1 Introduction

This read-me file describes a library written in C for fitting two-dimensional Gaussian functions. To test this library, we have a Python script which interfaces with it using ctypes [1]. The Python script generates a random set of points in $\mathbb{R}^2$ over a region x_{min}, x_{max}, y_{min} and y_{max} with a uniform distribution. We use $N = 10000$ and calculate the value $Z = f(X, Y) + \text{noise}$. Here, $f(X, Y)$ is a 2D Gaussian function:

$$f_{\overrightarrow{p}}(X, Y) = A \exp(-\frac{(X - \mu_x)^2}{2\sigma_x^2} - \frac{(Y - \mu_y)^2}{2\sigma_y^2}) + flr \tag{1}$$

The Python script also chooses the parameter vector of this function, $\overrightarrow{p} = [\mu_x, \mu_y, \sigma_x, \sigma_y, A, flr]^T$ at random. A successful test will retrieve the parameters used to generate the data set, within a margin of error. In addition, the Python script will graphically display the results of the fit in a plot.

2 Gauss-Newton method for iterative Least-Squares fitting

We use the Gauss-Newton method to fit a Gaussian function. This method can be summarized as follows:

Data: Column vectors of independent sample points $\overrightarrow{X}$ and $\overrightarrow{Y}$ and observations $\overrightarrow{Z}$ of length N

Result: Find the parameter vector $\overrightarrow{p}$ which minimizes $\left|\overrightarrow{Z} - f_{\overrightarrow{p}}(\overrightarrow{X}, \overrightarrow{Y})\right|$

First guess fit parameters $\overrightarrow{p} = \text{initial\_estimate}(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$;

while *True* **do**

  Evaluate the vector $\overrightarrow{r} = [r_1, r_2, ..., r_N]^T$ with $r_n = Z_n - f_p(X_n, Y_n)$;

  Evaluate Jacobean $\mathbf{J} = \begin{bmatrix} \frac{\partial r_1}{\partial p_1}, \frac{\partial r_1}{\partial p_2}, ..., \frac{\partial r_1}{\partial p_6} \\ \frac{\partial r_2}{\partial p_1}, \frac{\partial r_2}{\partial p_2}, ..., \frac{\partial r_2}{\partial p_6} \\ \vdots \\ \frac{\partial r_N}{\partial p_1}, \frac{\partial r_N}{\partial p_2}, ..., \frac{\partial r_N}{\partial p_6} \end{bmatrix}$;

  Solve the equation $\mathbf{J}\Delta\overrightarrow{p} = \overrightarrow{r}$ for the vector $\Delta\overrightarrow{p}$;

  Update the estimate of the parameters $\overrightarrow{p} = \overrightarrow{p} - \Delta\overrightarrow{p}$;

  if $|\Delta\overrightarrow{p}| < 0.001$ **then**

    exit loop;

  end

end

We will denote the fitted parameter vector as $\overrightarrow{\tilde{p}} = [\tilde{\mu_x}, \tilde{\mu_y}, \tilde{\sigma_x}, \tilde{\sigma_y}, \tilde{A}, \tilde{flr}]^T$ in the rest of our document.

2.1 finding the initial estimates

We estimate flr and A by simply finding the smallest and largest value of Z

$$flr^* = \min(Z) \tag{2}$$
$$A^* = \max(Z) - \min(Z) \tag{3}$$

Symbols with an asterisk mark * denote an initial estimate. The initial estimates for μ_x and μ_y are found by calculating the weighted center of mass of the (X, Y) sample points with the weight factors given by Z

$$\mu_x^* = \frac{\sum_{n=1}^{N}(Z_n - flr^*)X_n}{\sum_{n=1}^{N}(Z_n - flr^*)} \tag{4}$$

$$\mu_y^* = \frac{\sum_{n=1}^{N}(Z_n - flr^*)Y_n}{\sum_{n=1}^{N}(Z_n - flr^*)} \tag{5}$$

We find initial estimates for σ_x and σ_y by using the following property of the Gaussian function

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A\exp(-\frac{(X-\mu_x)^2}{2\sigma_x^2} - \frac{(Y-\mu_y)^2}{2\sigma_y^2})DST = 2A\pi\sigma_x\sigma_y \tag{6}$$

We approximate the double integral by a summation over Z and find:

$$\Delta X\Delta Y\sum_{n=1}^{N}(Z_n - flr^*) \cong 2A^*\pi\sigma_x\sigma_y \tag{7}$$

Here $\Delta X = \frac{x_{max}-x_{min}}{\sqrt{N}}$ and $\Delta Y = \frac{y_{max}-y_{min}}{\sqrt{N}}$. By assuming $\sigma_x \cong \sigma_y$ we can find approximate values for σ_x and σ_y individually.

