1 Introduction

- Today we will discuss curve fitting.
- (1) the least squares method, (2) implementing least squares with Scipy, (3) χ^2 analysis of goodness-of-fit (how well our model actually describes the data).

2 Least Squares Regression

- VARIABLES: when we collect data, we vary and independent variable(s) x_i and observe the dependent variable y_i. The data that we collect is of the form:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \tag{2.1}$$

- MODEL: a description of how we expect y_i to relate to x_i with parameters,

$$\lambda_1, \lambda_2, \ldots, \lambda_n = \vec{\lambda}; \quad m > n \tag{2.2}$$

  We write the model as,

$$f(\vec{\lambda}; x) \tag{2.3}$$

  We FIT our data to the model by finding the values of $\vec{\lambda}$ so that $\hat{y}_i = f(\vec{\lambda}; x)$ matches y_i as close as possible.

- We say the model is LINEAR if $f(\vec{\lambda}; x)$ is linear in $\vec{\lambda}$,

$$f(a\vec{\lambda}_0 + b\vec{\lambda}_1; x) = af(\vec{\lambda}_0; x) + bf(\vec{\lambda}_1; x) \tag{2.4}$$

  If $f(\vec{\lambda}; x)$ is linear, then,

$$f(\vec{\lambda}; x) = \sum_{j=0}^{n} \lambda_j \cdot f_j(x); \quad \text{where } f_j(x) \text{ are arbitrary functions} \tag{2.5}$$

  Examples:

  1. $f(\vec{\lambda}; x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2$
  2. $f(\vec{\lambda}; x) = \lambda_0 x + \lambda_1 \sin(x)$
  3. $f(\vec{\lambda}; x) = \lambda_0 \sin(x) + \lambda_1 \sin(x)$

- A model is NONLINEAR if $f(\vec{\lambda}; x)$ is not linear in $\vec{\lambda}$, Examples:

  1. $\sin(\lambda_0 x + \lambda_1 x)$
  2. $\lambda_0 x/(\lambda_1 + x)$

- LEAST SQUARES: how do we know when the parameters $\vec{\lambda}$ have been optimized? We define our heuristic ϕ as the sum of the square residues,

$$\boxed{\phi = \sum_{i=0}^{m} r_i^2 = \sum_{i=0}^{m} [y_i - f(\vec{\lambda}; x_i)]^2} \tag{2.6}$$

[Draw picture of residues]

Least squares regression is the task of minimizing ϕ. When ϕ is minimum,

$$\boldsymbol{\nabla}_{\vec{\lambda}}\, \phi = 0 \tag{2.7}$$

(the subscript $\vec{\lambda}$ means we are taking the partial derivative w.r.t. the parameters $\vec{\lambda}$.)

ϕ is a complicated function, computing it's gradient and finding the values of $\vec{\lambda}$ that make the gradient zero is not an easy task!

1. An exact solution for $\vec{\lambda}$ can be determined in the linear least squares case with some vector calculus (perhaps with some caveats I don't know of...I'm not a statistician)
2. There is no exact, general solution for the nonlienar least squares case. Instead, we use numerical techniques to APPROXIMATE solutions.
3. In practice: we will use an exact solution for the line, and use numerical fittings for all other cases (linear and nonlinear).

NOTE: $\boldsymbol{\nabla}_{\vec{\lambda}}\, \phi = 0$ only gives us the critical points. To be rigorous, we should show that the critical points are indeed minimum. In practice, we shall be a bit lazy and omit this step.

2.1 Least squares regression for a line

- We will use the model,

$$f(\vec{\lambda}; x) = \lambda_0 + \lambda_1 x \tag{2.8}$$

where λ_0 is the y-intercept and λ_1 is the slope.

