



Distance

# 距离

人是万物的尺度



两点之间最短的路径是一条直线。

*The shortest path between two points is a straight line.*

—— 阿基米德 (Archimedes) | 古希腊数学家、物理学家 | 287 ~ 212 BC



- ▶ matplotlib.pyplot.axhline() 绘制水平线
- ▶ matplotlib.pyplot.axvline() 绘制竖直线
- ▶ matplotlib.pyplot.contour() 绘制等高线图
- ▶ matplotlib.pyplot.contourf() 绘制填充等高线图
- ▶ np.abs() 计算绝对值
- ▶ numpy.meshgrid() 获得网格化数据
- ▶ plot\_wireframe() 绘制三维单色线框图
- ▶ sympy.abc() 引入符号变量
- ▶ sympy.lambdify() 将符号表达式转化为函数
- ▶ scipy.spatial.distance.mahalanobis() 计算欧氏距离

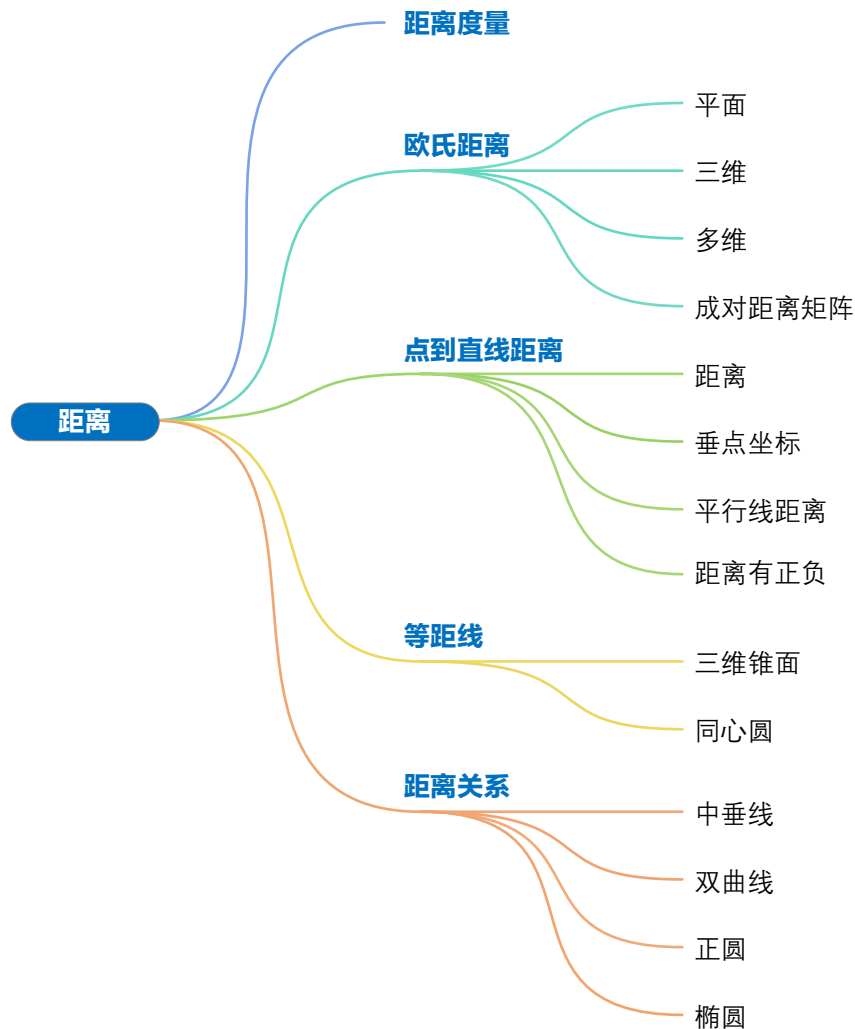
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

## 7.1 距离：未必是两点间最短线段

阿基米德说“两点之间最短的路径是一条线段”，这个道理看似再简单不过；扔个肉包子给狗，狗会径直冲向包子；它应该不会拐几个弯，跑出优美曲线给你看。

但是，哪怕最基本的生活经验也会告诉我们，两点之间最短的路径不能简单地用两点线段来描述。图 1 和图 2 所示的四个路径规划就很好地说明这一点。

如果城市街区以正方形方格规划，如图 1 (a) 所示，从  $A$  点到  $B$  点，有很多路径可以选择；但是不管怎么选择路径，会发现这些路径都是由横平竖直的线段组合而成，而不是简单的“两点一线”。

图 1 (b) 所示，若城市的街区都是整齐的平行四边形，那么从  $A$  点到  $B$  点的路径就要依照四边形的走势来规划；尽管如此规划得到的路径依然是直线段的组合。

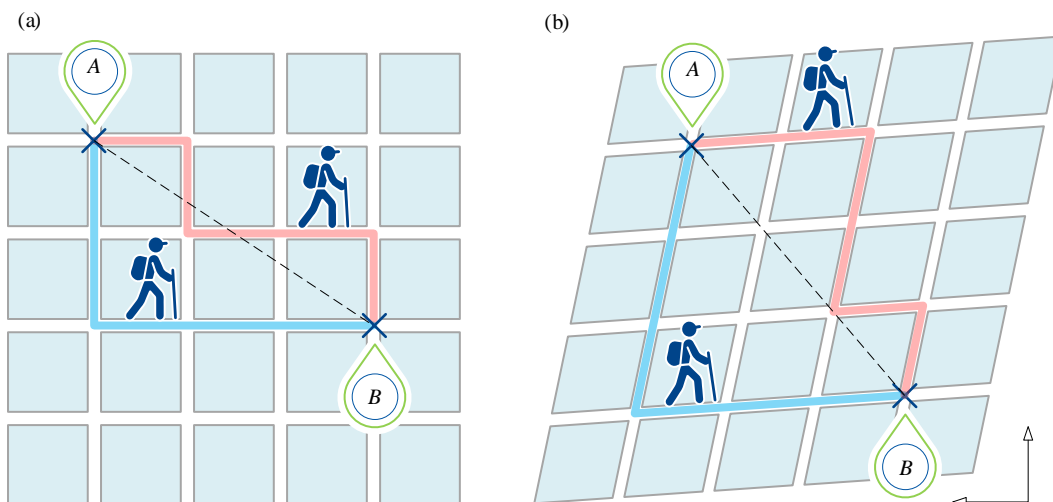


图 1. 两种直线段路径规划

有些城市街区的布置类似极坐标，呈现放射性网状，如图 2 (a) 中，从  $A$  点到  $B$  点的路径，便是直线段和弧线段的结合。

更常见的情况是，路径可能是由不规则曲线、折线构造得到。如图 2 (b) 所示，从  $A$  点到  $B$  点，别无选择只能按照一段自由曲线行走。

上述路径规划还是在平面上，这是一种高度理想化的情况；实际情况中度量“距离”要复杂得多。如图 3 所示，计算地球上相隔很远的两个大陆上两点的距离要考虑的是一段弧线的长度。

