CS410P

LAB 17L

Purpose

The purpose of this assignment is to introduce you to SciPy as well as give you more practice with matplotlib. You will use NumPy arrays for this assignment.

Scenario

The SciPy library, which is one component of the SciPy stack, provides many numerical routines.

SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the Python session by providing the user with high-level commands and classes for manipulating and visualizing data. Each group of functions are classified as sub-packages in SciPy.

Problem

(1) In an earlier programming assignment (5P) you wrote a Python program to compute a linear regression equation to fit a straight line using least squares approximation. We are going to do something very similar here but just use numPy arrays and plot it using matplotlib.

(2) We will also use SciPy to solve a basic mathematical problem, to use the linear least squares method for a linear line fit. In order to include this in your Python program you need to import the specific package related to Linear Algebra:

```
from scipy import linalg
```

We are going to use the same data files as we did for Programming Assignment 5P and use both the above techniques (1) & (2) to draw all 4 plots in 1 figure window (4 subplots) with the data from one input file going on to one subplot.

Our goal is to use matplotlib to plot that straight line that Linear Least Squares computes using the two techniques 1 & 2 above (see sample output). First to actually compute the best-fit function we use traditional calculus like we did in Programming Assignment 5P, and then we use the SciPy library to verify that it indeed can be done much more easily with the SciPy function available in its sub-package *linalg*.

A) You need to use NumPy arrays to store the data from the files. (You can choose to use portions of what you did before (for 5P), and just convert the lists to numPy arrays). **Plot this data set using blue circles to indicate the (x,y) co-ordinates.**

B) Linear Least-Squares (LLS) Method assumes that the data set falls on a straight line. Therefore,

$$f(x) = mx + b$$

where m and b are constants. However, due to experimental error, some data might not be on the line exactly. There must be error (residual) between the estimated function and real data. Linear Least-Squares Method (or $l_2$approximation) defined as the best-fit function is the function that minimizes the errors using:

$$\sum_{i=0}^{n-1}(y_i - f(x_i))^2$$

Using calculus, the following formulas for coefficients can be obtained (exactly what you did in your lab 15L for numpy tutorial)

$$\begin{aligned} s_x &= \sum_{i=0}^{n-1} x_i \\ s_y &= \sum_{i=0}^{n-1} y_i \\ s_{xy} &= \sum_{i=0}^{n-1} x_i y_i \\ s_{xx} &= \sum_{i=0}^{n-1} x_i^2 \\ m &= (s_x s_y - s_{xy} n)/(s_x^2 - s_{xx} n)) \\ b &= (s_y - s_x m)/n \end{aligned}$$

Use all NumPy functions that operate on arrays to obtain the best-fit function *f(x)* (in other words, no loops allowed in your code!)

Plot x versus f(x) on the same figure using red dashed lines.

C) Now use the linalg sub-package from SciPy to obtain the coefficients a and b (for slope and intercept respectively) for the above data. linalg.lstsq() needs matrices for input arguments. Once you get the coefficients you can compute f(x) again using array arithmetic directly (f(x) = ax + b).
Plot x versus f(x) on the same figure using green circles.

Your final figure should look like the one shown in the sample output with some meaningful text included as well (for axes and legend). You should also title the sub plots to show the data file name and values for slope and y-intercept (the m and b coefficients in (B) or a and b values from (C). I have provided some documentation with an example of how the linalg.lstsq() function works from SciPy.

Your grade for this program includes how efficiently you have written your code in terms of not being repetitive. Even though you have four data files and four different plots, you must create appropriate function(s) so that they get called in a loop four times once for each input file. The input file names can be hard-coded in your program, and you don't need to ask the user to supply the file names.

numpy.linalg.lstsq(*a*, *b*, *rcond=-1*)

Return the least-squares solution to a linear matrix equation.

Solves the equation *a x = b* by computing a vector *x* that minimizes the Euclidean 2-norm *|| b - a x ||^2*. The equation may be under-, well-, or over- determined (i.e., the number of linearly independent rows of *a* can be less than, equal to, or greater than its number of linearly independent columns). If *a* is square and of full rank, then *x* (but for round-off error) is the "exact" solution of the equation.

Parameters:

a *: (M, N) array_like*
"Coefficient" matrix.

b *: {(M,), (M, K)} array_like*
Ordinate or "dependent variable" values. If *b* is two-dimensional, the least-squares solution is calculated for each of the *K* columns of *b*.

rcond *: float, optional*
Cut-off ratio for small singular values of *a*. Singular values are set to zero if they are smaller than *rcond* times the largest singular value of *a*.

Returns:

x *: {(N,), (N, K)} ndarray*
Least-squares solution. If *b* is two-dimensional, the solutions are in the *K* columns of *x*.

residuals *: {(), (1,), (K,)} ndarray*
Sums of residuals; squared Euclidean 2-norm for each column in `b - a*x`. If the rank of *a* is < N or M <= N, this is an empty array. If *b* is 1-dimensional, this is a (1,) shape array. Otherwise the shape is (K,).

rank *: int*
Rank of matrix *a*.

s *: (min(M, N),) ndarray*
Singular values of *a*.

