

24

Fundamentals of Linear Algebra

鸡兔同笼 2

之线性回归风暴



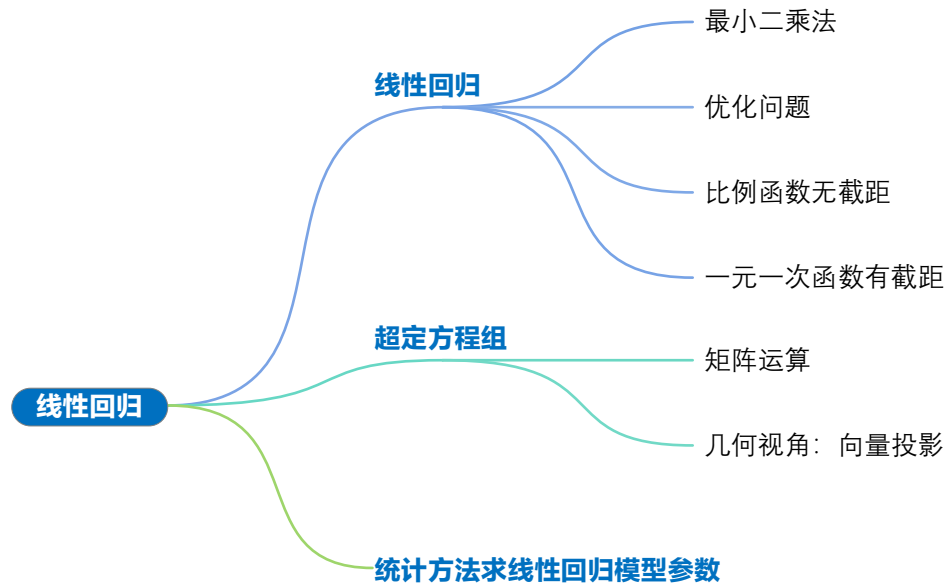
只有带着对数学纯粹的爱去接近她，数学才会向你展开它的神秘所在。

Mathematics reveals its secrets only to those who approach it with pure love, for its own beauty.

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



- ▶ matplotlib.pyplot.contour() 绘制等高线图
- ▶ matplotlib.pyplot.scatter() 绘制散点图
- ▶ numpy.array() 创建 array 数据类型
- ▶ numpy.linalg.inv() 矩阵求逆
- ▶ numpy.linalg.solve() 求解线性方程组
- ▶ numpy.linspace() 产生连续均匀向量数值
- ▶ numpy.meshgrid() 创建网格化数据
- ▶ numpy.random.randint() 产生随机整数
- ▶ numpy.random.seed() 设定初始化随机状态
- ▶ plot_wireframe() 绘制三维单色线框图
- ▶ seaborn.scatterplot() 绘制散点图
- ▶ sympy.abc 引入符号变量
- ▶ sympy.diff() 求解符号导数和偏导解析式
- ▶ sympy.evalf() 将符号解析式中未知量替换为具体数值
- ▶ sympy.simplify() 简化代数式
- ▶ sympy.solvers.solve() 符号方程求根
- ▶ sympy.symbols() 定义符号变量
- ▶ sympy.utilities.lambdify.lambdify() 将符号代数式转化为函数



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

24.1 鸡兔数量的有趣关系

清江一曲抱村流，长夏江村事事幽。

舶来的线性代数知识悄悄地改变着小村，村民们凡事都要用这个数学工具探究一番。

大家这次盯上了一个养鸡养兔的小妙招。老人常言“两鸡一兔，百毒不入”。也就是说，不管最开始养多少鸡兔，当鸡兔大概达到 2:1 这个比例，便达到某种神奇的平衡，鸡兔都健健康康。

农夫决定一探究竟，他搜集村中 20 户养鸡大户的鸡兔数量，总结在表 1。

表 1. 20 户农户鸡兔数量关系

养鸡数量, y	32	110	71	79	45	20	56	55	87	68	87	63	31	88	44	33	57	16	22	52
养兔数量, x	22	53	39	40	25	15	34	34	52	41	43	33	24	52	20	18	33	12	11	28

