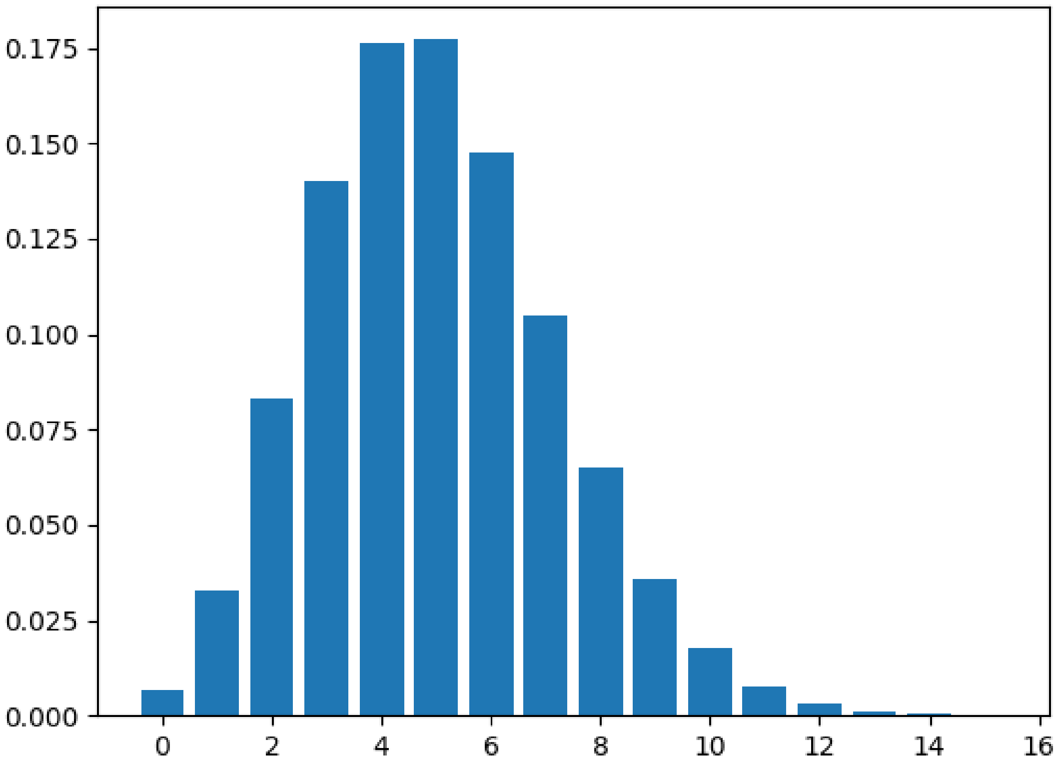
The Poisson(λ) appears as an approximation to the Bin(n, p) when n is large, p is small, and $\lambda = np$:

$$\begin{aligned}\binom{n}{k} p^k (1-p)^{n-k} &= \frac{n(n-1)\ldots(n-k+1)}{k!}\left(\frac{\lambda}{n}\right)^k \left(1 - \frac{\lambda}{n}\right)^{n-k} \\ &\approx \frac{\lambda^k}{k!}\left(1 - \frac{\lambda}{n}\right)^n \qquad \text{if } k \text{ small relative to } n \\ &\approx \frac{\lambda^k}{k!} e^{-\lambda} \qquad \text{if } n \text{ is large}\end{aligned}$$

در پارت سوم قیمت برای بلیط ها تغیین شده است و درصد داده شده است. قیمت بلیط ها در pay_t و احتمال متناسب در p_pay_t ریخته شده است. این احتمال خرید هر بلیط را بایستی از درصد کل استفاده کرد. $E[Y_i]$ همان p_mean می باشد. سپس $mean$ محاسبه می گردد. p_var همان واریانس Y_i است و واریانس کلی محاسبه شده و نتایج $print$ شده اند.

