# LECTURE 5

# PYTHON CODES

## PROBABILITY

DR. PRAPASSORN TANTIPHANWADI
INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING AT KHAMPAENGSAEN
DECEMBER 2565

# SCHEDULE

| ลำดับ | วันที่ | เนื้อหา | ชั่วโมงบรรยาย |
|---|---|---|---|
| 1 | สอน | ความรู้เบื้องต้นวิทยาศาสตร์ข้อมูล<br>(Introduction to data science) | 3 |
| 2-3 | สอน | การเขียนโปรแกรมไพธอน (& Excel-VBA) – รหัสและโครงสร้าง<br>(Python programming (& Excel-VBA) – Codes and Structures) | 6 |
| 4 | สอน | พีชคณิตเชิงเส้นด้วยไพธอน - พีชคณิตเชิงเส้นและการคำนวณเมทริกซ์<br>(Linear algebra with Python - Linear algebra and matrix computations) | 3 |
| 5 | 10 มค | ความน่าจะเป็นด้วยไพธอน - ความน่าจะเป็น ความน่าจะเป็นร่วม ความน่าจะเป็นแบบมีเงื่อนไข ทฤษฎีบทของเบย์ การแจกแจงความน่าจะเป็นแบบไม่ต่อเนื่องและต่อเนื่อง<br>( Probability with Python - Probability, Joint Probability, Conditional Probability, Bayes' Theorem, Discrete and Continuous Probability Distributions) | 3 |
| 6 | 17 มค | สถิติพื้นฐานด้วยไพธอน (& Excel-VBA) - ตัวแปรสุ่ม การกระจายของตัวอย่าง การทดสอบสมมติฐาน การถดถอยเชิงเส้นอย่างง่ายและแบบพหุคูณ การถดถอยพหุนาม<br>(Basic statistics with Python (& Excel-VBA) - Random variables, Sampling Distributions, Hypothesis testing, Simple and Multiple Linear Regression, Polynomial Regression) | 3 |
| 7-8 | 24, 31 มค | การวิเคราะห์ข้อมูล การสร้างภาพข้อมูล และการวิเคราะห์ด้วยภาพ<br>(Data Exploration, visualization and visual analysis – python & Excel VBA) | 6 |

# SCHEDULI

| ลำดับ | วันที่ | เนื้อหา | ชั่วโมงบรรยาย |
|---|---|---|---|
| 9 | ???? | การวิเคราะห์ข้อมูลของแอปพลิเคชันอินเตอร์เน็ตในทุกสรรพสิ่ง - การวิเคราะห์และการแสดงข้อมูล IoT บนเครื่องอาร์เอฟไอดี<br>(Data Analysis of IoT Application - IoT Data Analysis and Visualization on RFID machine) | 3 |
| 10-11 | 7, 14 กพ | การเรียนรู้ของเครื่องจักร - การเรียนรู้ของเครื่องจักรภายใต้การดูแล/ไม่มีผู้ดูแล การจัดกลุ่มด้วยเคมีน การจำแนกประเภทด้วยเคเอ็นเอ็น การจัดกลุ่มด้วยแผนผังการตัดสินใจ, เทคนิครวมกลุ่ม<br>(Machine learning - Supervised/unsupervised machine learning, K-mean Clustering, Classification with KNN, Clustering with Decision Tree, Ensemble Techniques) | 6 |
| 12 | 21 กพ | การประมวลผลภาพดิจิทัล - ความรู้เบื้องต้นเกี่ยวกับการประมวลผลภาพดิจิทัลสำหรับการตรวจสอบอัตโนมัติ<br>(Digital Image Processing - Introduction to Digital Image Processing for automated inspection) | 3 |
| 13-14 | 28 กพ, 7 มีค | การเรียนรู้เชิงลึก - ความรู้เบื้องต้นเกี่ยวกับคีราส และการไหลของเทนเซอร์ เครือข่ายประสาทเทียมแบบคอนโวลูชัน เครือข่ายแบบคอนโวลูชันตามส่วน<br>(Deep Learning - Introduction to Keras and Tensor flow, Introduction of neural networks, Convolutional Neural Network (CNN), Region Based Convolutional Neural Networks (R-CNN)) | 6 |
| 15 | 14 มีค | กรณีศึกษา: การเรียนรู้ของเครื่องจักรหรือการเรียนรู้เชิงลึก ในกระบวนการผลิตและการขนส่ง<br>(Case Studies: Machine Learning or Deep Learning in Manufacturing and Logistics) | 3 |
| **รวม** | | | **45** |