让我们再增加一些复杂性，考虑山势起伏，具体如图 4 所示。这时，规划  $A$  点到  $B$  点的路径难度便再提高。

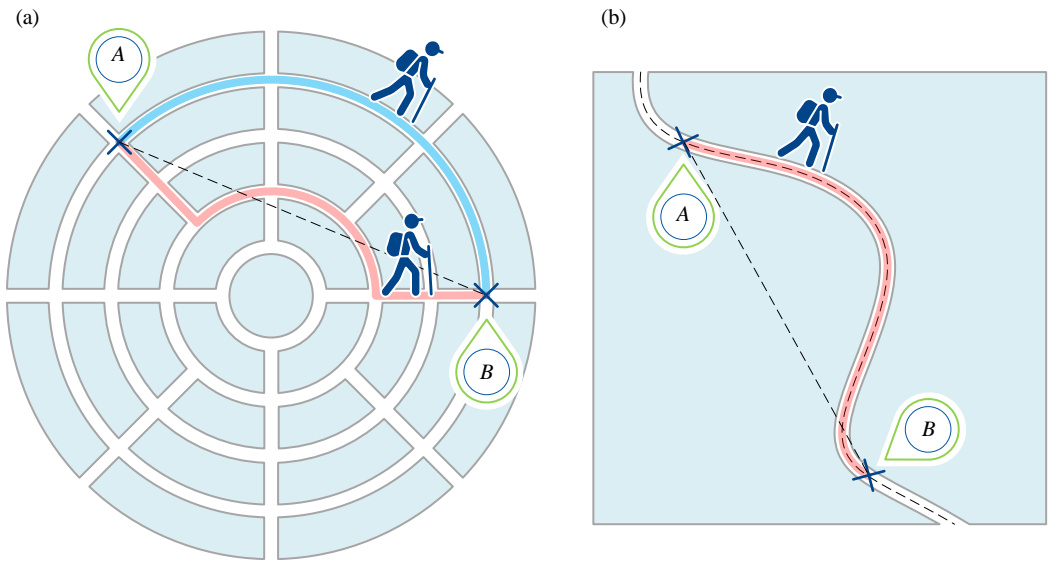


图 2. 两种非线性路径规划

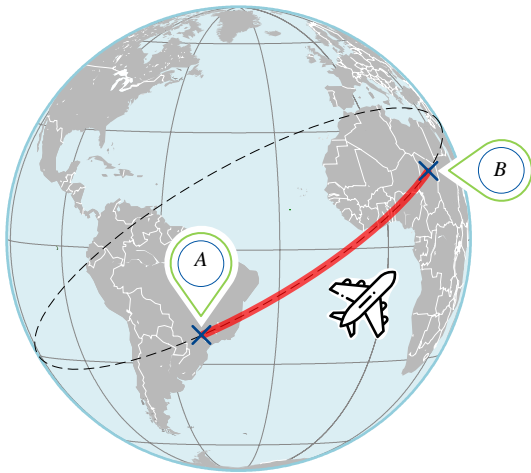


图 3. 地球表面两点距离

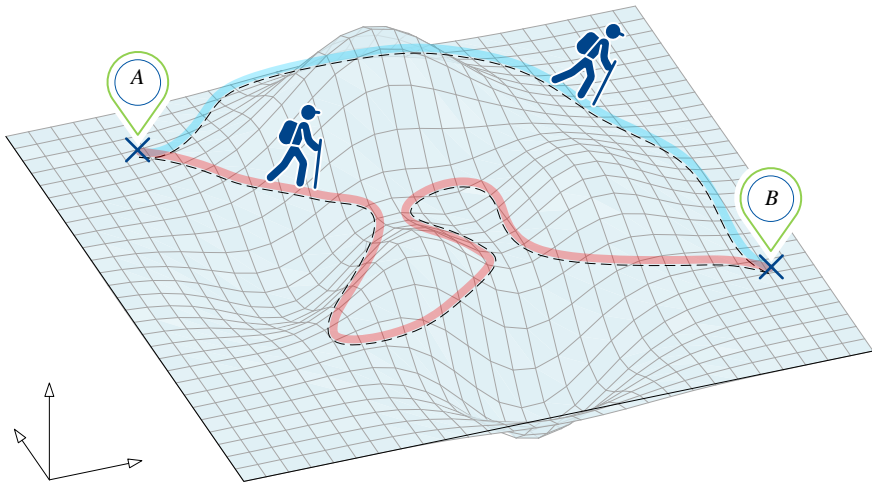


图 4. 考虑山势起伏的路径规划

此外，计算距离时还可以考虑数据的分布因素，得到的距离即所谓的**统计距离** (statistical distance)。

如图 5 所示， $A$ 、 $B$ 、 $C$  和  $D$  四点和  $Q$  点的欧氏距离相同；图中蓝色散点是样本数据的分布，考虑数据分布“紧密”情况，不难判断  $C$  点距离  $Q$  最近，而  $D$  距离  $Q$  最远。

也就是说，地理上的相近，不代表关系的靠近——相隔万里的好友，近在咫尺的路人。

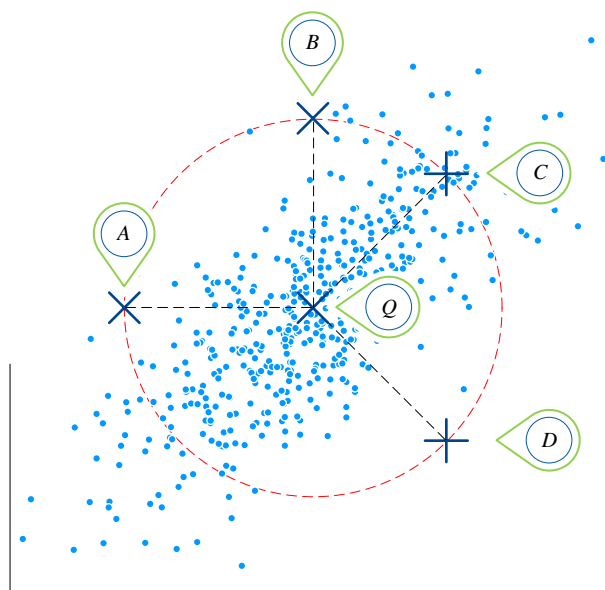


图 5. 考虑数据分布的距离度量

“距离”在数据科学和机器学习中非常重要。本章先从最简单的两点直线距离入手，和大家探讨距离这个话题；本系列丛书后续会不断扩展丰富“距离”这个概念。

## 7.2 欧氏距离：两点间最短线段

**欧几里得距离** (Euclidean distance)，或**欧氏距离**，是最简单的距离度量。

本书前文讲过的绝对值实际上就是一维数轴上的两点距离。如图 6 所示，一维数轴上有  $A$  和  $B$  两点，它们的坐标值分别为  $x_A$  和  $x_B$ ； $A$  和  $B$  两点欧氏距离就是  $x_A$  和  $x_B$  之差的绝对值。

$$\text{dist}(A, B) = |x_A - x_B| \quad (1)$$

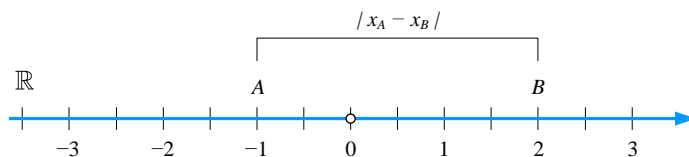


图 6. 实数轴上 A 和 B 距离

## 二维平面

平面直角坐标系中两点  $A(x_A, y_A)$  和  $B(x_B, y_B)$  的欧氏距离就是  $AB$  线段的长度，可以通过如下公式求得。

$$\begin{aligned} \text{dist}(A, B) &= \sqrt{|x_A - x_B|^2 + |y_A - y_B|^2} \\ &= \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \end{aligned} \quad (2)$$

上式用到的数学工具就是勾股定理；如图 7 所示，直角三角形的两个直角边边长为  $|x_A - x_B|$  和  $|y_A - y_B|$ 。

A 和 B 点连线的**中点** (midpoint)  $M$  的坐标为。

$$M = \left( \frac{x_A + x_B}{2}, \frac{y_A + y_B}{2} \right) \quad (3)$$

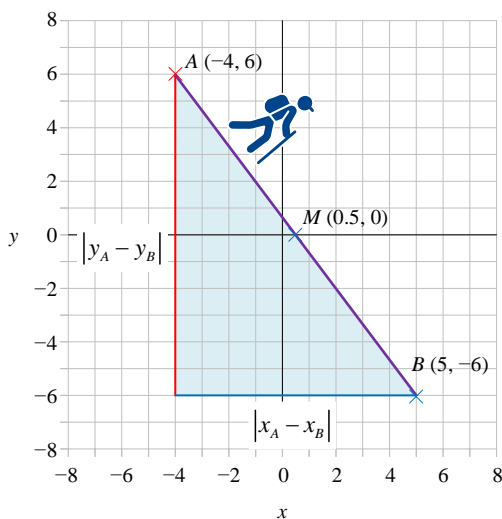
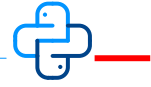


