

## Linear Regression and OLS

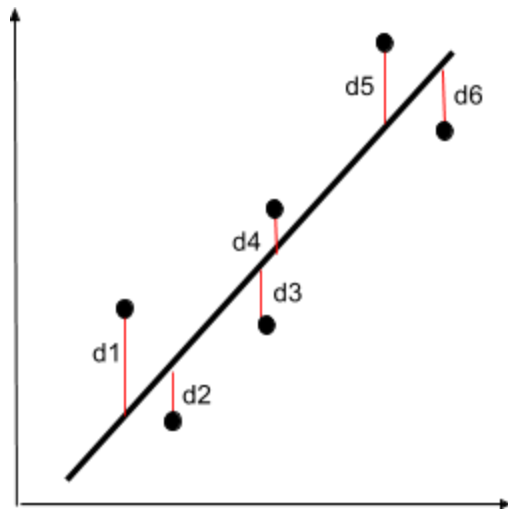
Points about linear regression: a line equation has 1 variable and 2 parameters

$$y = ax + b$$

Thus the number of degrees of freedom of the model is 1.

Ordinary Least Square Method:

calculates the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line.



So the idea is to choose  $a$  and  $b$  such that the sum of distances between  $y_{\text{predicted}}$  -  $y_{\text{observed}}$  is minimum.

$$a \text{ and } b \Rightarrow \min \sum_{i=1}^n |a x_i + b - y_i|$$

This can be done

**Numerically:**

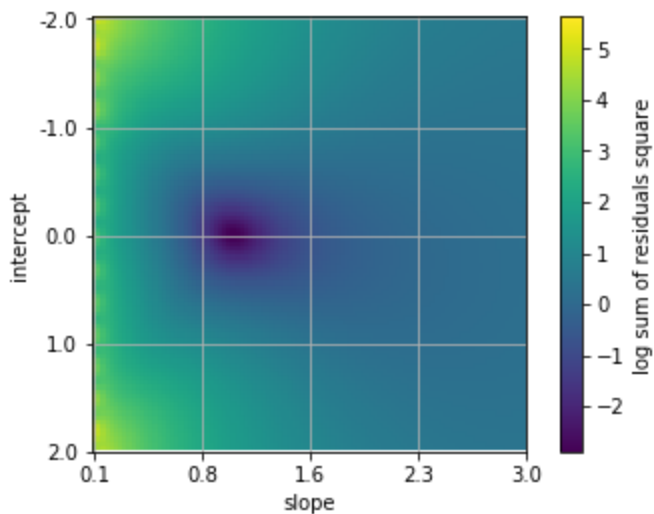
1. the simplest minimization algorithm is brute force: choose a reasonable range for each of the  $N$  variables in the minimization (here 2: the parameters of the line equation) and explore the  $N$ -dimensional (here 2-dimensional, i.e. a surface) parameter space. Very expensive computationally, and will only get you as close to the minimum as the step size you chose to explore the space

(This is the first code I ever wrote by the way, years ago in C)

```

a = -99e9 #some obviously unreasonable value
b = -99e9
residualsSum = np.zeros((100,100))
for i,aa in enumerate(np.linspace(-2, 2, 100)):
    for j,bb in enumerate(np.linspace(0.1, 3, 100)):
        residualsSum[i,j] =  $\sum_{i=1}^n |aa x_i + bb - y_i|$ 
        if residualsSum[i,j] < minVal:
            minVal = residualsSum[i,j]
            a = aa
            b = bb

```



2. Use a wiser minimization algorithm which explores the space looking for a minimum by following a strategy, for example only moving downhill.

```

def residuals(pars, data, model, error=None):
    """Calculates the residuals between model and data
    Arguments:
        data: series - 1D array of datapoints
        model: series - 1D array of model predictions
        error (optional): series - 1D array of errors on data
    Returns:
        sum of residuals square
    """
    residuals = (line(data, pars) - model)**2
    if error:
        residuals = residuals / error**2

```

```

return residuals.sum()
p0 = (0.5,1000)
#here the model is the female income given the male income: a regression line
##important!! never work with very large numbers! always try to reduce your numbers to
~unity when solving numerically
result = minimize(residuals, p0, args=(all_m/1e4, all_f/1e4))
a,b = result.x

```

**Analytically:** for this simple problem the linear algebra is solvable. The sum of the residual square is written in matrix form as

$$S = \sum_{i=1}^n (y_i - \sum_{j=1}^2 x_{ij} \theta_j)^2$$

Which is what we want to minimize and where y are the observations,

the dependent variables, x the location on independent variables (which may be more than one for an multilinear regression, leading to a double index for x, but is 1 in our current case), and  $\theta = (a,b)$  the parameters. Note that even with 1 independent variable to solve the problem analytically the n values of the independent variable need to be encoded in a 2D vector, which is what statsmodels.api.add\_constant() does! The vector of x's ( $x = (0,0.5,1,1.5,2,2.5)$  for example) needs to look like:

```

x = [[ 1., 0. ],
      [ 1., 0.5],
      [ 1., 1. ],
      [ 1., 1.5],
      [ 1., 2. ],
      [ 1., 2.5]]

```

Full derivation is beyond the scope here, but to minimize we take the derivative and set it to 0  $\delta r_i / \delta \theta_j = -X_{ij}$  are the partial derivatives we want to set to 0, with this **X** matrix we can write the problem as  $(X^T X) \hat{\theta} = X^T \hat{y}$  or:  $\hat{\theta} - (X^T X)^{-1} X^T \hat{y} = 0$  (which we can solve as long as we can invert  $(X^T X)$ ).