خروجی این کد به صورت زیر است:

```
P(X = k) =  0.17724760428872932
Poisson approximation of part a :  0.17546736976785077
mean:  105.00000000000001
Variance:  3305.9
```



فصل سوم

جواب سوال اول:

```
def calculate_expected_value(p1,p2,x1,x2):
  return (p1 * x1) + (p2 * x2)

def solving_chapter2_problem_c3_q1():
  py_1 = 1/3
  py_2 = 1 - py_1

  gx_1 = 1
  gx_2 = 2

  print('Showing PMF:')
  print(f'--py(1) = P(X <= 1/3) = {py_1}')
  print(f'--py(2) = P(X <= 1/3) = {py_2}')

  print('-' * 60)

  expected_value = calculate_expected_value(py_1,py_2,gx_1,gx_2)
  print('Showing expected value')
  print(f'--E[Y] = (1/3 * 1) + (2/3 * 2) = {expected_value}')
```

احتمال مقدار gx در سوال داده شده است. همچنین مقدار آنها ۱ و ۲ میباشند. در ابتدا PMF این مقادیر را رسم میکنیم و بعد از آن مقدار expected value یا مقدار مورد انتظار را بدست می آوریم. برای

بدست آوردن expected value این مقادیر، از تابع calculate_expected_value استفاده میکنیم و سپس آن را نمایش میدهیم.

خروجی این کد به صورت زیر است:

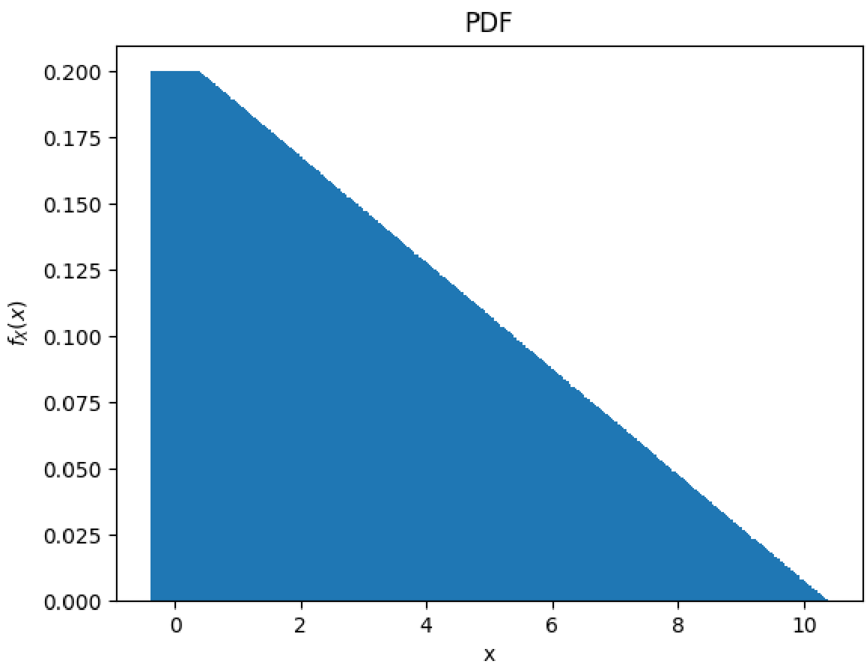
```
Showing PMF:
—py(1) = P(X <= 1/3) = 0.3333333333333333
—py(2) = P(X <= 1/3) = 0.6666666666666667
————————————————————————————————
Showing expected value
—E[Y] = (1/3 * 1) + (2/3 * 2) = 1.6666666666666667
```

