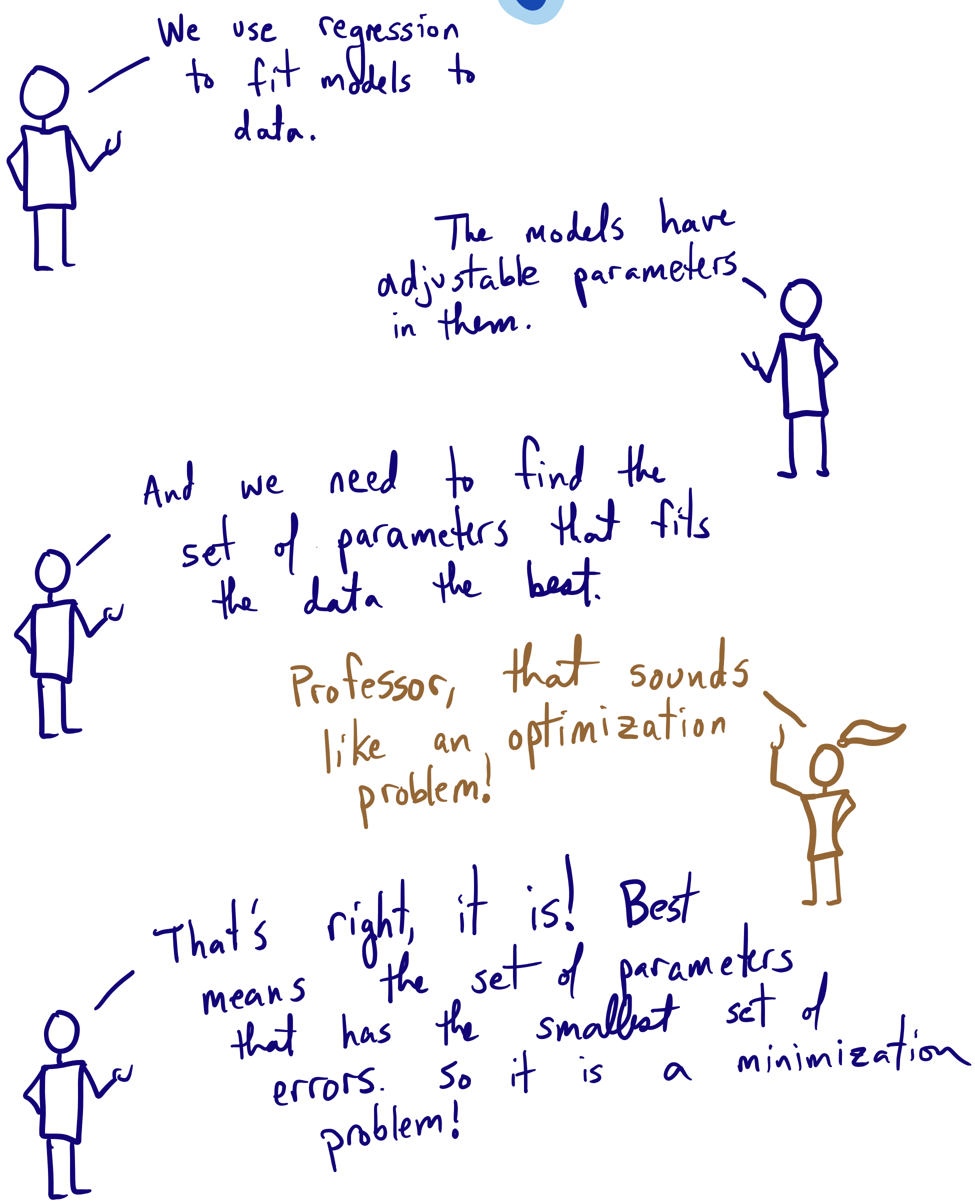
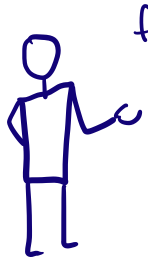
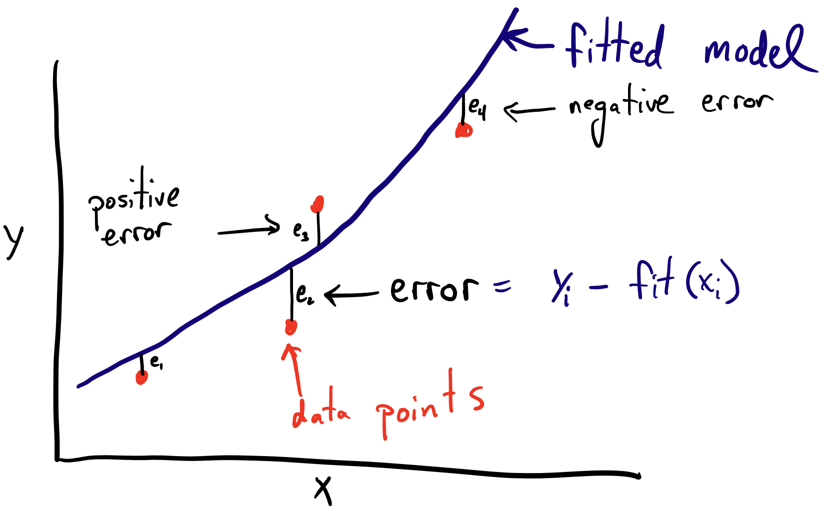
Nonlinear regression