图 7. 平面直角坐标系，两点之间距离

以下代码绘制图 7。



```
# Bk Ch7 01

import matplotlib.pyplot as plt
import numpy as np

x_A = -4
y_A = 6

x_B = 5
y_B = -6

x_M = (x_A + x_B)/2
y_M = (y_A + y_B)/2

fig, ax = plt.subplots()

plt.plot(x_A, y_A, 'x', markersize = 12)
plt.plot(x_B, y_B, 'x', markersize = 12)

x_points = [x_A, x_B, x_M]
y_points = [y_A, y_B, y_M]

plt.plot(x_points, y_points)
plt.plot(x_points, y_points)

labels = ['A', 'B', 'M']

plt.plot(x_points, y_points, 'x')

for label, i, j in zip(labels, x_points, y_points):
    plt.text(i, j+0.5, label + ' ({} , {})'.format(i, j))

plt.xlabel('x')
plt.ylabel('y')
plt.axhline(y=0, color='k', linestyle='--')
plt.axvline(x=0, color='k', linestyle='--')
plt.xticks(np.arange(-8, 8 + 1, step=1))
plt.yticks(np.arange(-8, 8 + 1, step=1))
plt.axis('scaled')

ax.set_xlim(-8,8)
ax.set_ylim(-8,8)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)

ax.grid(linestyle='--', linewidth=0.25, color=[0.5,0.5,0.5])
```

### 三维空间

类似地，三维直角坐标系中两点  $A(x_A, y_A, z_A)$  和  $B(x_B, y_B, z_B)$  的距离可以通过如下公式求得。

$$\text{dist}(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2} \quad (4)$$

图 8 给出一个计算三维空间两点欧氏距离的例子；容易发现，计算过程两次使用勾股定理。

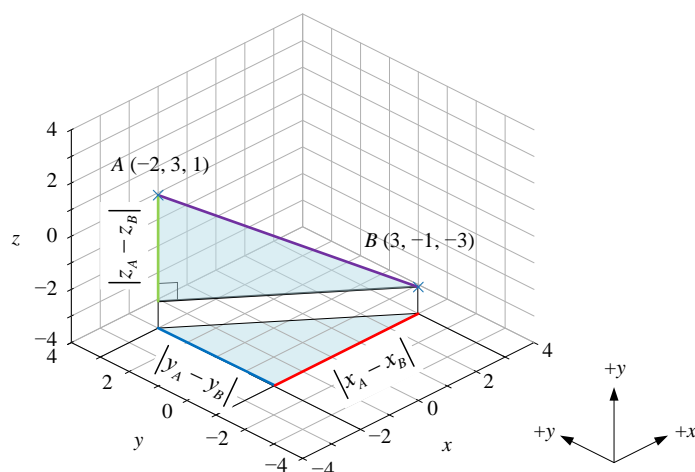


图 8. 三维直角坐标系，两点之间距离

我们也可以把 (4) 推广到多维； $D$  维空间两点  $A(x_{1,A}, x_{2,A}, \dots, x_{D,A})$  和  $B(x_{1,B}, x_{2,B}, \dots, x_{D,B})$  的距离可以通过如下公式求得。

$$\text{dist}(A, B) = \sqrt{(x_{1,A} - x_{1,B})^2 + (x_{2,A} - x_{2,B})^2 + \dots + (x_{D,A} - x_{D,B})^2} \quad (5)$$

以下代码绘制图 8。



```
# Bk_Ch7_02

import numpy as np
import matplotlib.pyplot as plt

# point A
x_A = -2
y_A = 3
z_A = 1

# point B
x_B = 3
y_B = -1
z_B = -3

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot([x_A, x_B], [y_A, y_B], [z_A, z_B], 'x', linestyle = '-')

plt.show()
ax.set_proj_type('ortho')

ax.set_xlim(-4, 4)
ax.set_ylim(-4, 4)
ax.set_zlim(-4, 4)

plt.tight_layout()
ax.set_xlabel('$\it{x}$')
ax.set_ylabel('$\it{y}$')
ax.set_zlabel('$\it{z}$')

ax.view_init(azim=-135, elev=30)
```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



```
ax.xaxis._axinfo["grid"].update({"linewidth":0.25, "linestyle" : ":"})
ax.yaxis._axinfo["grid"].update({"linewidth":0.25, "linestyle" : ":"})
ax.zaxis._axinfo["grid"].update({"linewidth":0.25, "linestyle" : ":"})
```

## 成对距离

在数据科学和机器学习实践中，我们经常遇到如图9所示这种多点**成对距离** (pairwise distance) 的情况。在图9这个平面上，一共有12个点；而这12点一共可以构造得到66 ( $C_{12}^2$ ) 个两点距离。

以什么结构存储、运算及展示这些距离值，成了一个问题。

这时，矩阵就可以派上大用场。

如图10所示，矩阵的形状为  $12 \times 12$ ，即12行、12列。矩阵的主对角线元素都是0，这是某点和自身的距离；矩阵非主对角线元素则代表成对距离。

很容易发现，这个矩阵关于主对角线对称；也就是说，我们只需要主对角线斜下方的66个元素，或者主对角线斜上方的66元素。这66个元素就是我们要保存的所有两点距离。

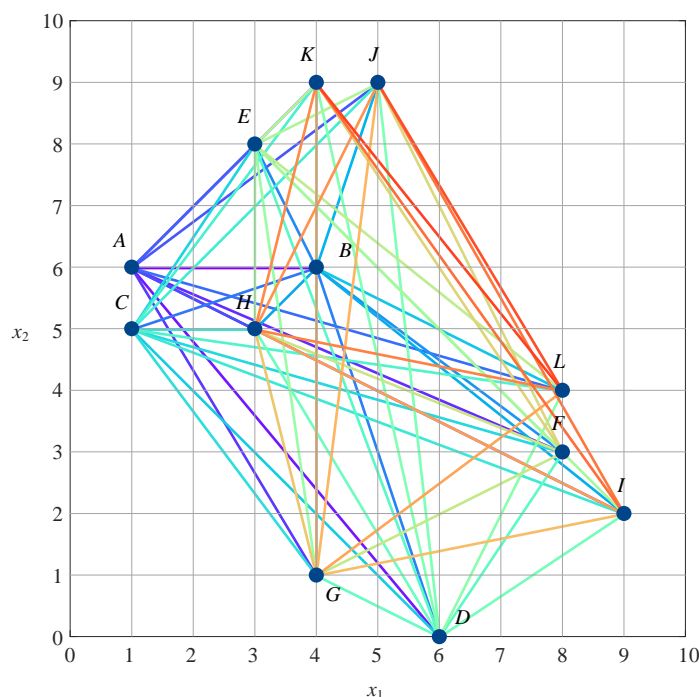


图9. 平面上12个点，成对距离

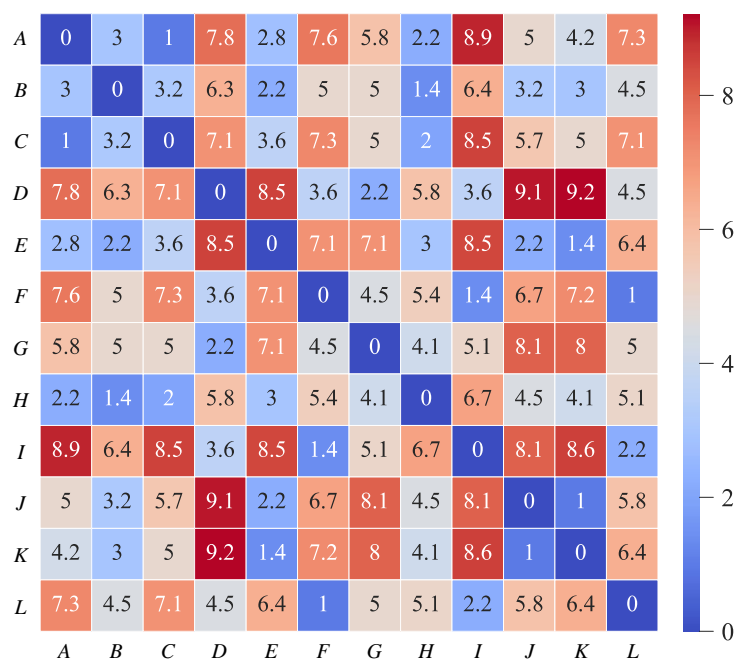


图 10. 成对距离矩阵

多讲一点，提取类似图 10 方阵中主对角线及其左下方元素，它们单独构成的矩阵叫做**下三角矩阵** (lower triangular matrix)  $L$ ；注意，矩阵其余元素为 0。