将表 1 数据以散点方式绘制在方格纸上得到图 1。老农隐隐觉得这个 2:1 的比例关系好像的确存在。

但是，农夫并不满足于于此，他想找到鸡兔达到平衡时确切的数学关系；于是乎，他想到了想到了比例函数和一元函数，决心探究一番。

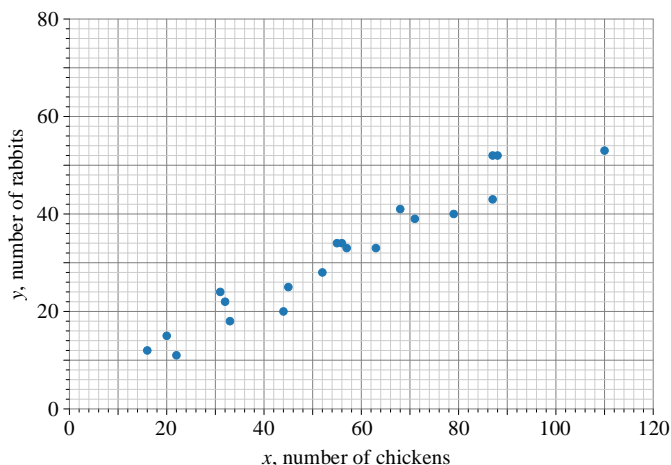


图 1. 平衡时，各家鸡兔数量关系

24.2 试试比例函数： $y = ax$

观察图 1，农夫首先想到用比例函数。

比例函数假设平衡时鸡兔数量好像呈现的某种比例关系：

$$\hat{y} = ax \quad (1)$$

其中，为了区分数据 y ， \hat{y} 上加了个帽子表示预测。

农夫在方格纸上，用红笔画出一系列通过原点的斜线，得到图 2。

老农先是觉得 a 取 0.5 比较好，但是又觉得 a 取 0.6 也不差。他隐约觉得 a 应该在 0.5 和 0.6 之间。

如何找到合理的 a 值？

这个问题让他陷入了沉思。显然，他需要找到一条红线足够靠近图 2 所有散点。那么，问题来了——如何量化“足够靠近”？

他决定先找几个值试试看。

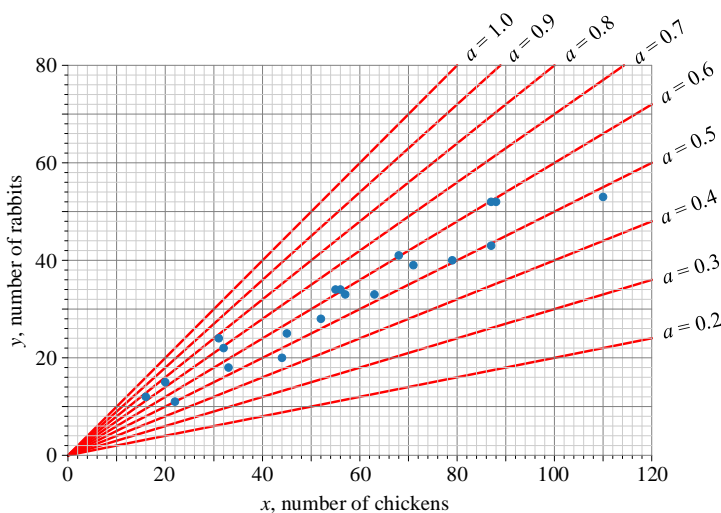


图 2. 平衡时，各家鸡兔数量好像呈现某种比例关系

$a = 0.4$

农夫先试了 $a = 0.4$ ，这时比例函数为：

$$\hat{y} = 0.4x \quad (2)$$

将 $x = 110$ (鸡的数量) 代入 (2)，得到 44 (兔的数量) 这个预测值。

$$\hat{y}|_{x=110} = 0.4 \times 110 = 44 \quad (3)$$

当 $x = 110$ 时，真实值 y 和预测值 \hat{y} 两者的误差 e 为：

$$e|_{x=100} = y - \hat{y} = 53 - 44 = 9 \quad (4)$$

农夫觉得从这个误差值入手，可能会找到合适的 a 值，并确定一条合理的比例函数。

于是乎，农夫开始计算 $\hat{y} = 0.4x$ 这个比例函数条件下，图 1 中每个点的误差值。

最后，他得到图 3。图中，竖直黄色线段代表实际数据和比例函数估值之间的误差，也就是不同 x 对应的 e 。

农夫翻阅舶来的数学典籍，发现了最小二乘法；仔细研读后，他决定拿来一试。

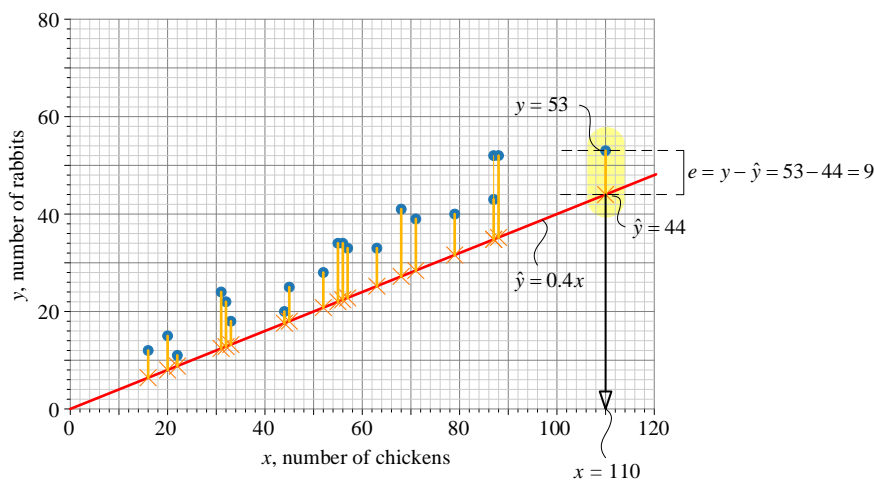


图 3. $a = 0.4$ 时，实际数据和比例函数估值之间的误差

24.3 最小二乘法

书上写道：“最小二乘法通过最小化误差的平方和，寻找数据的最佳回归参数匹配。”

误差平方和最小化

农夫已经得到了一系列 e 值，只需要对 e 平方。

他把计算得到的分步数据记录在表 2 中。表第一、二行数值分别为鸡、兔实际数量，第三行为 $\hat{y} = 0.4x$ 估算得到的兔子数量，第四行为误差 $e = y - \hat{y}$ ，即实际兔数减去估算兔数，第五行为误差的平方值 e^2 。

表 2 第五行 e^2 求和得到误差平方和为 1756.28。

表 2. a 取 0.4 时，估计值、误差、误差平方

养鸡数量 x	32	110	71	79	45	20	56	55	87	68	87	63	31	88
养兔数量 y	22	53	39	40	25	15	34	34	52	41	43	33	24	52
$\hat{y} = 0.4x$ 估算兔数	12.8	44	28.4	31.6	18	8	22.4	22	34.8	27.2	34.8	25.2	12.4	35.2
误差 e	9.2	9	10.6	8.4	7	7	11.6	12	17.2	13.8	8.2	7.8	11.6	16.8
误差平方 e^2	84.6	81	112.3	70.5	49	49	134.5	144	295.8	190.4	67.2	60.8	134.5	282.2

农夫突然意识到， e^2 不就是以 e 的绝对值为边长的正方形面积嘛！

有了这个几何视角，他绘制得到了图 4。图 4 中所有的正方形的边长为不同 x 位置的误差 e 。将这些蓝色正方形面积相加得到面积和，即误差之和：

$$\sum_{i=1}^{20} (e^{(i)})^2 = \sum_{i=1}^{20} (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{20} (y^{(i)} - ax^{(i)})^2 \quad (5)$$

