U.U.D.M. Project Report 2015:37

Bootstrap Estimate of Localization Error in Super Resolution Microscopy

Elias Nygren

Examensarbete i matematik, 15 hp
Handledare och examinator: Ingemar Kaj
December 2015

Bootstrap Estimate of Localization Error in Super Resolution Microscopy

Elias Nygren
Uppsala University

December 28, 2015

Abstract

EMCCD cameras are commonly used in microbiology for the purpose of super-resolution microscopy and single-molecule tracking where the location of biofluorecent molecules are tracked within living cells. Due to noise in the EMCCD pictures a localization algorithm need to be used to estimate the position of the molecules. The purpose of this paper is to examine how a bootstrap algorithm can be implemented to determine the accuracy of the localization estimate.

Three bootstrap setups were tested against simulated EMCCD pictures generated from a statistical model with good results in two of the setups. Since the bootstrap algorithms in this paper uses the statistical model for sampling and no tests on real data were done, it is uncertain how well the algorithms will perform on actual images. Testing on the algorithms' sensitivity to deviations between the statistical model used and the real world is thus something that still needs to be done in order to determine its validity for practical use.

Contents

1 Introduction

EMCCD cameras are sensitive enough to detect single photons which is why they have found extensive use in many types of photon counting experiments and are used in varied fields such as microbiology, physics and astronomy. For the purpose of super-resolution microscopy and single-molecule tracking EMCCD cameras are able to depict fluorophores (biofluorescent molecules) within living cells. The clarity of the images produced by the EMCCD cameras is however limited by a number of issues involved with imaging of fluorophores such as photobleaching, motion blur and blinking. In addition to these factors EMCCD microscopy also contain three main noise sources coupled with the EMCCD technology namely the inherent noise of the light source, noise from an electron multiplication process within the microscope and Gaussian readout noise.

To determine the location of the fluorophores in EMCCD pictures effected by various degrees of noise, localization algorithms are used[1]. The accuracy of an implemented localization algorithm is however also effected by the noise of the image. Knowledge of the uncertainty within a given molecule location has the advantage of allowing for more advanced and accurate models over the movements of the molecules, where for instance given location points can be accompanied by probabilistic confidence regions.

In this paper the localization error will be estimated by means of a few different bootstrap approaches with the goal of determining their effectiveness as well as some of the factors which effect their precision.

2 Camera model

Light emitted from a light source will randomly distribute photons according to a point spread function which describe the intensity of light at various distances from the light source. The shape of the PSF of a stationary light source is dependent on whether the light source is in focus or not and is often modelled by an Airy-function as it contains surface waves. Due to the complexity and costly calculations involved with using Airy-functions and more realistic models a Gaussian approximation[1],

$$E(\boldsymbol{x}) = b + \frac{N}{\sqrt{2\pi\sigma_E^2}} exp(-\frac{1}{2}(\frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sigma_E})^2), \tag{1}$$

is often used to describe the PSF. The intensity E is here given for a vector point $\boldsymbol{x}$ in the xy-plane of a PSF with centre $\boldsymbol{\mu}$. The b parameter is the background intensity, N is the overall intensity of the PSF and σ_E is a width parameter. Since an EMCCD picture is divided into pixels where each pixel cover an area of the PSF, the intensity E_i over a pixel with index i is given by the integral over that pixel area,

$$E_i = \int_{pixel\#i} E(\boldsymbol{x}) d\boldsymbol{x}. \tag{2}$$

As this integral has no closed form solution it has to be approximated. In Mortensen et al.[2] a Taylor series expansion on $E(\boldsymbol{x})$ is used to find an approximation for E_i. The Matlab code for the approximation used for this paper was provided by Martin Lindén and Vladimir Curic and is presented in the appendix.

When a picture is taken the photons arrive at random to the pixels of the camera by a shot noise function determined by the intensity E of that pixel. The distribution of photons n over a single pixel with intensity E will thus follow the Poisson distribution with parameter E, that is

$$P(n|E) = \frac{E^n e^{-E}}{n!}. \tag{3}$$

To make sure the signal from the photons doesn't drown out in readout noise the signal is amplified by electron multiplication where each photon generate an electric pulse. The number of electrons generated from each pulse can be considered to be approximately exponentially distributed with, depending on the parameterization used, a rate parameter α or a gain parameter $\beta = \alpha^{-1}$, which is a known variable for the camera. Thus letting S be the total number of electrons registered in a pixel hit by n photons the variable S will be the sum of n exponentially distributed random variables and hence S will be gamma distributed with rate α and shape n. In other words

$$P(S|n) \approx \frac{\alpha^n S^{n-1} e^{-\alpha S}}{\Gamma(n)}(1 - \delta(n)) + \delta(n)\delta(S), \tag{4}$$

where the delta functions makes sure that the number of generated electrons is zero in cases where no photons hit. The readout from the camera is then also affected by a readout noise in the form of Gaussian white noise as well as a known constant, the camera offset, which is applied to eliminate negative image counts for pixels with low intensities. Let ξ denote the readout noise having distribution

$$p(\xi) \approx \frac{1}{\sqrt{2\pi\sigma_\xi^2}} exp(-\frac{\xi^2}{2\sigma_\xi^2}), \tag{5}$$

