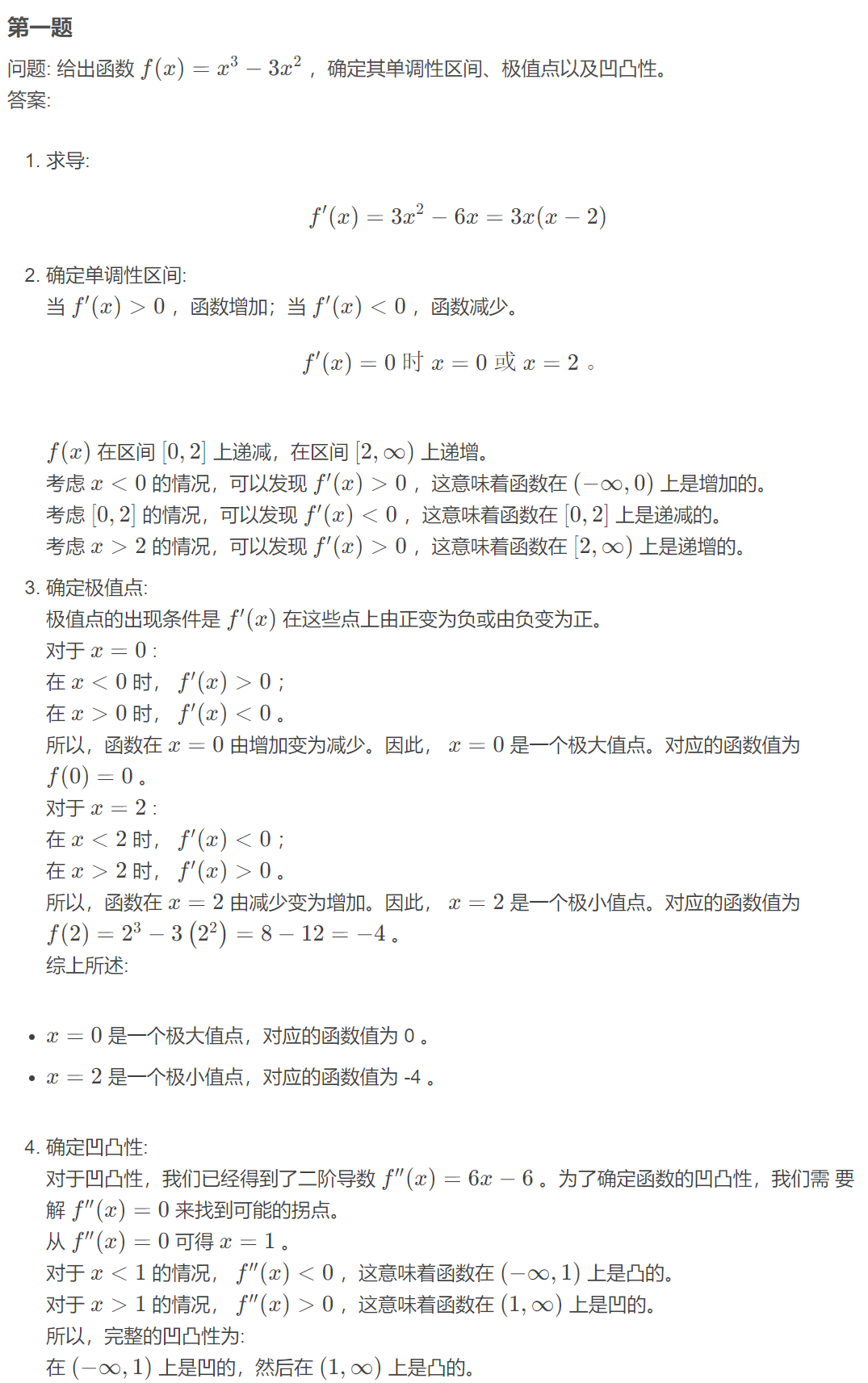
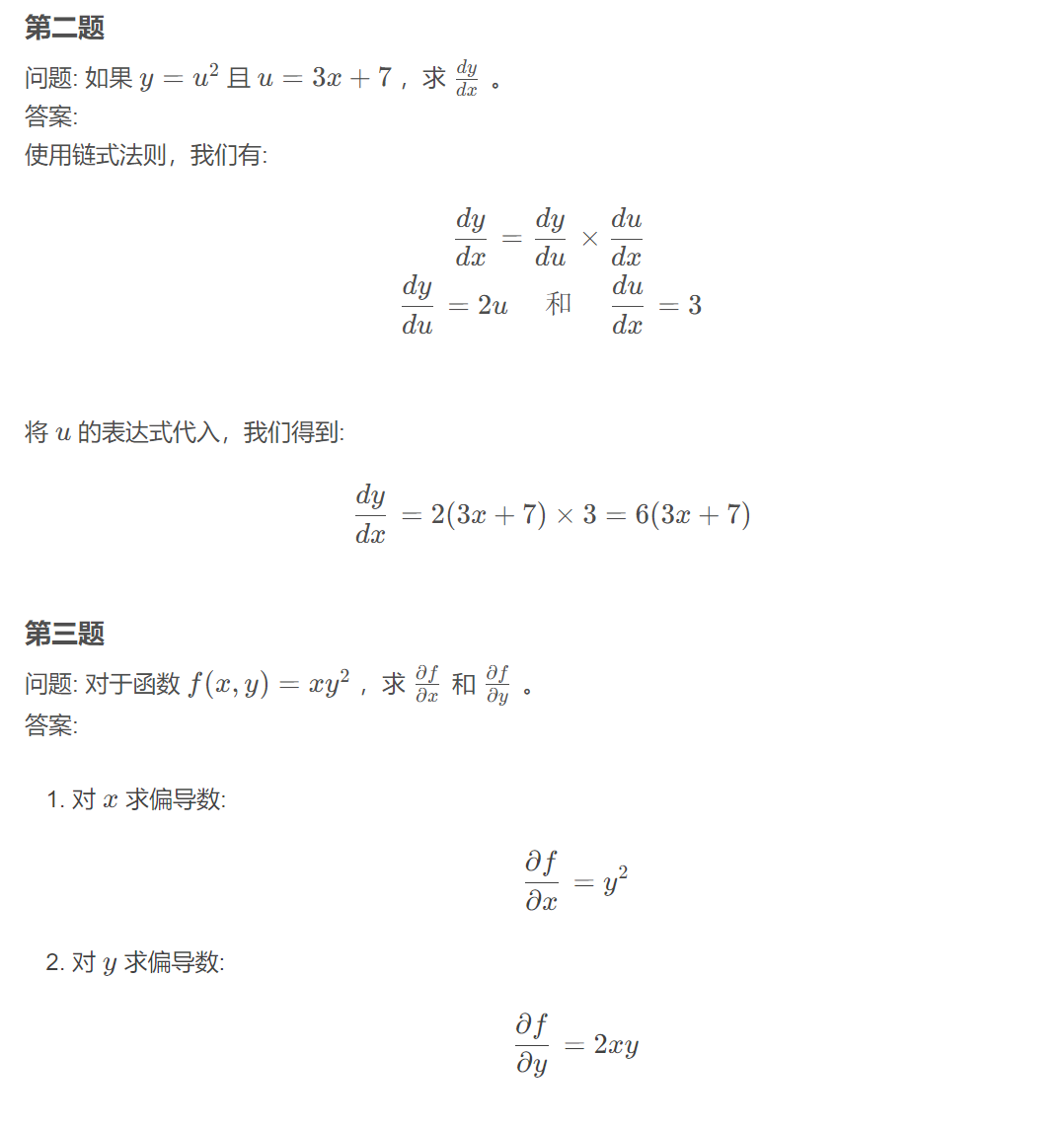
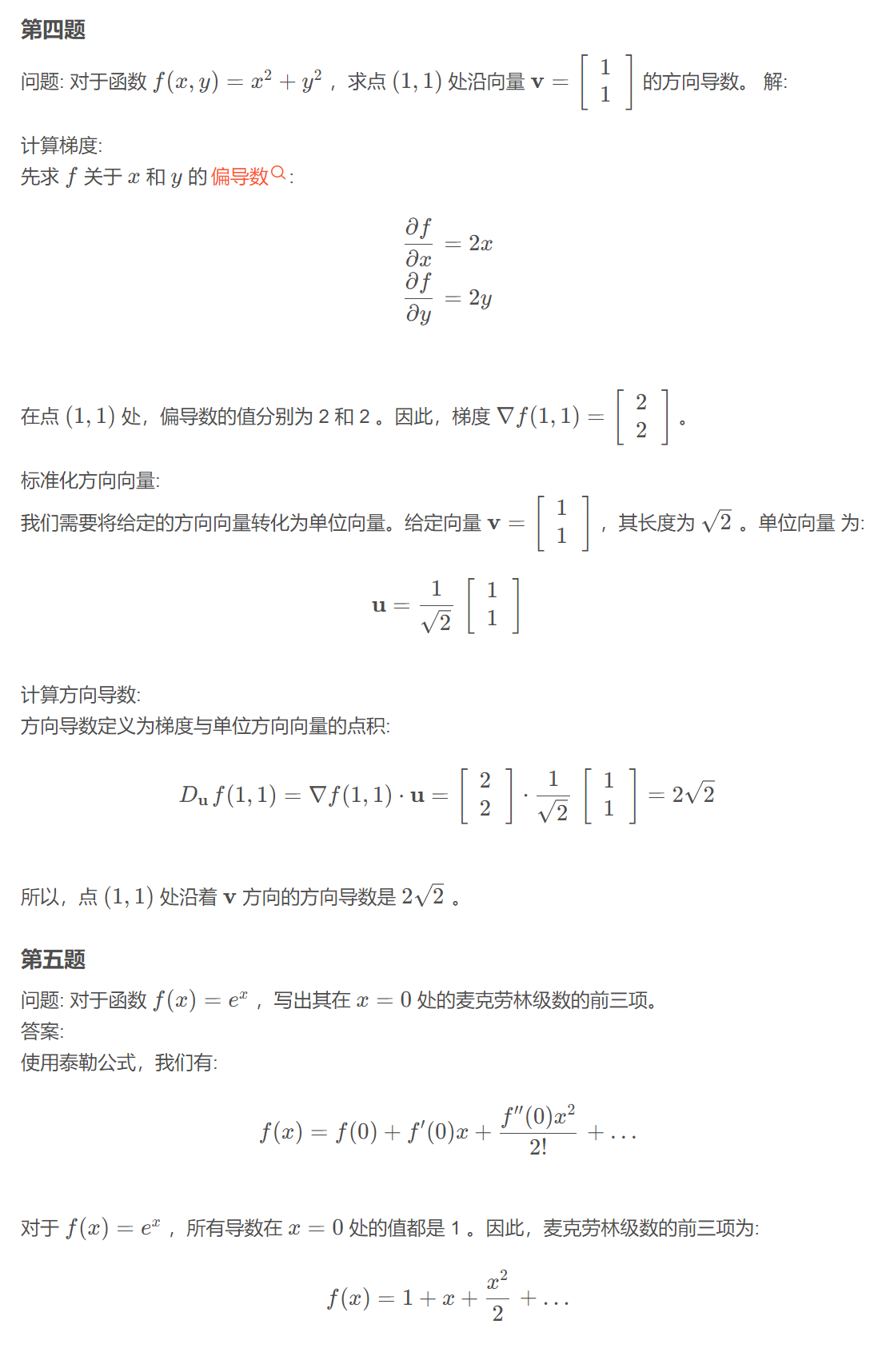