而主对角线和其右上方元素单独构成的矩阵叫做**上三角矩阵** (upper triangular matrix)  $U$ ，其余元素为 0。图 11 给出下三角矩阵和上三角矩阵的示意图；下三角矩阵  $L$  转置得到的  $L^T$  便是上三角矩阵  $U$ 。



图 11. 下三角矩阵和上三角矩阵



机器学习很多算法中常常用到亲密度 (affinity)；亲密度和距离正好相反。两点距离越远，两者亲密度越低；而当它们距离越近，亲密度则越高。

我们假设亲近度的取值范围在  $[0, 1]$  这个区间；亲近度为 1 代表两点重合，也就是距离为 0，亲近度最大；亲近度为 0 代表两点相距无穷远。

摆在我们面前的一个数学问题就是，如何把距离转化成亲近度。如图 12 所示，我们需要某种“映射”关系，将“欧氏距离”和“亲密度”一一联系起来。

自然而然地，我们就会想到代数中的“函数”；函数就是映射，完成数据转换。

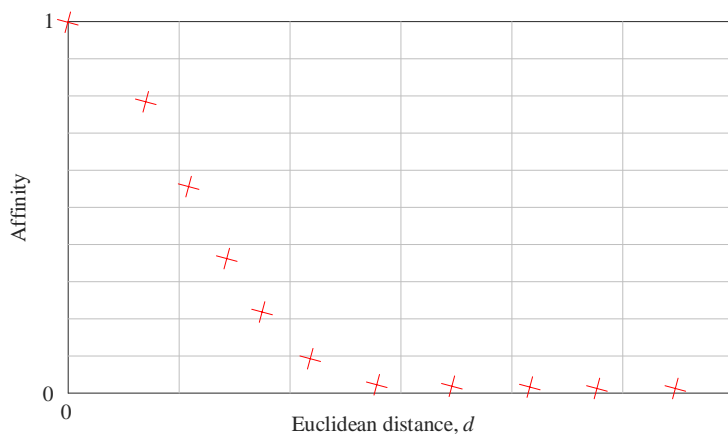


图 12. 如何设计“欧氏距离”到“亲密度”的映射关系

而众多函数当中，高斯函数便可以胜任这一映射要求。高斯函数的一般解析式为。

$$f(d) = \exp(-\gamma d^2) \quad (6)$$

图 13 所示为参数  $\gamma$  影响高斯函数右半侧曲线形状。本书后续会介绍高斯函数性质；本系列丛书在《机器学习》一册会专门介绍“亲近度”这个概念。

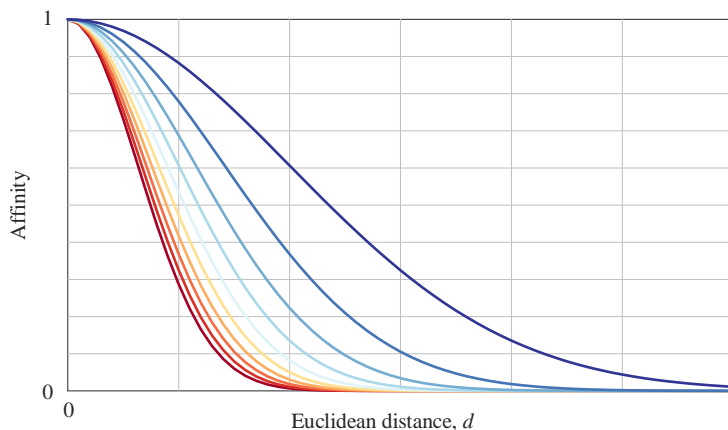
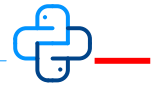


图 13. 参数  $\gamma$  影响高斯函数右半侧曲线形状

以下代码计算成对距离，并且绘制图 9 和图 10。



```
# Bk Ch7_03

import matplotlib.pyplot as plt
import itertools
from sklearn.metrics.pairwise import euclidean_distances
import numpy as np
import seaborn as sns
from matplotlib import cm

# define sample data
X = np.array([[1,6], [4,6], [1,5], [6,0],
              [3,8], [8,3], [4,1], [3,5],
              [9, 2], [5, 9], [4, 9], [8, 4]])

# define labels
labels = ['a','b','c','d','e','f','g','h','i','j','k','l']

colors = plt.cm.rainbow(np.linspace(0,1,int(len(X)*len(X)/2)))

fig, ax = plt.subplots()

for i, d in enumerate(itertools.combinations(X, 2)):
    plt.plot([d[0][0],d[1][0]], [d[0][1],d[1][1]], color = colors[i,:])

# plot scatter of sample data
plt.scatter(x=X[:, 0], y=X[:, 1], color=np.array([0, 68, 138])/255., alpha=1.0,
           linewidth = 1, edgecolor=[1,1,1])

for i, (x,y) in enumerate(zip(X[:, 0],X[:, 1])):
    # add labels to the sample data
    label = labels[i] + f"({x},{y})"

    plt.annotate(label, # text
                 (x,y), # point to label
                 textcoords="offset points",
                 xytext=(0,10),
                 # distance from text to points (x,y)
                 ha='center')
    # horizontal alignment center

ax.set_xticks(np.arange(0, 11, 1))
ax.set_yticks(np.arange(0, 11, 1))
ax.set_xlim(0, 10)
ax.set_ylim(0, 10)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
ax.grid(linestyle='--', linewidth=0.25, color=[0.5,0.5,0.5])
ax.set_aspect('equal')
plt.show()

%% compute pairwise distance matrix

Pairwise_d = euclidean_distances(X)

fig, ax = plt.subplots()

h = sns.heatmap(Pairwise_d, cmap="coolwarm",
               square=True, linewidths=.05, annot=True,
               xticklabels = labels, yticklabels = labels)
```

## 7.3 点到直线的距离

给定平面上一条直线  $l: ax + by + c = 0$ ，直线外一点  $A(x_A, y_A)$  到该直线的距离为。

$$\text{dist}(A, l) = \frac{|ax_A + by_A + c|}{\sqrt{a^2 + b^2}} \quad (7)$$

直线  $l$  上距离  $A$  最近点的坐标为  $H(x_H, y_H)$ 。

$$\begin{aligned} x_H &= \frac{b(bx_A - ay_A) - ac}{a^2 + b^2} \\ y_H &= \frac{a(-bx_A + ay_A) - bc}{a^2 + b^2} \end{aligned} \quad (8)$$

特别地，当  $a = 0$  时，直线  $l$  为水平线。  $A(x_A, y_A)$  到该直线的距离为。

$$\text{dist}(A, l) = \frac{|by_A + c|}{|b|} \quad (9)$$

当  $b = 0$  时，直线  $l$  为竖直线。  $A(x_A, y_A)$  到该直线的距离为。

$$\text{dist}(A, l) = \frac{|ax_A + c|}{|a|} \quad (10)$$

举个例子，图 14 给定直线  $x - 2y - 4 = 0$ ， $A(-4, 6)$  到直线距离最近点为  $H(0, -2)$ ； $A$  到直线的距离为 8.944。