# CONTENT

- Discrete random variables
- Continuous random variables
- Joint Probability
- Conditional Probability
- Bayes' Theorem

# DISCRETE RANDOM VARIABLES

- toss two dices, the outcome $\Omega = \{(1,1),(1,2),\ldots,(5,6),(6,6)\}$
- For example: the product of the respective measures of each element

$$\mathbb{P}((1,2)) = \mathbb{P}(\{1\})\mathbb{P}(\{2\}) = \frac{1}{6^2}$$

- the following question: what is the probability that the sum of the dice equals seven?
  - the first thing to do is characterize

$$X : (a,b) \mapsto (a+b).$$

  - Next, associate all of the (a, b) pairs with their sum

```
>>> d={(i,j):i+j for i in range(1,7) for j in range(1,7)}
```

  - The next step is to collect all of the (a, b) pairs that sum to each of the possible values from two to twelve.

```
>>> from collections import defaultdict
>>> dinv = defaultdict(list)
>>> for i,j in d.items():
...     dinv[j].append(i)
```

**Programming Tip**

The `defaultdict` object from the built-in collections module creates dictionaries with default values when it encounters a new key. Otherwise, we would have had to create default values manually for a regular dictionary.

# DISCRETE RANDOM VARIABLES

```
In [15]: import numpy as np
         import pandas as pd
```

```
In [16]: d={(i,j):i+j for i in range(1,7) for j in range(1,7)}
```

```
In [17]: from collections import defaultdict  #The defaultdict object from the built-in collections module creates dictionari
         # Otherwise, we would when it encounters a have had to create default values manually for a regular dictionary.
```

```
In [18]: dinv = defaultdict(list)
```

```
In [22]: for i,j in d.items():
             dinv[j].append(i)
```

```
In [23]: dinv[7] # contains the following list of pairs that sum to seven,
```

```
Out[23]: [(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)]
```

```
In [24]: X={i:len(j)/36. for i,j in dinv.items()}
```

```
In [25]: print(X)
```

```
{2: 0.027777777777777776, 3: 0.05555555555555555, 4: 0.08333333333333333, 5: 0.1111111111111111, 6: 0.1388888888888889, 7: 0.16666666666666
666, 8: 0.1388888888888889, 9: 0.1111111111111111, 10: 0.08333333333333333, 11: 0.05555555555555555, 12: 0.027777777777777776}
```

Sample space.

$S = (1,1), (1,2), (1,3), (1,4), (1,5), (1,6)$
$(2,1), (2,2), (2,3), (2,4), (2,5), (2,6)$
$(3,1), (3,2), (3,3), (3,4), (3,5), (3,6)$
$(4,1), (4,2), (4,3), (4,4), (4,5), (4,6)$
$(5,1), (5,2), (5,3), (5,4), (5,5), (5,6)$
$(6,1), (6,2), (6,3), (6,4), (6,5), (6,6)$

dinv [i+j]

2 : (1,1) : $P = \left(\frac{1}{36}\right) = 0.0278$
3 : (1,2), (2,1) : $P = \left(\frac{1}{36}\right) + \left(\frac{1}{36}\right) = 0.05556$
4 : (1,3), (2,2), (3,1)
5 : (1,4), (2,3), (3,2), (4,1)
6 : (1,5), (2,4), (3,3), (4,2), (5,1)
7 : (1,6), (2,5), (3,4), (4,3), (5,2), (6,1)

# DISCRETE RANDOM VARIABLES

- can ask other questions like what is the probability that half the product of three dice will exceed the their sum?

In [4]:
```
import numpy as np
import pandas as pd
from collections import defaultdict
```

In [10]:
```
# the probability that half the product of three dice will exceed the their sum
d={(i,j,k):((i*j*k)/2>i+j+k) for i in range(1,7)
                             for j in range(1,7)
                               for k in range(1,7)}
```

In [6]:
```
dinv = defaultdict(list)
```

In [7]:
```
for i,j in d.items():
    dinv[j].append(i)  #dinv contains only two keys, True and False
```

In [8]:
```
# because the dice are independent, the probability of any triple is 1/(6^3). Finally, we collect this for ea
X={i:len(j)/6.0**3 for i,j in dinv.items()}
```

In [11]:
```
print(X)
# Thus, the probability of half the product of three dice exceeding their sum is 136/6.0**3) = 0.63.
```

```
{False: 0.37037037037037035, True: 0.629629629629297}
```

# DISCRETE RANDOM VARIABLES



Consider the first problem with the two dice where we want the probability of a seven, but this time one of the dice is no longer fair. Find out the elementary probabilities.