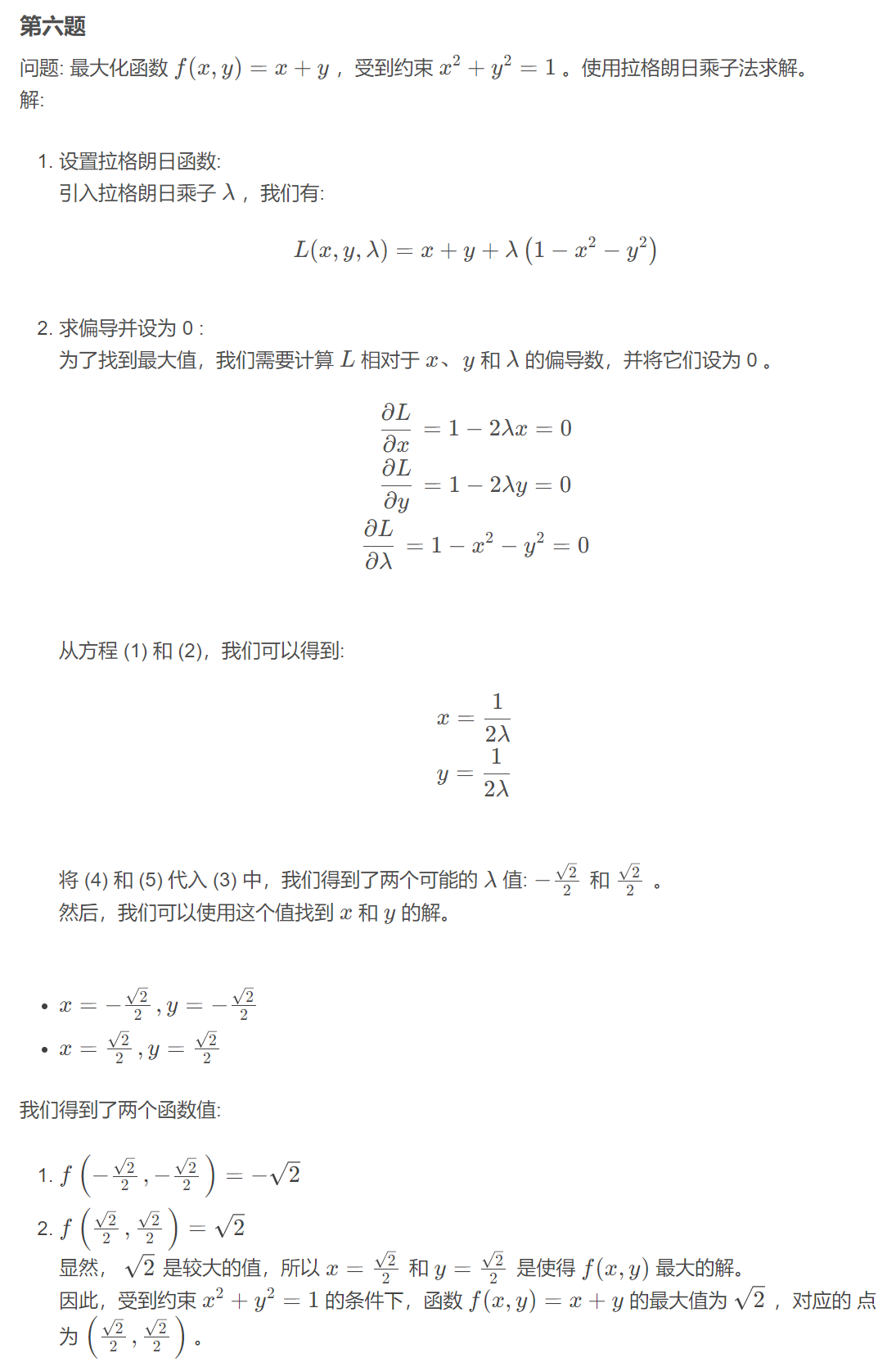
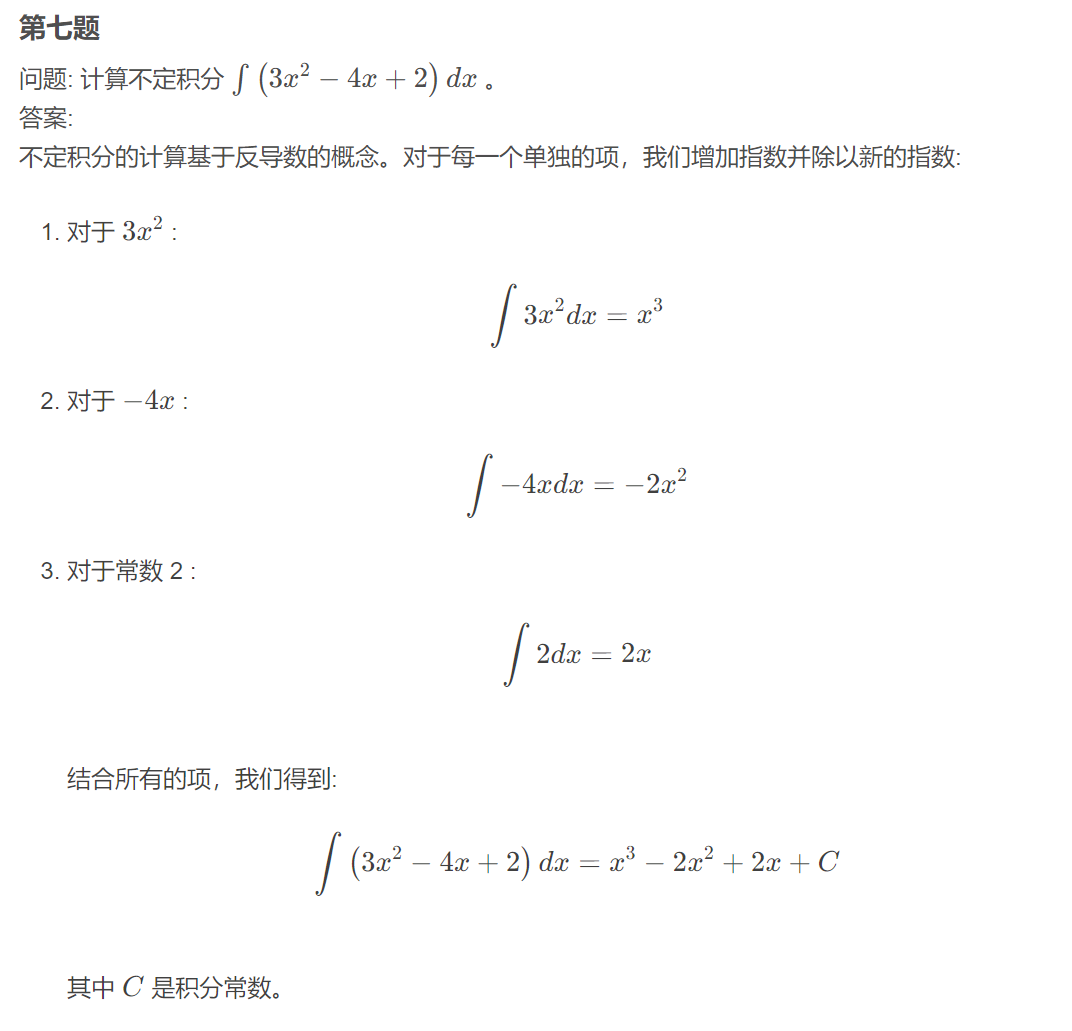
# 高数

