例 5.1 Python 程序代码

```
#一元线性回归模型
import scipy.stats as sst
import numpy as np
import matplotlib.pylab as plt
from matplotlib.widgets import Cursor
import csv


#对升序序列找到对匹配 val 值最接近的点坐标
def index_fit(y, val):
    if val >= y[-1]:
        return len(y) - 1
    for i, yi in enumerate(y):
        if val >= yi and val <= y[i + 1]:
            if abs(val - yi) <= abs(val - y[ i + 1]):
                fit_index = i
            else:
                fit_index = i+1
            break
    return fit_index

#用于计算 Lxx, Lyy
def laa(x):
    x_mean = np.mean(x)
    lxx = np.sum((x-x_mean)**2)
    return lxx

#用于计算 Lxy
def lab(x,y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    lxy = np.sum((x-x_mean)*(y-y_mean))
    return lxy


#一元线性回归模型
def polyfit_one(x, y, alpha):
    assert len(x) == len(y)
    n = len(x)
    assert n > 2
    lxx = laa(x)
    lyy = laa(y)
    lxy = lab(x, y)
```

```
    R = lxy/(np.sqrt(lxx) * np.sqrt(lyy))
    R2 = R*R   #计算相关系数与决定系数

    b_est = lxy/lxx  #计算 b 估计
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    a_est = y_mean - b_est * x_mean   #计算 a 估计
    Qe = lyy - b_est * lxy
    sigma_est2 = Qe / (n - 2)

    sigma_est = np.sqrt(sigma_est2) #sigma 估计

    test = np.abs(b_est * np.sqrt(lxx))/sigma_est
    test_level = sst.t.ppf(1 - alpha/2, df=n - 2)
    linear_test = test > test_level   #线性回归检验

    #a,b 的置信区间
    b_int = [b_est - test_level * sigma_est / np.sqrt(lxx), b_est + test_level *
sigma_est / np.sqrt(lxx)]
    a_int = [y_mean - b_int[1] * x_mean, y_mean - b_int[0] * x_mean]

    poly_int = (a_int, b_int)

    poly_val = (a_est, b_est)

    #返回回归模型相应参数
    test_val = {'R': R,
                'R2': R2,
                'linear_test': linear_test,
                'poly_int': poly_int,
                }
    process_val = {'lxx': lxx,
                   'lyy': lyy,
                   'lxy': lxy,
                   'sigma_est': sigma_est,
                   'x_mean': x_mean,
                   'y_mean': y_mean,
                   'test_level': test_level,
                   'ndim': n,
                   }
    return (poly_val, test_val, process_val)

#计算相应的预测区间
```

```
def confidence_interval(y0=None, *args, **kwargs):
    a_est, b_est = args
    sigma_est = kwargs['sigma_est']
    test_level= kwargs['test_level']
    lxx = kwargs['lxx']
    n = kwargs['ndim']
    x_mean = kwargs['x_mean']

    if isinstance(y0, (int, float, np.ndarray)):
        x0 = (y0 - a_est) / b_est
    elif isinstance(y0, (list, tuple)):
        y0 = np.array(y0)
        x0 = (y0 - a_est) / b_est
    else:
        return None

    conf_down = y0 - test_level * sigma_est * np.sqrt(1 + 1 / n + ((x0 - x_mean) **
2 / lxx))
    conf_up = y0 + test_level * sigma_est * np.sqrt(1 + 1 / n + ((x0 - x_mean) **
2 / lxx))

    confidence_interval = (conf_down, conf_up)

    return confidence_interval
```