FitGaussianBeam_18b

November 24, 2018

1 Fitting a Beam Profile to a Gaussian Beam

From the different papers, you can see that a Gaussian beam has a symmetric 2D intensity of

$$E(r) = E_{max} e^{-r^2/w^2},$$

where E_{max} is the electric field amplitude, r is the radius from the center of the beam, and w is the width of the beam where the electric field is $1/e$ of its maximum intensity. In Cartesian coordinates, $r^2 = x^2 + y^2$, so

$$E(r) = E_{max} e^{-(x^2+y^2)/w^2}.$$

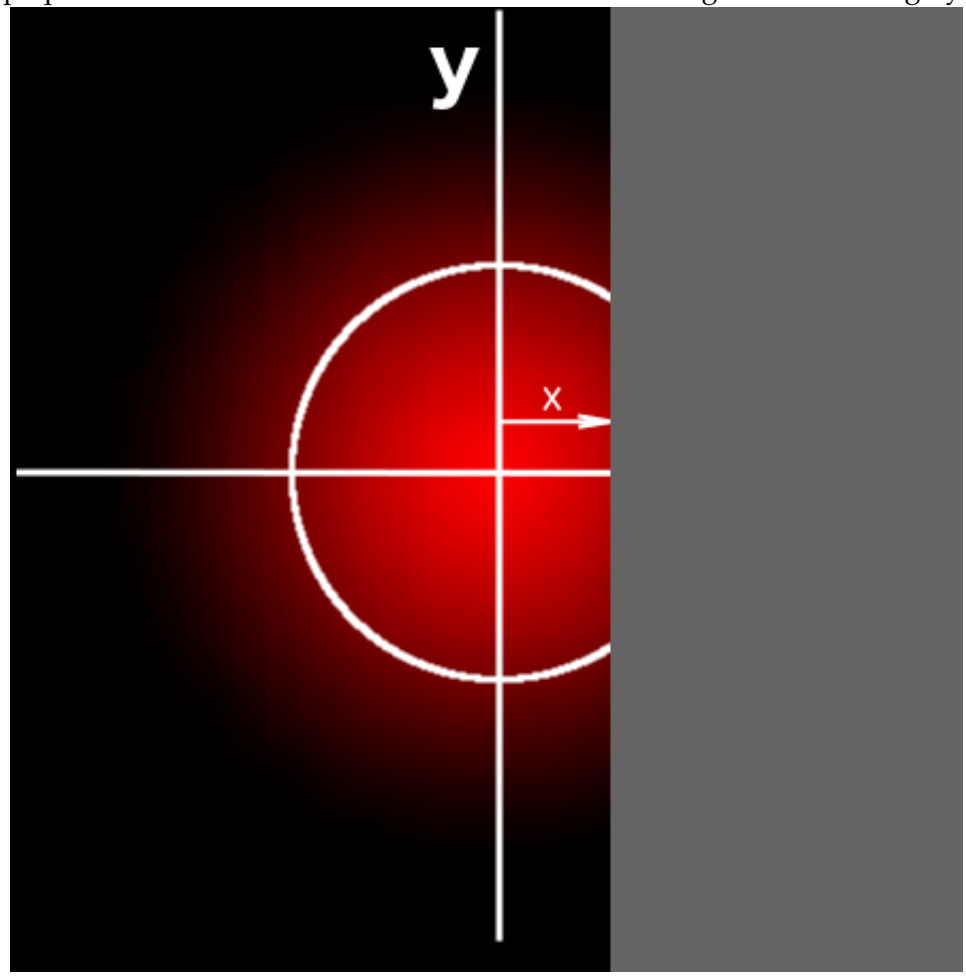
The intensity is proportional to E^2 so squaring the above result and grouping factors out front into I_{max}, you get

$$I(x, y) = I_{max} e^{-2(x^2+y^2)/w^2}.$$

Note that $I(r = w) = e^{-2} I_{max} = 0.1353\, I_{max}$.

1.1 Cutting a Beam

To profile a beam an opaque razor blade is moved across the beam. In the figure below the grey area is the razor blade.