let $c0$ denote the camera offset and let the readout from the camera be c. The readout of an EMCCD picture is then $c = S + \xi + c0$, which can be expressed as a probability measure as

$$p(c|S, \xi) = \delta(c - c0 - S - \xi). \tag{6}$$

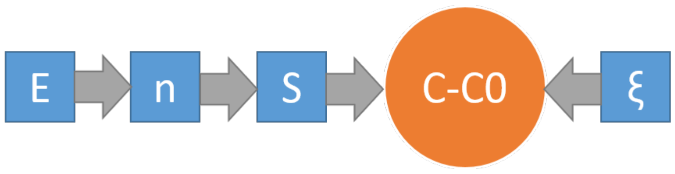


Figure 1: Relationship between the unobserved variables depicted as squares and the observed variable depicted with a circle.

.

The above model of estimations for E, $P(n|E)$, $P(S|n)$ and $P(\xi)$ is the camera model of Ulbrich and Isacoff[3] which has also been used by Mortensen et al.[2]. The dependencies between the variables can be drawn as in figure 1 where the squares indicate unobserved variables and the circle the observed one. It is worth noting that there are more ways of modelling the process in which the image count of an EMCCD picture is generated, for instance Tubbs[4] use a somewhat different distribution to describe the process in which the EM-register convert photons to electrons. For the purpose of this paper the only method analysed is that of Ulbrich and Isacoff[3].

The posterior distribution $p(\xi, S, n|c)$ will later on be needed for generating bootstrap samples via a Metropolis-Hastings algorithm, and can be derived using a Bayseian argument[1].
By using conditional relations,

$$\begin{aligned} p(c, \xi, S, n, E) &= p(c|\xi, S, n, E)p(\xi|S, n, E)p(S|n, E)p(n|E)p(E) \\ &= p(c|\xi, S)p(\xi)p(S|n)p(n|E)p(E). \end{aligned} \tag{7}$$

Integrating over E gives the probability

$$p(c, \xi, S, n) = p(c|\xi, S)p(\xi)p(S|n) \int p(n|E)p(E)dE. \tag{8}$$

Kolmogorovs definition,

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \tag{9}$$

can now be applied by setting $A = (\xi, S, n)$ and $B = c$ to give

$$p(\xi, S, n|c) = \frac{1}{p(c)} p(c|\xi, S)p(\xi)p(S|n) \int p(n|E)p(E)dE. \tag{10}$$

[1]The calculations to derive $p(\xi, S, n|c)$ was done by Martin Lindén, Department of Cell and Molecular Biology, Uppsala University.

Since the normalization constant $p(c)$ isn't needed in a Metropolis-Hastings algorithm it can be disregarded and we can settle for the distributions kernel,

$$p(\xi, S, n|c) \propto p(c|\xi, S)p(\xi)p(S|n) \int p(n|E)p(E)dE. \tag{11}$$

The integral of the kernel can be solved analytically if we define $p(E)$ by using a Jeffreys prior[5 6] which for a Poisson distribution is the improper distribution,

$$p(E) = 1/\sqrt{E}, \quad E > 0. \tag{12}$$

Using this distribution for $p(E)$ gives the integral as,

$$\int p(n|E)p(E)dE = \int dE \frac{e^{-E}E^n}{\Gamma(n+1)} E^{-1/2} = \frac{\Gamma(n+\frac{1}{2})}{\Gamma(n+1)}. \tag{13}$$

From this (11) can now be expressed as

$$\begin{aligned} p(\xi, S, n|c) \propto & \delta(c - c0 - S - \xi) e^{-\frac{\xi^2}{2\sigma^2}} \frac{\Gamma(n+\frac{1}{2})}{\Gamma(n+1)} \\ & \times \left\{ (1-\delta(n)) \frac{\alpha^n S^{n-1} e^{-\alpha S}}{\Gamma(n)} + \delta(n)\delta(S) \right\}. \end{aligned} \tag{14}$$

By noting that $\xi = c - c0 - S$ the kernel can be modified as a probability measure over only S and n,

$$p(S, n|c) \propto e^{-\frac{(c-c0-S)^2}{2\sigma^2}} \frac{\Gamma(n+\frac{1}{2})}{\Gamma(n+1)} \left\{ (1-\delta(n)) \frac{\alpha^n S^{n-1} e^{-\alpha S}}{\Gamma(n)} + \delta(n)\delta(S) \right\}. \tag{15}$$

3 Localization algorithm

For the main part of this paper a maximum likelihood estimate[7] of μ is used as the localization algorithm and if not otherwise specified μ^* is used to denote this estimator. For a given EMCCD picture $\hat{c}$ with pixel values $\hat{c}_i$ over k pixels the probability of pixel i having image count c_i equal to $\hat{c}_i$ given the intensity $E_i(\theta)$ can be expressed by the camera model as

$$p(c_i = \hat{c}_i | E_i(\theta)) = \sum_{S=0}^{\infty} \sum_{n=0}^{\infty} p(\xi = \hat{c}_i - S) p(S|n) p(n|E_i(\theta)). \tag{16}$$