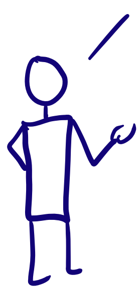




In this figure the blue line is the fitted model, and e_i is the error between the model and data point i.

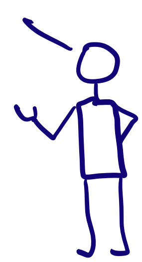


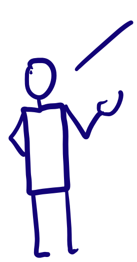
Some errors are positive and some are negative.



We need a quantity that represents the overall quality of the fit, and that accounts for all the errors.

We can't just sum the errors. Since some are positive, and some are negative, in a sum there may be cancellation.





Instead, we use the summed squared errors. These are always positive. Our goal is to find $f(x;p)$ that minimizes the summed squared errors.

the adjustable parameters.

minimize $\sum e_i^2$

Approach using minimize

x = some data } usually 1-d arrays
y = some data }

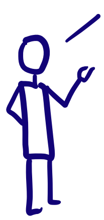
Note the global variables for x + y

```
def model(xvals, p)
    return some_function of xvals + p
```

```
def objective(pars)
    errs = y - f(x, pars)
    return np.sum(errs**2)
```

$\leftarrow \sum_i e_i^2$ AKA: SSE

```
sol = minimize(objective, pguess)
```

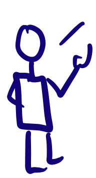


What do you think about this?

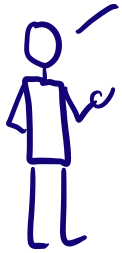
It looks like how we described the optimization problem.

It's kind of a lot of work though. We need to define two functions.

And the global variables can be confusing.



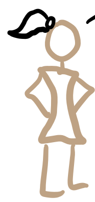
Those are all good points!



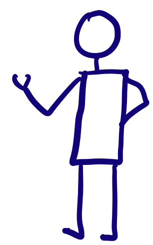
Lot's of people thought the Old-fashioned minimize approach was too much work.

Define my own objective? Yawn...

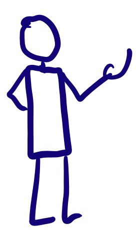
srsly, why should we do all that work?



Scipy agreed, and made curve_fit.



It doesn't come for free though. You have to learn a new syntax.



the model must list each parameter separately

```
def model(x, par_0, par_1, ...):
    return some_function of x and par_i
x = [data]
y = [data]
pars, pcov = curve_fit(model, x, y, P0)
```

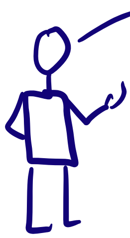
initial guess [par_0, par_1, ...]

the covariance matrix

best pars that minimize the summed squared error.

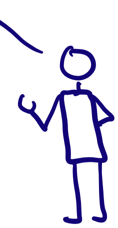
Then you evaluate the model at new x-values like this:

pred_y = model (newx, *pars)



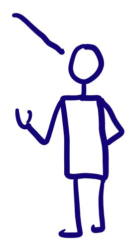
curve_fit requires you to write your model in a certain form: f(x, P_0, P_1, P_2, ...).

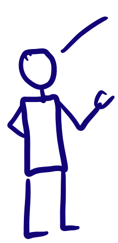
But you don't have to use global variables or define the SSE function.



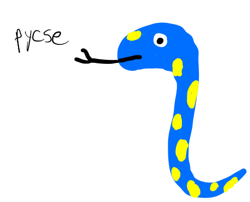
But professor, what about the uncertainty on the parameters?

Very good question! You should use nlinfit for that.





The pycse library provides nlinfit which builds on curve_fit + provides parameter uncertainty



$$\text{pars},\ \text{ci},\ \text{se} = \text{nlinfit}(\text{model}, x, y, p_0, \alpha)$$

still the initial guess

confidence level

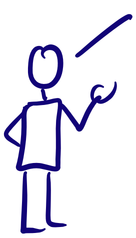
0.05 ⇒ 95% confidence

estimate of standard error on each parameter

confidence interval on each parameter

$(p_i \pm se * tval)$

parameters that minimize the summed squared errors.



pycse has to be installed:

```
pip install pycse
```

Things to remember:

1. The higher your confidence level, the wider the intervals will be

2. If the confidence interval includes $\emptyset$, That parameter may be unnecessary