You have to answer the question, "How much light is *passed*?" The answer is the integral

$$I_{profile}(X) = I_{max} \int_{x=-\infty}^{X} \int_{y=-\infty}^{\infty} dx\, e^{-2(x^2+y^2)/w^2} dx\, dy.$$

Fortunately because

$$e^{-2(x^2+y^2)/w^2} = e^{-2x^2/w^2} e^{-2y^2/w^2}$$

the integral factors into

$$I_{profile}(X) = I_{max} \left(\int_{-\infty}^{\infty} e^{-2y^2/w^2}\, dy \right) \left(\int_{-\infty}^{X} dx\, e^{-2x^2/w^2} \right).$$

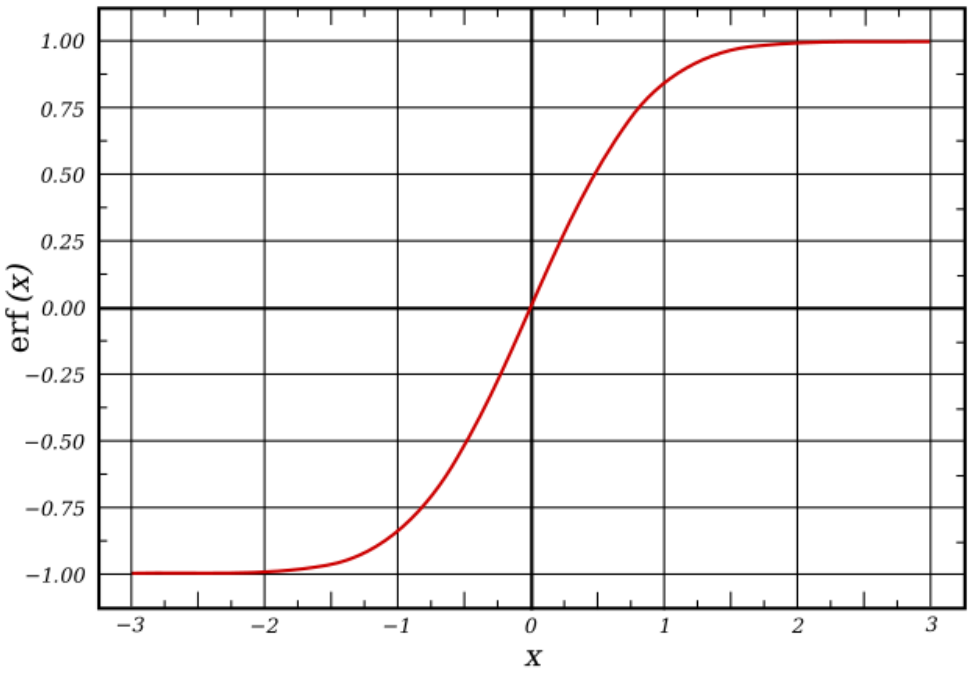
The first integral is a definite integral that evaluates to $w\sqrt{\pi/2}$, so you can just combine that back into I_{max}. So now you have

$$I_{profile}(X) = I_{max} \left(\int_{-\infty}^{0} dx\, e^{-2x^2/w^2} + \int_{0}^{X} dx\, e^{-2x^2/w^2} \right).$$

1.2 Doing the Integral

For the second integral, you can start with the definition of the *error function* or $erf(z)$ which is defined as

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$



A graph of the function is below

The `erf` function is in the `scipy` library. But you need to get your function in the same form as the $erf(x)$. This suggests making the substituion

$$\frac{2x^2}{w^2} = u^2$$

Then $u = \frac{\sqrt{2}x}{w}$ so $dx = \frac{w}{\sqrt{2}}du$. For the limits, when $x = 0$ then $u = 0$, when $x = -\infty$ then $u = -\infty$, and when $x = X$ then $u = \frac{\sqrt{2}}{w}X$. Putting this all together,

$$I_{profile}(X) = I_{max}\frac{w}{\sqrt{2}}\left(\int_{-\infty}^{0} du\, e^{-u^2} + \int_0^{\frac{\sqrt{2}}{w}X} dx\, e^{-u^2}\right).$$

To get this looking like the $erf()$, absorb the $\frac{w}{\sqrt{2}}$ into I_{max} then pull out a factor of $\frac{2}{\sqrt{\pi}}$. Next switch the limits of integration for the first integral. The result is

$$I_{profile}(X) = I_{max}\left(erf(\infty) + erf\left(\frac{\sqrt{2}}{w}X\right)\right).$$

$$I_{profile}(X) = \frac{1}{2}I_{max}\left(1 + erf\left(\frac{\sqrt{2}}{w}X\right)\right).$$

I put in a factor $\frac{1}{2}$ in of the last equation. This is the final form you will use to fit your data. Note the limits $I(-\infty) = 0$, $I(0) = \frac{1}{2}I_{max}$ and $I(0) = I_{max}$.