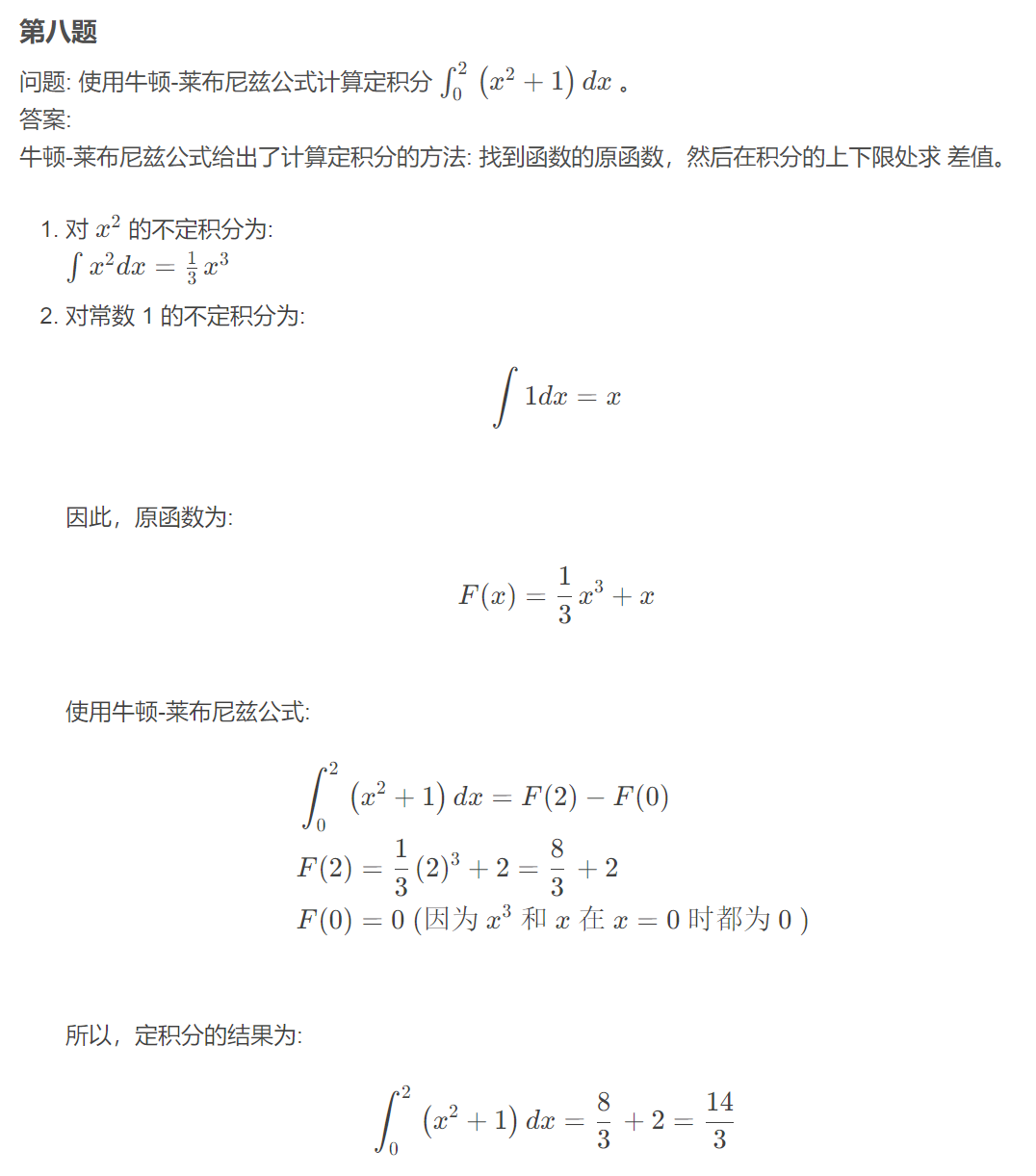












**第九题**

def manual\_gradient(f, variables, h=1e-5):

"""Compute the gradient of a function with respect to the given variables without using external libraries."""

gradient\_values = [] # 初始化一个空列表来存储每个变量的偏导数函数

for var in variables: # 对于输入的每个变量

def function\_at\_point(\*point):

"""Substitute the variables in the function with the given point."""

substitutions = {var: value for var, value in zip(variables, point)} # 创建一个字典，将变量映射到点的坐标

return eval(f, {}, substitutions) # 使用eval来计算函数在给定点的值

def partial\_derivative(point, var\_index=variables.index(var)):

"""Compute the partial derivative using the central difference method."""

point\_before = list(point) # 创建点的一个拷贝

point\_before[var\_index] -= h # 将对应变量的值减小一个小量 h

point\_after = list(point) # 再次创建点的一个拷贝

point\_after[var\_index] += h # 将对应变量的值增加一个小量 h

# 使用中心差分方法计算偏导数

return (function\_at\_point(\*point\_after) - function\_at\_point(\*point\_before)) / (2 \* h)

gradient\_values.append(partial\_derivative) # 将偏导数函数添加到列表中

return gradient\_values # 返回偏导数函数的列表

# Test the manual\_gradient function

f = "x\*\*2 + y\*\*2"

variables = ['x', 'y']

gradient\_funcs = manual\_gradient(f, variables)

# Evaluate the gradient at point (1, 2)

point = (1, 2)

gradient\_at\_point = [func(point) for func in gradient\_funcs]

gradient\_at\_point

**第十题**

def chain\_rule\_derivative(outer\_function, inner\_function, variable, h=1e-5):

"""使用链式法则计算导数，不使用外部库。"""

# 使用有限差分法计算外部函数在内部函数处的导数

def outer\_derivative\_at\_inner(point):

# 在给定的点处评估内部函数的值

inner\_value = eval(inner\_function, {variable: point})

# 为有限差分法在稍微移动的点上评估内部函数的值

inner\_value\_h = eval(inner\_function, {variable: point + h})

# 计算外部函数值的差分，并除以内部函数的值变化来近似导数

return (eval(outer\_function, {"u": inner\_value\_h}) - eval(outer\_function, {"u": inner\_value})) / (inner\_value\_h - inner\_value)