The probability $p(c = \hat{c}|E(\theta))$, of getting an outcome where $c_i = \hat{c}_i$ for all i will then be

$$p(c = \hat{c}|E(\theta)) = \prod_{i=1}^{k} p(c_i = \hat{c}_i | E_i(\theta)). \tag{17}$$

The maximum likelihood estimate of μ can be found by finding the values $\theta = (b, N, \sigma_E, \mu)$ of the PSF which maximizes the probability $p(c = \hat{c}|E_\theta)$. Since the possible values of S and n could theoretically expand over all positive integers an estimate[2] of $log(p(c = \hat{c}|E_\theta))$ is used together with the fminunc matlab function to find the θ which minimizes
$-log(p(c = \hat{c}|E_\theta))$.

The MLE estimate is one of the most accurate localization algorithms to use for EMCCD pictures as discussed in the paper by Mortensen et al. [2]. The accuracy comes with a cost however as it does have the drawback of being rather costly to calculate since it contains numerous error functions which require a lot of computational power.

4 Variance within the camera model

The camera model can be used to simulate EMCCD pictures for a given PSF. The importance of the electron multiplication process can be visualized in figure 2. Since the number S of electrons generated from a single photon is distributed as $Exp(\alpha)$ with expected value α^{-1}, a camera calibrated with $\alpha^{-1} = 2\sigma_\xi$ will generate an expected image count equal to two standard deviations of the readout noise for each photon, thus virtually eliminating the effect of the readout noise.

Even though the $S + \xi$ image is immensely better than the $E + \xi$ image in figure 2 the randomness of the EM-register together with the still faint readout noise results in variance when the PSF parameters are estimated. Estimating the variance of the estimator μ^* for a given PSF can be done numerically by generating a large number n of images from the PSF using the camera model and then use the estimator μ^* on all the images resulting in n estimates. Let μ_i^* denote the i:th estimate and $\bar{\mu^*}$ the sample mean over all the estimates. The variance $Var(\mu^*) = \sigma_{\mu^*}^2$ can then be estimated by the usual sample variance

$$\sigma_{\mu^*}^2 \approx s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\mu_i^* - \bar{\mu^*})^2. \tag{18}$$

Using the E layout from figure 2 $1,000$ simulated EMCCD pictures were generated and an MLE estimate of μ were done on all pictures resulting in the plot in figure 3, with a standard deviation estimate of $\sigma_{\mu^*} = 0.1939$.

[2]The matlab code use to estimate $p(c = \hat{c}|E_\theta)$ was provided by Vladimir Curic, Department of Cell and Molecular Biology, Uppsala University.

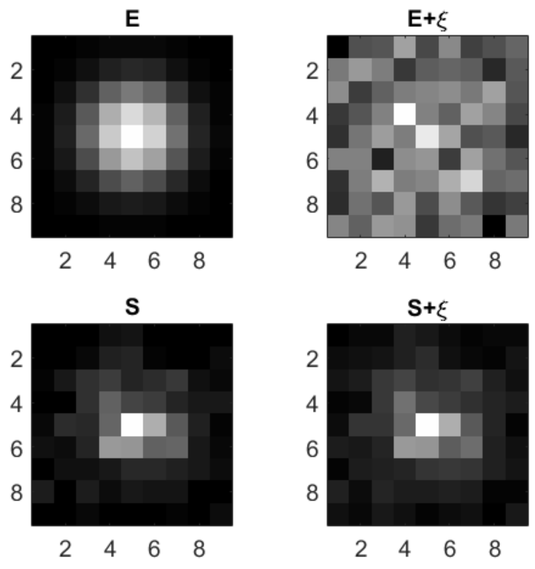


Figure 2: Effect of electron multiplication to increase image counts from photons. The generated images use $\alpha = 1/30$ and $\sigma_\xi = 15$ and each subplot is individually normalized so the overall image count in E is far less than the image count in S.

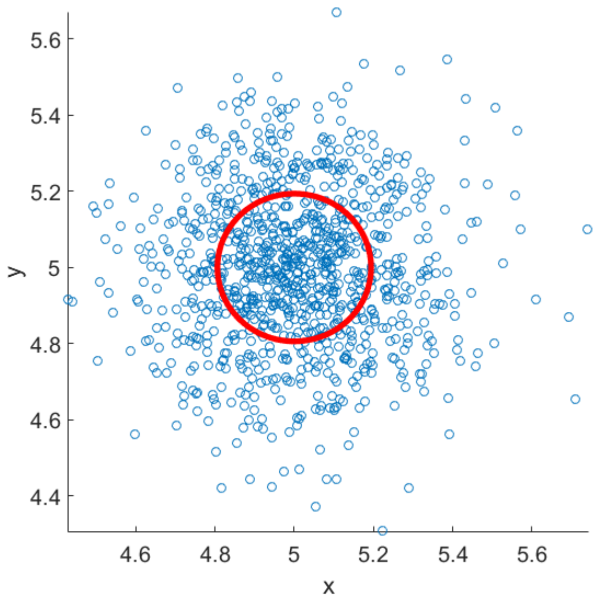


Figure 3: Scatter plot over $1,000$ μ estimates with σ_{μ^*} marked as the red circle.