找到让上式值最小的 a ，就可以让图 4 中正方形面积之和最小。

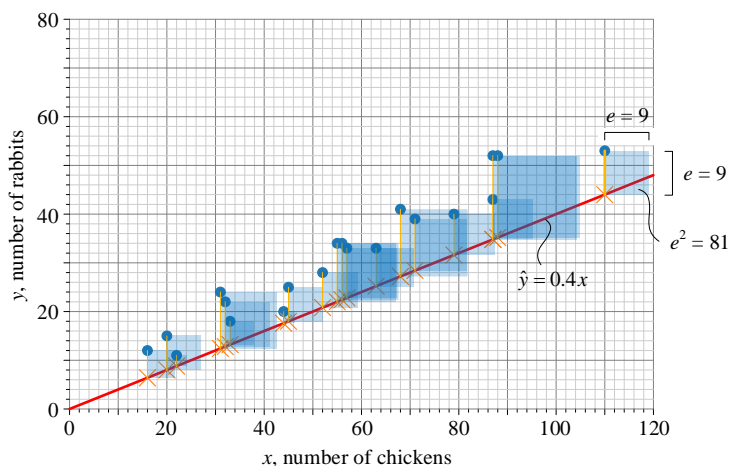


图 4. $a = 0.4$ 时，可视化误差平方

$a = 0.5$

他决定再试几个值，比如 $a = 0.5$ 时，

$$\hat{y} = 0.5x \quad (6)$$

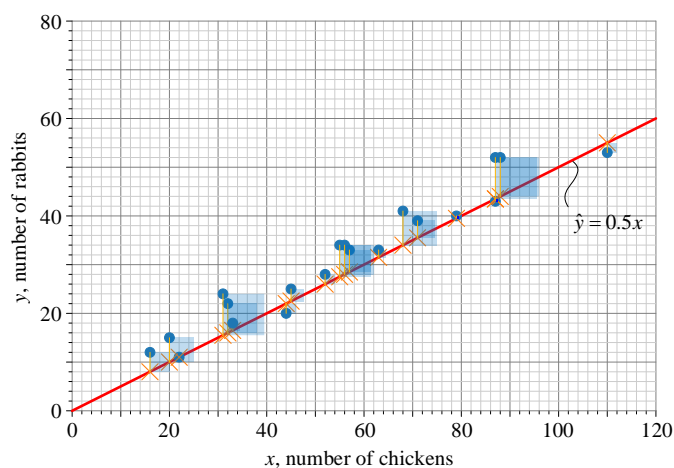
表 3 给出 a 取 0.5 时，不同 x 对应的估计值 \hat{y} 、误差 e 、误差平方 e^2 。

经过计算可以发现 (6) 这个比例函数模型条件下，误差平方和为 422。

几何角度来看，图 5 中的正方形面积之和看上去确实比图 4 要小。

表 3. a 取 0.5 时，估计值、误差、误差平方

养鸡数量 x	32	110	71	79	45	20	56	55	87	68	87	63	31	88
养兔数量 y	22	53	39	40	25	15	34	34	52	41	43	33	24	52
$\hat{y} = 0.5x$ 估算兔数	16	55	35.5	39.5	22.5	10	28	27.5	43.5	34	43.5	31.5	15.5	44
误差 e	6	-2	3.5	0.5	2.5	5	6	6.5	8.5	7	-0.5	1.5	8.5	8
误差平方 e^2	36	4	12.25	0.25	6.25	25	36	42.25	72.25	49	0.25	2.25	72.25	64

图 5. $a = 0.5$ 时，可视化误差平方

$a = 0.6$

农夫又试了试 $a = 0.6$ ，比例函数为：

$$\hat{y} = 0.6x \quad (7)$$

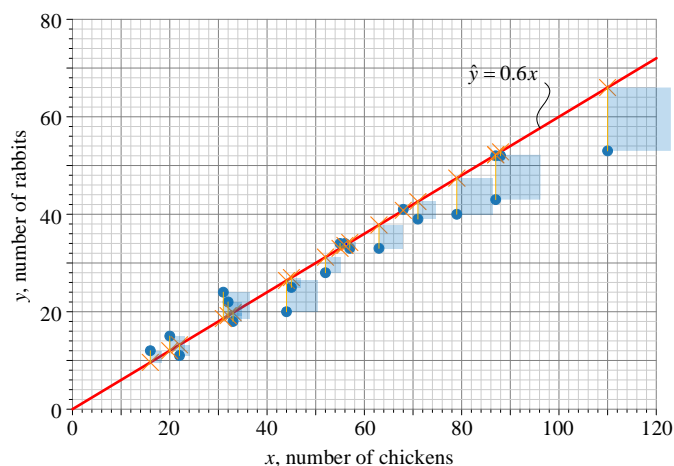
经过表 4 计算求得误差平方和为 396.28。图 6 可视化误差平方和。

农夫感觉到，似乎在 0.5 和 0.6 之间存在一个更好的 a ，能够让误差平方和最小。

但是，这样徒手计算，一个一个值算，终究不是办法。

表 4. a 取 0.6 时，估计值、误差、误差平方

养鸡数量 x	32	110	71	79	45	20	56	55	87	68	87	63	31	88
养兔数量 y	22	53	39	40	25	15	34	34	52	41	43	33	24	52
$\hat{y} = 0.6x$ 估算兔数	19.2	66	42.6	47.4	27	12	33.6	33	52.2	40.8	52.2	37.8	18.6	52.8
误差 e	2.8	-13	-3.6	-7.4	-2	3	0.4	1	-0.2	0.2	-9.2	-4.8	5.4	-0.8
误差平方 e^2	7.84	169	12.96	54.76	4	9	0.16	1	0.04	0.04	84.64	23.04	29.16	0.64

图 6. $a = 0.6$ 时，可视化误差平方

目标函数

观察 (5)，他发现 $x^{(i)}$ 和 $y^{(i)}$ 都是给定数值，而式中唯一的变量就是 a 。也就是说，把 a 看做一个未知数，(5) 可以写成一个函数 $f(a)$ ：

$$f(a) = \sum_{i=1}^{20} (y^{(i)} - ax^{(i)})^2 \quad (8)$$

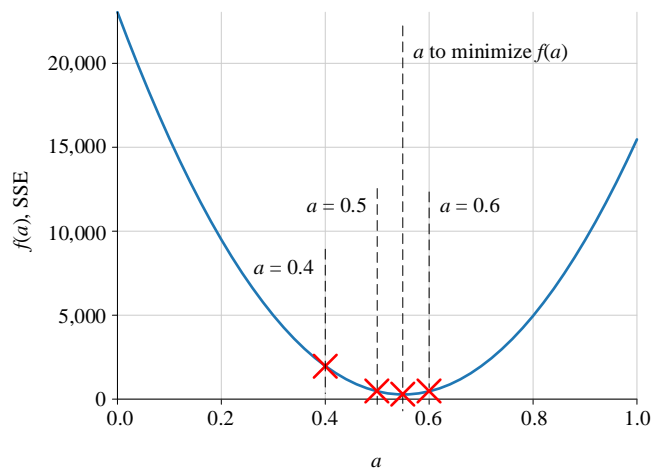
而最小化误差对应的就是让上述函数值取得最小值！农夫想到这里，高兴地不住拍手。

农夫把所有的 $x^{(i)}$ 和 $y^{(i)}$ 代入上式，整理并得到函数具体解析式。

$$f(a) = 65428a^2 - 72228a + 20179 \quad (9)$$

他惊奇地发现，竟然得到了一元二次函数！这个函数，我懂啊！

如图 7 所示，这个一元二次函数的图像是一条开口朝上的抛物线，具有凸性。显然，函数在对称轴处取得最小值。而这个一元二次函数就是优化问题中的目标函数，优化变量为 a 。