# 使用有限差分法计算内部函数的导数

def inner\_derivative(point):

# 计算内部函数值的差分，并除以 h 来近似导数

return (eval(inner\_function, {variable: point + h}) - eval(inner\_function, {variable: point})) / h

# 结合两个导数来应用链式法则

def combined\_derivative(point):

# 链式法则通过相乘两个导数来结合它们

return outer\_derivative\_at\_inner(point) \* inner\_derivative(point)

# 返回表示组合导数的函数

return combined\_derivative

# Example usage:

outer\_func = "u\*\*2"

inner\_func = "3\*x + 7"

variable = "x"

# Get the combined derivative function using chain rule

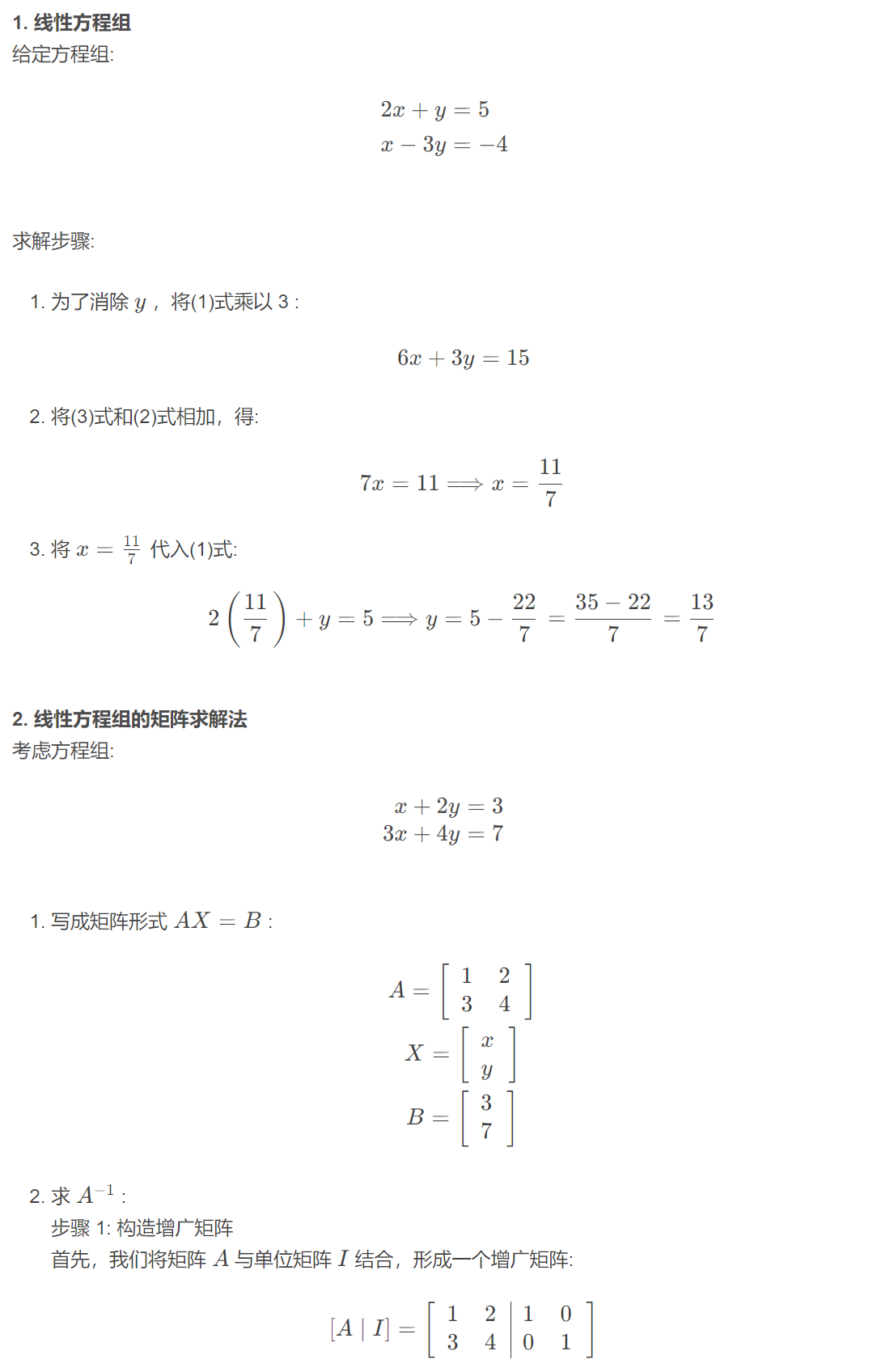
chain\_derivative\_func\_corrected = chain\_rule\_derivative(outer\_func, inner\_func, variable)

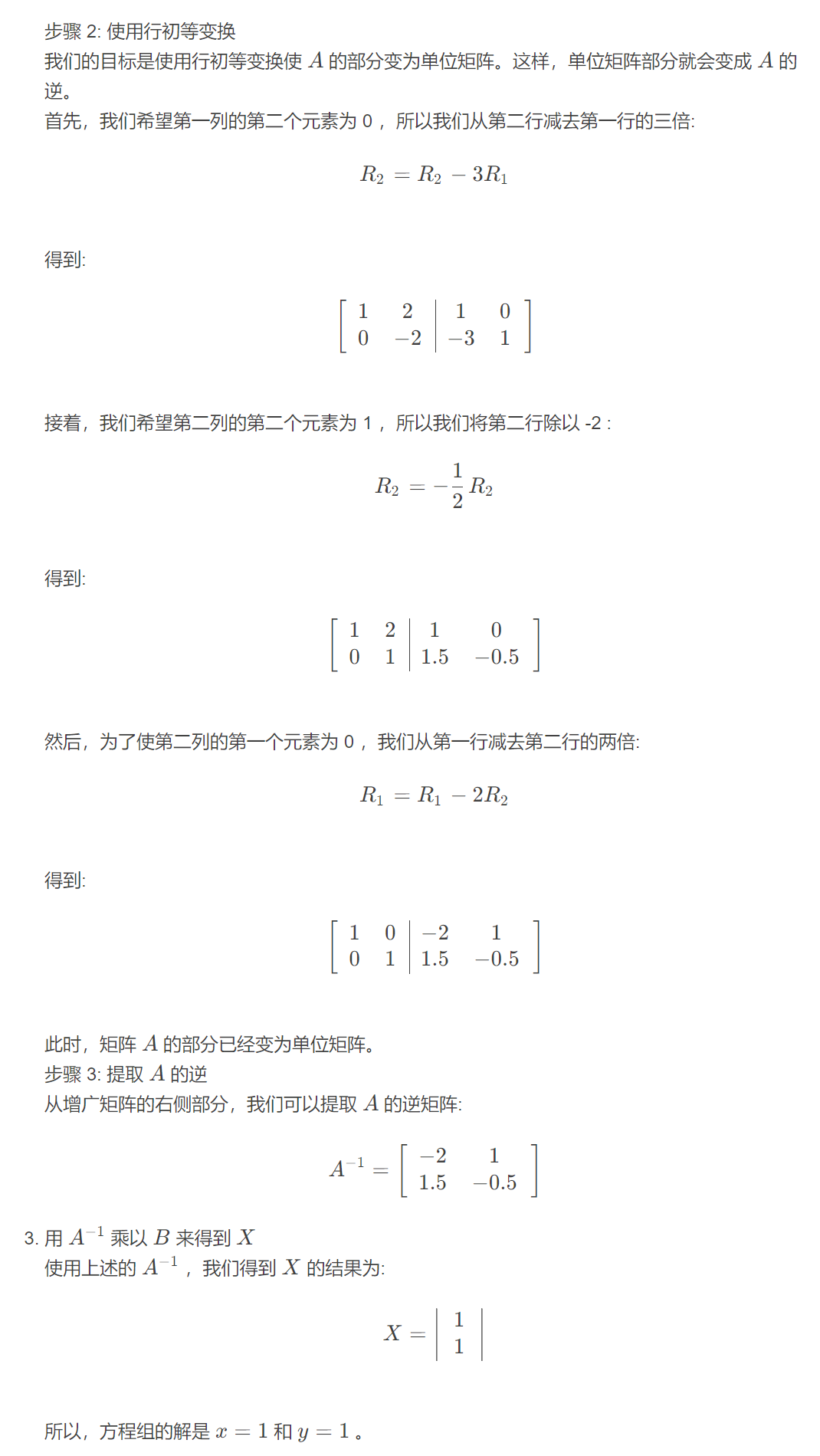
# Evaluate the derivative at the point x=1