The problem in reality is however how this variance can be estimated when all we have access to is the image count from the readout. The approach of bootstrapping used in this paper bears similarities to the numerical estimate above with the exception that new images are generated from a given readout instead of a given intensity.

5 Bootstrapping

The bootstrap method published by Efron[8] has been widely used ever since increasing computer power allows for more and more computationally heavy algorithms to be employed and new variations of bootstrap methods frequently pop up. The most straight forward method called non-parametric bootstrap is used in cases where we have a sample $X = (x_1, ..., x_n)$ of n i.i.d. observations from a distribution F_θ and would like to not only estimate θ by some estimator θ^* but also estimate the variance of the estimator θ^*. As long as X is a reasonable approximation of F_θ which can usually be assumed if n is large enough, then the bootstrap method can be used to estimate $Var(\theta^*)$. Algorithm 1 explains the procedure.

Algorithm 1 non-parametric Bootstrap algorithm

```
1: Given iid sample $X = (x_1, ..., x_n)$.
2: Initiate an empty vector $\theta^*_{bs}$ of length m for the bootstrap estimates.
3: for iteration $i = 1, ..., m$ do
4:     $X_{bs}$ = Resample $X$ with replacement.
5:     $\theta^*_{bs}(i) = \theta^*(X_{bs})$.
6: end for
7: Var($\theta^*$)=$\frac{1}{m-1}\sum_{i=1}^{m}(\theta^*_{bs}(i) - \bar{\theta^*_{bs}})^2$.
```

The number of bootstrap samples m used in the algorithm should be as large as possible to generate good results. The re-sampling in line four of the algorithm is done by drawing n samples from $(x_1, ..., x_n)$ where each value has an equal probability of being drawn each of the n times. The bootstrap samples will then contain tuples of some values resulting in different bootstrap samples. The bootstrap method is basically a way of viewing the sample X as the true population and its empirical distribution as the true distribution which is why the size of X is important. Note here that no assumptions of either distribution or estimator is used, the only requirement is that the observations are iid (independent and identically distributed) and sufficiently large to approximate the distribution of X.

To illustrate the basic procedure consider a sample $X = (x_1, ..., x_n)$ taken from a normal distribution $N(\mu, \sigma^2)$ where the parameters μ and σ are

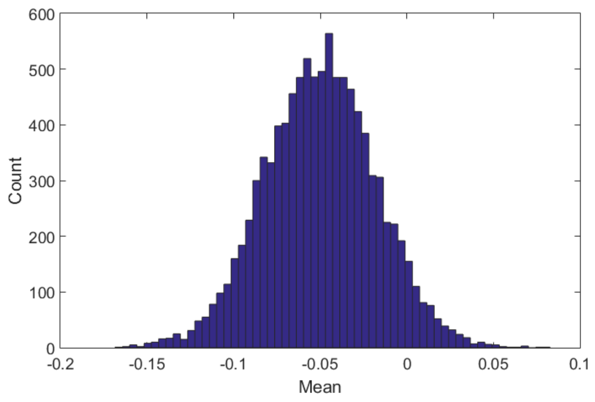


Figure 4: Histogram of mean estimates on bootstrap samples generated from original sample $X \sim N(0,1)$ having sample mean -0.0497.

unknown and we would like to estimate μ. To do this we use the maximum likelihood method giving us the estimator

$$\mu^*(X) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{19}$$

known as the sample mean. The variance of this estimator can easily be calculated to

$$\sigma^2_{\mu^*} = \frac{\sigma^2}{n} \approx \frac{s^2}{n}. \tag{20}$$

Since the sum of independent normally distributed random variables is again normally distributed we get,

$$\mu^*(X) \sim N(\mu, \sigma^2/n). \tag{21}$$

Suppose now that we instead use the bootstrap method as presented in algorithm 1 to estimate the variance $\sigma^2_{\mu^*}$ with $\theta^* = \mu^*$. Using Matlab with a sample X consisting of $1,000$ i.i.d. observations from a $N(0,1)$ distribution an estimate $\mu^*(X) = -0.0497$ is obtained (this is of course dependent on the random outcome of X). Using $10,000$ bootstrap samples of X gives us the estimate $\sigma^2_{\mu^*} = 0.001$ which align perfectly with the theoretical value $\sigma^2_{\mu^*} = \sigma^2/n = 1/1000 = 0.001$ from (20). A histogram of the values $\mu^*_{bs}(i)$, $i = 1, ..., 10,000$ is presented in figure 4, note that the mean of the bootstrap samples here are centred around the estimate of μ and not the actual true value $\mu = 0$.

The implementation in the case of EMCCD images would then be to generate new bootstrap images from a given image, use the localization algorithm on each of the bootstrap images and estimate the variance of the localization on the original image from the variance within the localization in the bootstrap images. The main problem here is how the bootstrap images should be generated. Since it is the photons that are i.i.d. and not the image counts, using Efrons bootstrap algorithm directly on the image counts of an EMCCD picture would be incorrect. If the camera output on the other hand would be the actual photon counts of each pixel bootstrapping would be quite straight forward as it could be performed as in algorithm 1.