图 7. 函数 $f(a)$ 图像

解析法求解优化问题

利用导数这个数学工具，对 $f(a)$ 求一阶导数，得到 $f'(a)$ ：

$$f'(a) = 130856a - 72228 \quad (10)$$

$f'(a) = 0$ 得到 $f(a)$ 取得最小值时 a 的值，记做 a^* ：

$$a^* = \frac{18057}{32714} \approx 0.552 \quad (11)$$

这个 a^* 就是农夫要找的最佳 a 值，它让误差平方和最小。

此时，对应的最优比例函数为。

$$\hat{y} = 0.552x \quad (12)$$

带回检验

农夫决定用“土办法”再算算 a^* 对应的估计值、误差、误差平方这几个数值，他得到表 5。

此时，误差平方和为 245.32，明显小于 $a = 0.5$ 或 $a = 0.6$ 这两种情况。

他不忘绘制图 6，看看正方形的面积到底怎样。

表 5. a 取 0.5504 时，估计值、误差、误差平方

养鸡数量 x	32	110	71	79	45	20	56	55	87	68	87	63	31	88
养兔数量 y	22	53	39	40	25	15	34	34	52	41	43	33	24	52
$\hat{y} = 0.5504x$ 估算兔数	17.6	60.5	39.1	43.5	24.8	11.0	30.8	30.3	47.9	37.4	47.9	34.7	17.1	48.4
误差 e	4.4	-7.5	-0.1	-3.5	0.2	4.0	3.2	3.7	4.1	3.6	-4.9	-1.7	6.9	3.6
误差平方 e^2	19.2	56.9	0.0	12.1	0.1	15.9	10.1	13.9	16.9	12.8	23.9	2.8	48.1	12.7

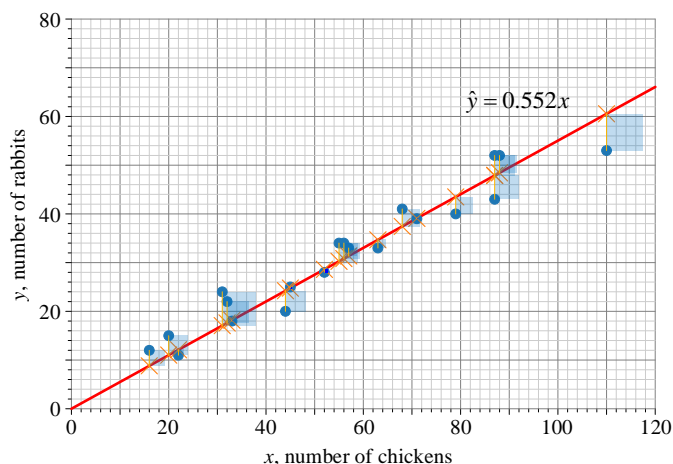
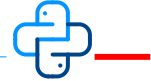


图 8. $a = 0.552$ 时，可视化误差平方

农夫如获至宝，不住地说“最小二乘法，好！真好！”。

他回过头再次翻阅数学典籍，又仔仔细细把最小二乘方法反复研读几遍。兴奋之余，他想让自己的数学模型再复杂一点，决定试试一元一次函数。

以下代码绘制本节图像，并求解最优化问题。



```
# Bk3_Ch24_1

from sympy.abc import a
import numpy as np
import matplotlib.pyplot as plt

def fig_decor(ax):

    plt.xlabel('$x$ (number of chickens)')
    plt.ylabel('$y$ (number of rabbits)')

    plt.axis('scaled')
    ax.set_xlim([0, 120])
    ax.set_ylim([0, 80])

    plt.xticks(np.arange(0, 120 + 1, step=10))
    plt.yticks(np.arange(0, 80 + 1, step=10))

    plt.minorticks_on()
    ax.grid(which='minor', linestyle=':',
            linewidth='0.5', color=[0.8, 0.8, 0.8])

    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])

# generate data
num_chickens = np.array([32, 110, 71, 79, 45, 20, 56, 55, 87, 68, 87, 63, 31, 88])
num_rabbits = np.array([22, 53, 39, 40, 25, 15, 34, 34, 52, 41, 43, 33, 24, 52])
# scatter plot

fig, ax = plt.subplots()

plt.scatter(num_chickens, num_rabbits)

fig_decor(ax)

%% generate f(a), sum of squared errors (SSE), symbolic

from sympy import *

y_pred = a*num_chickens

f_a_SSE = np.sum((num_rabbits - y_pred)**2)

f_a_SSE = simplify(f_a_SSE)

print(f_a_SSE)
```

```

%% plot f(a) versus a
a_array = np.linspace(0,1,51)
f_a_SSE_fcn = lambda f(a, f_a_SSE)
SSE_array = f_a_SSE_fcn(a_array)

# first-order differential
df_da_SSE = diff(f_a_SSE, a)
print(df_da_SSE)

# solution of a
a_star_only = solve(df_da_SSE, a)

print(a_star_only)

a_star_only = a_star_only[0].evalf()

SSE_min = f_a_SSE_fcn(a_star_only)

fig, ax = plt.subplots()

plt.plot(a_array, SSE_array)
plt.axvline(x=a_star_only, linestyle = '--')
plt.plot(a_star_only, SSE_min, 'rx', markersize = 16)

plt.xlabel('a, slope')
plt.ylabel('f(a), sum of squared errors, SSE')
ax.set_xlim([a_array.min(), a_array.max()])
ax.set_ylim([0, SSE_array.max()])

ax.grid(linestyle=':', linewidth='0.5', color=[0.8, 0.8, 0.8])

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

%% y = ax model
def plot_square(x,y1,y2):
    if y2 > y1:
        temp = y2;
        y2 = y1;
        y1 = temp;

    d = y1 - y2;

    plt.fill(np.vstack((x, x + d, x + d, x)),
             np.vstack((y2, y2, y1, y1)),
             facecolor='b', edgecolor='none',
             alpha = 0.3)

x_array = np.linspace(0,150,10)[: , None]
y_pred = a_star_only*x_array

fig, ax = plt.subplots()

plt.plot(x_array, y_pred, color = 'r')
plt.scatter(num_chickens, num_rabbits)

num_rabbits_predicted = a_star_only*num_chickens

plt.plot(np.vstack((num_chickens,num_chickens)),
         np.vstack((num_rabbits, num_rabbits_predicted)),
         color = np.array([255,182,0])/255)

for i in range(0,len(num_rabbits_predicted)):
    plot_square(num_chickens[i],num_rabbits[i],num_rabbits_predicted[i]);

```

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

```
fig_decor(ax)
```

24.4 再试试一次函数: $y = ax + b$

农夫知道，比例函数通过原点，也就是纵轴截距为 0。而一元函数则没有这个限制。

他决定试一下如下这个一元函数，看看是否有更好的结果：

$$\hat{y} = ax + b \quad (13)$$

这个一元函数对应的误差平方和为。

$$\sum_{i=1}^{20} (e^{(i)})^2 = \sum_{i=1}^{20} (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{20} (y^{(i)} - ax^{(i)} - b)^2 \quad (14)$$