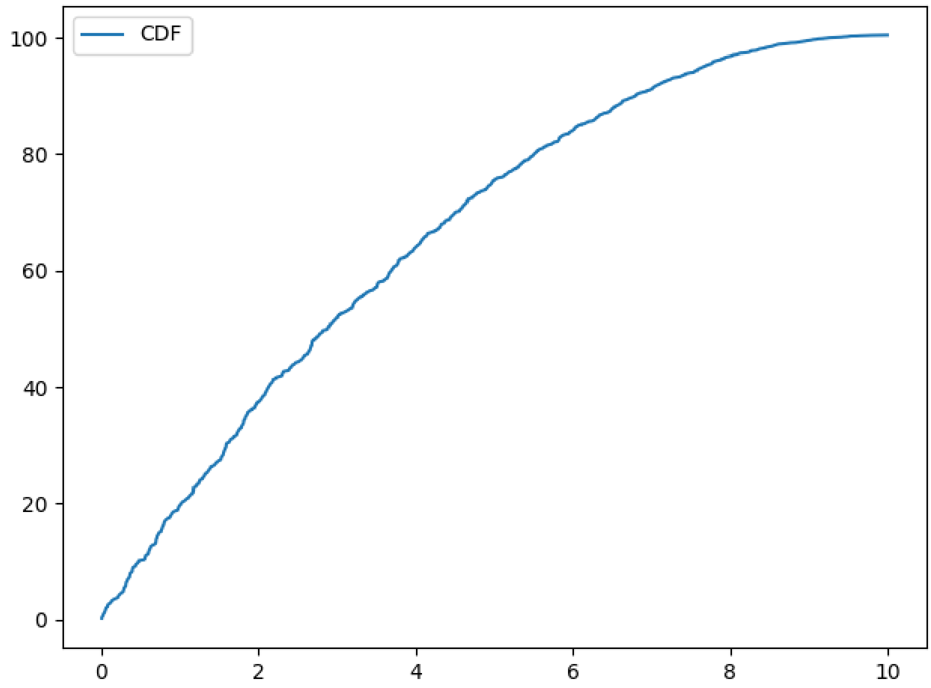
جواب سوال پنجم

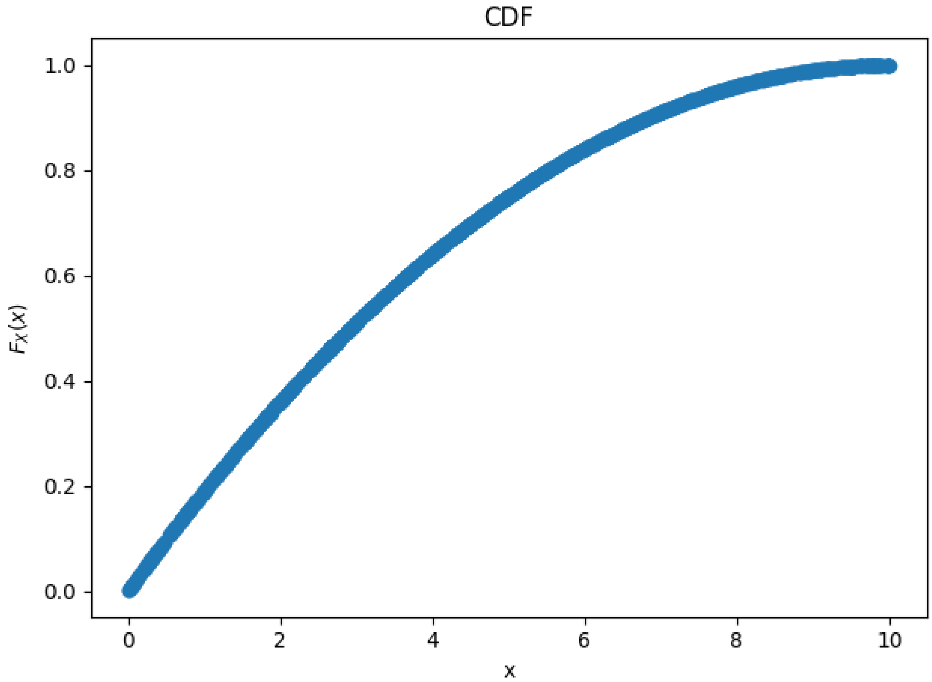
```
def solving_chapter2_problem_c3_q5():
  h = 10 #
  x = np.random.uniform(0,h, 1000)
  x_s = sort(x)

  pdf =[ 2*(h - t) / h**2 for t in x_s]

  plt.xlabel('x')
  plt.ylabel('$f_{X}(x)$')
  plt.title('PDF')
  plt.bar(x_s, pdf)
  plt.show()

  plt.rcParams["figure.autolayout"] = True
  cdf = cumsum(pdf)
  plt.plot(x_s, cdf, label="CDF")
  plt.legend()
  plt.show()

  cdf =[1-((h-t)/h)**2 for t in x_s]
  plt.xlabel('x')
  plt.ylabel('$F_{X}(x)$')
  plt.title('CDF')
  plt.plot(x_s, cdf, marker='o')
  plt.show()
```

h ارتفاع و x متغیر تصادفی است. نخست pdf را محاسبه کرده ایم و سپس cdf را به دو روش محاسبه نموده ایم. ابتدا بوسیله pdf و دوم به وسیله فرمول cdf.