6 Bootstrapping pictures of photon counts

To demonstrate how the bootstrap procedure would look in the case where the image count was of photons and not of electrons with added noise, let the given picture of photon counts be c containing n_i photons in pixel i with overall photon count $N = \sum n_i$. For the estimation of μ the sample mean in both the x and y direction of the picture is used as our estimator μ^*. The picture c can here be considered as a sample containing the pixel indexes of N photons where each index is an independent random variable determined by the distribution F_E given by the intensity E. The bootstrap procedure in this case is to draw each bootstrap sample from c or equivalently draw N samples from the empirical distribution with probabilities;

$$P(x_i^j = k) = \frac{n_k}{N}, \quad \forall k \in c. \tag{22}$$

x_i^j is here the pixel index of photon i in bootstrap sample j. Each bootstrap sample $c_{bs}^j = (x_1^j, ..., x_N^j)$ will then by line five in algorithm 1 result in a bootstrap estimate $\mu_j^* = \mu^*(c_{bs}^j)$. Finally line seven gives the sought after variance of the localization.

In figure 5 the numerical estimate is plotted against the bootstrapped estimate with the corresponding standard deviation of each estimate. Each point in the plot is generated from the same PSF where the over all intensity has been varied between the values $(50, 75, 100, ..., 525)$. For each intensity $1,000$ photon-count images were generated and for each generated image an additional $1,000$ bootstrap images were generated. For both the generated images and the bootstrapped ones the localization algorithm was then used resulting in $1,000$ μ^* estimates and $1,000,000$ μ_{bs}^* estimates. The numerical estimate of $Var(\mu^*)$ is then calculated by the sample variance of μ^*. The $1,000,000$ bootstrap samples gives $1,000$ bootstrapped variance estimates $Var(\mu_{bs}^*)_i$ corresponding to each of the generated images where the i subscript is the image index. The bootstrapped estimate in each point in the plot is then the sample mean

$$\sigma^2_{\mu_{bs}^*} = \frac{1}{1000} \sum_{i=1}^{1000} Var(\mu_{bs}^*)_i \tag{23}$$

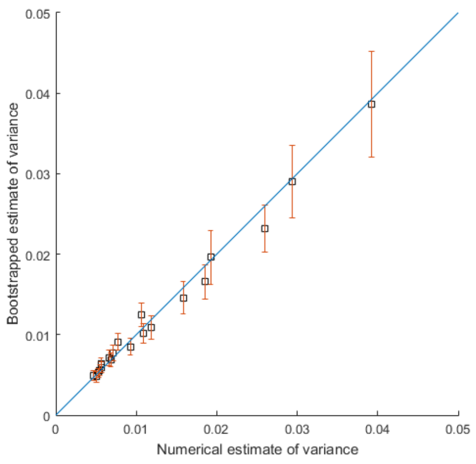


Figure 5: Numerical variance estimate plotted against bootstrapped variance estimate with standard deviation. The pictures used here are of photon counts and not generated EMCCD pictures.

and the given error bars of the points is the standard deviation of $Var(\mu^*_{bs})_i$.

As the plot shows the bootstrapped estimations correspond well with the numerical estimates with a proportional increase in variance as intensities drop. At low intensities the confidence region for the bootstrap estimates increase as well even though the number of bootstrap samples stay constant over all intensity levels. The reason for this is a direct result of lower intensities having a smaller sample of photons resulting in a sample containing less information on the density F_E. One of the conditions for bootstrapping as previously explained is that the sample used should be a good representation of the population which get progressively worse as intensities drop. The bootstrap estimate of variance will thus itself have an inherent variance and give more uncertain estimations for lower intensities.

As the additional aspect of the EM-register and readout noise is added the above method encounters problems since we no longer have image counts of photons but rather image counts $c = c0 + S + \xi$, which are no longer independent as they arrive to a large extent in clusters of electrons S by the EM-register. Bootstrapping with the offset $c0$ still present will cause the sample which is bootstrapped to contain multiple entries which are not random to be considered as a result of the PSF. The effect of $c0$ will because of this decrease the contribution of S and ξ in each bootstrap sample and in effect produce bootstrap images with a lower variance. If on the other hand $c0$ is removed before the bootstrap algorithm is applied the additional problem of the readout noise producing negative values needs to be dealt with. By removing $c0$ and setting all negative values to zero one could bootstrap the image as is done for an image of photon counts but the assumption that each image count in this case would be independent leads to issues. Since the photon count involved with creating an EMCCD picture is hidden and considerably lower than the image count of $S + \xi$, by ignoring the dependency between photon counts and electrons the sample changes from being viewed as a hidden sample of n events to be a much larger sample of $S + \xi$ events. Viewing the sample as $S+\xi$ events will lead to the misinterpretation that the image contain more information of the underlying distribution than it actually does and produce bootstrap images which are too similar to the EMCCD picture they were generated from.

One solution to this problem is to simply try and include the dependency between photons and the electrons into the bootstrap method and correctly bootstrap from a sample of close to n independent events. Since the values of n and S aren't known the bootstrap method need to be used together with some inference regarding them. For this paper this inference was conducted with a Metropolis-Hastings algorithm which samples values of n and S from $p(S, n|c)$.