- Minimizing ϕ,

$$0 = \boldsymbol{\nabla}_{\vec{\lambda}}\, \phi = \boldsymbol{\nabla}_{\vec{\lambda}} \sum_{i=0}^{m} [y_i - f(\vec{\lambda}; x_i)]^2 = \boldsymbol{\nabla}_{\vec{\lambda}} \sum_{i=0}^{m} [y_i - \lambda_0 - \lambda_1 x_i]^2 \tag{2.9}$$

$$= \frac{\partial}{\partial \lambda_0} \sum_{i=0}^{m} [y_i - \lambda_0 - \lambda_1 x_i]^2 \tag{2.10}$$

$$= \frac{\partial}{\partial \lambda_1} \sum_{i=0}^{m} [y_i - \lambda_0 - \lambda_1 x_i]^2 \tag{2.11}$$

- First component (2.10)

$$
\begin{aligned}
0 &= \frac{\partial}{\partial \lambda_0} \sum_{i=0}^{m} [y_i - \lambda_0 - \lambda_1 x_i]^2 \\
&= \frac{\partial}{\partial \lambda_0} \sum_{i=0}^{m} [y_i^2 - \lambda_0 y_i - \lambda_1 x_i y_i - \lambda_0 y_i + \lambda_0^2 + \lambda_0 \lambda_1 x_i - \lambda_1 x_i y_i + \lambda_0 \lambda_1 x_i + \lambda_1^2 x_i^2] \\
&= \sum_{i=0}^{m} \frac{\partial}{\partial \lambda_0} [y_i^2 - 2\lambda_0 y_i - 2\lambda_1 x_i y_i + \lambda_0^2 + 2\lambda_0 \lambda_1 x_i + \lambda_1^2 x_i^2] \\
&= \sum_{i=0}^{m} (-2y_i + 2\lambda_0 + 2\lambda_1 x_i) = \sum_{i=0}^{m} y_i - \lambda_0 - \lambda_1 x_i) \\
&= \left(\sum_{i=0}^{m} y_i \right) - m\lambda_0 - b \left(\sum_{i=0}^{m} x_i \right)
\end{aligned}
$$

$$
\lambda_0 = \frac{(\sum_i y_i) - b\,(\sum_i x_i)}{m} = \bar{y} - \lambda_1 \bar{x} \tag{2.12}
$$

- Second component (2.11)

$$
\begin{aligned}
0 &= \frac{\partial}{\partial \lambda_1} \sum_{i=0}^{m} [y_i - \lambda_0 - \lambda_1 x_i]^2 \\
&= \sum_{i=0}^{m} \frac{\partial}{\partial \lambda_1} [y_i^2 - 2\lambda_0 y_i - 2\lambda_1 x_i y_i + \lambda_0^2 + 2\lambda_0 \lambda_1 x_i + \lambda_1^2 x_i^2] \\
&= \sum_{i=0}^{m} (-2x_i y_i + 2\lambda_0 x_i + 2\lambda_1 x_i^2) = \sum_{i=0}^{m} (\lambda_1 x_i^2 + \lambda_0 x_i - x_i y_i) \\
&= \sum_{i=0}^{m} (\lambda_1 x_i^2 + (\bar{y} - \lambda_1 \bar{x}) x_i - x_i y_i) = \sum_{i=0}^{m} (\lambda_1 x_i^2 + x_i \bar{y} - \lambda_1 x_i \bar{x} - x_i y_i) \\
&= \lambda_1 \sum_i (x_i^2 - x_i \bar{x}) + \sum_i (x_i \bar{x} - x_i y_i)
\end{aligned}
$$

We will need below that: $\sum_i (\bar{x}^2 - x_i \bar{x}) = \sum_i (\bar{x}\bar{y} - y_i \bar{x}) = 0$

$$
\begin{aligned}
\lambda_1 &= \frac{\sum_i (x_i y_i - x_i \bar{x})}{\sum_i (x_i^2 - x_i \bar{x})} = \frac{\sum_i (x_i y_i - x_i \bar{x}) + \sum_i (\bar{x}\bar{y} - y_i \bar{x})}{\sum_i (x_i^2 - x_i \bar{x}) + \sum_i (\bar{x}^2 - x_i \bar{x})} \\
&= \frac{\frac{1}{m} \sum_i (x_i - \bar{x})(y_i - \bar{y})}{\frac{1}{m} \sum_i (x_i - \bar{x})^2} = \frac{Cov(x, y)}{Var(x)}
\end{aligned}
$$

$$
\boxed{\lambda_1 = \frac{Cov(x, y)}{Var(x)}; \quad \lambda_0 = \bar{y} - \lambda_1 \bar{x}} \tag{2.13}
$$