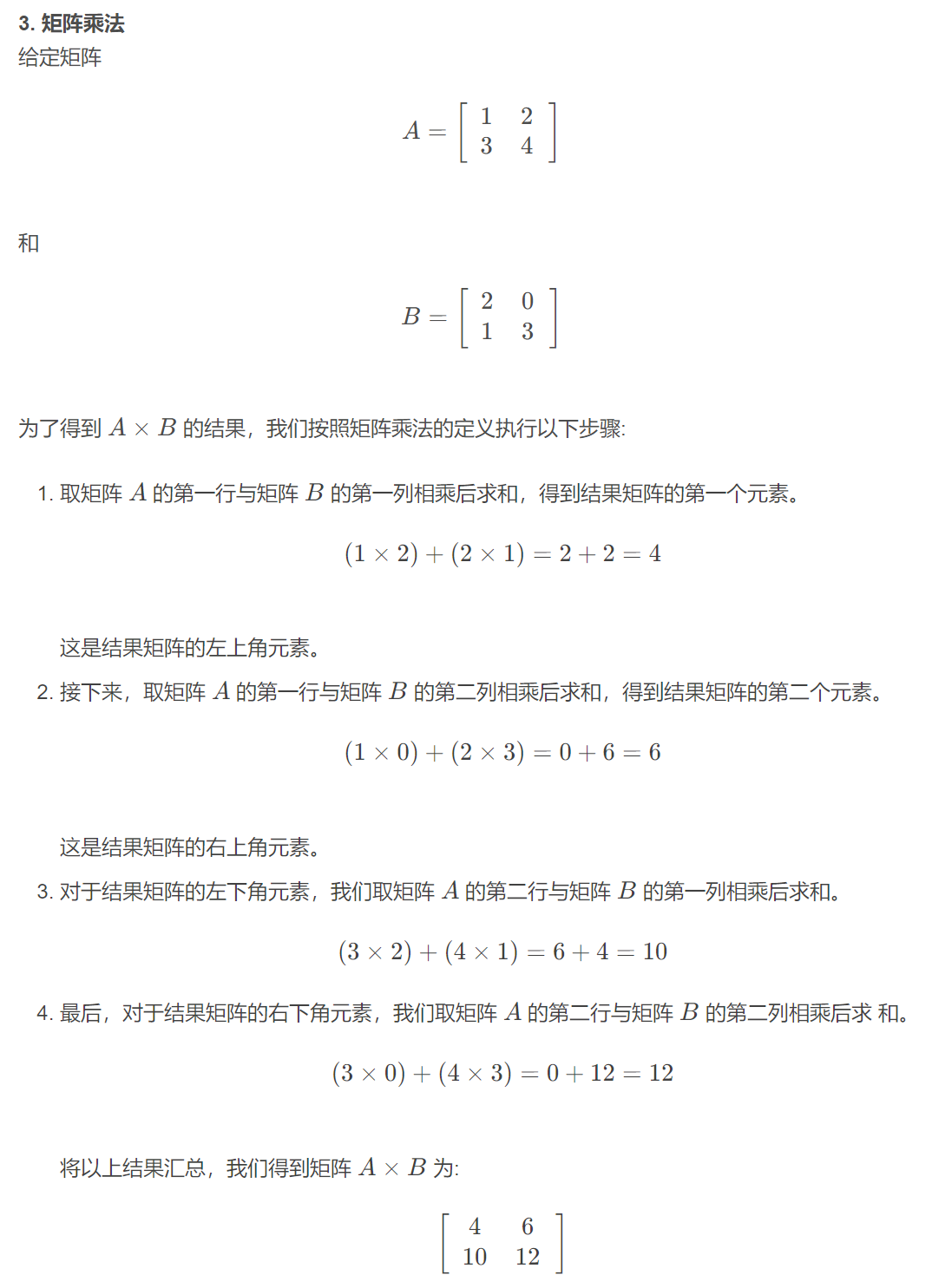
derivative\_at\_point\_corrected = chain\_derivative\_func\_corrected(1)