2 Code to Read Data and Fit It

2.1 Import Libraries

The first line allows plots to show up inline in the browser. The second line import the numerical array library. Next the standard plotting library, *matplotlib* is imported. It is a work-alike to the plotting in the commercial product `MATLAB`. From the `scipy` library you import the curve fitting module `curve_fit` and the `erf` function. Finally the `glob` function lets you final all of the file in a folder with a `.csv` extension.

Note: The `csv` format stands for `C`omma `S`eparated `V`alues. It is a common format for spreadsheet data. `csv` files can be opend and saved by `Excel` or your favorite spreadsheet program. It is also a text format, so it can easily be created by an Arduino or python program. You will see below it can be read into python with a single line of code.

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.optimize import curve_fit
        from scipy.special import erf
        from glob import glob
```

2.2 Read in Your Data

I put my data in a folder named `Data` to keep it separate from python code. The first seven lines find the `csv` files, prints them out in a numbered list, and asks which one yo uwant to open.

The `genfromtxt` function reads text files. In the function call I tell it the *comment* character, so it ignores lines beginning with this character. I also tell it the delimiter character to use, a comma, `,`.

The next group of lines read the data into two arrays, `pos` for the position, and `I` for the intensity. I then see if the data starts out at full intensity or zero intensity. If it is backwards, I reverse the data so it corresponds to the case you analyzed above.

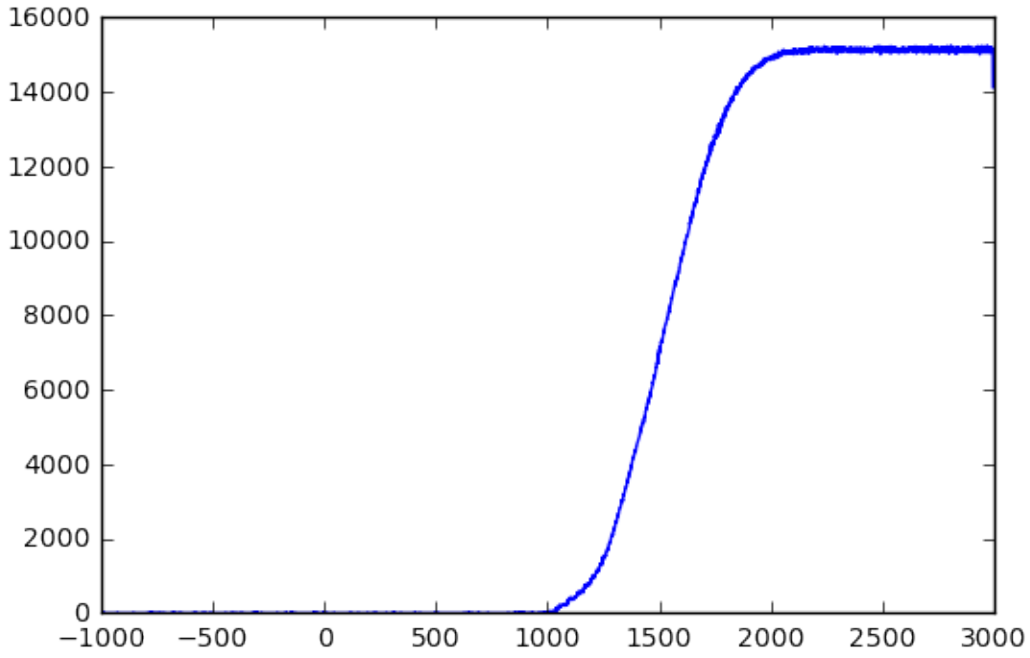
I do a quick plot so I can see the data right away.

Finally I grab the first part of the data file name, so plotted figures will have the same base name as the data file.