$$\mathbb{P}(\{1\}) = \mathbb{P}(\{2\}) = \mathbb{P}(\{3\}) = \frac{1}{9}$$

$$\mathbb{P}(\{4\}) = \mathbb{P}(\{5\}) = \mathbb{P}(\{6\}) = \frac{2}{9}$$

Out[79]:

| | sm |
|---|---|
| 2 | 0.018519 |
| 3 | 0.037037 |
| 4 | 0.055556 |
| 5 | 0.092593 |
| 6 | 0.12963 |
| 7 | 0.166667 |
| 8 | 0.148148 |
| 9 | 0.12963 |
| 10 | 0.111111 |
| 11 | 0.074074 |
| 12 | 0.037037 |

Name: p, dtype: object

All we need to change is the probability computation of each of elements.

$$\mathbb{P}((1,6)) = \mathbb{P}(1)\mathbb{P}(6) = \frac{1}{9} \times \frac{1}{6} \qquad \mathbb{P}((2,5)) = \mathbb{P}(2)\mathbb{P}(5) = \frac{1}{9} \times \frac{1}{6}$$

$$\mathbb{P}_X(7) = \frac{1}{9} \times \frac{1}{6} + \frac{1}{9} \times \frac{1}{6} + \frac{1}{9} \times \frac{1}{6} + \frac{2}{9} \times \frac{1}{6} + \frac{2}{9} \times \frac{1}{6} + \frac{2}{9} \times \frac{1}{6} = \frac{1}{6}$$

# DISCRETE RANDOM VARIABLES

Expectation $\mathbf{E}[X]$ where $X$ is uniform from $[0,1]$

```
import numpy as np
X  = np.random.rand(10000)
mX = np.mean(X)
```

**Mean from PMF**

```
# Python code to compute the expectation
import numpy as np
p = np.array([0.25, 0.5, 0.25])
x = np.array([0, 1, 2])
EX = np.sum(p*x)
```

## Mean of geometric random variable

The expectation of a random variable $X$ where

$$p_X(k) = \frac{1}{2^k} \quad k = 1, 2, 3 \ldots .$$

```
# Python code to compute the expectation
import numpy as np
k = np.arange(100)
p = np.power(0.5,k)
EX = np.sum(p*k)
```

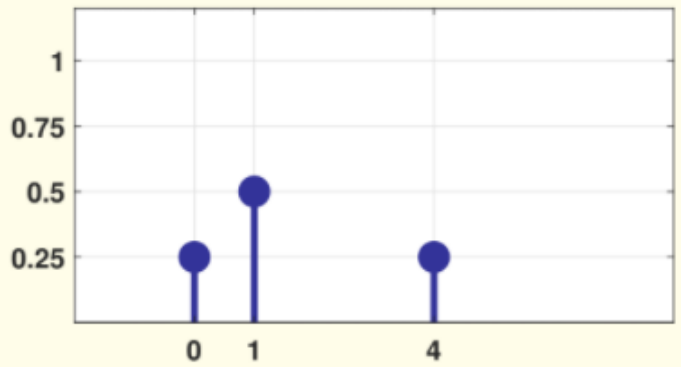
```
# Python code to generate 1000 Bernoulli random variables
import numpy as np
import matplotlib.pyplot as plt
p = 0.5
n = 1
X = np.random.binomial(n,p,size=1000)
plt.hist(X,bins='auto')
```

# DISCRETE RANDOM VARIABLES

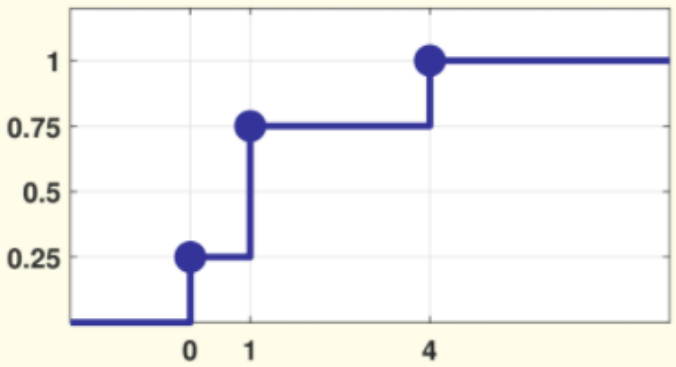
**Example 3.6**. Consider a random variable $X$ with PMF $p_X(0) = \frac{1}{4}$, $p_X(1) = \frac{1}{2}$ and $p_X(4) = \frac{1}{4}$. The CDF of $X$ can be computed as

$$F_X(0) = \mathbb{P}[X \leq 0] = p_X(0) = \frac{1}{4},$$
$$F_X(1) = \mathbb{P}[X \leq 1] = p_X(0) + p_X(1) = \frac{3}{4},$$
$$F_X(4) = \mathbb{P}[X \leq 4] = p_X(0) + p_X(1) + p_X(4) = 1.$$

As shown in **Figure 3.13**, the CDF of a discrete random variable is a staircase function.