خروجی کار به صورت زیر است:


PDF
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
$f_X(x)$
0
2
4
6
8
10
x


CDF
100
80
60
40
20
0
0
2
4
6
8
10



جواب سوال سیزدهم:

```
def normalProbabilityDensity(x):
  constant = 1.0 / np.sqrt(2*np.pi)
  return(constant * np.exp((-x**2) / 2.0))

def solving_chapter2_problem_c3_q13():
  Fahrenheit = 59
  mean = 10
  e = 10
  Celsius = (Fahrenheit - 32) * 5.0/9.0
  z_upper = (Celsius - e)/mean
  temperature, _ = quad(normalProbabilityDensity, np.NINF, z_upper)
  print('Probability: ', temperature)
```

نخست در کد فارنهایت رو به سلسیوس تبدیل کرده ایم که می شود ۱۵ درجه، بعد مقدار z رو محاسبه نموده ایم که از رابطه $0.5 = 10/(10 - 15)$ بدست می آید. سپس بایستی این مقدار را از جدول بررسی کنیم که در پایتون با دستور زیر محاسبه می گردد.

$$quad(normalProbabilityDensity, np.NINF, z_upper)$$

خروجی این کد به صورت زیر است:

```
Probability: 0.6914624612740132
```

جواب سوال بیست و یکم:

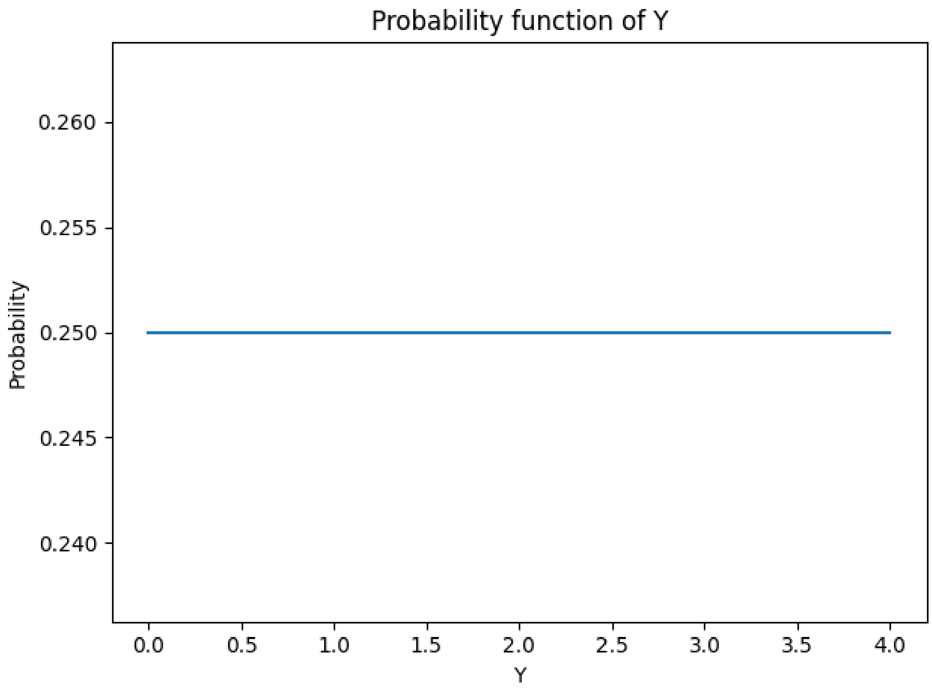
```
def solving_chapter2_problem_c3_q21():
  l = input("Please enter length l: ")
  l = int(l)
```