```
In [2]: dataFolder = "Data"
        globMatch = "*.csv"
        files = glob("/".join([dataFolder,globMatch]))
        for (i, f) in enumerate(files):
            print("{}: {}".format(i, f))
        iFile = int(input("Enter file number: "))
        dataFile = files[iFile]

        # Read the data into a variable named `data`, then separate it into positio
        print("Getting data from {}".format(dataFile))
        data = np.genfromtxt(dataFile, comments='#', delimiter=',')
        pos = data[:,2]
        I = data[:,3]
```

```
        # Find the median of the first and last ten lines.
        IStart = np.median(I[:10])
        IEnd = np.median(I[-10:])
        # If data starts with full intensity and ends completely blocked, flip the
        if IEnd < IStart:
            I = I[::-1]

        # Do a quick plot
        plt.plot(pos, I)

        # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
        # Grab the first part of the data file name so you can save figures with th
        idx = dataFile.rfind('/')
        if idx >= 0:
            fileBase = dataFile[dataFile.rfind('/')+1:]
        else:
            fileBase = dataFile
        print(fileBase)
        fileBase.index('.')
        figureFile = fileBase[:fileBase.index('.')] + '.png'
        print("Figure file is {}".format(figureFile))
```

```
0: Data/20181112a.csv
Enter file number: 0
Getting data from Data/20181112a.csv
20181112a.csv
Figure file is 20181112a.png
```



2.3 Estimate Fit Parameters

When ever you do a non-linear fit of a model to data, you usually have to estimate the parameters to get the fit function started.

Here is define the fit function first. The parameters are: * `pos` - the position * `I0` - the intensity of the full beam * `pos0` - the position of the center of beam * `width` - distance from the center of the beam to the e^{-2} point * `background` - a constant background term

See the code for how I estimate the parameters.

```
In [3]: # define the fit function
        # I added a constant background term in case there is stray light
        def beamProfile(pos, I0, pos0, width, background):
            return (0.5 * I0) * (1.0 + erf(np.sqrt(2)*(pos - pos0) / width)) + back

        # Estimate the starting and ending intensity
        IStart = np.median(I[:10])
        IEnd = np.median(I[-10:])

        # background estimate = starting intensity
        backgroundEst = np.min([IStart, IEnd])
        # I_Max (I0) intensity = ending intensity
        I0Est = np.max([IStart, IEnd]) - backgroundEst
        print IStart, IEnd, I0Est, backgroundEst

        # This finds the index of the data closes to half intensity.
        halfIdx = (np.abs(I - (0.5 * I0Est + backgroundEst))).argmin()
        # Then x0 is the position of the half intensity position
        posEst = pos[halfIdx]
        print halfIdx, posEst

        # Width estimate is distance from 1/e**2 intensity to half intensity
        eIdx = (np.abs(I - 0.5 * I0Est / np.e**2)).argmin()
        widthEst = np.abs(pos[halfIdx] - pos[eIdx])
        print eIdx, pos[eIdx], I[eIdx], widthEst, backgroundEst
```

```
0.0 15112.565 15112.565 0.0
2585 1519.49
2258 1200.78 1013.13 318.71 0.0
```

2.4 Plot the Data and Initial Guess

From my experience this is always a good step. You can check to see if your initial estimates are close.

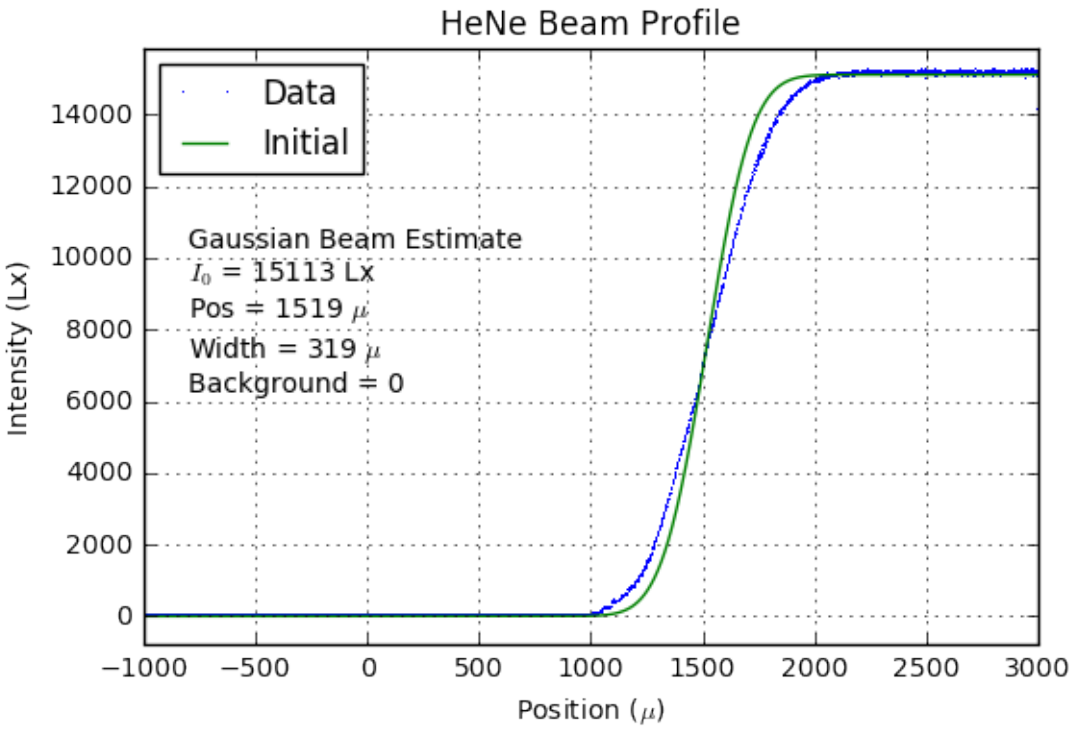
2.4.1 Good Python Plotting

This cell is also the first example of good `matplotlib` python plotting. See comments in the code below for details.