(a) PMF $p_X(k)$



(b) CDF $F_X(k)$

```
% Python code to generate a PMF and a CDF
import numpy as np
import matplotlib.pyplot as plt
p = np.array([0.25, 0.5, 0.25])
x = np.array([0, 1, 4])
F = np.cumsum(p)

plt.stem(x,p,use_line_collection=True); plt.show()
plt.step(x,F); plt.show()
```

# DISCRETE RANDOM VARIABLES

**Example 3.8**. Let $X$ be a random variable with PMF $p_X(0) = 1/4$, $p_X(1) = 1/2$ and $p_X(2) = 1/4$. We can show that the expectation is

$$\mathbb{E}[X] = (0)\underbrace{\left(\frac{1}{4}\right)}_{p_X(0)} + (1)\underbrace{\left(\frac{1}{2}\right)}_{p_X(1)} + (2)\underbrace{\left(\frac{1}{4}\right)}_{p_X(2)} = 1.$$

```
# Python code to compute the expectation
import numpy as np
p = np.array([0.25, 0.5, 0.25])
x = np.array([0, 1, 2])
EX = np.sum(p*x)
```

*The* **variance** *of a random variable $X$ is*

$$\mathrm{Var}[X] = \mathbb{E}[(X - \mu)^2],$$

```
% Python code to compute the variance
import numpy as np
X = np.random.rand(10000)
vX = np.var(X)
sX = np.std(X)
```

# DISCRETE RANDOM VARIABLES

Alternatively, we can call the scipy.stats library

```
# Python code to call scipy.stats library
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
p = 0.5
X  = stats.bernoulli.rvs(p,size=1000)
plt.hist(X,bins='auto');
```

To generate Bernoulli random variable objects, we can call the scipy.stats library

```
# Python code to generate a Bernoulli rv object
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
p = 0.5
rv = stats.bernoulli(p)
mean, var = rv.stats(moments='mv')
print(mean, var)
```

```
# Python code to generate 5000 Binomial random variables
import numpy as np
import matplotlib.pyplot as plt
p = 0.5
n = 10
X = np.random.binomial(n,p,size=5000)
plt.hist(X,bins='auto');
```

```
# Python code to plot CDF of a binomial random variable
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
p = 0.5
n = 10
rv = stats.binom(n,p)
x  = np.arange(11)
F  = rv.cdf(x)
plt.plot(x, F, 'bo', ms=10);
plt.vlines(x, 0, F, colors='b', lw=5, alpha=0.5);
```

# DISCRETE RANDOM VARIABLES

Compute Bernoulli random variables:
Mean and variance

```
# Python code to compute the mean and var of a binomial rv
import scipy.stats as stats
p = 0.5
n = 10
rv = stats.binom(n,p)
M, V = rv.stats(moments='mv')
print(M, V)
```

**Practice Exercise 3.9**. Show that the binomial PMF sums to 1.

**Solution**. We use the binomial theorem to prove this result:

$$\sum_{k=0}^{n} p_X(k) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} = (p + (1-p))^n = 1.$$

```
# Python code to compute the mean and var of a binomial rv
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
p = 0.5
n = 10
rv = stats.binom(n,p)
x  = np.arange(11)
F  = rv.cdf(x)
plt.plot(x, F, 'bo', ms=10);
plt.vlines(x, 0, F, colors='b', lw=5, alpha=0.5);
```

# DISCRETE RANDOM VARIABLES

Compute Geometric random variables:

$$p_X(k) = \underbrace{(1-p)^{k-1}}_{k-1 \text{ failures}} \underbrace{p}_{\text{final success}}.$$

```
# Python code to generate 1000 geometric random variables
import numpy as np
import matplotlib.pyplot as plt
p = 0.5
X = np.random.geometric(p,size=1000)
plt.hist(X,bins='auto');
```

```
# Python code to generate 1000 geometric random variables
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x  = np.arange(1,11)
rv = stats.geom(p)
f  = rv.pmf(x)
plt.plot(x, f, 'bo', ms=8, label='geom pmf')
plt.vlines(x, 0, f, colors='b', lw=5, alpha=0.5)
```