式中， $x^{(i)}$ 和 $y^{(i)}$ 都是给定的样本数据。也就是说，上式有两个自变量，有两个需要优化的参数 a 、 b 。

农夫还是决定暴力求解一番，将 $x^{(i)}$ 和 $y^{(i)}$ 代入 (14)，整理并得到二元函数 $f(a, b)$ 解析式。

$$f(a, b) = 65428a^2 + 1784ab - 72228a + 14b^2 - 1014b + 20179 \quad (15)$$

这不就是二元二次函数，我也懂啊！几何角度不就是个开口朝上的旋转椭圆面嘛！农夫再次惊叹数学的精妙！

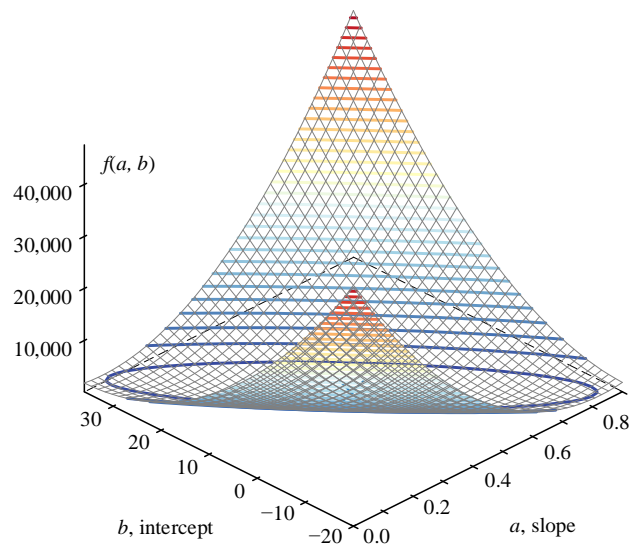


图 9. 误差平方和 $f(a, b)$ 随 a 、 b 变化构造的开口向上抛物曲面

用偏导求极值点

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

计算 $f(a, b)$ 最小值极值点处，利用 $f(a, b)$ 对 a 、 b 求偏导为 0 为条件，构造两个等式。

$$\begin{cases} \frac{\partial f}{\partial a} = 130856a + 1784b - 72228 = 0 \\ \frac{\partial f}{\partial b} = 1784a + 28b - 1014 = 0 \end{cases} \quad (16)$$

联立等式，求得最优解：

$$\begin{cases} a^* = \frac{513}{1157} \approx 0.4434 \\ b^* = \frac{18429}{2314} \approx 7.9641 \end{cases} \quad (17)$$

图 10 告诉我们这个最优解就在旋转椭圆中心位置。农夫看着图 10，嘴里叨叨着“椭圆真是个好东西！哪都离不开它！”

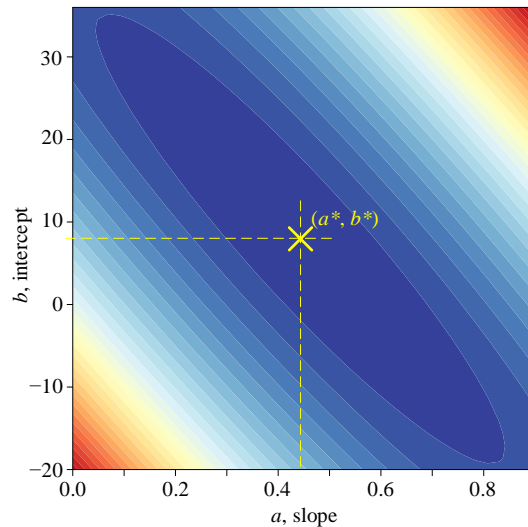


图 10. $f(a, b)$ 平面等高线和最优解位置

(17) 对应的一次函数：

$$\hat{y} = 0.4434x + 7.9641 \quad (18)$$

这就是农夫要找的最佳一次函数！

带回检验

农夫还是想用“土办法”再验算一遍！

他用 (18) 一步步仔细运算，并在将分步结果记录在表 6 中。农夫最终，并求得误差平方和为 128.67，这比之前的比例函数对应的误差平方和还要小。

他不怕麻烦，又画了图 11。图中，一次函数的截距为正。

表 6. $a = 0.4434$ 、 $b = 7.9641$ 时，估计值、误差、误差平方

养鸡数量 x	32	110	71	79	45	20	56	55	87	68	87	63	31	88
养兔数量 y	22	53	39	40	25	15	34	34	52	41	43	33	24	52
$\hat{y} = 0.4863x + 4.312$	19.9	57.8	38.8	42.7	26.2	14.0	31.5	31.1	46.6	37.4	46.6	34.9	19.4	47.1
误差 e	2.1	-4.8	0.2	-2.7	-1.2	1.0	2.5	2.9	5.4	3.6	-3.6	-1.9	4.6	4.9
误差平方 e^2	4.5	23.1	0.0	7.5	1.4	0.9	6.0	8.7	28.9	13.1	13.1	3.8	21.3	23.9