2.2 Gauss-Newton method for nonlinear least squares [Optional]

- This is an approximate least squares method for a general nonlinear regression $f(\vec{\lambda}; x)$. Recall that the goal of the least squares method is to minimize,

$$\phi = \sum_i [y_i - f(\vec{\lambda}; x_i)]^2 \tag{2.14}$$

- Recall Newton's method: a method to approximate the roots of a function $f(x)$.
  1. We make an initial guess for the root x_0
  2. We approximate the function at x_0 as a line, and define the next approximation x_1 as the zero of the line

$$y(x) = f'(x_0)(x - x_0) + f(x_0) \tag{2.15}$$
$$0 = f'(x_0)(x_1 - x_0) + f(x_0) \tag{2.16}$$
$$x_1 = x_0 - f(x_0)/f'(x_0) \tag{2.17}$$

  Generalizing,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{2.18}$$

- We can use Newton's method to approximate the root of $\boldsymbol{\nabla}_{\vec{\lambda}}\, \phi$ [NOTE: it is implied that all further derivatives are w.r.t $\vec{\lambda}$, so I will omit the subscript moving forward. In particular, the ∇^2 is not the Laplacian, but is really the Hessian $\nabla^2_{\vec{\lambda}} = H$],

$$\vec{\lambda}_{k+1} = \vec{\lambda}_k - \frac{\boldsymbol{\nabla}\phi(\vec{\lambda}_k)}{\nabla^2\phi(\vec{\lambda}_k)} \tag{2.19}$$

- Before we find an expression for $\vec{\lambda}_{k+1}$, recall the JACOBIAN is the generalization of the derivative operator to vector-valued functions. For the vector-valued function $\mathbf{f}(\vec{\lambda}_k) = \Big(f(\vec{\lambda}_k; x_1), f(\vec{\lambda}_k; x_2), \ldots, f(\vec{\lambda}_k; k_n)\Big)^T$,

$$\mathbf{J} = \begin{pmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial \lambda_1} & \cdots & \frac{\partial f_1}{\partial \lambda_n} \\ \frac{\partial f_2}{\partial \lambda_1} & \cdots & \frac{\partial f_2}{\partial \lambda_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \lambda_1} & \cdots & \frac{\partial f_m}{\partial \lambda_n} \end{pmatrix} \tag{2.20}$$

- The GRADIENT. Examining a single component,

$$\Big(\boldsymbol{\nabla}\phi(\vec{\lambda}_k)\Big)_j = \frac{\partial}{\partial \lambda_j} \sum_i [y_i - f(\vec{\lambda}_k; x_i)]^2 = \sum_i -2[y_i - f(\vec{\lambda}_k; x_i)] \frac{\partial f(\vec{\lambda}_k; x_i)}{\partial \lambda_j} \tag{2.21}$$

Therefore,

$$\boldsymbol{\nabla}\phi(\vec{\lambda}_k) = -2\,\mathbf{J}^T\mathbf{r} \tag{2.22}$$

where $\mathbf{r}$ is the vector of residuals, $r_i = y_i - f(\vec{\lambda}_k; x_i)$. NOTICE: $\mathbf{J}$ and $\mathbf{r}$ are both functions of $\vec{\lambda}_k$.

- The HESSIAN (what is the Hessian? See *Gradient, Jacobian, Hessian, Laplacian and all that*). Examining a single component,

$$\begin{aligned} H_{l,m} = \nabla^2_{\lambda_l,\lambda_m}\phi(\vec{\lambda}_k) &= \frac{\partial}{\partial\lambda_l}\frac{\partial}{\partial\lambda_m}\sum_i [y_i - f(\vec{\lambda}_k; x_i)]^2 \\ &= \frac{\partial}{\partial\lambda_l}\sum_i -2[y_i - f(\vec{\lambda}_k; x_i)]\frac{\partial f(\vec{\lambda}_k; x_i)}{\partial\lambda_m} \\ &= -2\sum_i \left(-\frac{\partial f(\vec{\lambda}_k; x_i)}{\partial\lambda_l}\frac{\partial f(\vec{\lambda}_k; x_i)}{\partial\lambda_m} + r_i\frac{\partial^2 f(\vec{\lambda}_k; x_i)}{\partial\lambda_l\partial\lambda_m}\right) \\ &\approx 2\sum_i \frac{\partial f(\vec{\lambda}_k; x_i)}{\partial\lambda_l}\frac{\partial f(\vec{\lambda}_k; x_i)}{\partial\lambda_m} = 2\mathbf{J}^T\mathbf{J} \end{aligned}$$

- Putting everything together,

$$\vec{\lambda}_{k+1} = \vec{\lambda}_k - (\mathbf{J}_k^T\mathbf{J}_k)^{-1}\mathbf{J}_k^T\mathbf{r} \tag{2.23}$$

When implementing this algorithm, we can determine $\mathbf{J}_k$ numerically (i.e. estimating the derivatives).