# DISCRETE RANDOM VARIABLES

```
# Python code to plot the Poisson PMF
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import factorial
lambda_set = [1, 4, 10]
p = np.zeros((20, 3))
k = np.arange(0, 20)
for i in range(0, 3):
    lambd = lambda_set[i]
    p[:,i] = lambd**k/factorial(k)*np.exp(-lambd)

plt.plot(k, p[:,0], 'bo')
plt.vlines(k, 0 , p[:,0], colors='b', lw=2)
plt.plot(k, p[:,1], 'go')
plt.vlines(k, 0 , p[:,1], colors='g', lw=2)
plt.plot(k, p[:,2], 'yo')
plt.vlines(k, 0 , p[:,2], colors='y', lw=2)
labels = ["lambda = 1", "lambda = 4", "lambda = 10"]
plt.legend(labels=labels)
plt.grid()
plt.show()
```

```
# Python code to plot the Poisson CDF
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import factorial
lambda_set = [1, 4, 10]
p = np.zeros((20, 3))
k = np.arange(0, 20)
for i in range(0, 3):
    lambd = lambda_set[i]
    p[:,i] = lambd**k/factorial(k)*np.exp(-lambd)

plt.step(k, np.cumsum(p[:,0]), 'b')
plt.step(k, np.cumsum(p[:,1]), 'g')
plt.step(k, np.cumsum(p[:,2]), 'y')
plt.plot(k, np.cumsum(p[:,0]), 'bo')
plt.plot(k, np.cumsum(p[:,1]), 'go')
plt.plot(k, np.cumsum(p[:,2]), 'yo')
labels = ["lambda = 1", "lambda = 4", "lambda = 10"]
plt.legend(labels=labels)
plt.grid()
plt.show()
```

# DISCRETE RANDOM VARIABLES

```
# Python code to approximate binomial using Poisson
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
n = 5000; p = 0.01
rv1 = stats.binom(n,p)
X   = rv1.rvs(size=10000)
plt.figure(1); plt.hist(X,bins=np.arange(0,100));
rv2 = stats.poisson(n*p)
f   = rv2.pmf(bin)
plt.figure(2); plt.plot(f);
```

# DISCRETE RANDOM VARIABLES

$$p_X(k) = \frac{\lambda^k}{k!} e^{-\lambda}, \qquad k = 0, 1, 2, \ldots,$$

```
# Python code to generate 5000 Poisson random variables
import numpy as np
import matplotlib.pyplot as plt
lambd = 1
X = np.random.poisson(lambd,size=5000)
plt.hist(X,bins='auto');
```

```
# Python code to plot the Poisson PMF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.arange(0,11)
rv = stats.poisson(lambd)
f  = rv.pmf(x)
plt.plot(x, f, 'bo', ms=8, label='geom pmf')
plt.vlines(x, 0, f, colors='b', lw=5, alpha=0.5)
```

```
# Python code to compute Poisson statistics
import scipy.stats as stats
lambd = 1
rv = stats.poisson(lambd)
M, V = rv.stats(moments='mv')
```

# CONTINUOUS RANDOM VARIABLES

```
# Python code to generate the PDF and CDF of a uniform random variable
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.linspace(-5,10,1500)
f = stats.uniform.pdf(x,-3,4)
F = stats.uniform.cdf(x,-3,4)
plt.plot(x,f); plt.show()
plt.plot(x,F); plt.show()
```

```
# Python code to generate 1000 uniform random numbers
import scipy.stats as stats
a = 0; b = 1;
X = stats.uniform.rvs(a,b,size=1000)
plt.hist(X);
```

```
# Python code to generate the PDF and CDF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.linspace(-5,10,1500)
f = stats.expon.pdf(x,2)
F = stats.expon.cdf(x,2)
plt.plot(x,f); plt.show()
plt.plot(x,F); plt.show()
```

```
# Python code to compute empirical mean, var, median, mode
X = stats.uniform.rvs(a,b,size=1000)
M = np.mean(X)
V = np.var(X)
Med = np.median(X)
Mod = stats.mode(X)
```

```
# Python code to compute the probability P(0.2 < X < 0.3)
import scipy.stats as stats
a = 0; b = 1;
F = stats.uniform.cdf(0.3,a,b)-stats.uniform.cdf(0.2,a,b)
```