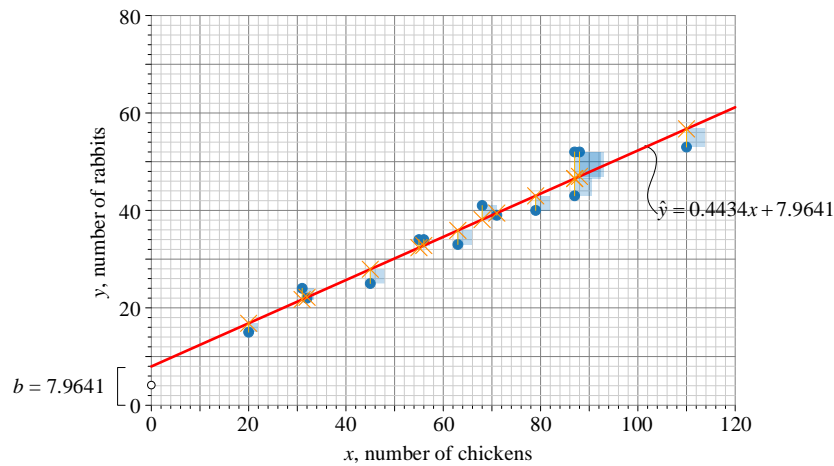


图 11. $a = 0.4434$ 、 $b = 7.9641$ 时，可视化误差平方

以下代码绘制本节图像，并求解优化问题。



```
# Bk3 Ch24 2

import numpy as np
import matplotlib.pyplot as plt

def fig_decor(ax):

    plt.xlabel('$x$ (number of chickens)')
    plt.ylabel('$y$ (number of rabbits)')

    plt.axis('scaled')
    ax.set_xlim([0, 120])
    ax.set_ylim([0, 80])

    plt.xticks(np.arange(0, 120 + 1, step=10))
    plt.yticks(np.arange(0, 80 + 1, step=10))
```

```

plt.minorticks_on()
ax.grid(which='minor', linestyle=':',
        linewidth='0.5', color=[0.8, 0.8, 0.8])

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])

num_chickens = np.array([32, 110, 71, 79, 45, 20, 56, 55, 87, 68, 87, 63, 31, 88])
num_rabbits = np.array([22, 53, 39, 40, 25, 15, 34, 34, 52, 41, 43, 33, 24, 52])

%% generate f(a, b), sum of squared errors (SSE), symbolic
from sympy.abc import a, b
from sympy import *

y_pred = a*num_chickens + b

f_ab_SSE = np.sum((num_rabbits - y_pred)**2)
f_ab_SSE = simplify(f_ab_SSE)

print(f_ab_SSE)

%% plot f(a) versus a
a_array = np.linspace(0,0.9,40)
b_array = np.linspace(-20,36,40)

aa,bb = np.meshgrid(a_array,b_array)

f_ab_SSE_fcn = lambdify((a,b), f_ab_SSE)

SSE_matrix = f_ab_SSE_fcn(aa,bb)
# SSE_matrix = SSE_matrix.evalf()

# first-order partial differential
df_da_SSE = diff(f_ab_SSE, a)
print(df_da_SSE)

df_db_SSE = diff(f_ab_SSE, b)
print(df_db_SSE)

# solution of (a,b)

sol = solve([df_da_SSE, df_db_SSE], [a, b])

print(sol)

a_star = sol[a]
b_star = sol[b]

a_star = a_star.evalf()
b_star = b_star.evalf()

print(a_star)
print(b_star)

SSE_min = f_ab_SSE_fcn(a_star,b_star)
print(SSE_min)

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

ax.plot_wireframe(aa,bb, SSE_matrix,
                  color = [0.5,0.5,0.5],
                  linewidth = 0.25)

plt.plot(a_star, b_star, SSE_min,
         marker = 'x', markersize = 12)

```

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

```

colorbar = ax.contour(aa,bb, SSE_matrix,30,
                      cmap = 'RdYlBu_r')

fig.colorbar(colorbar, ax=ax)

ax.set_proj_type('ortho')

ax.set_xlabel('$a$, slope')
ax.set_ylabel('$b$, intercept')
ax.set_zlabel('$Sum of squared errors')
plt.tight_layout()
ax.set_xlim(aa.min(), aa.max())
ax.set_ylim(bb.min(), bb.max())

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

ax.grid(False)
plt.show()

fig, ax = plt.subplots()

colorbar = ax.contourf(aa,bb, SSE_matrix, 30, cmap='RdYlBu_r')
fig.colorbar(colorbar, ax=ax)
plt.plot(a_star, b_star, marker = 'x', markersize = 12)

ax.set_xlim(aa.min(), aa.max())
ax.set_ylim(bb.min(), bb.max())

ax.set_xlabel('$a$, slope')
ax.set_ylabel('$b$, intercept')
# plt.gca().set_aspect('equal', adjustable='box')

plt.show()

### y = ax + b model

def plot_square(x,y1,y2):

    if y2 > y1:
        temp = y2;
        y2 = y1;
        y1 = temp;

    d = y1 - y2;

    plt.fill(np.vstack((x, x + d, x + d, x)),
             np.vstack((y2, y2, y1, y1)),
             facecolor='b', edgecolor='none',
             alpha = 0.3)

x_array = np.linspace(0,150,10)[: , None]

y_pred = a_star*x_array + b_star

fig, ax = plt.subplots()

plt.plot(x_array, y_pred, color = 'r')
plt.scatter(num_chickens, num_rabbits)

num_rabbits_predicted = a_star*num_chickens + b_star

plt.plot(np.vstack((num_chickens,num_chickens)),
         np.vstack((num_rabbits, num_rabbits_predicted)),
         color = np.array([255,182,0])/255)

plt.plot(num_chickens, num_rabbits_predicted,
         linestyle = 'None', marker = 'x',
         markerfacecolor = 'darkorange',
         markeredgecolor = 'darkorange',
         markersize = 10)

```

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


```
for i in range(0, len(num_rabbits_predicted)):
    plot_square(num_chickens[i], num_rabbits[i], num_rabbits_predicted[i]);

fig_decor(ax)
```

24.5 再探黄鼠狼惊魂夜：超定方程组

突然间，一道灵光闪过！

农夫回想，那夜黄鼠狼来偷鸡抓兔，邻居甲乙丙丁四人数头数、脚数时，为了估算鸡兔数量，他采用的舶来线性代数典籍中的超定方程组的求解方法。

回过头来看自己手中的数学问题，“这不也是一个超定方程组吗？！”

比例函数

他立刻摊开纸，把表 1 数据写成列向量形式。

$$\mathbf{x} = \begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 22 \\ 53 \\ \vdots \\ 28 \end{bmatrix} \quad (19)$$

农夫将比例函数模型写成：

$$\mathbf{y} = a\mathbf{x} \quad (20)$$

即

$$\underset{\mathbf{y}}{\begin{bmatrix} 22 \\ 53 \\ \vdots \\ 28 \end{bmatrix}} = a \underset{\mathbf{x}}{\begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix}} \quad (21)$$

只有一个未知数 a ，但是方程组有 20 个方程，这显然也是一个超定方程组！

农夫顿时兴奋起来，他用黄鼠狼惊魂夜一模一样的方法求解。

$$a = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y} \quad (22)$$

实际上， $\mathbf{x}^T \mathbf{x}$ 是一个 1×1 矩阵，也就是一个数字；它的逆就是 $\mathbf{x}^T \mathbf{x}$ 这个数的倒数。

将 \mathbf{x} 和 \mathbf{y} 具体数值代入 (22)，得到。

$$a = \left(\begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix}^T @ \begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix} \right)^{-1} @ \begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix}^T @ \begin{bmatrix} 22 \\ 53 \\ \vdots \\ 28 \end{bmatrix} = 0.552 \quad (23)$$

农夫惊呼，“得来全不费工夫啊！”这个结果和他用最小二乘法得到结果完全一致。

一元函数

灵光再现，他立刻疾步多取回些纸笔，将一元函数这个模型也写成矩阵形式：

$$\underset{y}{\begin{bmatrix} 22 \\ 53 \\ \vdots \\ 28 \end{bmatrix}} = a \underset{x}{\begin{bmatrix} 32 \\ 110 \\ \vdots \\ 52 \end{bmatrix}} + b \underset{I}{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}} \quad (24)$$

即

$$\hat{y} = ax + bI \quad (25)$$

I 叫做全 1 列向量。

只有二个未知数 a 、 b ，但是方程组有 20 个方程，这明显也是一个超定方程组。

将 (25) 写成：

$$\hat{y} = \underbrace{\begin{bmatrix} I & x \end{bmatrix}}_X \begin{bmatrix} b \\ a \end{bmatrix} \quad (26)$$