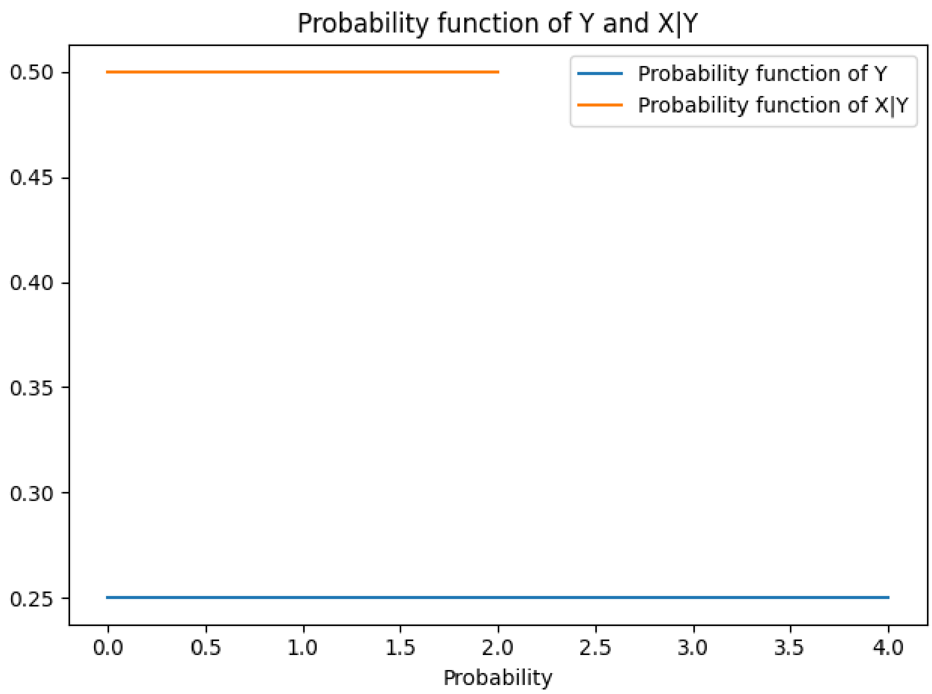
```
x = [0 , l]
y = [(1/l) , (1/l)]
plt.plot(x, y)
plt.xlabel('Y')
plt.ylabel('Probability')
plt.title('Probability function of Y')
plt.show()
y_in = input("Please enter length of first cut: ")
y_in = int(y_in)


x1 = [0, l]
y1 = [(1 / l), (1 / l)]
# plotting the line 1 points
plt.plot(x1, y1, label="Probability function of Y")

# line 2 points
x2 = [0, y_in]
y2 = [(1 / y_in), (1 / y_in)]
# plotting the line 2 points
plt.plot(x2, y2, label="Probability function of X|Y")

# naming the x axis
plt.xlabel('Probability')
# naming the y axis
plt.ylabel('')
# giving a title to my graph
plt.title('Probability function of Y and X|Y')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()

input("Please enter any key to calculate joint PDF of Y and X ")
x = [0, y_in]
y = [(1 / l)*(1 / y_in) , (1 / l)*(1 / y_in)]
plt.plot(x, y)
plt.xlabel('X,Y')
plt.ylabel('Probability')
plt.title('Joint probability function of X,Y')
plt.show()
print('————————————————————————————————————————')
print()
print('The marginal PDF of X is: 1/l ln(l/x)')
print('E[X] = l/4')
```

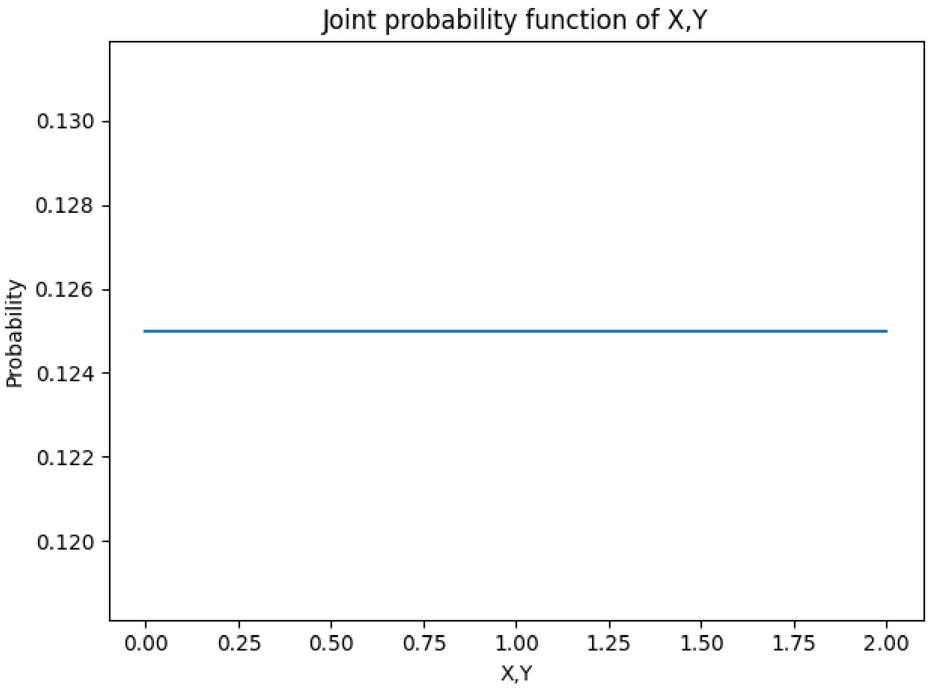
خروجی این کد بستگی به مقدار پارامتر ورودی دارد. در زیر بر اساس مقدار l برابر ۴ و اولین برش برابر ۲ نتایج آورده شده است:

```
Please enter length l: 4
Please enter length of first cut: 2
Please enter any key to calculate joint PDF of Y and X
```