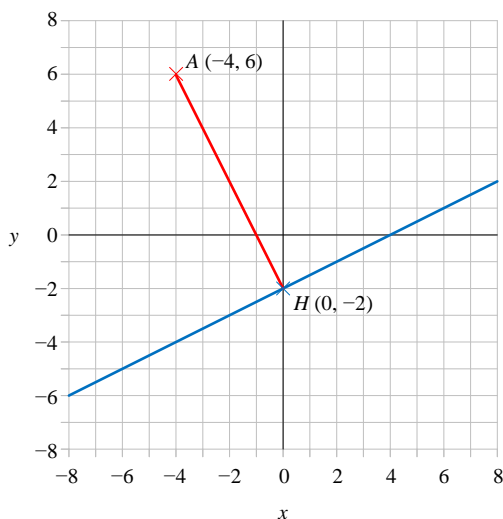
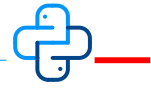


图 14. 平面直角坐标系，点到直线距离

以下代码计算点 A 到直线距离，并绘制图 14。



```
# Bk Ch7 04

import matplotlib.pyplot as plt
import numpy as np

def dist(a, b, c, x_A, y_A):

    dist_AH = np.abs(a*x_A + b*y_A + c)/np.sqrt(a**2 + b**2)

    x_H = (b*(b*x_A - a*y_A) - a*c)/(a**2 + b**2)
    y_H = (a*(-b*x_A + a*y_A) - b*c)/(a**2 + b**2)
    return x_H, y_H, dist_AH

x_A = -4
y_A = 6

a = 1
b = -2
c = -4

x_array = np.linspace(-8,8,10)
y_array = -a/b*x_array - c/b

x_H, y_H, dist_AH = dist(a, b, c, x_A, y_A)

fig, ax = plt.subplots()

plt.plot(x_array, y_array)

plt.plot(x_A, y_A, 'x', markersize = 12)

plt.plot(x_H, y_H, 'x', markersize = 12)

x_points = [x_A, x_H]
y_points = [y_A, y_H]

plt.plot(x_points, y_points)

labels = ['A', 'H']

plt.plot(x_points, y_points, 'x')

for label, i, j in zip(labels, x_points, y_points):
    plt.text(i, j+0.5, label + ' ({} , {})'.format(i, j))

plt.xlabel('x')
plt.ylabel('y')
plt.axhline(y=0, color='k', linestyle='-')
plt.axvline(x=0, color='k', linestyle='-')
plt.xticks(np.arange(-8, 8 + 1, step=1))
plt.yticks(np.arange(-8, 8 + 1, step=1))
plt.axis('scaled')

ax.set_xlim(-8,8)
ax.set_ylim(-8,8)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)

ax.grid(linestyle='--', linewidth=0.25, color=[0.5,0.5,0.5])
```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

## 平行线间距离

给定如下两条平行线  $l_1$  和  $l_2$ ,

$$\begin{cases} ax + by + c_1 = 0 \\ ax + by + c_2 = 0 \end{cases} \quad (11)$$

其中,  $c_1 \neq c_2$ 。

这两条平行线的距离为。

$$\text{dist}(l_1, l_2) = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}} \quad (12)$$

在本系列丛书《矩阵力量》一册, 我们会利用线性代数运算工具来求解点到直线距离, 及两条平行线之间的距离。

## 距离也可以有“正负”

本章前文介绍的距离都是“非负值”; 但是, 在机器学习算法中, 我们经常会给距离度量加个正负号。下面举几个例子。

如图 15 所示, 以  $Q$  点作为比较的基准点, 距离  $AQ$  和  $BQ$  的定义分别为。

$$\begin{aligned} \text{dist}(A, Q) &= |x_A - x_Q| \\ \text{dist}(B, Q) &= |x_B - x_Q| \end{aligned} \quad (13)$$

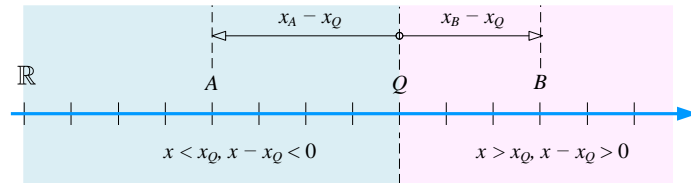


图 15. 一维数轴上距离的正负

将上两式中绝对值去掉, 得到。

$$\begin{aligned} \text{dist}(A, Q) &= x_A - x_Q \\ \text{dist}(B, Q) &= x_B - x_Q \end{aligned} \quad (14)$$

图 15 中,  $A$  在  $Q$  的左边, 因此  $x_A - x_Q < 0$ , 也就是距离为“负”; 而  $B$  在  $Q$  的右边, 因此  $x_B - x_Q > 0$ , 也就是距离为“正”。

距离的绝对值告诉我们两点的远近, 距离的“正负”符号多了相对位置这层信息。

上一章介绍的不等式有划定区域作用；也就是说，(14) 这种含“正负”的距离把不等式“区域”这层信息也囊括进来。

同理，将 (7) 分子上的绝对值符号去掉，点  $A$  和直线  $l$  的距离为。

$$\text{dist}(A, l) = \frac{ax_A + by_A + c}{\sqrt{a^2 + b^2}} \quad (15)$$

以图 16 为例，图中直线  $l$  的解析式为  $x + y - 1 = 0$ ；这条直线把平面直角坐标系划分成两个区域—— $x + y - 1 > 0$  (暖色背景) 和  $x + y - 1 < 0$  (冷色背景)。

根据 (15)，计算  $A$  和  $B$  点到直线  $l$  的距离分别为。

$$\text{dist}(A, l) = \frac{3}{\sqrt{2}}, \quad \text{dist}(B, l) = \frac{-5}{\sqrt{2}} \quad (16)$$

根据距离的“正负”符号，可以判断  $A$  点在  $x + y - 1 > 0$  这个区域， $B$  点在  $x + y - 1 < 0$  这个区域。

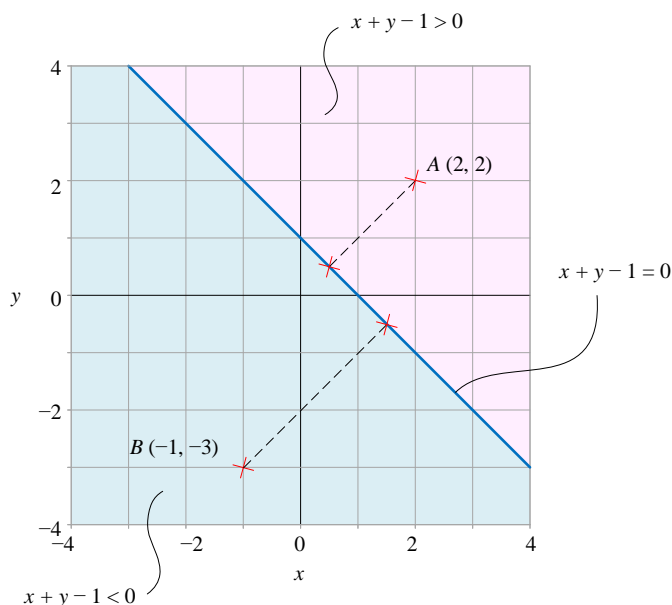


图 16. 点到直线距离的正负

请大家思考去掉 (12) 分子中绝对值符号后，两条平行线距离分别为正负值所代表的几何含义。



支持向量机 (Support Vector Machine, SVM) 是非常经典的机器学习算法之一。支持向量机既可以用来分类，也可以用来处理回归问题。图 17 所示为支持向量机核心思路。



如图 17 所示，一片湖面左右散布着蓝色 ● 红色 ● 礁石，游戏规则是，皮划艇以直线路径穿越水道，保证船身恰好紧贴礁石；寻找一条路径，让该路径通过的皮划艇宽度最大，也就是图 17 中两条虚线之间宽度最大。这个宽度叫做间隔 (margin)。