# CONTINUOUS RANDOM VARIABLES

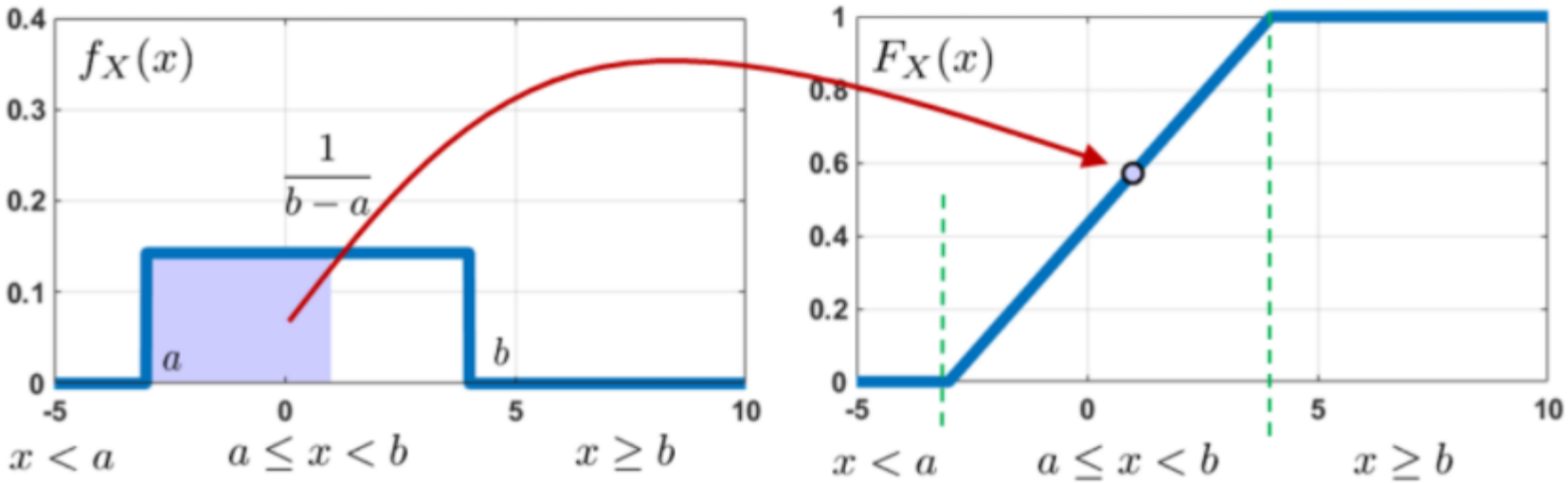



**Figure 4.6:** Example: $f_X(x) = 1/(b-a)$ for $a \le x \le b$. The CDF has three segments.

```
# Python code to generate the PDF and CDF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.linspace(-5,10,1500)
f = stats.uniform.pdf(x,-3,4)
F = stats.uniform.cdf(x,-3,4)
plt.plot(x,f); plt.show()
plt.plot(x,F); plt.show()
```

# CONTINUOUS RANDOM VARIABLES

```
# Python code to generate PDF and CDF of an exponential random variable
lambd1 = 1/2
lambd2 = 1/5
x = np.linspace(0,1,1000)
f1 = stats.expon.pdf(x,scale=lambd1)
f2 = stats.expon.pdf(x,scale=lambd2)
plt.plot(x, f1)
plt.plot(x, f2)

F1 = stats.expon.cdf(x,scale=lambd1)
F2 = stats.expon.cdf(x,scale=lambd2)
plt.plot(x, F1)
plt.plot(x, F2)
```

```
# Python code to generate Gaussian from uniform
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

mu = 3
sigma = 2
U  = stats.uniform.rvs(0,1,size=10000)
gU = sigma*stats.norm.ppf(U)+mu
plt.hist(U); plt.show()
plt.hist(gU); plt.show()
```

```
# Python code to generate standard Gaussian PDF and CDF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.linspace(-10,10,1000)
f = stats.norm.pdf(x)
F = stats.norm.cdf(x)
plt.plot(x,f); plt.show()
plt.plot(x,F); plt.show()
```

# CONTINUOUS RANDOM VARIABLES

**Practice Exercise 4.11.** (**Exponential random variable**) Let $X$ be a continuous random variable with PDF $f_X(x) = \lambda e^{-\lambda x}$ for $x \geq 0$, and 0 otherwise. Find the CDF of $X$.