$$\sigma_x^* = \sqrt{\frac{\Delta X\Delta Y\sum_{n=1}^{N}(Z_n - flr^*)}{2A^*\pi}} \tag{8}$$
$$\sigma_y^* = \sigma_x^* \tag{9}$$

3 Implementation and API

The main function which does the fitting of a two-dimensional Gaussian function has the following prototype

prototype	
	`int gaussian2DFit(double x[], double y[], double intensities[], int dataSize, double (*paramsArray)[])`
inputs	
x	Array representing the x-coordinate of the points where we have sampled the observable
y	Array representing the y-coordinate of the points where we have sampled the observable
intensities	Array representing the values of the observables
dataSize	The length of the x, y and intensities arrays
outputs	
paramsArray	The fitted parameters $[\mu_x, \mu_y, \sigma_x, \sigma_y, A, flr]$
return value	
EXIT_FAIL (1)	If the fit has failed (e.g. the data is too noisy)
EXIT_SUCCESS (0)	If the fit has succeeded

In addition to the main fitting function, the library also exposes some underlying functions which are useful for general fitting purposes.

prototype	

	`int gaussNewton(`
	`matrix x,`
	`matrix y,`
	`matrix paramsInit,`
	`int (*fitFunction)(matrix,matrix,matrix*),`
	`double (*derivatives[])(matrix, matrix),`
	`matrix *paramsFinal)`

description	This function uses the Gauss-Newton method to iteratively find the parameter vector $\overrightarrow{p}$ which minimizes $\left\|y - f_{\overrightarrow{p}}(x)\right\|$, where f is the function we are fitting our data to. Note that this implementation requires that the derivatives of the fit function have known closed forms.
data structures	

matrix	

```
// In this code, matrix data is stored in
// double arrays in a column wise fashion
// So if array A holds data of a 3-by-4
// matrix, this should be interpreted as
//
//       A[0] A[1] A[2]  A[3]
//  A =  A[4] A[5] A[6]  A[7]
//       A[8] A[9] A[10] A[11]
//
// Then, A.m = 3 and A.n = 4
typedef struct matrix
{
    double *data;
    int m;
    int n;
}matrix;
```

inputs

x	m-by-n Matrix holding the m independent variables in $\mathbb{R}^n$ space
y	m-by-1 Matrix holding the m dependent observables
paramsInit	p-by-1 matrix holding the initial estimates of p parameters
fitFunction	The first argument of this function is an m-by-n matrix holding the m independent variables in $\mathbb{R}^n$ space. The second argument is a p-by-1 matrix holding p parameters. The third argument is an output argument holding the result of the evaluation of the function at the locations of the independent variables, represented by an m-by-1 matrix. Memory for the matrix holding the evaluations will be allocated by the function being called
derivatives	An array of function pointers representing the derivatives of the fit function with respect to the parameter vector. The length of this function pointer array is equal to the number of parameters. The first argument of a derivative function is an m-by-n Matrix holding the m independent variables in $\mathbb{R}^n$ space. The second argument is a p-by-1 matrix holding p parameters. The functions will return a single double value

outputs

paramsFinal	The fitted parameters as an p-by-1 matrix

return value

EXIT_FAIL (1)	If the fit has failed (e.g. the data is too noisy or maximum iterations reached)
EXIT_SUCCESS (0)	If the fit has succeeded

prototype	`int linearLeastSquares( matrix X, matrix Y, matrix *b)`
description	Find **b** for which $\|\mathbf{Xb} - \mathbf{Y}\|^2$ is minimized. We use Cholesky factorization to accomplish this
inputs	
x	**X** is a m-by-n matrix
y	**Y** is a m-by-o matrix
output	
b	**b** is a n-by-o matrix
return value	
EXIT_FAIL	If the matrix $\mathbf{XX}^T$ is not invertible
EXIT_SUCCESS	If the fit has succeeded

4 Test Results

As explained in the introduction, a Python script is used to test the library and visualize the results. We use ctypes to interface to the library, matplotlib for plotting and numpy for general math and statistical functions.

An array X of random values between the limits x_{min}, x_{max} of length N represents the x-coordinates where we sample function 1. Similarly, an array Y between y_{min}, y_{max} of length N represents the y-coordinates. We then construct a Voronoi diagram based on these X and Y positions, where the interior color of the Voronoi cells are determined by the value of function 1 at that point. Figure 1 illustrates this.