NOTICE: this procedure is sensitive to the initial condition $\vec{\lambda}_0$. The successive approximation is not guaranteed to converge, and if it does, it is not guaranteed to be the absolute minimum. Rather, different initial conditions may lead to different local minimums.

3 curve_fit with scipy.optimize

- In our discussion of the Gauss-Newton method, we saw that the solution to the least squares method can be numerically approximated. This is what often happens in practice, and is what we commonly mean by curve fitting.
- The package scipy has the function `scipy.optimize.curve_fit` which performs a least squares regression using the Levenberg-Marquardt algorithm This algorithm is similar to the Gauss-Newton method, with a few modifications that make it a bit more robust.
- First, let's create define our model, a line in this case,

```
def f(x, a, b):
    '''
```

```
        Parameters
                x: ndarray of x-values
                a: slope
                b: intercept

        Returns: ndarray of line
        '''
        return a*x + b

# plot
x = np.linspace(0,10,100)
y = f(x, 1, 5)
plt.plot(x,y)
plt.show()
```

- Generate some random data,

```
n = 20
x = np.linspace(0,10,n)
y = 2*x + 3 + np.random.normal(scale=3, size=n)
plt.plot(x,y,'.')
plt.show()
```

- Least squares regression with `curve_fit`

Parameters: **f** : *callable*

The model function, f(x, …). It must take the independent variable as the first argument and the parameters to fit as separate remaining arguments.

xdata : *array_like or object*

The independent variable where the data is measured. Should usually be an M-length sequence or an (k,M)-shaped array for functions with k predictors, but can actually be any object.

ydata : *array_like*

The dependent data, a length M array - nominally `f(xdata, ...)`.

p0 : *array_like, optional*

Initial guess for the parameters (length N). If None, then the initial values will all be 1 (if the number of parameters for the function can be determined using introspection, otherwise a ValueError is raised).

Returns: **popt** : *array*

Optimal values for the parameters so that the sum of the squared residuals of `f(xdata, *popt) - ydata` is minimized.

pcov : *2-D array*

The estimated covariance of popt. The diagonals provide the variance of the parameter estimate. To compute one standard deviation errors on the parameters use `perr = np.sqrt(np.diag(pcov))`.

```
popt, pcov = curve_fit(f, x, y)
```

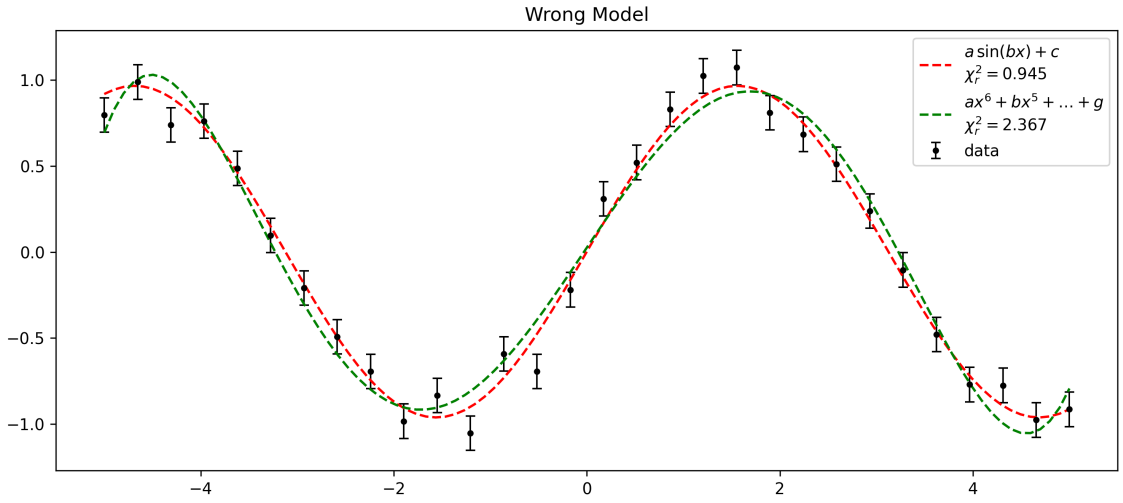
```
perr = np.sqrt(np.diag(pcov))

label = f'a={popt[0]:.2f}$\pm${perr[0]:.2f}'
label += f'\nb={popt[1]:.2f}$\pm${perr[1]:.2f}'

fig, ax = plt.subplots()
ax.plot(x,y,'.', label='data')
ax.plot(x, f(x,*popt), label=label)
ax.legend()
plt.show()
```