```
In [4]: pos0 = posEst
        width = widthEst
        I0 = I0Est
        background = backgroundEst

        # First create a "figure"
        fig = plt.figure()
        # Next add a set of axes to the figure
        ax = fig.add_subplot(111)
        # The basic plot command. The ',' plots a single pixel at the data positio
        # The `b` plots the data in the color blue
        # Add a label to this plot for use in the legend later.
        ax.plot(pos, I, ',b', label='Data')
        # I want to expand the y-axis a bit to see the bottom and top of the data
        # calculate the range of the data in y, then add 0.05 to the top and botto
        extraY = 0.05 * (I0 - background)
        ax.set_ylim(background-extraY,I0+background+extraY)
        # I think the grids look nice
        ax.grid()
        # Add another plot to the axes with the calculate model using the paramete
        # Make the line green with 'g' and add a label
        # The y values are the calculated function values
        ax.plot(pos, beamProfile(pos, I0, pos0, width, background), 'g-', label='I
        # Add the legend. `loc=0` places it in the upper right.
        ax.legend(loc=0)
        # Add x and y labels with units
        ax.set_xlabel('Position ($\mu$)')
        ax.set_ylabel('Intensity (Lx)')
        # Give the plat a title
        ax.set_title('HeNe Beam Profile')
        # Add text with the values of the estimated parameters
        # Note that I use a triple quoted string which extends over multiple lines
        # I also use the "new" formatting style. The string {:.0f} get replaced wit
        # in the format function. They are all formatted as floats with no decimal
        # Also note that $\mu$ is a Latex command that inserts a greek mu characte
        estText = \
            """Gaussian Beam Estimate
        $I_0$ = {:.0f} Lx
        Pos = {:.0f} $\mu$
        Width = {:.0f} $\mu$
        Background = {:.0f}""".\
            format(I0, pos0, width, backgroundEst)
        # Place the text in data coordinates. Note alignment
```

```
        ax.text(0.05, 0.7, estText, horizontalalignment='left', verticalalignment='
                transform=ax.transAxes)
```

```
Out[4]: <matplotlib.text.Text at 0x11031a510>
```



2.5 Do a Least-Squares Fit

I googled `scipy curve_fit` to figure out how to use the curve fit function.

```
In [5]: # pack the parameters into a numpy array
        initParams = np.array((I0Est, posEst, widthEst, backgroundEst))
        # This calls the non-linear curve fit function.
        # It returns `popt`, the optimal or best fit parameter valuse
        # It also returns the covarience matrix which has the uncertainties of the
        # And also the correlations in the paramters.
        popt, pcov = curve_fit(beamProfile, pos, I, p0=initParams)
        # print results
        print "Initial Parameters = ", initParams
        print "Best Parameters = ", popt
        perr = np.sqrt(np.diag(pcov))
        print "Uncertainty in parameters = ", perr