令，

$$X = \begin{bmatrix} I & x \end{bmatrix} = \begin{bmatrix} 1 & 32 \\ 1 & 110 \\ \vdots & \vdots \\ 1 & 52 \end{bmatrix} \quad (27)$$

(26) 则写成：

$$\hat{y} = X \begin{bmatrix} b \\ a \end{bmatrix} \quad (28)$$

求解超定方程组，得到。

$$\begin{bmatrix} b \\ a \end{bmatrix} = (X^T X)^{-1} X^T y \quad (29)$$

将 X 和 y 具体数值代入 (29)，得到。

$$\begin{bmatrix} b \\ a \end{bmatrix} = \left(\begin{bmatrix} 1 & 32 \\ 1 & 110 \\ \vdots & \vdots \\ 1 & 52 \end{bmatrix}^T \begin{bmatrix} 1 & 32 \\ 1 & 110 \\ \vdots & \vdots \\ 1 & 52 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 32 \\ 1 & 110 \\ \vdots & \vdots \\ 1 & 52 \end{bmatrix}^T \begin{bmatrix} 22 \\ 53 \\ \vdots \\ 28 \end{bmatrix} = \begin{bmatrix} 14 & 892 \\ 892 & 65428 \end{bmatrix}^{-1} \begin{bmatrix} 507 \\ 36114 \end{bmatrix} = \begin{bmatrix} 7.9641 \\ 0.4434 \end{bmatrix} \quad (30)$$

几何视角

农夫知道，凡是有向量的地方，就有几何！

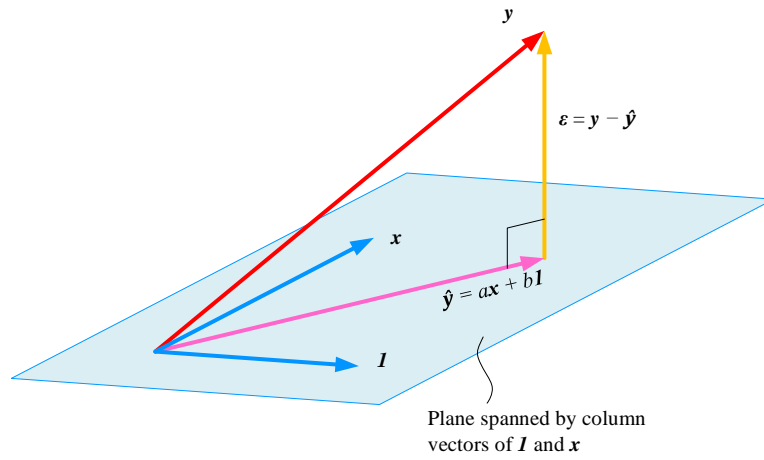


图 12. 几何角度解释一元最小二乘结果，二维平面

上述解法肯定可以通过几何角度解释。如图 12 所示，将 y 向量向 x 和 1 张成的平面 H 投影，得到结果为向量 \hat{y} ；而误差 ε 可以写成。

$$\varepsilon = y - \hat{y} = y - (ax + b1) \quad (31)$$

误差 ε 显然垂直于 H ，即 ε 分别垂直 1 和 x 。

也就是说：

$$\begin{aligned} \varepsilon \perp 1 &\Rightarrow 1^T \varepsilon = 0 \Rightarrow 1^T (y - (ax + b1)) = 0 \\ \varepsilon \perp x &\Rightarrow x^T \varepsilon = 0 \Rightarrow x^T (y - (ax + b1)) = 0 \end{aligned} \quad (32)$$

以上两式合并：

$$\underbrace{\begin{bmatrix} 1 & x \end{bmatrix}}_X^T \left(y - X \begin{bmatrix} b \\ a \end{bmatrix} \right) = 0 \quad (33)$$

整理得到：

$$\mathbf{X}^T \mathbf{X} \begin{bmatrix} b \\ a \end{bmatrix} = \mathbf{X}^T \mathbf{y} \quad (34)$$

等式左右分别左边乘以 $\mathbf{X}^T \mathbf{X}$ 的逆，不就得到 (29) 嘛！

“嗟夫！我的神仙姑奶奶！”，这个结果让农夫惊呆了半晌。

带回检验

醒过神来，他把比例函数和一次函数对应的图像都画在一幅图上，如图 13。

“朝闻道夕死可矣！”线性代数的魅力让农夫彻底折服。

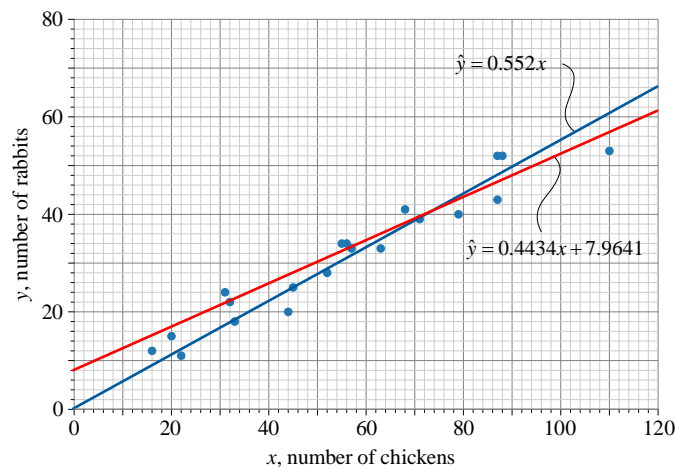
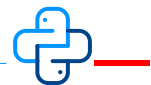


图 13. 比较比例模型和线性模型

以下代码求解本节优化问题，并绘制图 13。



```
# Bk3 Ch24 3

import numpy as np
import matplotlib.pyplot as plt

def fig_decor(ax):

    plt.xlabel('$x$ (number of chickens)')
    plt.ylabel('$y$ (number of rabbits)')

    plt.axis('scaled')
    ax.set_xlim([0, 120])
    ax.set_ylim([0, 80])

    plt.xticks(np.arange(0, 120 + 1, step=10))
    plt.yticks(np.arange(0, 80 + 1, step=10))
```

```

plt.minorticks_on()
ax.grid(which='minor', linestyle=':',
        linewidth='0.5', color=[0.8, 0.8, 0.8])

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])

num_chickens = np.array([32, 110, 71, 79, 45, 20, 56, 55, 87, 68, 87, 63, 31, 88])
num_rabbits = np.array([22, 53, 39, 40, 25, 15, 34, 34, 52, 41, 43, 33, 24, 52])

# scatter plot
fig, ax = plt.subplots()

plt.scatter(num_chickens, num_rabbits)

fig_decor(ax)

%% proportional function, y = ax
x_array = np.linspace(0, 150, 10)[:, None]

x = num_chickens[:, None]
y = num_rabbits[:, None]

a_star_only = np.linalg.inv(x.T @ x) @ x.T @ y

y_pred = a_star_only * x_array

fig, ax = plt.subplots()

plt.plot(x_array, y_pred, color='r')
plt.scatter(num_chickens, num_rabbits)

fig_decor(ax)

%% linear function, y = ax + b
X = np.hstack((np.ones_like(x), x))

sol = np.linalg.inv(X.T @ X) @ X.T @ y

a_star_ = sol[0]
b_star_ = sol[1]

a_star, b_star = np.polyfit(num_chickens, num_rabbits, 1)
y_pred = a_star * x_array + b_star

fig, ax = plt.subplots()

plt.plot(x_array, y_pred, color='r')
plt.scatter(num_chickens, num_rabbits)

fig_decor(ax)