4 χ^2 Goodness-of-fit

- For a more thorough treatment of this topic: link
- How do we know our model is correct and that we have a good fit? *Can't we just look at the sum of the square residuals? If it's a small value, then the fit must be good.* NO! For example, we can have the wrong model, but still have small residuals,



The data points are generated by adding normal noise to a sin curve. Both a sin and an order-6 polynomial provide a good fit (by inspection). But the incorrect polynomial fit has a reduced $\chi^2 > 1$, indicating something is off!

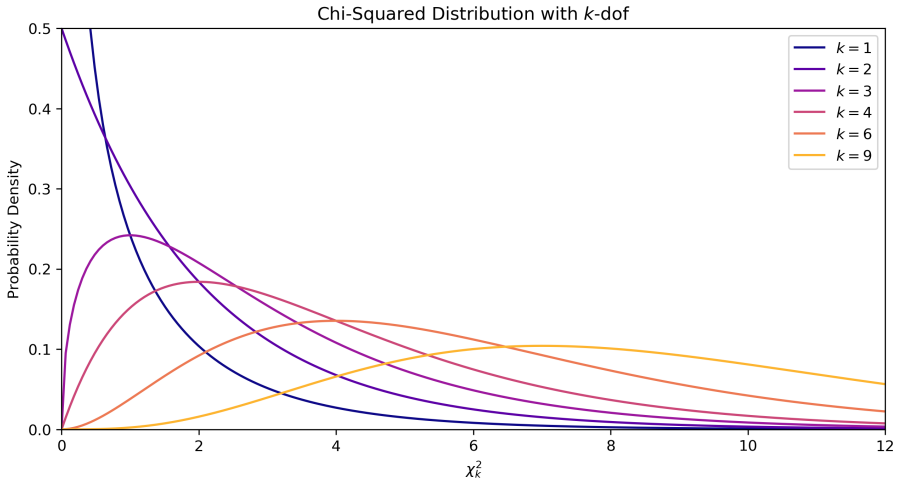
So what is this REDUCED CHI-SQUARED?

- CHI-SQUARED DISTRIBUTION: suppose $X_1, X_2, \ldots, X_k$ are k i.i.d. standard normal distributions $X_i \sim N(0,1)$. We define the chi-squared distribution with k degrees of freedom as the sum of the squared X_i distributions,

$$\chi_k^2 = \sum_{i=1}^{k} X_i^2 \tag{4.1}$$

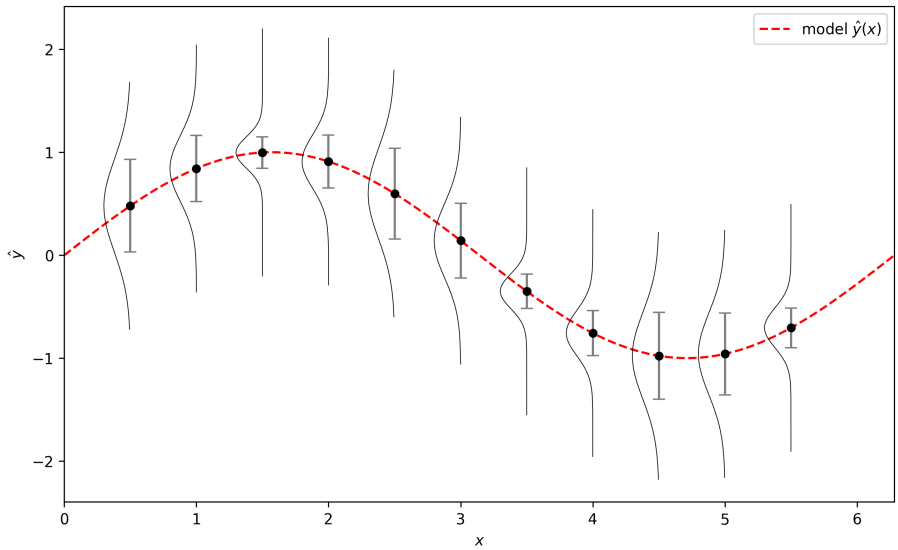
The chi-squared distribution has probability density function (derivation here),

