Partly BEC image fitting

Li Jing

December 11, 2015

1 Density Distribution

In theory, partly BEC has following density distribution:

$$n(\mathbf{r}) = n_{th} Li_{3/2}(\prod_{i=1}^{3} \exp(-x_i^2/x_{i,th,0}^2)) + n_c \max(1 - \sum_{i=1}^{3} \frac{x^2}{x_{i,c,0}^2}, 0) \tag{1}$$

where $n_t h$ is thermal atom number, n_c condensated atom number. Li is polylog function, whose definition is

$$Li_n(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^n} \tag{2}$$

However, polylog function is quite difficult to use.

1. In matlab, polylog can only work for integal n.
2. In python, there is polylog in package **mpmath**. But it is in type **mpc**.
3. In order to fit, this function should work for array.

So usually, we replace polylog by Gaussian, which is good enough for thermal clouds.

NOTE: for fermion fitting, we should not replace polylog by Gaussian. Because the most important property in fermion T/T_F just represents the difference between polylog and Gaussian.

Now we can write down the approximate density distribution of the atom clouds.

$$n(\mathbf{r}) = n_{th} \exp(-\sum_{i=1}^{3} x_i^2/x_{i,th,0}^2) + n_c \max(1 - \sum_{i=1}^{3} \frac{x^2}{x_{i,c,0}^2}, 0) \tag{3}$$

2 fitting function

Our images are 2D. So we need to integrat our funciton in one dimension.

$$n_{2D} = n_{th} \exp(-\sum_{i=1}^{2} x_i^2/x_{i,th,0}^2) + n_c \max(1 - \sum_{i=1}^{2} \frac{x^2}{x_{i,c,0}^2}, 0)^{\frac{3}{2}} \tag{4}$$

But 2D image is still quite difficult to fit, or very slow. So usually, we integrate the image to 1D and fit these two directions separately. That is, we slide the image in x axis to fit y-distribution and then repeat by change x and y. Then, we need our 1D distribution:

$$n_{1D} = n_{th}\exp(-x_i^2/x_{i,th,0}^2) + n_c \max(1 - \frac{x^2}{x_{i,c,0}^2}, 0)^2 \tag{5}$$

3 Pseudo Code for fitting

```
function 2D_partlyBEC_fit(2D_image):
        \\ get 1D slides
        slidesX = sum(2D_image, 0);
        slidesY = sum(1D_image, 1);
        \\ define 1D distribution inside 2D fit:
        \\ CENTER is position for cloud center
        \\ SIGMA is guassian size, WIDTH is parabolic size,
        \\ OFFSET is background value.
        \\ N_G and N_P are atom numbers respectively.
        \\ x is coordinate.
        function 1D_partlyBEC(x, CENTER, SIGMA, WIDTH, N_G, N_P, OFFSET):
                return OFFSET + N_G * (-(x-CENTER)^2/SIGMA^2)
                        + N_P * max(0, 1-(x-CENTER)^2/WIDTH^2)
        \\ then do initial guess. This step is very important.
        Do initial guess

        \\ use built-in fit functions
        [centerX, sigmaX, widthX, n_GX, n_PX, offsetX]
        = built-in-fit-function(1D_partlyBEC, slidesX, initial guess)

        [centerY, sigmaY, widthY, n_GY, n_PY, offsetY]
        = built-in-fit-function(1D_partlyBEC, slidesY, initial guess)
        \\ combine two directions
        Transfer results to 2D

        return output
```

NOTE:

- Initial guess is very important before fit.
- Validations that make sure atom number positive are useful.
- Transfers from 1D to 2D are required before output.
- In python, *curve_fit* in scipy is good for fit.

4 Example

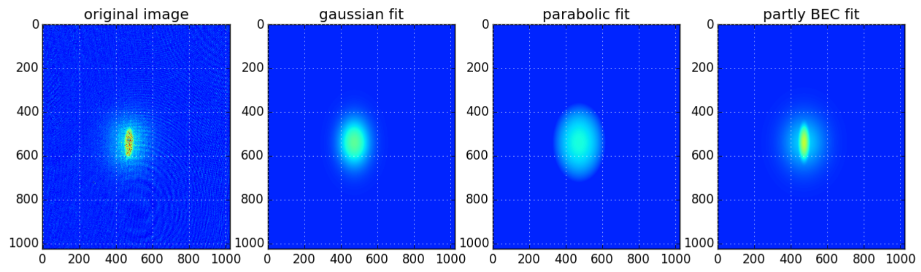

Figure 1: 60% condensate

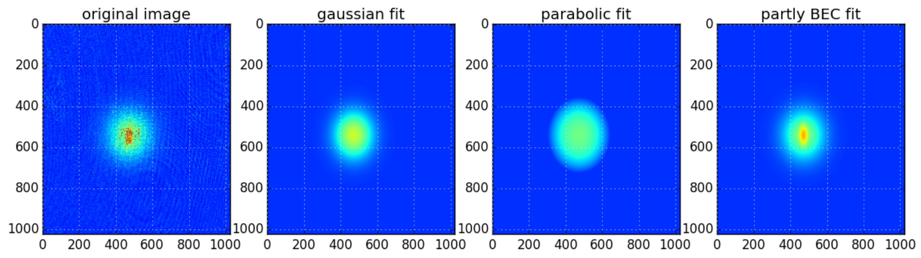

Figure 2: 30% condensate