第三回少数性生物学トレーニングコース　画像解析 (2)

三浦耕太　http://cmci.embl.de
所属：

1. 自然科学研究機構研究力推進本部欧州拠点
2. 基礎生物学研究所
3. EMBL・先端光学顕微鏡施設・分子細胞イメージングセンター（ドイツ）

20150804,5 於　吹田・大阪大学産業科学研究所

輝点の位置測定の精度

超解像度顕微鏡法では、画像中の輝点の座標を2次元ガウス分布をフィットして測定する。フィットの結果として、サブピクセル解像度の位置座標を得ることができ、それを元に点描(pointilism)によって精細な画像を再構成すつことができる。ここで注目するのは、フィットの結果得られる座標の信頼性である・

フィティングによる測定結果のクオリティは、正確さ（accuracy）と精度(precision)によって評価できる。

正確さとは、その点が本当にある座標をどの程度正確に測定しているか、ということで、真の座標と、測定した座標の間の距離がそれに相当する。0ならば無論よいのだが、そもそも真の座標を知ることは、あまり簡単なことではない。

例えば、重さであれば「真の重さ」とされる標準が存在し、それをどれだけ正確に測定できるか、ということで正確さを求めることができる。あるいは、pHの測定であれば、標準の溶液を使って、測定システムの正確度を求めることができる（心もとなくはあるが）。では位置に関しては、真の座標が既知のサンプルの測定を行って測定結果を検討すればよいではないか、ということになるのだが、真の座標が分かっているサンプル・測定環境を用意すること自体が難しい。このため、その正確度の判定は困難であり、シミュレーション画像を使った方法が一般的である。またDNAオリガミ等を利用した方法が現在模索されている。

測定の精度は測定値自体がどの程度の誤差を持つか、という、平均誤算(standard error of the mean)に他ならない。真の値を測定しているかどうか、ということとは関係なく、測定そのものの信頼性を評価するための数値である。超解像度顕微鏡法の輝点の測定精度は、輝点を構成する光子数が多ければ多いほど、精度が高まることが知られている。

精度推定の理論

ガウス分布をフィッティングすることによって得られる位置情報の精度は次の式が提唱されている(Thompson, 2002)。

$$\langle(\sigma)\rangle = \sqrt{\frac{s^2}{N} + \frac{a^2/12}{N} + \frac{4\sqrt{\pi}s^3b^2}{aN^2}}$$

ここで、sはPSF（点像分布関数）のs.d.を、Nはフォトン数、aはピクセルサイズ, bは背景ノイズ（背景光と検出器ノイズを含む）を表している。ルートの中は三つの項の足し算になっているが、それぞれの項は

- 第一項：　フォトンノイズによる誤差
- 第二項：　離散化ノイズによる誤差
- 第三項：　背景ノイズによる誤差

に由来する誤差であり、その和が全体の平均誤差となる。

この式の導き方は詳しい解説を新井さんが書いてくれたので参照にしてほしい。

なお、二次元のガウス分布のフィッティングの場合、先ほどの式を一般化し、精度は次のような式になる。

$$\langle(\sigma_i)\rangle = \sqrt{\frac{s_i^2}{N} + \frac{a^2/12}{N} + \frac{8\pi s^4b^2}{a^2N^2}}$$

iはxないしはy方向を表す。

一次元の位置測定とその精度の推定

輝点の位置の測定は通常二次元ないしは三次元画像でおこなうが、ここでは一次元で測定を行うことで手順を単純にし、「キャプチャーした光子数が多ければ多いほど、精度が高まる」ことを実測で確認することを目的とする。

測定の手順の流れは次のようである。

1. 画像の前処理。
2. 輝点をひとつ選ぶ
3. Y軸に沿って投射を行い、x軸に沿った輝度のプロファイルを得る。
4. この輝度のプロファイルを一次元ガウス分布関数でフィッティングする。ピークの位置と、分布の標準偏差を得る。標準誤差が「精度」となる。
5. 同じ測定をいくつかの輝点で行う。
6. 横軸に各輝点の総輝度（光子数に比例）、縦軸に精度をプロットする。
7. Thompsonの結果にあるような右下がりのカーブになることを確認する。

ステップ2と3は、スクリプトで自動化してある。

画像の前処理

取得した画像はかなり高い輝度のベースラインになる。この部分を差し引くために、各フレームで輝度の平均値を取得し、この平均値を全てのピクセルから引き算する。このためのコードを以下に示す。

https://gist.github.com/cmci/be839c4dfb09b4cd395f

```
1  # subtract mean
2  from ij import IJ
3  from ij.process import ImageStatistics as IS
4  #import math
5
6  imp = IJ.getImage()
7  stack = imp.getStack()
8  #stat = IS()
9  for i in range(stack.getSize()):
10     ip = stack.getProcessor(i + 1)
11     stats = ip.getStatistics()
12     meanInt = stats.mean
13     ip.add(-1 * round(meanInt))
14 imp.updateAndDraw()
```