$$f_{\chi^2_k}(x) = \frac{e^{-x/2}\ x^{(k/2-1)}}{2^{k/2}\ \Gamma(k/2)} \tag{4.2}$$



The distribution χ^2_k has mean $= k$ and variance $= 2k$. The expectation value makes sense: $E(\sum_i X_i) = \sum_i E(X_i) = k \cdot N(0,1)^2$ and the expectation value of the standard normal squared is 1.

- UNCERTAINTY: the data that we collect has uncertainty, i.e. our data will have some amount of error. We assume that our measurments are normally distributed and that our uncertainty for a given point is the standard deviation. [NOTE: chi-squared analysis hold ONLY if errors are normally distributed]



Then, for each x_i, we measure y_i with uncertainty σ_i, and,

$$\frac{y_i - \hat{y}(\vec{a}, x_i)}{\sigma_i} \equiv X_i = N(0,1) \tag{4.3}$$

- REDUCED CHI-SQUARED TEST: therefore,

$$\sum_n \left(\frac{y_i - \hat{y}(\vec{a}, x_i)}{\sigma_i} \right)^2 = \sum_i X_i^2 = \chi_k^2 \tag{4.4}$$

This is the expression for the χ^2 value for our data.

Q: For m data points and n fit parameters, how many dof are there? *A: we will use $m-n$ dof.* A hand-wavy argument is that there are m dof from the data points y_i, but n constrains because we are minimizing the squares w.r.t the n parameters. Hence, a total of $m-n$ dof. However, this is in general a tricky topic (see one interpretation).