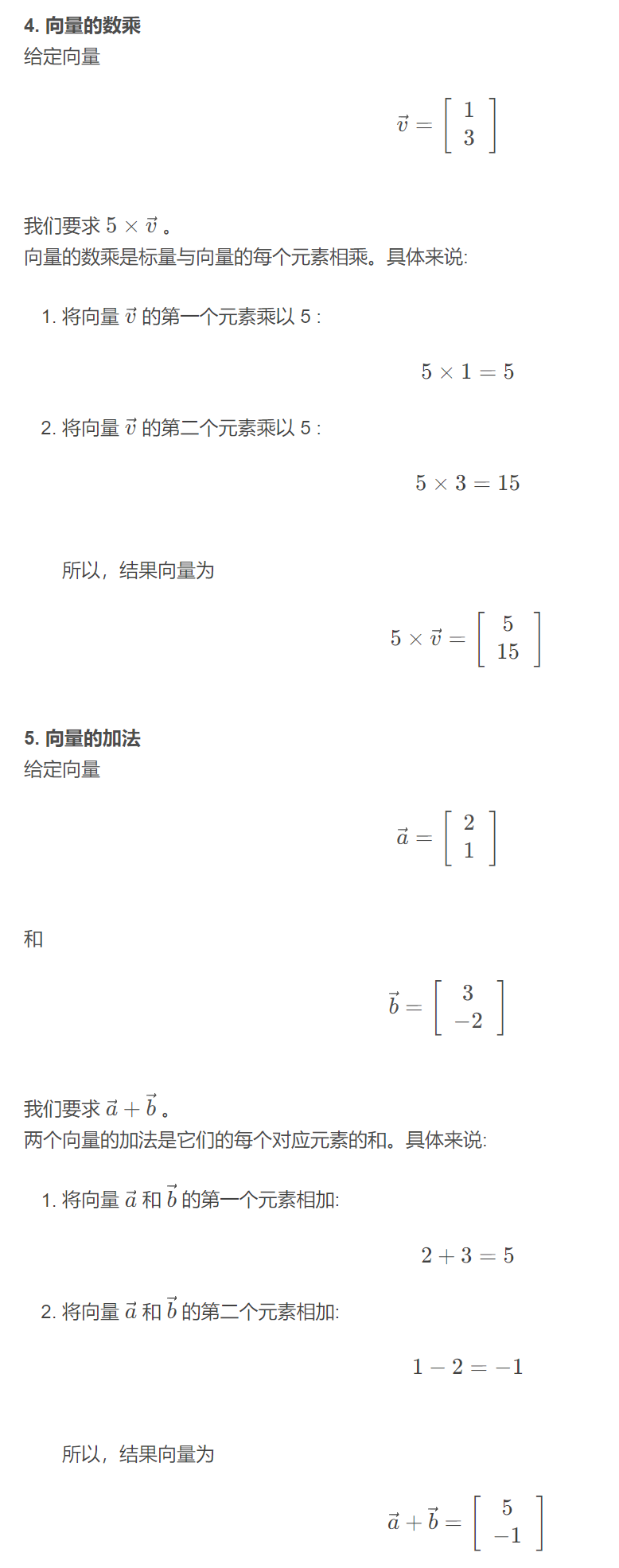
derivative\_at\_point\_corrected

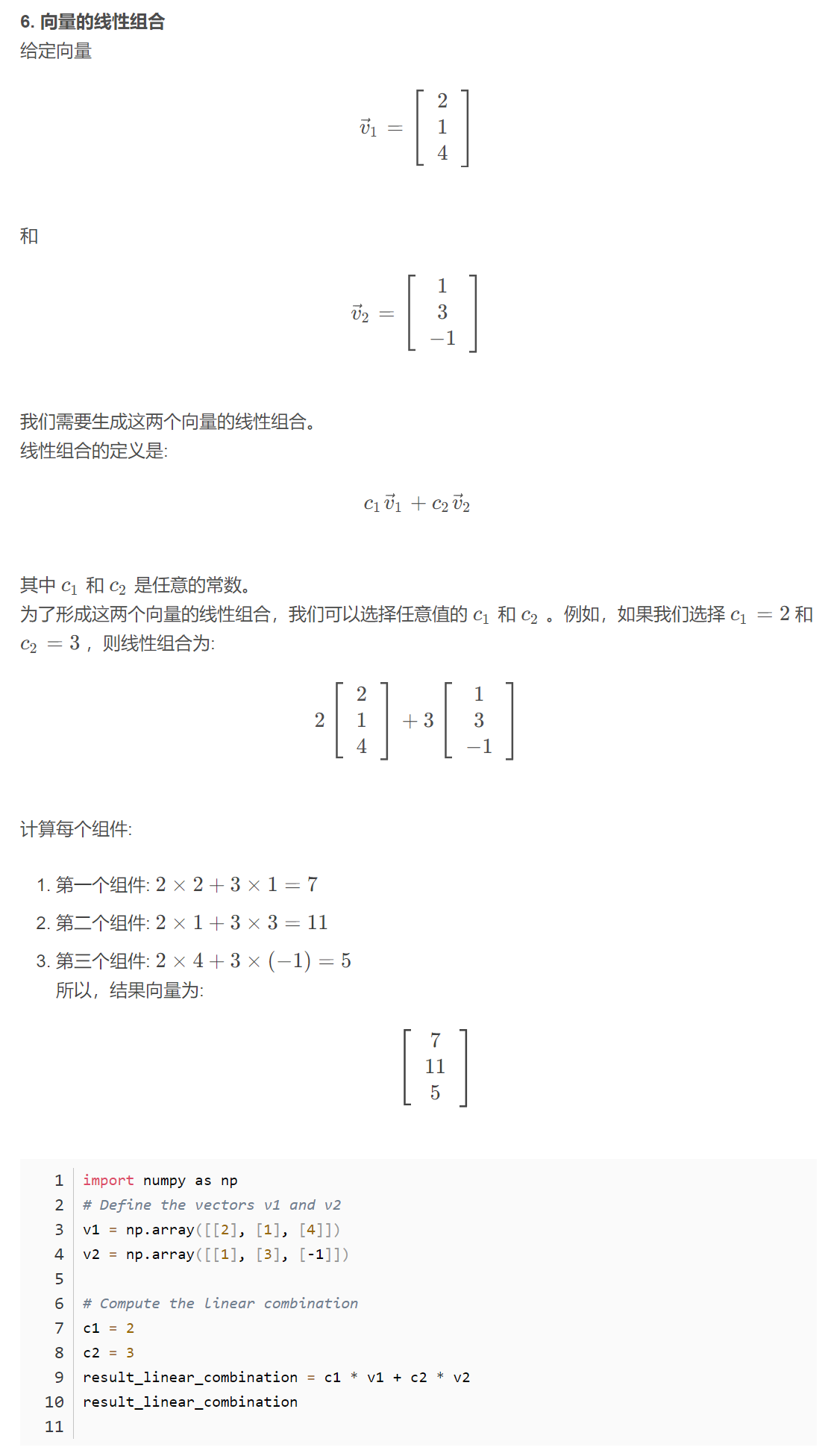