7 Metropolis-Hastings

7.1 Introduction

To generate n, S and ξ from a given picture c a Metropolis-Hastings[9] algorithm can be used on the distribution $P(S, n|c)$. The remaining ξ values can then be derived from $\xi = c - c0 - S$. The benefit of using a Metropolis-Hastings algorithm for this case is that it works even in cases where only the kernel of the distribution is known which is the case for the $P(S, n|c)$ distribution. The MCMC algorithm is presented in algorithm 2.

Here π is our target distribution $P(S, n|c - c0)$. The Markov Chain denoted as x in the algorithm will convert to its stationary distribution π after a certain unknown number of iterations. Meaning that $x^{(i)}$ will be distributed as π if i is sufficiently large.

Even though there is no way to know for sure when the chain has converged there are a number of different convergence diagnostics which gives a good indication of both how fast the chain converges to the stationary distribution and how well the chain explores the probability space of the target distribution which is an aspect known as 'Mixing'. The choice of the proposal density $q(x^{(i)}|x^{(i-1)})$ is crucial for getting acceptable levels of mixing as well as convergence.

After the chain x has been generated the first few values must be removed

Algorithm 2 Metropolis-Hastings algorithm

```
 1: Define the proposal distribution $q(x^{(i)}|x^{(i-1)})$ and $x^{(0)}$
 2: for iteration $i = 1, 2, ...$ do
 3:     Propose: $x^{cand} \sim q(x^{(i)}|x^{(i-1)})$
 4:     Acceptance probability:
          $\alpha(x^{cand}|x^{(i-1)}) = \min(1, \frac{\pi(x^{cand})q(x^{(i-1)}|x^{cand})}{\pi(x^{(i-1)})q(x^{cand}|x^{(i-1)})})$
 5:     Draw $u \sim \text{Uniform}(0, 1)$
 6:     if $u < \alpha$ then
 7:         $x^{(i)} \Leftarrow x^{cand}$
 8:     else
 9:         $x^{(i)} \Leftarrow x^{(i-1)}$
10:     end if
11: end for
```

since these have not yet converged to the stationary distribution of the chain. The number of initial chain values which are removed is known as the 'burn' of the chain and can be considered as sort of a warm up of the algorithm to find its bearing towards its stationary distribution. After the burn has been removed the remaining part of the chain will be correlated by an auto correlation function determined by the proposal distribution used. Since we need independent samples, lowering the correlation within the samples is crucial and can be achieved by only including every n:th entry in the chain. The size of n is known as the sweepsize of the chain.

7.2 Proposal Distribution

For notation purposes let δ_c denote $c - c0$ and δ_{ci} the image count of $c - c0$ in pixel i. An initial guess of a good proposal distribution is often one which closely resembles the target distribution since this will insure that our candidate move x^{cand} will have a good spread over the probability space and a high acceptance probability. For instance the case where $q(x^{(i)}|x^{(i-1)}) = \pi$ will give the acceptance probability in line 4 of algorithm 2 as

$$\alpha(x^{cand}|x^{(i-1)}) = \min(1, \frac{\pi(x^{cand})\pi(x^{(i-1)})}{\pi(x^{(i-1)})\pi(x^{cand})}) = 1. \tag{24}$$

As an initial guess of a viable proposal distribution some plots of the density for higher intensities seem to be shaped as a bivariate normal distribution over the (n,S) space. Because of this setting

$$q(x^{(i)}|x^{(i-1)}) \sim \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \boldsymbol{\mu} = x^{(i-1)} \tag{25}$$

seem like a natural choice. $\boldsymbol{\Sigma}$ should have zero entries except for its diagonal consisting of σ_n^2 and σ_S^2. The question now remains on what constitutes good

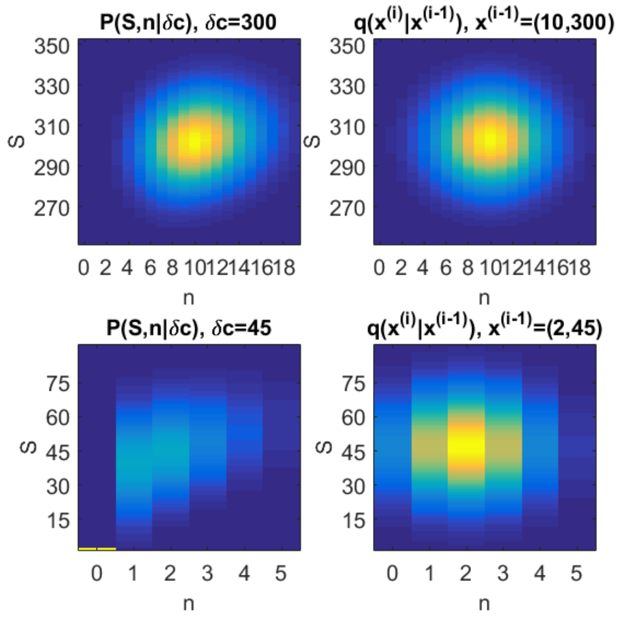


Figure 6: Plot of the target density $P(S, n|\delta_c)$ and the proposal density of equation 14 for $\delta_c = 300$ and $\delta_c = 45$.