        # Unpack the best fit values into variables with more memorable names
```

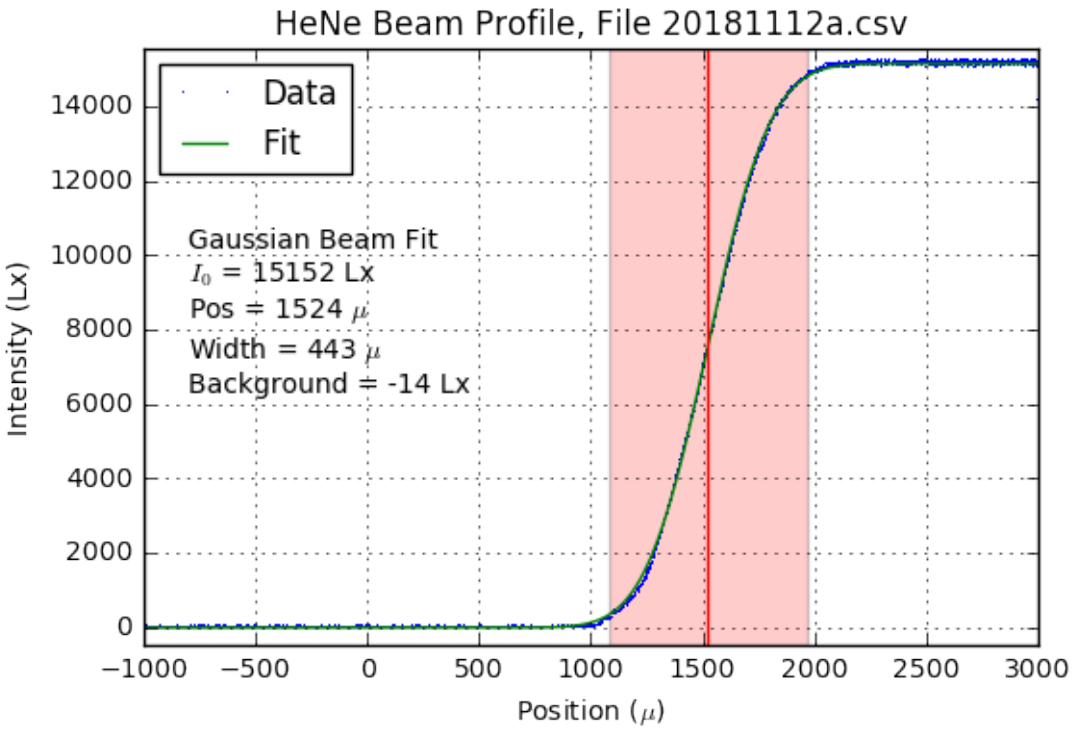
```
        (I0Fit, posFit, widthFit, backgroundFit) = popt
        # Calculate the function using the best fit parameter values.
        IFit = beamProfile(pos, I0Fit, posFit, widthFit, backgroundFit)