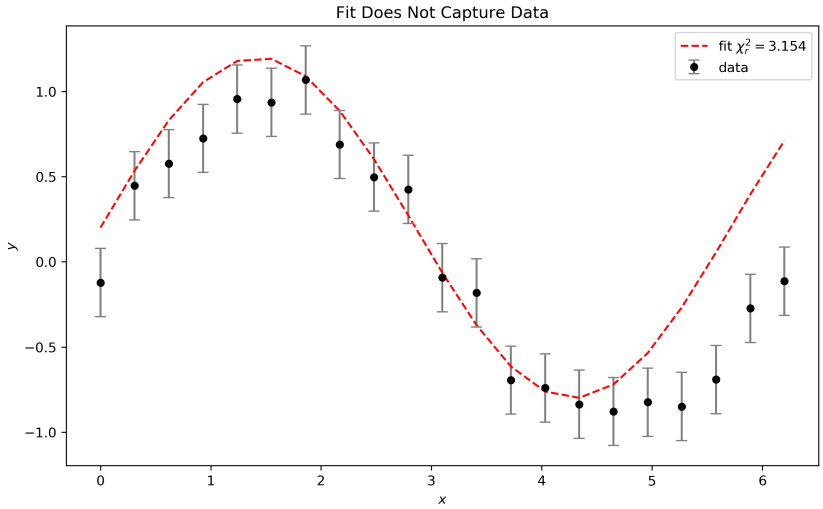
$$\text{dof} = m - n \tag{4.5}$$

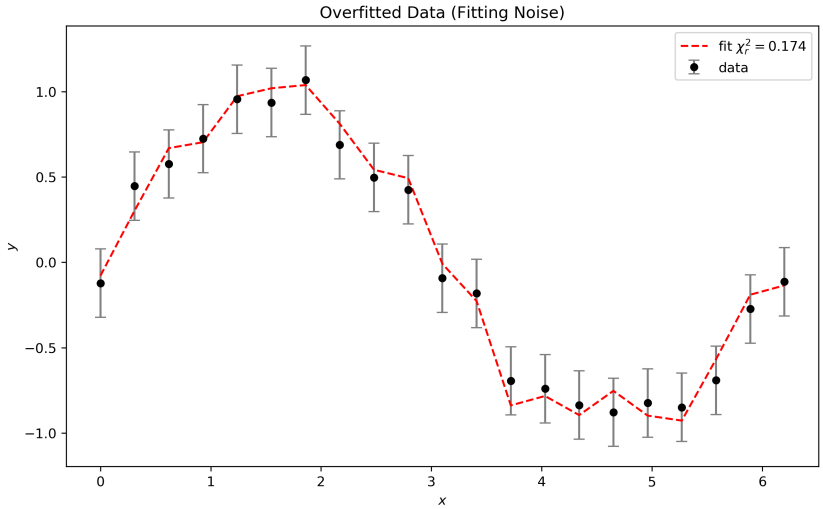
The reduced chi-squared is,

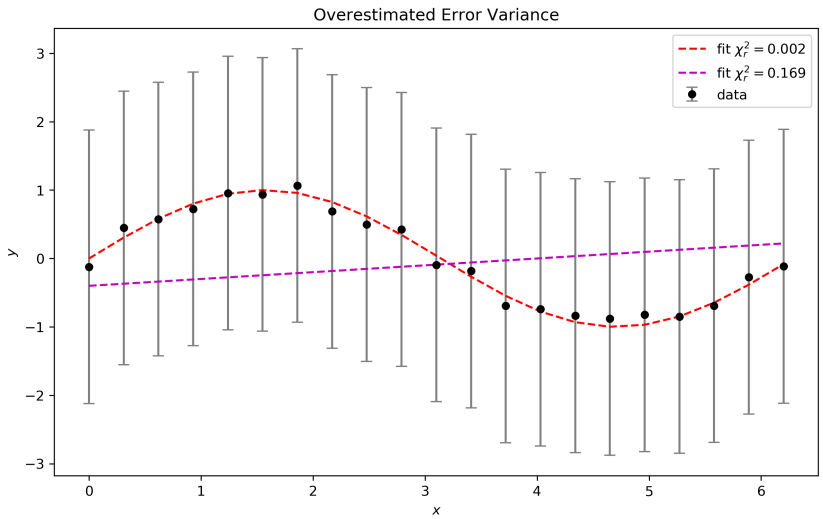
$$\boxed{\chi_r^2 = \frac{\chi_k^2}{k} = \frac{1}{\text{dof}} \sum_n \left(\frac{y_i - \hat{y}(\vec{a}, x_i)}{\sigma_i} \right)^2} \tag{4.6}$$

The χ_k^2 distribution has mean k, so we expect χ_r^2 to have mean 1. Isn't this a bit surprising?

1. $\chi_r^2 \gg 1$: the residues are too large (bad model, incorrect fit parameters) or the uncertainty is too small (we underestimated the uncertainty our of data so even a good model will have a large χ_r^2).
2. $\chi_r^2 \ll 1$: the residues are too small (over-fitted data, fitting noise) or the uncertainty is too large (we overestimated the uncertainty of our data so even a bad model will have a small χ_r^2).
3. The ideal value of χ_r^2 is not 0, which corresponds to no residuals and a perfect fit. This is because σ_i is never 0, we EXPECT our measurements to deviate by a reasonable amount and not perfectly match our model. The value WE EXPECT of χ^2 is 1 per dof.
4. RULE OF THUMB: a good fit has $0.8 < \chi_r^2 < 1.5$ (I couldn't trace where this comes from, but it's likely arbitrary to a certain extent)


Fit Does Not Capture Data
fit $\chi_r^2 = 3.154$
data
y
x


Overfitted Data (Fitting Noise)
fit $\chi_r^2 = 0.174$
data
y
x


Overestimated Error Variance
fit $\chi_r^2 = 0.002$
fit $\chi_r^2 = 0.169$
data
y
x

5 Summary

- We discussed the least squares method, and it's implementation in python.
- We explained the reduced chi-squared distribution and it's usefulness in determining goodness-of-fit.