# 线代

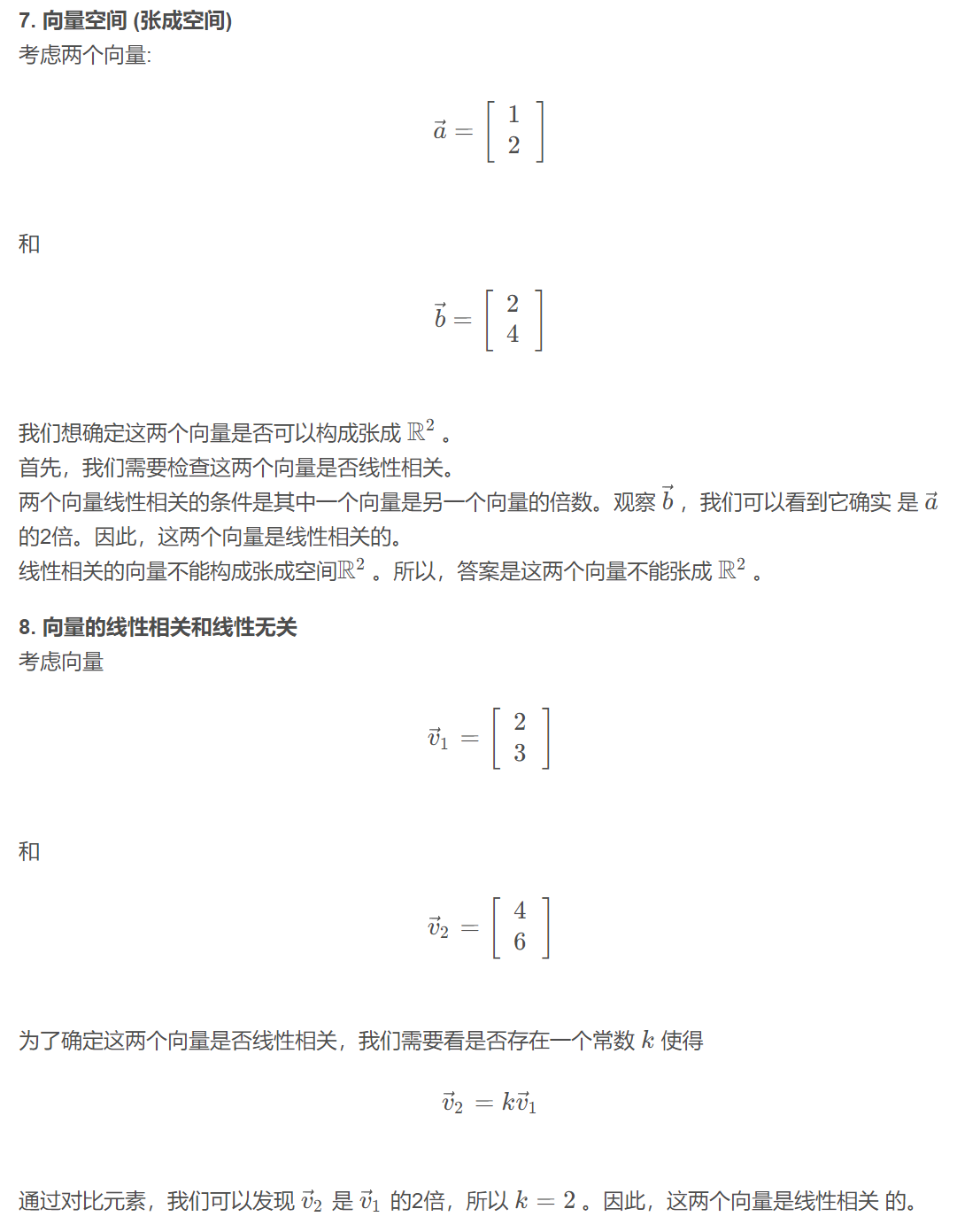


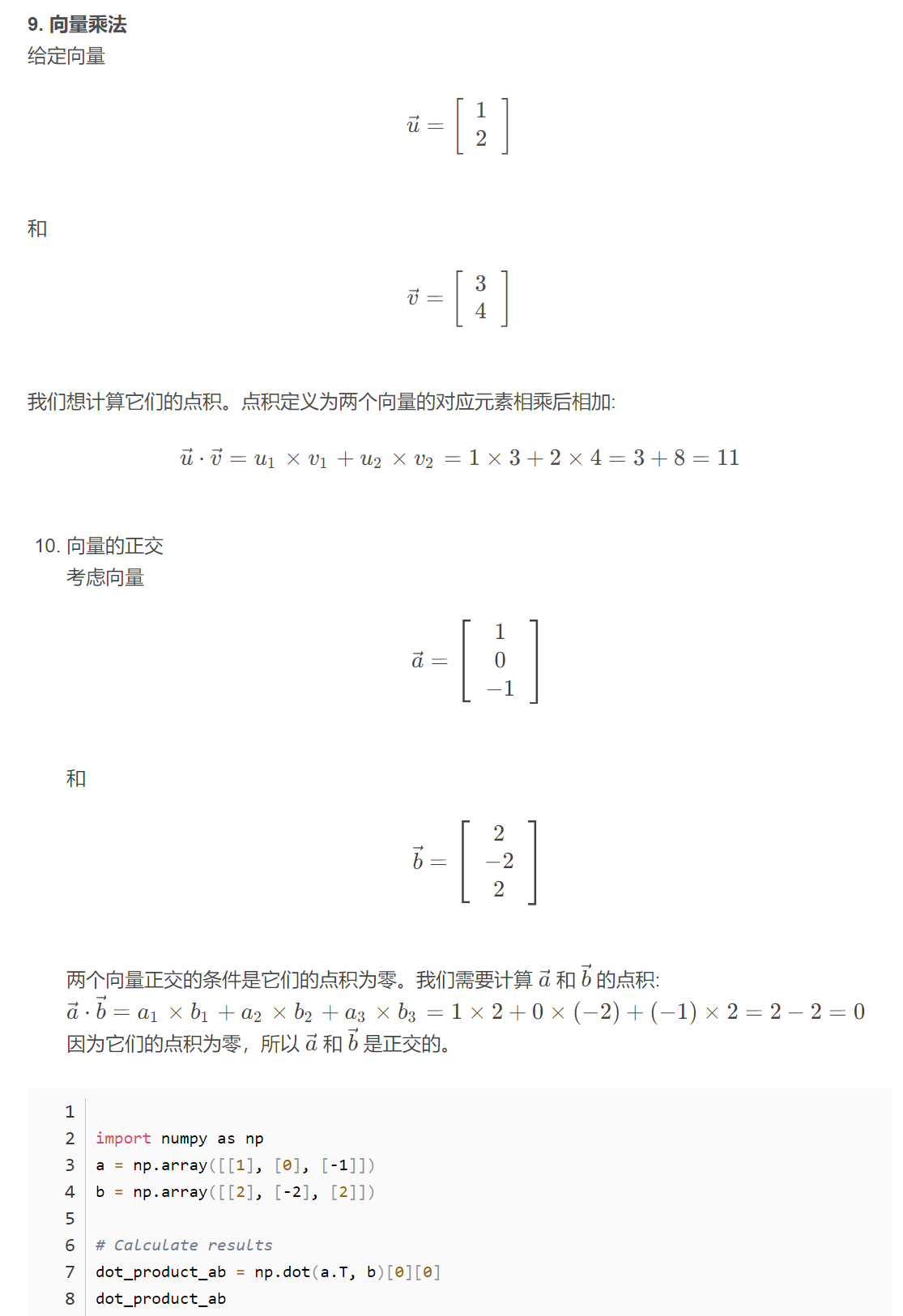




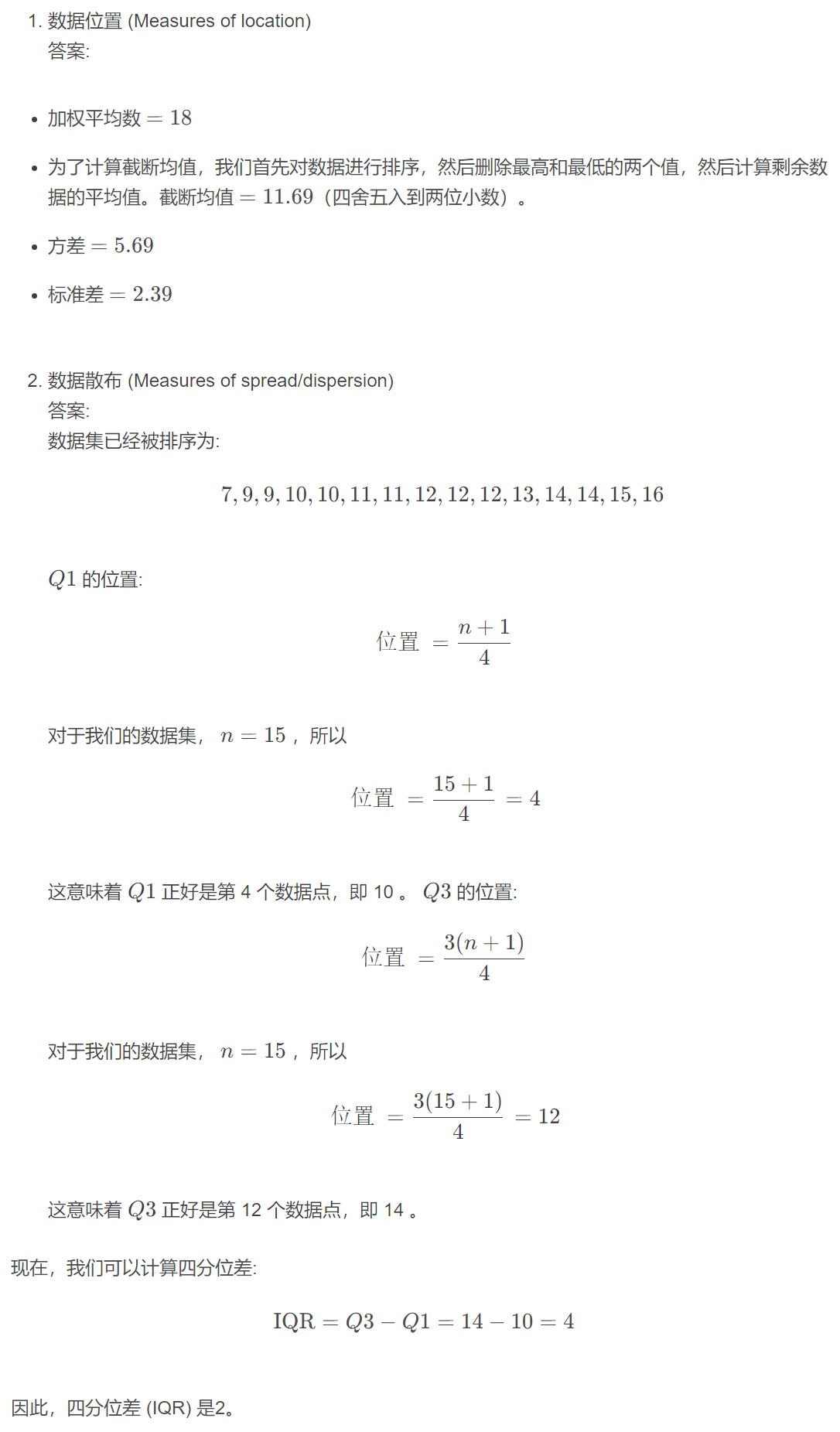