コードの解説

1. 6行目：アクティブな画像をグラブ。
2. 7行目：ImageStackのインスタンスをstackとして取得
3. 9行目：各フレームを処理するためのループ。
4. 10行目：フレームのImageProcessorインスタンスをipとして取得
5. 11行目：ipの統計情報をImageStatisticsのインスタンスstatsとして取得。
6. 12行目：statsのフィールド値meanに平均輝度が格納されている。
7. 13行目：ipのピクセル値から、平均輝度を引く。
8. 14行目：impのインスタンスを再描画。

さて、測定にはこの引き算を行った画像を用いるが（以下、"測定画像"と呼ぶ）、輝点を探しやすくするために、探索用の画像を以下の手順で用意する。

1. 測定画像を複製する。[Image > Duplicate] （"Duplicate Stack"をチェックする）
2. メディアンフィルタにより、ノイズを除去する。[Process > Filters > Median...] （Radius=2とする）。
3. 1-100フレーム、101-200フレームの最大輝度投射画像を作成する。[Image > Stack > Z Projection...] 'Maximum'を選ぶ。フレーム範囲は上記の二種類。
4. 結果した投射画像が、探索用の画像である。

輝点の探索と測定

探索用の画像から、輝点を選び、それを四角ROIで囲んで選択する。測定用の画像に戻り、同じ位置にROIを作成する。ショートカットはcommand-shift-E。メニューからだと `[Edit > Selection > Restore Selection]`

この状態で、以下のコード（gaussFitPrecision1D.py）を実行する。

https://gist.github.com/cmci/d3c8eadb2a69abf04858

```
from ij import IJ
from ij.gui import Plot
from ij.measure import CurveFitter, ResultsTable

from java.awt import Color
import math, sys, jarray


## function for getting pixel intensity profile projected along y-axis.
def yprojection(ip):
    xprof = []
    for i in range(ip.getWidth()):
        col = [ip.getPixel(i, j) for j in range(ip.getHeight())]
        #print col
        xprof += [sum(col)]
    return xprof

## grab image.

imp = IJ.getImage()
imgtitle = imp.getTitle()
ww = imp.getWidth()
hh = imp.getHeight()
cf = imp.getSlice()

## get imageprosessor instance from ImagePlus instance
## if ROI is set, then use that ROI.

iporg = imp.getProcessor()
if iporg.getRoi() is not None:
    ip = iporg.crop()
    roi = iporg.getRoi()
    rx = roi.x
    ry = roi.y
    rw = roi.width
    rh = roi.height
else:
```

```
    ip = iporg
    rx = 0
    ry = 0
    rw = ww
    rh = hh

## projection along Y-axis, using the function defined above.

xprof = yprojection(ip)

## prepare index array.
xval = range(0,len(xprof))

## convert to java arrays.
xvalj = jarray.array(xval,  'd')
xprofj = jarray.array(xprof, 'd')

## curve fitting.
cf = CurveFitter(xvalj, xprofj)
cf.doFit(CurveFitter.GAUSSIAN)
para = cf.getParams()
offset = para[0]
height = para[1]
xpos = para[2]
sd = para[3]
R2 = cf.getFitGoodness()
formula = cf.getFormula()
print formula
print 'a =', offset
print 'b =', height
print 'c =', xpos
print 'd =', sd
print 'R^2=', R2

### total intensity within SD range
xmin = int(math.floor(xpos - sd))
xmax = int(math.floor(xpos + sd))

print xmin, xmax

totalint = 0
for i in range(xmin, xmax):
    totalint += int(xprofj[i] - round(offset))

print 'total intensity =', totalint

if totalint <= 0:
    print 'no photon found'
```

```
    sys.exit()

###### precision
# camera pixel 6.5um
# pixelsize 65nm

cp = 6500.0
pscale = 65.0
background = 0.5 ## estimated from vacant space

photons = round(totalint * 0.46) ## 0.46 from Arai
print 'photons =', photons

photonNoise = math.pow(sd*pscale, 2) / photons
pixelationNoise = math.pow(pscale, 2) / 12.0 / photons
backgroundNoise = 4 * math.pow(math.pi, 0.5) * \
   math.pow(sd*pscale, 3) * math.pow(background, 2) / \
   pscale / math.pow(photons, 2)

print 'ph noise error', photonNoise, '[nm^2]'
print 'pix noise error', pixelationNoise, '[nm^2]'
print 'back noise error', backgroundNoise, '[nm^2]'

precision = math.pow(photonNoise + pixelationNoise + backgroundNoise, 0.5)
print 'precision', precision, '[nm]'
print '----'
print 'xlocation ± precision', xpos * pscale , '±', precision, '[nm]'

###### plotting


xvalscaled = [x * pscale for x in xval]
factor = 10
fitx = range(len(xval) * factor)
fitx =  [x * 1.0 / factor for x in fitx]
fity = [cf.f(para, x) for x in fitx]
fitxscaled = [x * pscale for x in fitx]


datamax = max(xprof) * 1.1
datamin = min(xprof) * 0.9


fitplot = Plot("x-profile Gaussian fit", "x", "sum of intensity")
fitplot.setLimits(-1, xvalscaled[-1]+1, datamin, datamax)
fitplot.setColor(Color.red)
#fitplot.addPoints(xvalj, xprofj, Plot.CIRCLE)
fitplot.addPoints(xvalscaled, xprof, Plot.CIRCLE)
```

```
fitplot.setColor(Color.blue)
#fitplot.addPoints(fitx, fity, Plot.LINE)
fitplot.addPoints(fitxscaled, fity, Plot.LINE)
fitplot.show()

#### ResultsTable

rt = ResultsTable.getResultsTable()
cc = rt.getCounter()

rt.setValue("image", cc, imgtitle)
rt.setValue("rx", cc, rx)
rt.setValue("ry", cc, ry)
rt.setValue("rw", cc, rw)
rt.setValue("rh", cc, rh)
rt.setValue("offset", cc, offset)
rt.setValue("height", cc, height)
rt.setValue("xpos", cc, xpos)
rt.setValue("sd", cc, sd)
rt.setValue("R2", cc, R2)
rt.setValue("totalInt", cc, totalint)
rt.setValue("Photons", cc, photons)
rt.setValue("scaledLocation", cc, xpos * pscale)
rt.setValue("precision", cc, precision)

rt.show('Results')
```

