Basic Plotting of Data

We have already seen some nice examples where plotting data was useful. In this section, we want to consider a "base" example, upon which other cases can be built (using matplotlib).

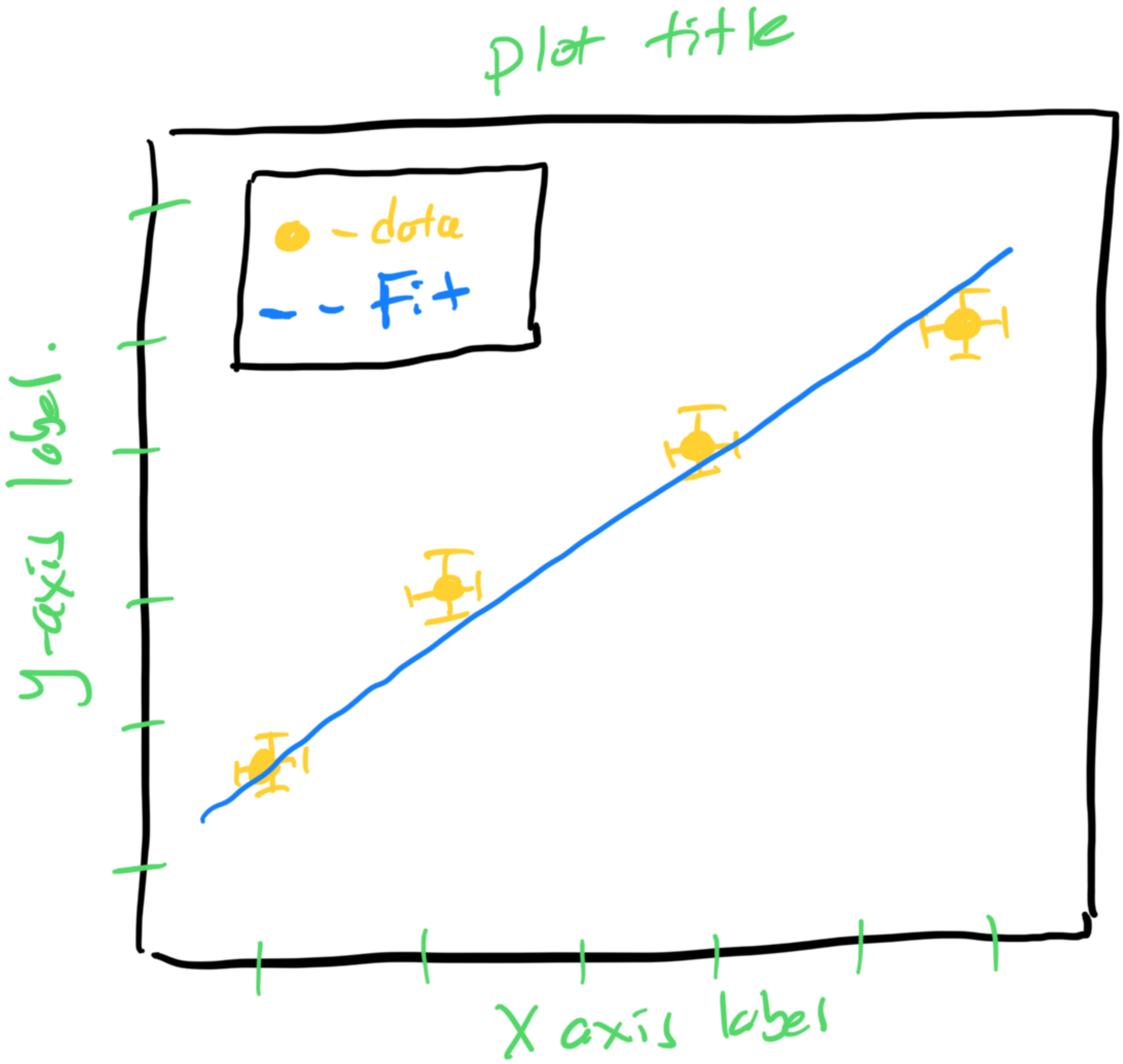
Base Case

We have a data file, in CSV format, of the form

```
XLabel, YLabel, dX Label, dY Label
  1.0 ,  3.0,  0.01,  0.03
  2.0 ,  3.2,  0.02,  0.02
   ⋮      ⋮      ⋮      ⋮
```

x $\quad y \quad \delta x \quad \delta y \quad N$ data points

We seek to create the following plot:



1. title, x-axis label, y-axis label
2. data points, with both x- and y- error bars
3. fit to the data, with

(i) fit parameters

extraction of ...

(ii) uncertainties in the fit parameters.

i.e. $y = (m \pm \delta m)x + (b \pm \delta b)$

(4) A legend, describing the data and the fit.

(5) Appropriate x and y axis limits

(6) Possibly, a choice of logarithmic x and/or y axes.

(7) Possibly, displaying an "error band" around the fit, to give a visualization of the fit parameter uncertainties.

Step 1: Read the data into appropriate data structures.

```
import csv

def read_data (filename):
    x = [ ], y = [ ], dx = [ ], dy = [ ]

    with open(filename, 'r') as file:
        reader = csv.reader (file)
        headers = next (reader, None)
        for row in reader:
            x.append (float (row[0]))
            y.append (float (row[1]))
            dx.append (float (row[2]))
            dy.append (float (row[3]))

    return headers, x, y, dx, dy
```

Step 2: Create a basic plot

```
import matplotlib.pyplot as plt

filename = "testdata.csv"

header_values, xi, yi, dxi, dyi
```

= read_data (filename)

```
plt.errorbar(xi, yi, dxi, dyi, "o")
plt.title("Basic Plotting Example")
plt.xlabel(header_values[0])
plt.ylabel(header_values[1])

plt.show()
```

→ at this point, we have a basic plot, suitable for fitting! Python will also figure out automatic axis limits, <u>BUT</u>: Be careful of y-axis zero suppression!!

```
plt.ylim(0,0)
```

→ Python will choose the upper limit!!

→ So, we have completed requirements ①, ②, and ⑤ above, and for this data, we do not need ⑥

Step 3 - Fitting.

Honestly, I could spend like half a course on this topic! And in PHYS 441, we kinda do. It's an amazing and wonderful part of data analysis. That encompases a lot of cool things in math and CS.

So, this is a <u>really</u> simple summary, and I am going to have to ask you to accept some things.

Step 3a: Choose a fitting function.

(i) may be from Theory

(ii) may be based on visualization.

In this case, I think that

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

$$(\text{pollen count}) = a(\text{temperature})^2 + b(\text{temperature}) + c$$

→ what I want is to calculate the "best fit" values of this fit function.

$a \pm \delta a$

$b \pm \delta b$

c ± o c

→ There are tons of packages to do this, but all of them will require us to specify the fit function!!

pointer!

```
def fitfunction(x, *param):
    return param[0] * x**x
        + param[1] * x
        + param[2]
```

Step 3b: Call the curve_fit package from scipy

```
from scipy.optimize import curve_fit
init_vals = [0.0 for x in range(3)]   # initize parameter values
popt, pcov = curve_fit(fitfunction,
```

Optimal fit parameters

covariance matrix (3 x 3)

xi, yi,
p0 = init_vals,
sigma = dyi,
absolute_sigma = True)

Step 3c:

Extract fit parameters and Uncertainties

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

a = popt[0], b = popt[1],
c = popt[2]

da = perr[0], db = perr[1],
dc = perr[2]

Step 3d:

→ Plot the fit

min (xi) - 2

xlow = ...

xhigh = max (xi) + 2

xfit = np.linspace (xlow, xhigh, 100)

yfit = fitfunction (xfit, *popt)

plt. plot (xfit, yfit, "r--")