The marginal PDF of X is: 1/1 ln(1/x)
E[X] = 1/4







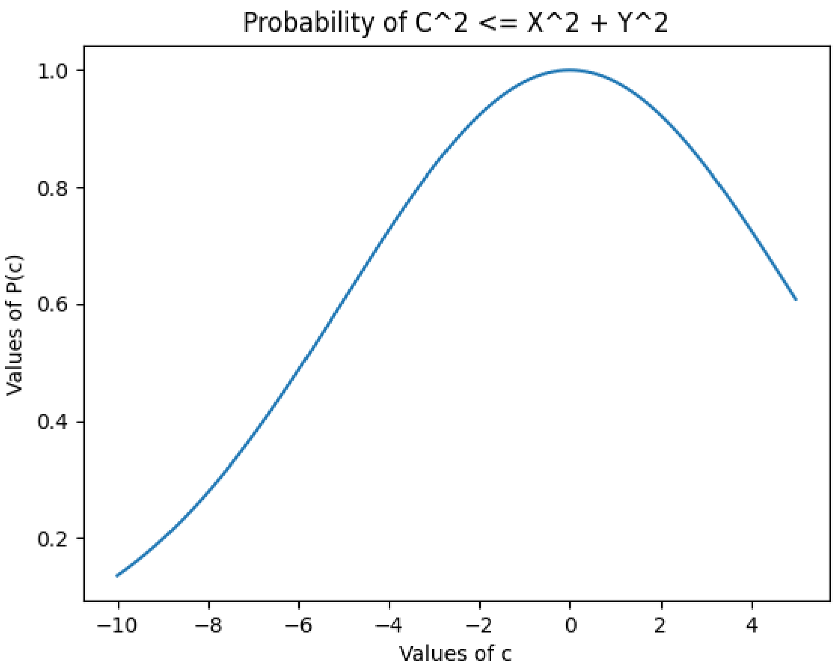
جواب سوال بیست و پنجم:

```
def solving_chapter2_problem_c3_q25():
  sigma = input("Please enter  value: ")
  sigma = float(sigma)
  c = np.arange(-10, 5, 0.01)
  y = np.exp(-((pow(c,2))/(2*pow(sigma,2))))
  print('Values of c: ', c)
  print('Values of P(c): ', y)

  plt.plot(c, y)

  plt.title("Probability of C^2 <= X^2 + Y^2")
  plt.xlabel("Values of c")
  plt.ylabel("Values of P(c)")
  plt.show()
```

در این سوال پس از اجرای کد مقدار سیگما از کاربر اخذ می شود و دریافت این مقدار نمودار آن رسم می گردد.

جواب این سوال با توجه به مقدار ورودی فرق میکند. در زیر خروجی به ازای مقدار ورودی ۵ آمده است:

```
Please enter  value: 5
Values of c:  [-10.    -9.99   -9.98 ...    4.97    4.98    4.99]
Values of P(c):  [0.13533528 0.13587744 0.13642122
  ... 0.6101698  0.60895677 0.60774372]
```


Probability of C^2 <= X^2 + Y^2
Values of P(c)
Values of c
1.0
0.8
0.6
0.4
0.2
−10
−8
−6
−4
−2
0
2
4