图 17 (b) 中加黑圈 ○ 的五个点，就是所谓的支持向量 (support vector)。图 17 中深蓝色线就是水道，叫做决策边界 (decision boundary)。决策边界将标签分别为蓝色 ● 红色 ● 数据点“一分为二”，也就是分类。

很明显，图 17 (b) 中规划的路径好于图 17 (a)。而本节介绍的“距离”这个概念在支持向量机算法中扮演重要角色。

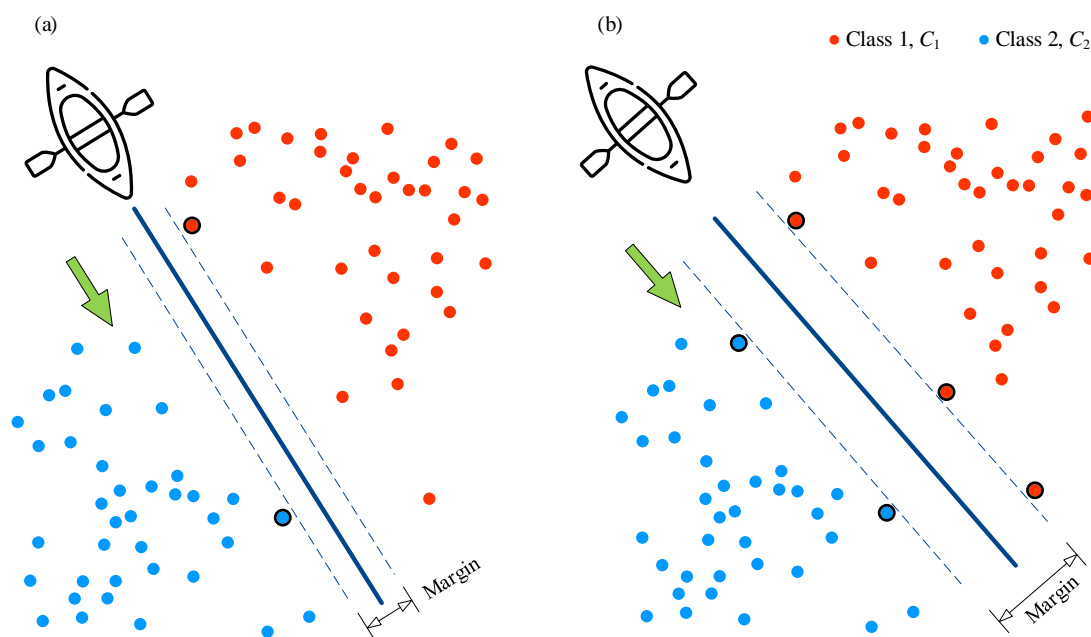


图 17. 支持向量机原理

如图 18 所示，计算“支持向量” A、B、C、D、E 和“决策边界”的距离，用到的就是本节讲到的“点到直线距离”；计算  $l_1$  和  $l_2$  “间隔”宽度用到的是“平行线间距离”

而图 18 中暖色和冷色两个区域就是通过不等式划定的区域。暖色区域的样本点分类为红色 ●，即  $C_1$ ；冷色区域的样本点分类为蓝色 ●，即  $C_2$ 。

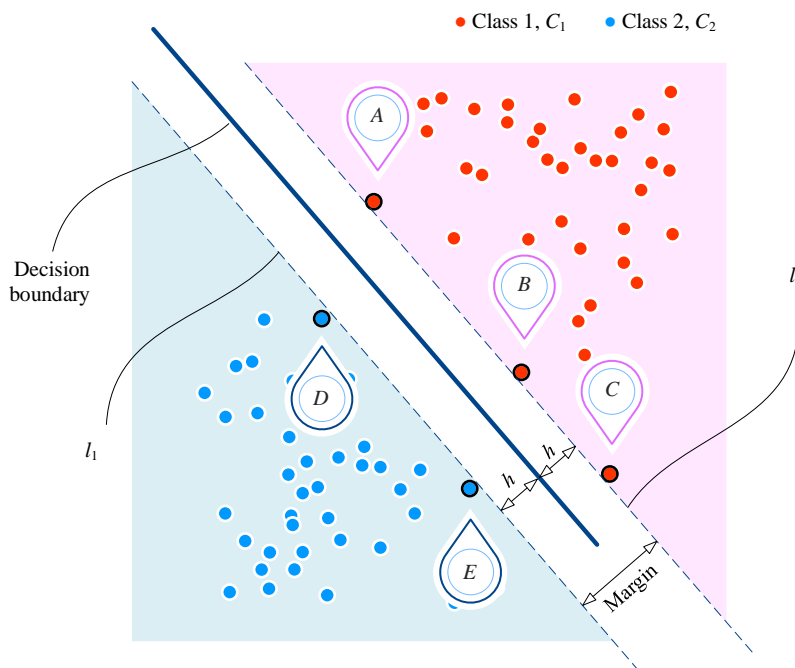


图 18. “距离”在 SVM 算法中扮演的角色

## 7.4 等距线：换个视角看距离

任意一点  $P(x, y)$  距离原点  $O$  的欧氏距离为  $r$ ，对应的解析式为。

$$\text{dist}(P, O) = \sqrt{x^2 + y^2} = r \quad (17)$$

上式左右两侧平方得到下式。

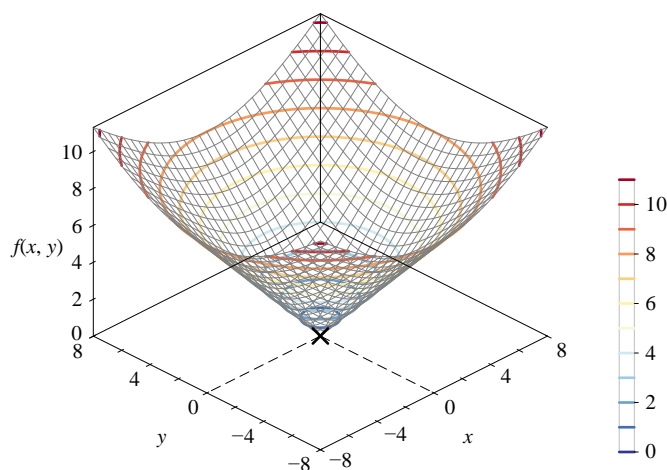
$$x^2 + y^2 = r^2 \quad (18)$$

这样，我们得到一个圆心位于原点、半径为  $r$  的正圆的解析式。

构造如下二元函数  $f(x, y)$ ； $x$  和  $y$  为自变量。

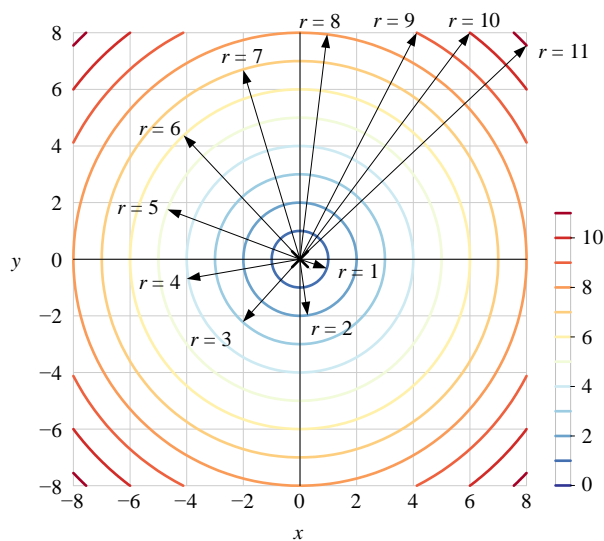
$$f(x, y) = \sqrt{x^2 + y^2} \quad (19)$$

图 19 所示为  $f(x, y)$  在三维直角坐标系的曲面形状；这个曲面为一个圆锥。曲面上我们还绘制了等高线 (contour line 或 contour)。上一章介绍过，等高线指的是  $f(x, y)$  上值相等的相邻各点所连成的曲线。