Raises:

LinAlgError
If computation does not converge.

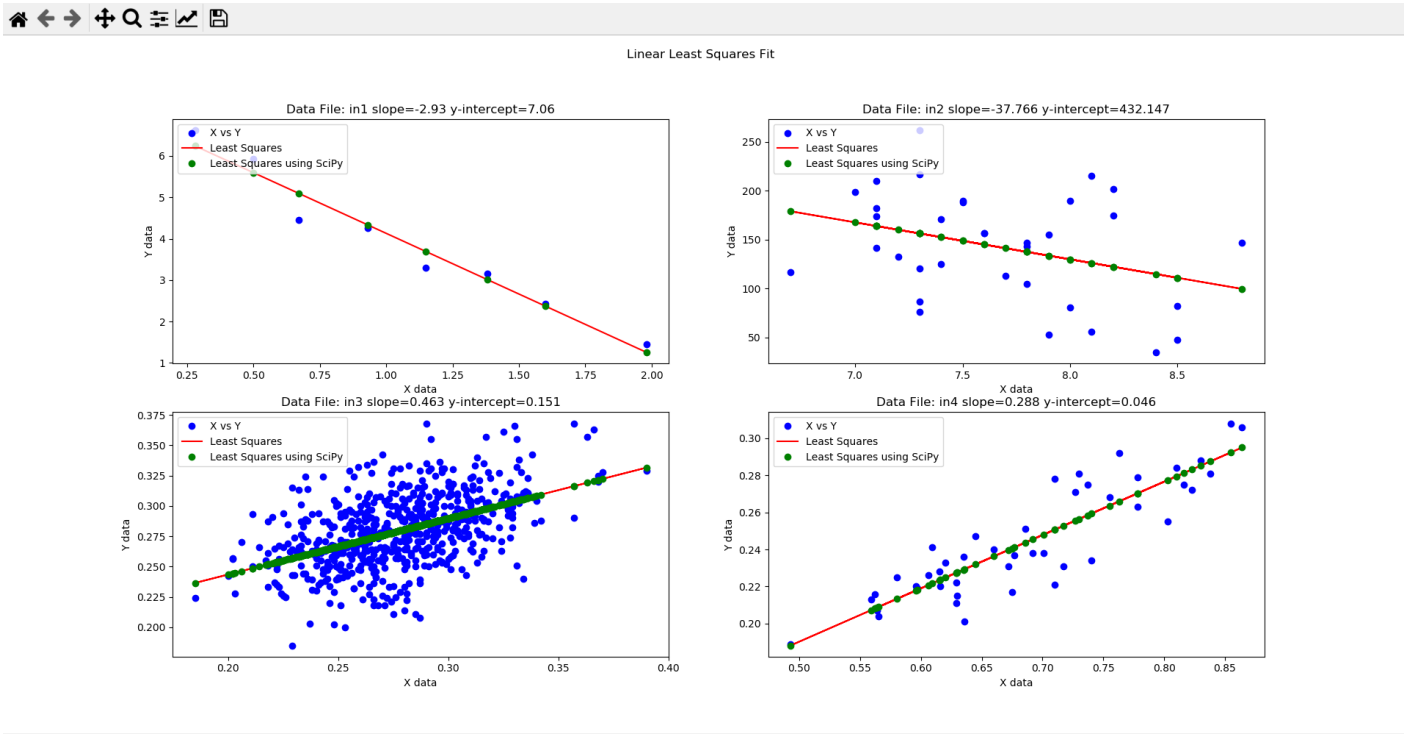
```
>>> x = np.array([0, 1, 2, 3])
>>> y = np.array([-1, 0.2, 0.9, 2.1])
```

By examining the coefficients, we see that the line should have a gradient of roughly 1 and cut the y-axis at, more or less, -1.

We can rewrite the line equation as `y = Ap`, where `A = [[x 1]]` and `p = [[m], [c]]`. Now use lstsq to solve for *p*:

```
>>> A = np.vstack([x, np.ones(len(x))]).T
>>> A
array([[ 0.,  1.],
       [ 1.,  1.],
       [ 2.,  1.],
       [ 3.,  1.]])
```

```
>>> m, c = np.linalg.lstsq(A, y)[0]
>>> print m, c
1.0 -0.95
```


Linear Least Squares Fit
Data File: in1 slope=-2.93 y-intercept=7.06
Data File: in2 slope=-37.766 y-intercept=432.147
Data File: in3 slope=0.463 y-intercept=0.151
Data File: in4 slope=0.288 y-intercept=0.046
X vs Y
Least Squares
Least Squares using SciPy
X data
Y data