On this Voronoi diagram we superimpose a plot of an ellipse given by the implicit equation;

$$\left(\frac{x-\tilde{\mu_x}}{\tilde{\sigma_x}}\right)^2 + \left(\frac{y-\tilde{\mu_y}}{\tilde{\sigma_y}}\right)^2 = 1 \tag{10}$$

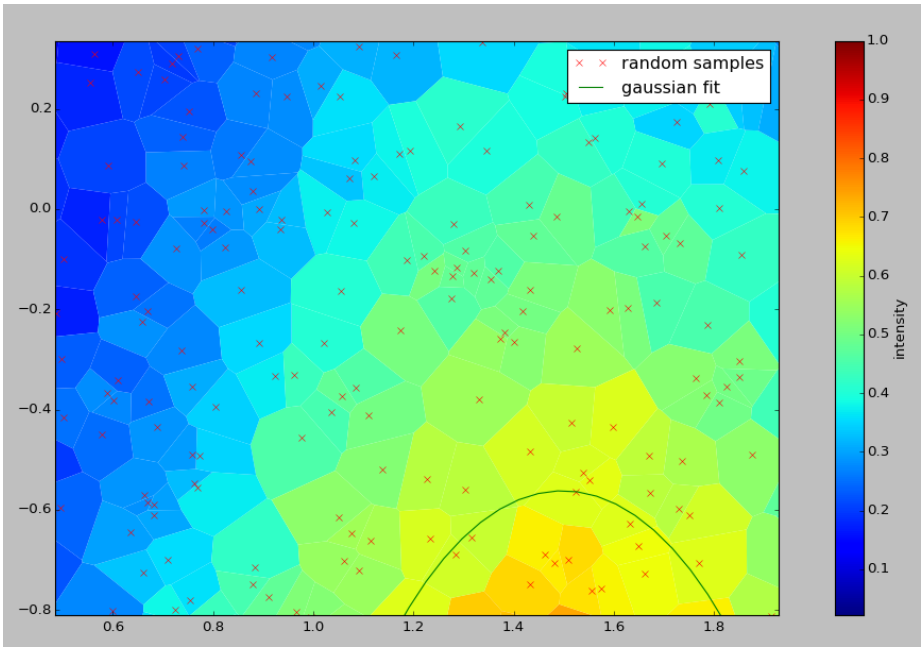

Figure 1: To visualize the fit results, we construct a Voronoi diagram based on the sample points (X, Y), shown here as the red crosses. The interior color of the Voronoi cells are based on the value of the observation Z at that point. The green solid line is an ellipse given by the implicit equation 10

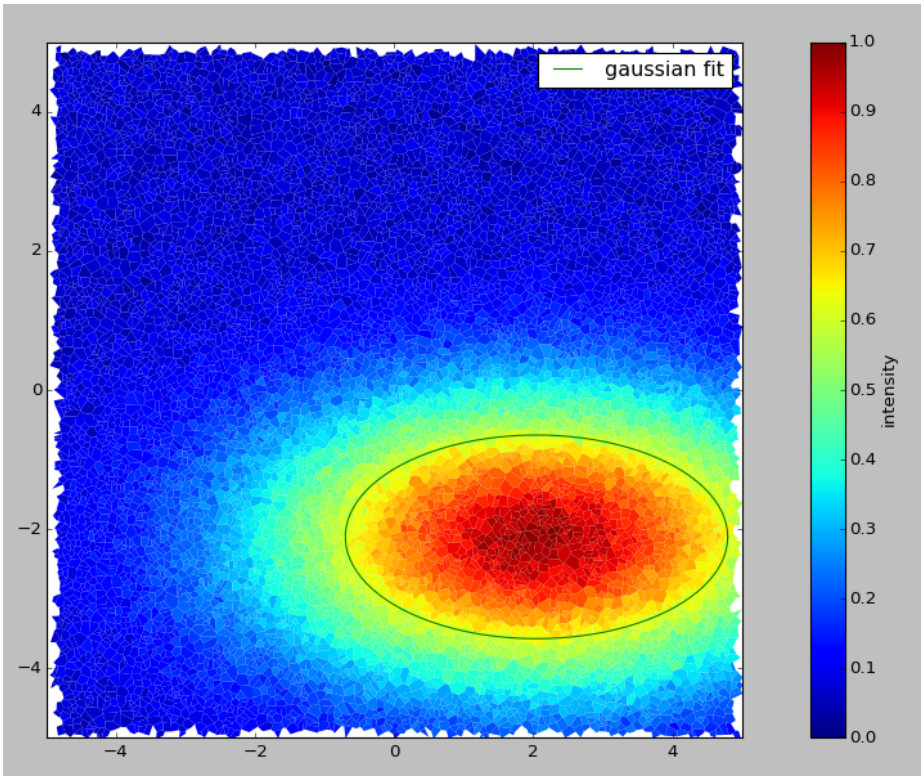

Figure 2: Here we see the results of our fitting library on the entire data set

To reiterate, $\tilde{\mu_x}$, $\tilde{\mu_y}$, $\tilde{\sigma_x}$ and $\tilde{\sigma_y}$ are estimates of the parameters used to generated the data set (X, Y, Z) by applying the least squares fit as outlined in section 2

The results of a complete fit are shown in figure 2.

References

[1] Guy K. Kloss. Automatic C Library Wrapping - Ctypes from the Trenches. *The Python Papers*, 3(3), December 2008.