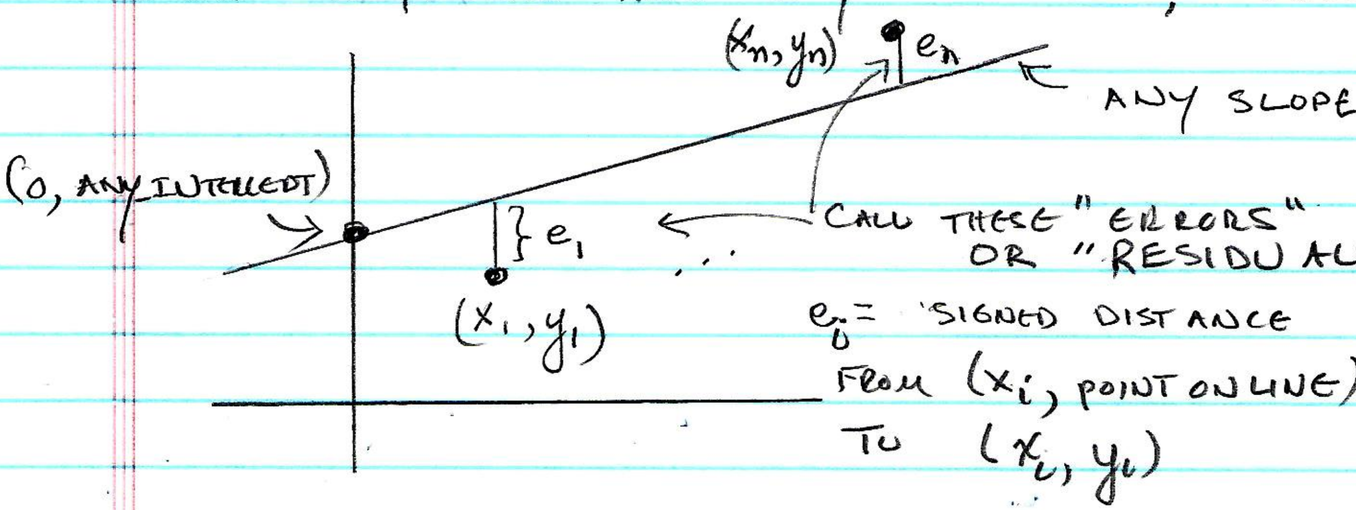
"LEAST SQUARES LINE" – ANOTHER APPROACH
(BUT ARRIVE AT THE SAME "REGRESSION LINE TO MEAN")

GIVEN. (x, y) COORDINATE PAIRS $\{x_i, y_i\}_{i=1}^{n}$

$i =$	$x_i =$	$y_i =$
1	x_1	y_1
2	x_2	y_2
⋮	⋮	⋮
n	x_n	y_n

FOR ANY SLOPE AND ANY INTERCEPT, CONSIDER



CALL THESE "ERRORS" OR "RESIDUALS"

e_i = SIGNED DISTANCE FROM $(x_i$, POINT ON LINE$)$ TO (x_i, y_i)

COMPUTE $\sqrt{\dfrac{\sum_{i=1}^{n} e_i^2}{n}}$

① SQUARE THE ERRORS AND SUM
② FIND THE MEAN
③ TAKE THE SQUARE ROOT

HENCE CALLED THE ROOT MEAN SQUARE ERROR
③ ← ② ← ① ←

OR rmse

A "NUMERICAL OPTIMIZATION" FUNCTION IN SCIPI LIBRARY

IF WE MINIMIZE rmse AMONG <u>ALL</u> POSSIBLE LINES. ONE GETS THE

"LEAST SQUARES LINE"

IT TURNS OUT, THAT

[LINEAR] REGRESSION LINE $\equiv$ LEAST SQUARES LINE

LAST TIME

IT ALSO TURNS OUT THAT THE LEAST SQUARES LINE <u>PROCESS</u> CAN BE USED TO FIND A BEST FITTING 2ND DEGREE POLYNOMIAL

$$y = ax^2 + bx + c$$

OR A 3rd DEGREE POLYNOMIAL

$$y = ax^3 + bx^2 + cx + d$$

ETC.

So, we find the “least squares line” by writing a function that requires arguments “any_slope”, and “any_intercept” and returns the “~~Root~~ Mean Square Error”. For example:

```
def shotput_linear_rmse(any_slope, any_intercept):
    x = shotput.column('Weight Lifted')
    y = shotput.column('Shot Put Distance')
    estimate = any_slope*x + any_intercept
    return np.mean((y - estimate) ** 2) ** 0.5
```

We “minimize” the function with Python numerical optimization capability in scipy library to return a best slope-intercept pair.

```
best_line = minimize(shotput_linear_rmse)
best_line

array([0.09834382, 5.95962883])
```

The following function calls from earlier are still useful functions for comparison, plotting, etc.

```
def standard_units(arr):
    return (arr - np.average(arr))/np.std(arr)

def correlation(t, x, y):
    x_standard = standard_units(t.column(x))
    y_standard = standard_units(t.column(y))
    return np.average(x_standard * y_standard)

def slope(t, x, y):
    r = correlation(t, x, y)
    y_sd = np.std(t.column(y))
    x_sd = np.std(t.column(x))
    return r * y_sd / x_sd

def intercept(t, x, y):
    x_mean = np.mean(t.column(x))
    y_mean = np.mean(t.column(y))
    return y_mean - slope(t, x, y)*x_mean
```

A fitted values function is also useful:

```
def fitted_values(t, x, y):
    """Return an array of the regressions estimates at all the x values"""
    a = slope(t, x, y)
    b = intercept(t, x, y)
    return a*t.column(x) + b
```