**Solution.** Clearly, for $x < 0$, we have $F_X(x) = 0$. For $x \geq 0$, we can show that

$$F_X(x) = \int_0^x f_X(x')\,dx' = \int_0^x \lambda e^{-\lambda x'}\,dx' = 1 - e^{-\lambda x}.$$

Therefore, the complete CDF is (see **Figure 4.7** for illustration):

$$F_X(x) = \begin{cases} 0, & x < 0, \\ 1 - e^{-\lambda x}, & x \geq 0. \end{cases}$$

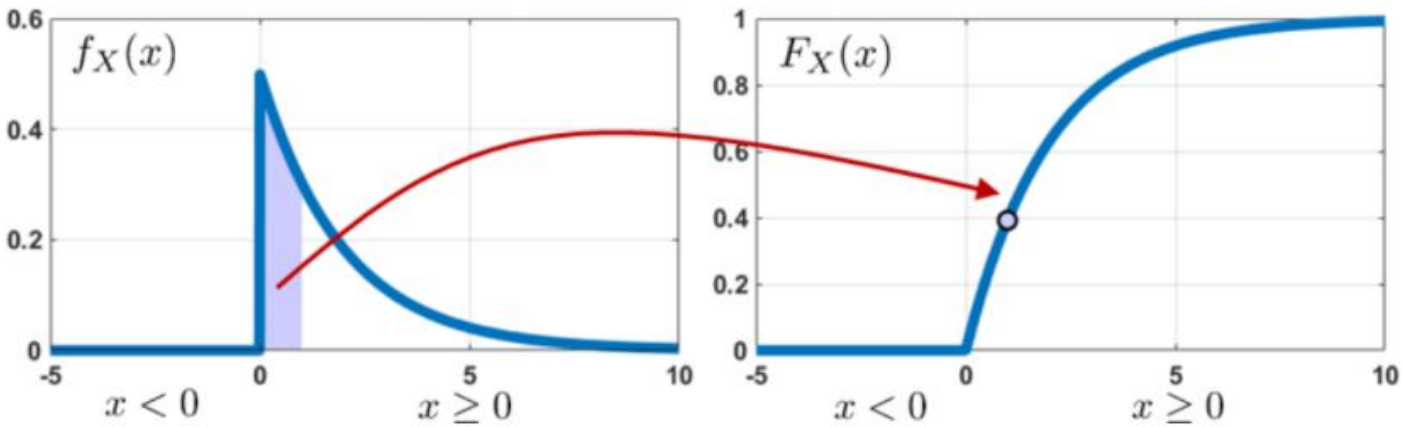



**Figure 4.7:** Example: $f_X(x) = \lambda e^{-\lambda x}$ for $x \geq 0$. The CDF has two segments.

```
# Python code to generate the PDF and CDF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
x = np.linspace(-5,10,1500)
f = stats.expon.pdf(x,2)
F = stats.expon.cdf(x,2)
plt.plot(x,f); plt.show()
plt.plot(x,F); plt.show()
```

# JOINT DISTRIBUTION

NO

```
# Python code to compute the correlation coefficient
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
x = stats.multivariate_normal.rvs([0,0], [[3,1],[1,1]], 10000)
plt.figure(); plt.scatter(x[:,0],x[:,1])
rho,_ = stats.pearsonr(x[:,0],x[:,1])
print(rho)
```

```
# Python code to compute a mean vector
import numpy as np
import scipy.stats as stats
X = stats.multivariate_normal.rvs([0,0],[[1,0],[0,1]],100)
mX = np.mean(X,axis=1)
```

```
# Python code to compute covariance matrix
import numpy as np
import scipy.stats as stats
X = stats.multivariate_normal.rvs([0,0],[[1,0],[0,1]],100)
covX = np.cov(X,rowvar=False)
print(covX)
```

```
# Python code: Overlay random numbers with the Gaussian contour.
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

X = stats.multivariate_normal.rvs([0,0],[[0.25,0.3],[0.3,1.0]],1000)
x1 = np.arange(-2.5, 2.5, 0.01)
x2 = np.arange(-3.5, 3.5, 0.01)
X1, X2 = np.meshgrid(x1,x2)
Xpos = np.empty(X1.shape + (2,))
Xpos[:,:,0] = X1
Xpos[:,:,1] = X2

F = stats.multivariate_normal.pdf(Xpos,[0,0],[[0.25,0.3],[0.3,1.0]])

plt.scatter(X[:,0],X[:,1])
plt.contour(x1,x2,F)
```