```

25.6 统计方法求解回归参数

突然间，农夫想起前几日在一本叫做《概率统计》的数学典籍中一个有趣的公式，它赶忙取来典籍，找到如下这个公式：

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

$$y = \rho_{x,y} \frac{\sigma_y}{\sigma_x} (x - \mu_x) + \mu_y = \underbrace{\rho_{x,y} \frac{\sigma_y}{\sigma_x}}_a x + \underbrace{\left(-\rho_{x,y} \frac{\sigma_y}{\sigma_x} \mu_x + \mu_y \right)}_b \quad (35)$$

其中, μ_x 为 X 均值, μ_y 为 Y 均值; σ_x 为 X 的标准差, σ_y 为 Y 的标准差; $\rho_{x,y}$ 为 X 和 Y 的相关性系数。

农夫意识到, 从统计角度, 也可以用 (35) 计算一元一次线性回归模型。

他赶紧利用表 1 数据计算得到均值、标准差和相关性系数等值:

$$\begin{cases} \mu_x = 63.714 \\ \mu_y = 36.214 \end{cases}, \begin{cases} \sigma_x = 25.712 \\ \sigma_y = 11.826 \end{cases}, \rho_{x,y} = 0.96397 \quad (36)$$

这样可以计算得到参数 a 和 b :

$$\begin{aligned} a &= \rho_{x,y} \frac{\sigma_y}{\sigma_x} = 0.96397 \times \frac{11.826}{25.712} = 0.4434 \\ b &= -\rho_{x,y} \frac{\sigma_y}{\sigma_x} \mu_x + \mu_y = -0.96397 \times \frac{11.826}{25.712} \times 63.714 + 36.214 = 7.9641 \end{aligned} \quad (37)$$

这和前面的几种方法结果完全吻合! 农夫顿悟, 原来最小二乘法线性回归是几何、向量、优化、概率统计的完美合体!

他不忘绘制图 14 这幅图; 图中回归直线通过 (μ_x, μ_y) 这点。

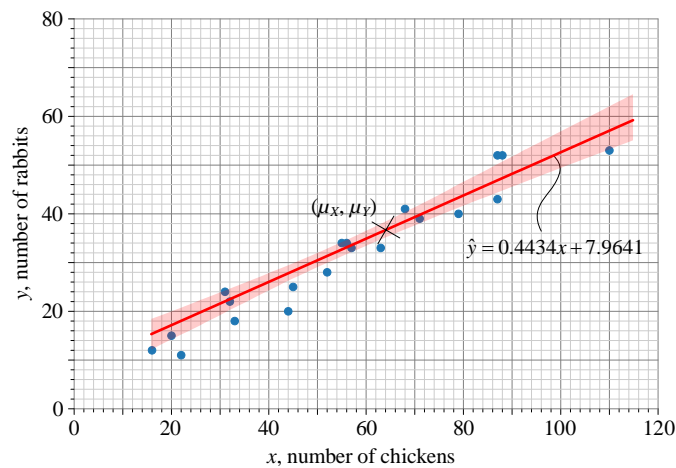
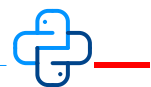


图 14. 利用统计方法获得线性模型

以下代码绘制图 14。



```

# Bk3 Ch24_4

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

def fig_decor(ax):

    plt.xlabel('$x$ (number of chickens)')
    plt.ylabel('$y$ (number of rabbits)')

    plt.axis('scaled')
    ax.set_xlim([0, 120])
    ax.set_ylim([0, 80])

    plt.xticks(np.arange(0, 120 + 1, step=10))
    plt.yticks(np.arange(0, 80 + 1, step=10))
    plt.minorticks_on()
    ax.grid(which='minor', linestyle=':',
            linewidth='0.5', color=[0.8, 0.8, 0.8])

    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])

num_chickens = np.array([32, 110, 71, 79, 45, 20, 56, 55, 87, 68, 87, 63, 31, 88])
num_rabbits = np.array([22, 53, 39, 40, 25, 15, 34, 34, 52, 41, 43, 33, 24, 52])

sigma_X = num_chickens.std(ddof = 1)
sigma_Y = num_rabbits.std(ddof = 1)
rho_XY = np.corrcoef(num_chickens, num_rabbits)[1][0]
mean_X = num_chickens.mean()
mean_Y = num_rabbits.mean()

a = rho_XY*sigma_Y/sigma_X
b = -a*mean_X + mean_Y

print('=== Slope, a ===')
print(a)
print('=== Intercept, b ===')
print(b)

x_array = np.linspace(0, 120, 20)
fig, ax = plt.subplots()
sns.regplot(x=num_chickens, y=num_rabbits, ax = ax, truncate=False,
            line_kws={"color": "red"});

plt.plot(mean_X, mean_Y, marker = 'x', markerfacecolor = 'r',
         markersize = 12)
fig_decor(ax)

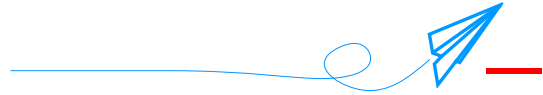
%% use sklearn

from sklearn.linear_model import LinearRegression

x = num_chickens.reshape((-1, 1))
y = num_rabbits

model = LinearRegression().fit(x, y)
print('Slope, a:', model.coef_)
print('Intercept, b:', model.intercept_)

```



农夫落笔刹那，毫无防备之间，黑云压城城欲摧，窗外云浪翻腾。

农夫赶忙起身关紧门窗，只见道道长龙电光从西方汹汹而来！它们撕开天幕，列缺霹雳，丘峦崩摧。

瞬时，天河倾注，拳头大的雨滴敲击着大地，冲刷每一条沟壑，涤荡每一片浮尘。

农夫却毫无惧色，他喜出望外，连声说道，“天上之水啊！上善若水啊！好水，好水！”

未完待续。