σ_n^2 and σ_S^2 values to be used.

By inspection of high values of δ_c it seem like the variance from the readout noise is the most prevalent in the S direction thus setting $\sigma_S^2 = \sigma_\xi^2$ seem like a good choice. As for the choice of a good σ_n^2 value, examination of the distribution $P(S, n|\delta_c)$ in the n direction by considering S as a constant give the kernel of $P(S, n|\delta_c)$ as

$$\frac{\Gamma(n+\frac{1}{2})}{\Gamma(n+1)} \frac{\alpha^n S^{n-1}}{\Gamma(n)} \approx \frac{(\alpha S)^{n-1}}{\Gamma(n)}. \tag{26}$$

All though the approximation is rather rough it is in fact the kernel of the Poisson distribution with $\lambda = \alpha S$. Since the Poisson distribution has mean and variance equal to λ, σ_n^2 is set to $\alpha\delta_c$ where δ_c is used as an approximation of S. The initial guess of good proposal distribution will then be

$$q(x^{(i)}|x^{(i-1)}) \sim \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \boldsymbol{\mu} = x^{(i-1)} = \begin{pmatrix} n^{(i-1)} \\ S^{(i-1)} \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \alpha\delta_c & 0 \\ 0 & \sigma_\xi^2 \end{pmatrix}. \tag{27}$$

This type of proposal density also has computational advantages since it is symmetric $(q(x^{(i)}|x^{(i-1)}) = q(x^{(i-1)}|x^{(i)}))$ the acceptance probability reduces to

$$\alpha(x^{cand}|x^{(i-1)}) = \min(1, \frac{\pi(x^{cand})}{\pi(x^{(i-1)})}) \tag{28}$$

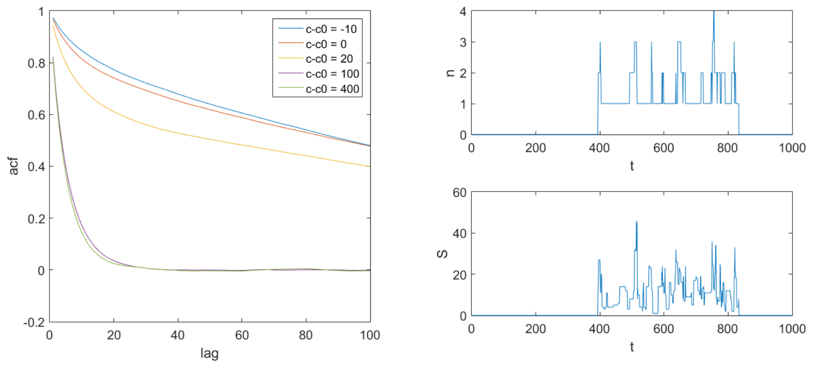


Figure 7: Plot of the auto-correlation function for different δ_c values and trace plots of n and S for $\delta_c = 0$.

which require less computer power to calculate.

In figure 6 it's clear that the proposal density has a similar shape to $P(S, n|\delta_c)$ when δ_c is 300 but for the case with a low image count of 45 a potential problem arises. As image counts drop an increasing ratio of the probability mass get assigned to the $(0, 0)$ point while the shape of the proposal density remains the same. Points (n, S) of $x^{(i-1)}$ which are not in a close proximity to $(0, 0)$ as for instance $(2, 45)$ in the figure will have a low probability of suggesting $(0, 0)$ as the next point in the chain leading to a case where the probability space outside $(0, 0)$ is explored for long runs of the chain. When $(0, 0)$ is finally suggested it will due to the high difference in probability between the two points lead to a very high if not certain probability that the $(0, 0)$ move will be accepted. When the chain reaches $(0, 0)$ the effect if flipped and long runs of only $(0, 0)$ entries will arise in the chain mainly because any candidate move near $(0, 0)$ will have a much lower probability mass. The effect of the proposal density's inability to smoothly transition between $(0, 0)$ and non-$(0, 0)$ points is displayed in figure 7 where low image counts result in high correlation.

7.3 Fast mixing at low intensities

To combat the issue of poor mixing at low intensities the proposal density needs to be modified to better adapt to pixels with low image counts. In the case where the chain is located outside the $(0, 0)$ point the proposal density is modify to more accurately suggest this as a proposed step by assigning this point a constant probability mass $\kappa(\delta_c)$ regardless of where $x^{(i-1)}$ is. Since this probability increase and decrease depending on the image count $\kappa(\delta_c)$ needs to be dependent on δ_c. As we want $\kappa(\delta_c)$ to be as close as possible to $P(S = 0, n = 0|\delta_c)$ some numerical testing using different values on the normal cdf gave

$$\kappa(\delta_c) = max\Big(\frac{1}{2}(1 + erf(1 - \frac{\delta_c}{2\sigma_\xi})), 10^{-5}\Big) \tag{29}$$

as a viable option. $\kappa(\delta_c)$ tend to somewhat underestimate $P(S = 0, n = 0|\delta_c)$ which is why a lower limit here chosen as 10^{-5} is needed to insure the value doesn't drop too close to zero for higher image counts.