图 19. 三维直角坐标系,  $f(x, y)$  函数曲面

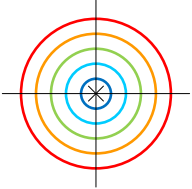
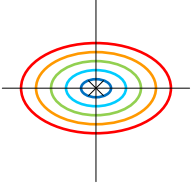
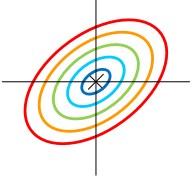
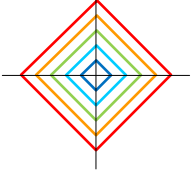
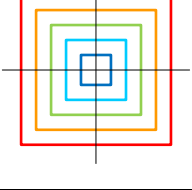
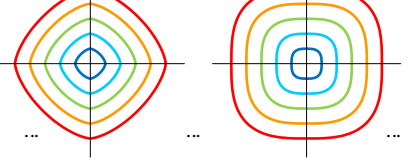
将图 19 等高线投影到  $xy$  平面上, 我们便得到如图 20 所示的平面等高线。图 20 中每条等高线对应的就是  $f(x, y) = r$  截面图像;  $r$  取不同值时对应一系列同心圆。

也就是说, 距离原点  $O$  的欧氏距离取不同值时, “等距线”是一系列同心圆。

图 20.  $f(x, y)$  函数平面等高线, 也就是欧氏距离等距线

再次强敌, 在机器学习中, 欧氏距离是最基础的距离度量; 本系列丛书会和大家一起探讨各种距离度量, 每种距离度量都有自己独特的“等距线”。表 1 总结常见距离度量, 和它们在平面直角坐标系中的等距线形状。本系列丛书将一一揭开表 1 距离度量的面纱。

表 1. 常见距离定义及等距线形状

距离度量	平面直角坐标系中等距线形状
欧氏距离 (Euclidean distance)	
标准化欧氏距离 (standardized Euclidean distance)	
马氏距离 (Mahalanobis distance)	
城市街区距离 (city block distance)	
切比雪夫距离 (Chebyshev distance)	
闵氏距离 (Minkowski distance)	

以下代码绘制图 19 和图 20。请读者修改代码绘制 $f(x, y) = x^2 + y^2$ 这个函数的曲面三维等高线和平面等高线图。



```

import numpy as np
from sympy import lambdify, sqrt
from sympy.abc import x, y
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm

O = [0, 0]

num = 301; # number of mesh grids
x_array = np.linspace(-8,8,num)
y_array = np.linspace(-8,8,num)

xx,yy = np.meshgrid(x_array,y_array)

dist_OP = sqrt((x - O[0])**2 + (y - O[1])**2)

dist_OP_fcn = lambdify([x,y],dist_OP)

dist_OP_zz = dist_OP_fcn(xx,yy)

### mesh plot

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, dist_OP_zz,
                  color = [0.5,0.5,0.5],
                  rstride=10, cstride=10,
                  linewidth = 0.25)

colorbar = ax.contour(xx,yy, dist_OP_zz,
                      levels = np.arange(0,11 + 1),
                      cmap = 'RdYlBu_r')

fig.colorbar(colorbar, ax=ax)

ax.set_proj_type('ortho')

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
plt.tight_layout()
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())

ax.view_init(azim=-135, elev=30)

ax.grid(False)
plt.show()

### contour plot

fig, ax = plt.subplots()

plt.plot(O[0],O[1], color = 'k',
         marker = 'x', markersize = 12)

colorbar = ax.contour(xx,yy, dist_OP_zz,
                      levels = np.arange(0,11 + 1),
                      cmap='RdYlBu_r')

fig.colorbar(colorbar, ax=ax)

plt.xlabel('x')
plt.ylabel('y')
plt.axhline(y=0, color='k', linestyle='-')
plt.axvline(x=0, color='k', linestyle='-')
plt.xticks(np.arange(-10, 10, step=2))
plt.yticks(np.arange(-10, 10, step=2))

```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

```
plt.axis('scaled')
ax.set_xlim(x_array.min(), x_array.max())
ax.set_ylim(y_array.min(), y_array.max())
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.grid(linestyle='--', linewidth=0.25, color=[0.8, 0.8, 0.8])
```

## 7.5 距离间的量化关系

在平面直角坐标系，给定  $A$  和  $B$  两点，任意一点  $P$  到点  $A$  和点  $B$  的距离分别为  $AP$  和  $BP$ 。本节讨论  $AP$  和  $BP$  之间存在的一些常见量化关系，以及对应  $P$  的运动轨迹。这一节同时也引出本书下两章有关圆锥曲线的内容。

如果， $AP$  和  $BP$  等距，我们得到的是， $A$  和  $B$  两点的中垂线。

$$AP = BP \quad (20)$$

如图 21 所示， $A$  和  $B$  两点的中垂线垂直于  $AB$  线段，并且将  $AB$  等分。图 21 中的两组等高线，对应的是到  $A$  和  $B$  两点等距线，相同颜色代表相同距离。两组相同颜色等距线的交点显然都在中垂线上。

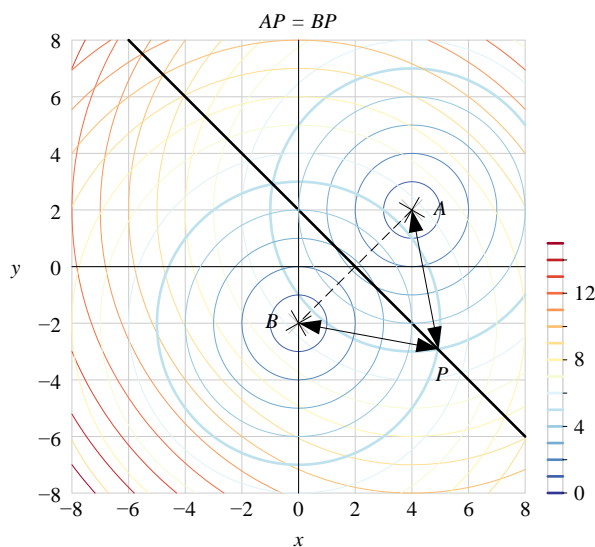


图 21.  $A$  和  $B$  的中垂线

再看一种情况， $AP$  和  $BP$  相差定值。再

$$AP - BP = c \quad (21)$$

比如， $AP$  比  $BP$  长 3，即。

$$AP - BP = 3$$

(22)

如图 22 所示，我们发现满足 (22) 这种数值关系的  $P$  构成了一条双曲线 (hyperbola)。双曲线等圆锥曲线 (conic section) 是本书下两章要介绍的重要内容。

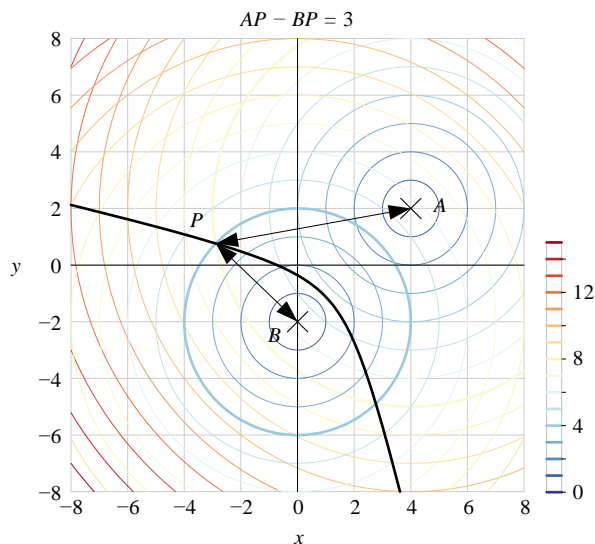


图 22.  $AP - BP = 3$  距离关系构成双曲线左下方一条

将 (22) 中的 3 变成 -3，也就是说  $AP$  比  $BP$  短 3，对应如下等式。

$$AP - BP = -3$$

(23)