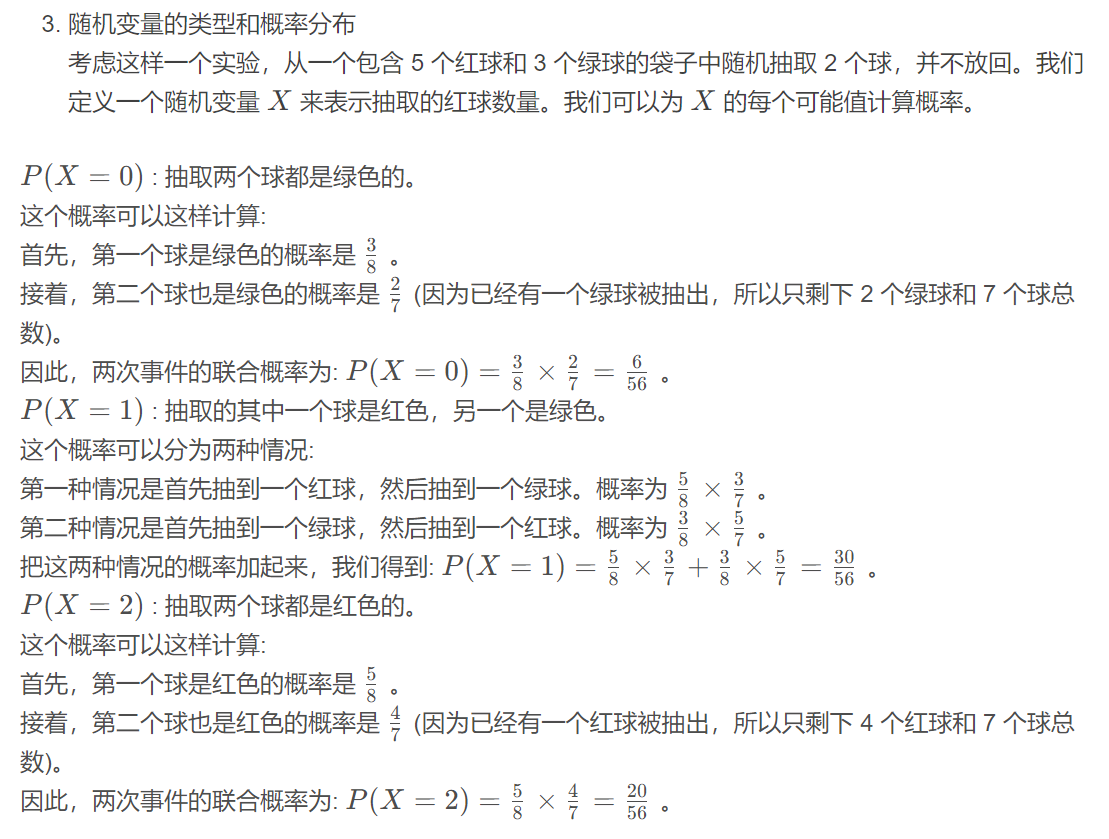


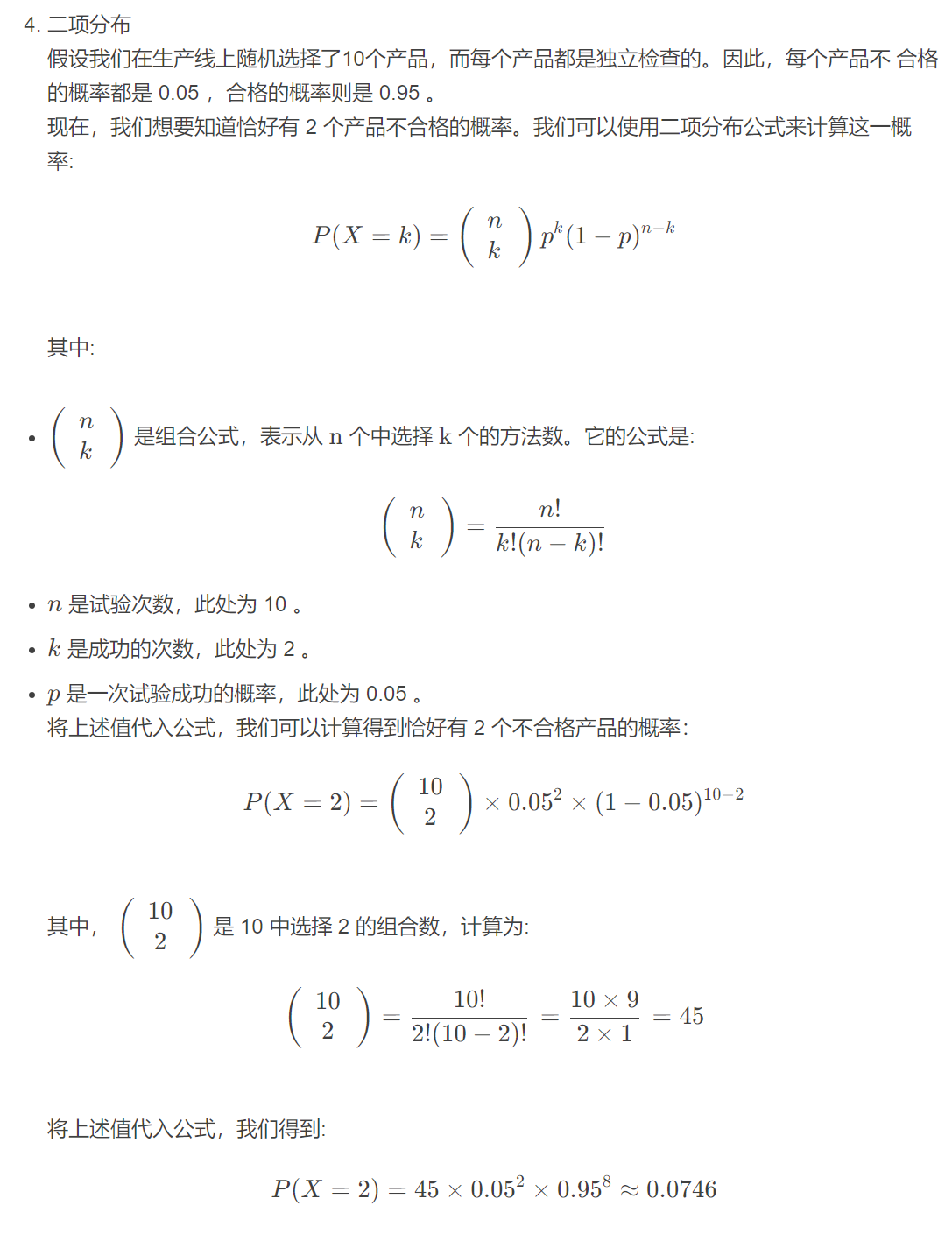


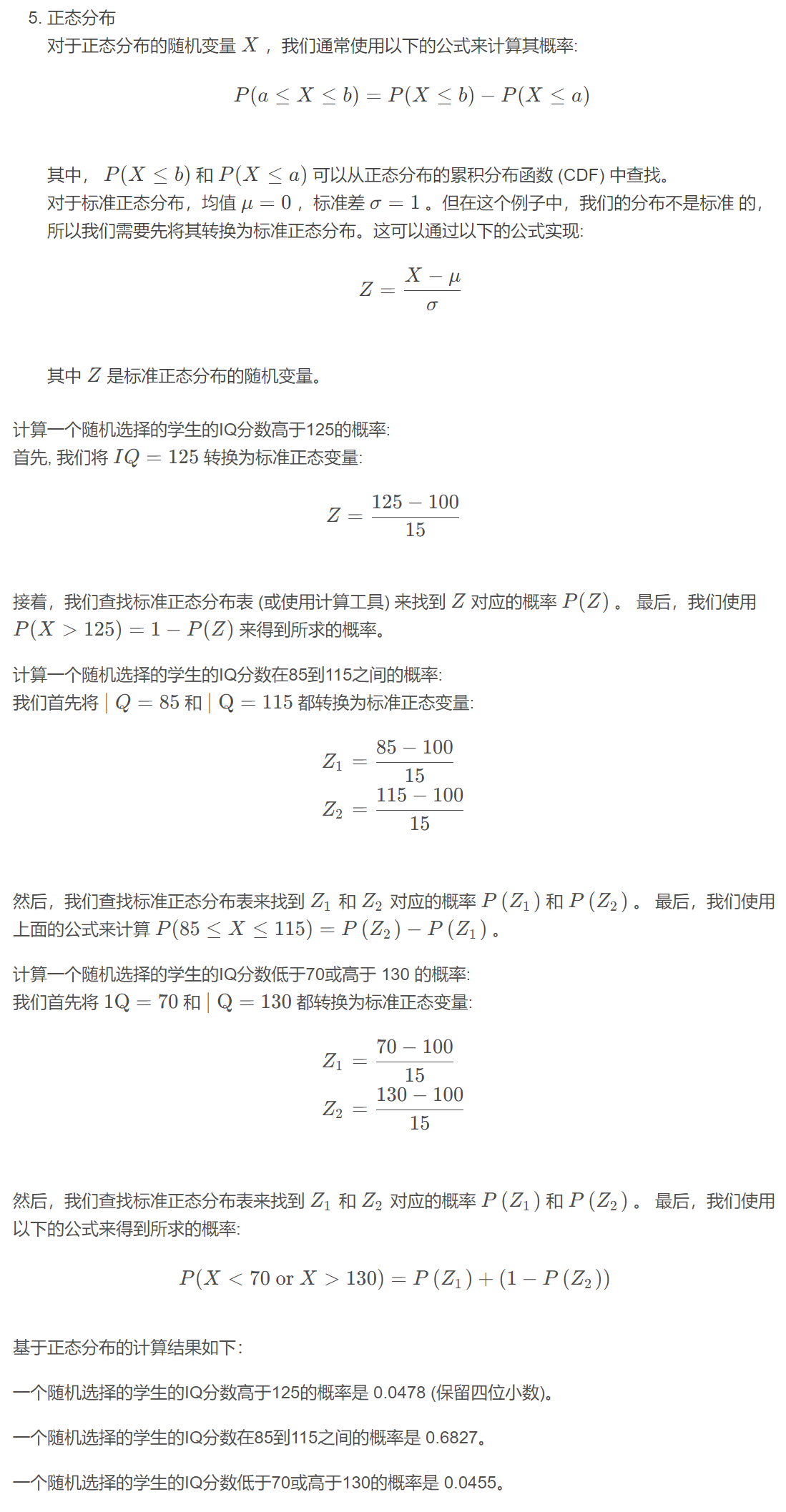


# 概率论









# 统计学