コードの解説をしよう。全体は157行のコードであるが、110行目あたりからは、グラフのプロットと、結果を表に出力するためのコードである。一番重要なのは、カーブフィッティングの部分と、その結果を使った計算の部分になる。カーブフィッティングは次の部分である。

```
cf = CurveFitter(xvalj, xprofj)
cf.doFit(CurveFitter.GAUSSIAN)
para = cf.getParams()
offset = para[0]
height = para[1]
xpos = para[2]
sd = para[3]
R2 = cf.getFitGoodness()
formula = cf.getFormula()
```

CurveFitterのインスタンスを生成するコンストラクタが最初の行で、この生成の際に、xの配列と、yの配列を与える。なお、ここで使う配列はpythonの配列ではなくJavaの配列を使うことが必要となるため、事前にjarrayを使って変換を行っている。実際のフィッテングは二行目で行っている。引数は、フィットするモデル関数。その後の部分は結果の取得で、getParamsメソッドで返されるガウス分布の係数の値は配列

になっているため、それぞれインデックスを指定して取り出している。面倒に見えるかもしれないが、このあとに続く精度の計算は、式がややこしいのでちゃんと名前をつけておいたほうが混乱が少ない。

Resultsのウィンドウに表示される結果のうち、Precisionが精度、Photonsが光子数になる。

暗いものもふくめて、輝点を少なくとも50点測定してデータを集める。なお、測定に失敗したら、Resultsのウィンドウで最後の行を右クリックし、データを削除すると良い。

Thomposonのプロット

Resultsに集積した結果は、CSVファイルに保存してRなどでPrecision vs Photonsのプロットをするのが一番速いだろう。あるいはExcelでプロットすることも可能である。今回はせっかくなので、ImageJのプロット機能を使う。以下のコードを使って、プロットしよう。

https://gist.github.com/cmci/16377f3da76eed5cd455

```
from ij.measure import ResultsTable
from ij.gui import Plot
from java.awt import Color
import jarray

rt = ResultsTable.getResultsTable()
cc = rt.getCounter()

photonA = []
precisionA = []
for i in range(cc):
    photonA += [rt.getValue("Photons", i)]
    precisionA += [rt.getValue("precision", i)]

photonMax = max(photonA)
precisionMax = max(precisionA)
photonMin = min(photonA)
precisionMin = min(precisionA)

photonAj = jarray.array(photonA,  'd')
precisionAj = jarray.array(precisionA,  'd')

fitplot = Plot("Thompson Plot", "Photons", "Precision [nm]")
fitplot.setLimits(photonMin*0.9, photonMax*1.1, precisionMin*0.9, precisionMax*1.1)
fitplot.setColor(Color.red)
fitplot.addPoints(photonAj, precisionAj, Plot.CIRCLE)
fitplot.show()
```

参考文献

Thompson, R. E., Larson, D. R., & Webb, W. W. (2002). Precise nanometer localization analysis for individual fluorescent probes. Biophysical Journal, 82(5), 2775–83. doi:10.1016/S0006-3495(02)75618-X