图 23 给出的是 (23) 对应的图像，这时候  $P$  的轨迹是双曲线右上方那一条。图 22 和图 23 构成一对完整的双曲线。

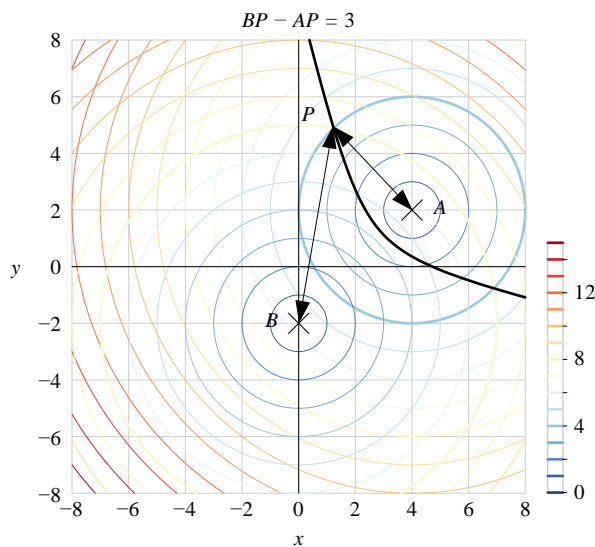


图 23.  $BP - AP = 3$  距离关系构成双曲线右上方一条

若线段  $AP$  和  $BP$  满足倍数关系。

$$AP = c \cdot BP \quad (24)$$

举个例子， $AP$  是  $BP$  的两倍。

$$AP = 2BP \quad (25)$$

如图 24 所示，(25) 中  $P$  轨迹对应的是正圆。有兴趣的读者可以推导这个正圆的解析式。

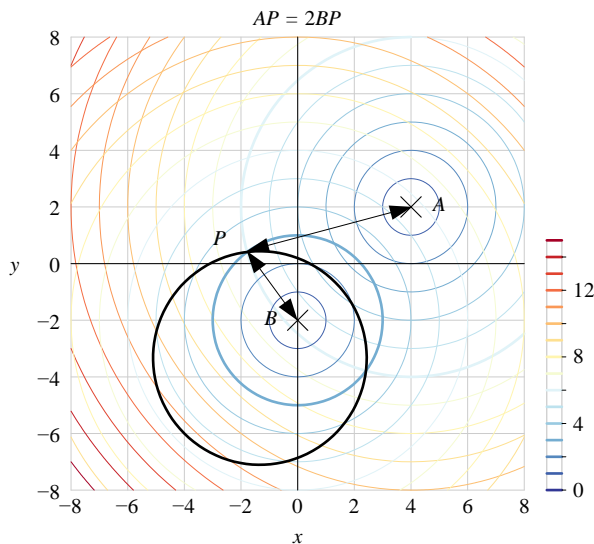


图 24.  $AP = 2BP$  距离关系构成正圆

再看一种情况， $AP$  和  $BP$  之和为定值，即。

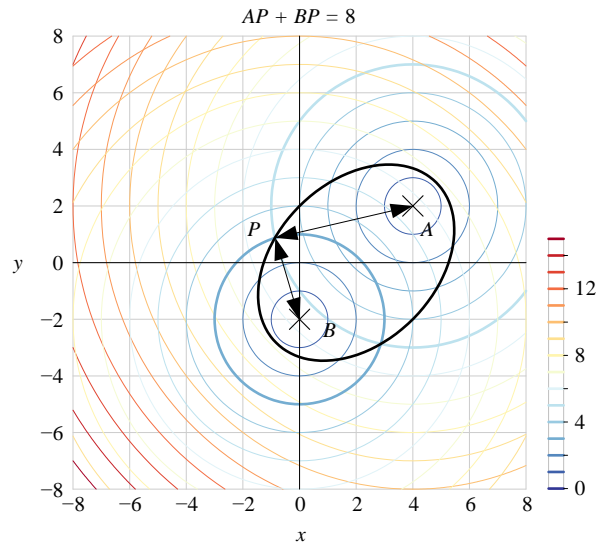
$$AP + BP = c \quad (26)$$

举个例子， $AP$  和  $BP$  之和为 8。

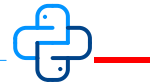
$$AP + BP = 8 \quad (27)$$

(27) 中  $P$  对应的轨迹为一个椭圆 (ellipse)，如图 25 所示。



图 25.  $AP + BP = 8$  距离关系构成椭圆

以下代码绘制图 21 ~ 图 25。



```
# Bk Ch7 06

import numpy as np
from sympy import lambdify, sqrt
from sympy.abc import x, y
import numpy as np
from matplotlib import pyplot as plt

def plot_fcn(A,B,dist AX zz,dist BX zz,distance):

    fig, ax = plt.subplots()

    plt.plot(A[0],A[1], color = 'k',
             marker = 'x', markersize = 12)

    colorbar = ax.contour(xx,yy, dist_AX_zz,
                          levels = np.arange(0,15 + 1),
                          cmap='RdYlBu_r')

    plt.plot(B[0],B[1], color = 'k',
             marker = 'x', markersize = 12)

    colorbar = ax.contour(xx,yy, dist_BX_zz,
                          levels = np.arange(0,15 + 1),
                          cmap='RdYlBu_r')

    ax.contour(xx,yy, distance,
               levels = 0,
               colors = 'k')

    fig.colorbar(colorbar, ax=ax)

    plt.xlabel('x')
    plt.ylabel('y')
```

```

plt.axhline(y=0, color='k', linestyle='-')
plt.axvline(x=0, color='k', linestyle='-')
plt.xticks(np.arange(-10, 10, step=2))
plt.yticks(np.arange(-10, 10, step=2))
plt.axis('scaled')

ax.set_xlim(x_array.min(), x_array.max())
ax.set_ylim(y_array.min(), y_array.max())
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)

ax.grid(linestyle='--', linewidth=0.25, color=[0.8, 0.8, 0.8])

A = [4, 2]
B = [0, -2]

num = 301;
# number of mesh grids
x_array = np.linspace(-8, 8, num)
y_array = np.linspace(-8, 8, num)

xx, yy = np.meshgrid(x_array, y_array)

dist_AX = sqrt((x - A[0])**2 + (y - A[1])**2)
dist_BX = sqrt((x - B[0])**2 + (y - B[1])**2)

dist_AX_fcn = lambdaify([x, y], dist_AX)
dist_BX_fcn = lambdaify([x, y], dist_BX)

dist_AX_zz = dist_AX_fcn(xx, yy)
dist_BX_zz = dist_BX_fcn(xx, yy)

# AX - BX = 0
distance = dist_AX_zz - dist_BX_zz
plot_fcn(A, B, dist_AX_zz, dist_BX_zz, distance)

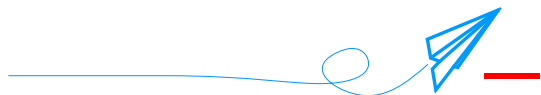
# AX - BX - 3 = 0
distance = dist_AX_zz - dist_BX_zz - 3
plot_fcn(A, B, dist_AX_zz, dist_BX_zz, distance)

# AX - BX + 3 = 0
distance = dist_AX_zz - dist_BX_zz + 3
plot_fcn(A, B, dist_AX_zz, dist_BX_zz, distance)

# AX - 2*BX = 0
distance = dist_AX_zz - 2*dist_BX_zz
plot_fcn(A, B, dist_AX_zz, dist_BX_zz, distance)

# BX + AX - 8 = 0
distance = dist_BX_zz + dist_AX_zz - 8
plot_fcn(A, B, dist_AX_zz, dist_BX_zz, distance)

```



为了更好地度量这些距离，需要读者具备解析几何、线性代数、统计学等知识。随着大家掌握更多数学工具，本系列丛书将慢慢揭开各种距离度量的面纱，以及它们在数据科学、机器学习中的重要作用。