# JOINT DISTRIBUTION

NO

```
# Python code: Gaussian(mu,sigma) --> Gaussian(0,1)
import numpy as np
import scipy.stats as stats
from scipy.linalg import fractional_matrix_power
y  = np.random.multivariate_normal([1,-2],[[3,-0.5],[-0.5,1]],100)
mY = np.mean(y,axis=0)
covY  = np.cov(y,rowvar=False)
covY2 = fractional_matrix_power(covY,-0.5)
x     = np.dot(covY2, (y-np.matlib.repmat(mY,100,1)).T)
```

```
# Python code to perform the principal component analysis
import numpy as np
x  = np.random.multivariate_normal([1,-2],[[3,-0.5],[-0.5,1]],1000)
covX = np.cov(x,rowvar=False)
S, U = np.linalg.eig(covX)
print(U)
```

# HOMEWORK

- จงเขียน python code

1. Find expectation value

Consider a game in which we flip a coin 3 times. The reward of the game is

- \$1 if there are 2 heads
- \$8 if there are 3 heads
- \$0 if there are 0 or 1 head

There is a cost associated with the game. To enter the game, the player has to pay \$1.50. We want to compute the net gain, on average.

To answer this question, we first note that the sample space contains 8 elements: HHH, HHT, HTH, THH, THT, TTH, HTT, TTT. Let $X$ be the number of heads. Then the PMF of $X$ is

$$p_X(0) = \frac{1}{8}, \quad p_X(1) = \frac{3}{8}, \quad p_X(2) = \frac{3}{8}, \quad p_X(3) = \frac{1}{8}.$$

We then let $Y$ be the reward. The PMF of $Y$ can be found by "adding" the probabilities of $X$. This yields

$$p_Y(0) = p_X(0) + p_X(1) = \frac{4}{8}, \quad p_Y(1) = p_X(2) = \frac{3}{8}, \quad p_Y(8) = p_X(3) = \frac{1}{8}.$$

The expectation of $Y$ is

$$\mathbb{E}[X] = (0)\left(\frac{4}{8}\right) + (1)\left(\frac{3}{8}\right) + (8)\left(\frac{1}{8}\right) = \frac{11}{8}.$$

Since the cost of the game is $\frac{12}{8}$, the net gain (on average) is $-\frac{1}{8}$.

# HOMEWORK

- จงเขียน python code

2. Show the geometric PMF sums to one.

Show that the geometric PMF sums to one.

**Solution**. We can apply infinite series to show the result:

$$\begin{aligned}\sum_{k=1}^{\infty} p_X(k) &= \sum_{k=1}^{\infty}(1-p)^{k-1}p \\ &= p\cdot\sum_{k=1}^{\infty}(1-p)^{k-1}, \qquad \ell = k-1 \\ &= p\cdot\sum_{\ell=0}^{\infty}(1-p)^{\ell} \\ &= p\cdot\frac{1}{1-(1-p)} = 1.\end{aligned}$$

**Exercise 3**

Let

$$g(X) = \begin{cases} 1, & \text{if } X > 10 \\ 0, & \text{otherwise.} \end{cases} \quad \text{and} \quad h(X) = \begin{cases} X-10, & \text{if } X-10 > 0 \\ 0, & \text{otherwise.} \end{cases}$$

(a) Find $\mathbb{E}[g(X)]$ for $X$ as in Problem 1(a) with $S_X = \{1, \ldots, 15\}$.

(b) Find $\mathbb{E}[h(X)]$ for $X$ as in Problem 1(b) with $S_X = \{1, \ldots, 15\}$.

# HOMEWORK

- จงเขียน python code

**Exercise 4**

A voltage $X$ is uniformly distributed in the set $\{-3, \ldots, 3, 4\}$.

(a) Find the mean and variance of $X$.

(b) Find the mean and variance of $Y = -2X^2 + 3$.

(c) Find the mean and variance of $W = \cos(\pi X/8)$.

(d) Find the mean and variance of $Z = \cos^2(\pi X/8)$.

**Exercise 5**

(a) If $X$ is Poisson($\lambda$), compute $\mathbb{E}[1/(X+1)]$.

(b) If $X$ is Bernoulli($p$) and $Y$ is Bernoulli($q$), compute $\mathbb{E}[(X+Y)^3]$ if $X$ and $Y$ are independent.

(c) Let $X$ be a random variable with mean $\mu$ and variance $\sigma^2$. Let $\Delta(\theta) = \mathbb{E}[(X-\theta)^2]$. Find $\theta$ that minimizes the error $\Delta(\theta)$.

(d) Suppose that $X_1, \ldots, X_n$ are independent uniform random variables in $\{0, 1, \ldots, 100\}$. Evaluate $\mathbb{P}[\min(X_1, \ldots, X_n) > \ell]$ for any $\ell \in \{0, 1, \ldots, 100\}$.

# THE END