Initial Parameters =  [ 15112.565   1519.49     318.71        0.   ]
Best Parameters =  [  1.51524963e+04   1.52421788e+03   4.43147134e+02  -1.3532010
Uncertainty in parameters =  [ 2.19372119  0.11800652  0.32207942  1.23950717]
```

2.6 Plot the Data and Best Fit Function

This cell parallel the previous plot, but with the best fit parameters. I also do more decoration by showing a vertical line for the center of the beam and a shaded rectangle for the width of the beam. Note that the best fit function is almost right through the data points!

I also save the figure to a png file you can use in a document. I created the figure file name above. In the `savefig` function I use a dpi (dot per inch) of 300 to give a nice high resolution figure file.

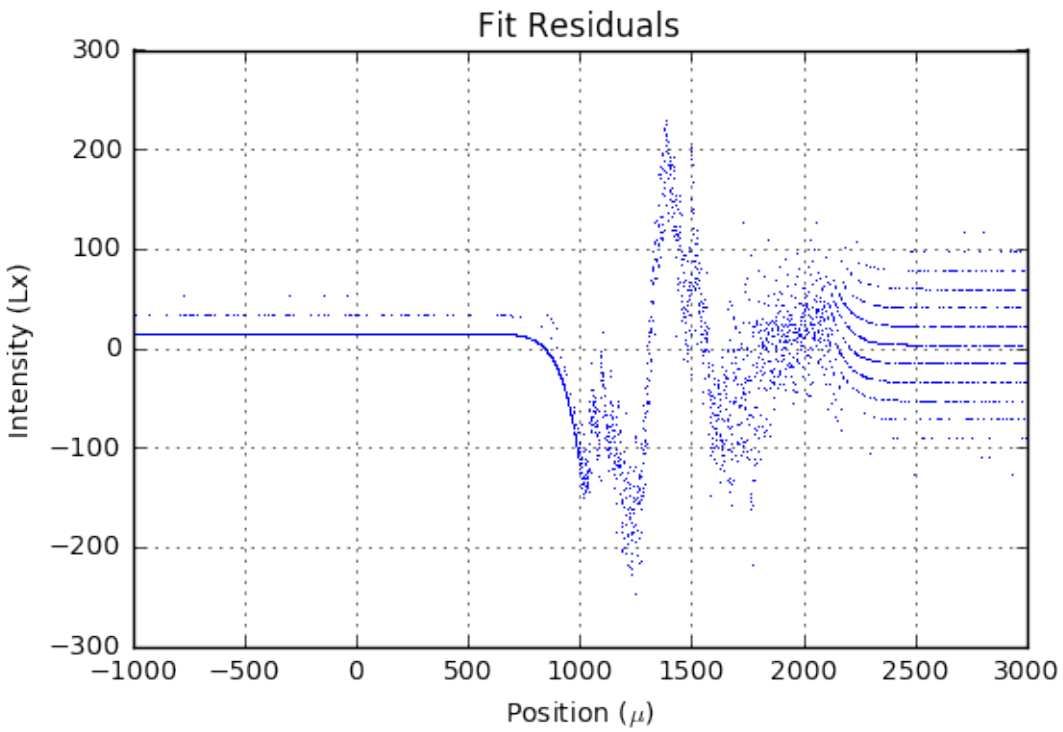
```
In [6]: fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(pos, I, ',b', label='Data')
        yLim = (IStart-0.03*I0Fit, IEnd+0.03*I0Fit)
        ax.set_ylim(yLim)
        ax.grid()
        ax.plot(pos, IFit, 'g', label='Fit')
        ax.legend(loc=0)
        fitText = """Gaussian Beam Fit
        $I_0$ = {:.0f} Lx
        Pos = {:.0f} $\mu$
        Width = {:.0f} $\mu$
        Background = {:.0f} Lx""".\
        format(I0Fit, posFit, widthFit, backgroundFit)
        ax.text(0.05, 0.7, fitText, horizontalalignment='left', verticalalignment='
                transform=ax.transAxes)
        ax.set_xlabel('Position ($\mu$)')
        ax.set_ylabel('Intensity (Lx)')
        ax.set_title('HeNe Beam Profile, File {}'.format(fileBase))

        # Decorate with a vertical red line at the center of the beam
        ax.plot((posFit,posFit),yLim, 'r', lw=1)
        # Now decorate the area showing the width of the beam
        # `alpha=0.2` makes the rectangle partially transparent
        ax.fill_betweenx(yLim, posFit-widthFit, posFit+widthFit, facecolor='red', a
        #ax.fill_betweenx(x, y1, y2, facecolor='gray', alpha=0.2)
        fig.savefig(figureFile, dpi=300)
```



2.7 Plot the Residuals

The *residuals* can help decide how good a fit is. The residuals are the difference between the data and the best fit function. An ideal fit would have the residuals randomly distributed around zero. This fit is pretty good. You can see though that around 1000 - 1300 μ the data are systematically below the function and between 1300 - 1600 μ, the data are high. Overall a deviation of 200 Lx out of 15,000 Lx is not much of an error.

```
In [7]: fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(pos[:-1], (I - IFit)[:-1], ',')
        ax.grid()
        ax.set_xlabel('Position ($\mu$)')
        ax.set_ylabel('Intensity (Lx)')
        ax.set_title('Fit Residuals')

Out[7]: <matplotlib.text.Text at 0x1113b8210>
```



3 Summary

This notebook gives you a tutorial to developing a function to fit a Gaussian profile beam, reading data from a file, programming a function to model the data, finding the best fit values for the function parameters, and, finally, making a publication quality figure.

```
In [1]: from IPython.core.display import HTML

        def css_styling():
            styles = open("custom.css", "r").read()
            return HTML(styles)
        css_styling()
```

```
Out[1]: <IPython.core.display.HTML object>
```

```
In [ ]:
```