For the case where $x^{(i-1)} = (0, 0)$ the problem of suggesting a reasonable candidate move x^{cand} can be solved by using the Poisson approximation from equation (26) for the n-direction of the step and a normal distribution centred at δ_c with variance $\sigma^2_{readout}$ for the S-direction of the step.

The new proposal density given these changes is now

$$q((0, 0)|(\hat{S}, \hat{n})) = \kappa(\delta_c), \tag{30}$$

$$q((\bar{S}, \bar{n})|(\hat{S}, \hat{n})) = \frac{1 - \kappa(\delta_c)}{2\pi\sigma_\xi\sqrt{\alpha\delta_c}} exp\Bigg(- \frac{1}{2}\Big(\frac{(\bar{S} - \hat{S})^2}{\sigma^2_\xi} + \frac{(\bar{n} - \hat{n})^2}{\alpha\delta_c}\Big)\Bigg), \tag{31}$$

$$q((\bar{S}, \bar{n})|(0, 0)) = \frac{1}{\sigma_\xi\sqrt{2\pi}} exp\Big(- \frac{\bar{S}^2}{2\sigma^2_\xi} + \alpha\delta_c\Big)\frac{(\alpha\delta_c)^{\bar{n}}}{\bar{n}!}, \tag{32}$$

$$(\bar{S}, \bar{n}) \subseteq \mathbb{Z}^2 \backslash (0, 0), \quad (\hat{S}, \hat{n}) \subseteq \mathbb{N}^2.$$

In figure 8 the improvement of the correlation time is visible both in the acf plot and in the trace plots. Since this proposal density is not symmetric the acceptance probability is not as easy to calculate as the one expressed in (28) because the q-probabilities in the fraction no longer cancel-out. This results in a more expensive MCMC algorithm to run but the upside of the over all lower correlation time means that a sweepsize (lag) of 20 is sufficient enough to produce samples with a correlation time that is low enough to be used. The modification of the step when the chain is in the (0,0) position also has the upside of having any chain starting in (0,0) to quickly find its way to the areas having a high probability mass regardless of the value δ_c. The benefit of this is that the chain can start in the (0,0) position for any value of δ_c and still with little burn converge to its stationary distribution.

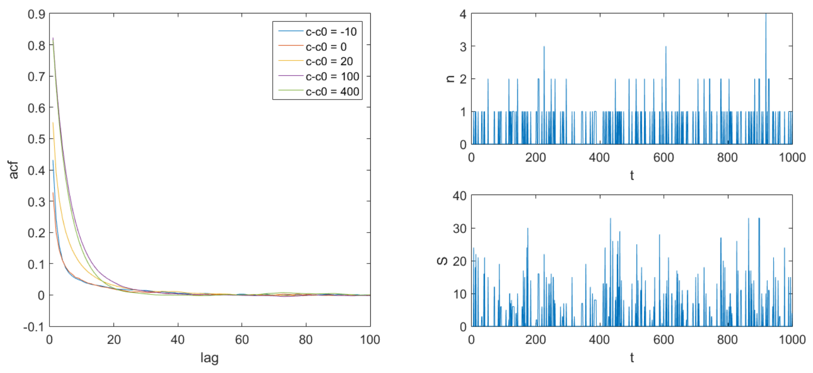


Figure 8: Plot of the auto-correlation function for different δ_c values and trace plots of n and S for $\delta_c = 0$. These plots are done using the improved proposal density.

8 Bootstrapping EMCCD pictures

A way to preserve the dependency between n and S when bootstrapping is to take the image and by implementing the MCMC algorithm presented above generate possible outcomes of the values n, S and ξ. Using these values the bootstrap images can then be created in several different ways. In this paper three different methods which incorporate n, S and ξ were used. In each method each bootstrap sample picture is generated in two steps, first the MCMC sample of n, S and ξ is generated (one MCMC sample for each bootstrap sample) and then depending on the method a single bootstrap sample is drawn from a re-sampling scheme which either uses only MCMC values or a mixture of MCMC values and new generated values from the camera model. Figure 9 is a flowchart of 'Method 3' where re-sampling is done on all the values.

Method 1: Bootstrap n and generate new S and ξ.

For this method a single n sample is generated for each bootstrap sample using the MCMC algorithm. The bootstrap sample is then generated by re-sampling the n values in the same way as was done in the section of bootstrapping pictures of photon counts and then simply generate new S and ξ values using $p(S|n)$ and $p(\xi)$ from the camera model. By adding S and ξ the new bootstrap picture is obtained.

Method 2: Bootstrap n and S and generate new ξ.

For this method both the n and S values from the MCMC algorithm is used. Since the n values from the MCMC algorithm is coupled with corresponding S values this method preserves this dependency which the first method does not. Each electron count in each pixel is here split randomly over the photons in the same pixel thus assigning each photon a certain number of electrons. A re-sampling over these photons with assigned electrons is then done to produce a new bootstrapped electron count, S_{bs}. A new read-out noise is then generated from the camera model and added to the bootstrapped electron count to give the new bootstrap picture.

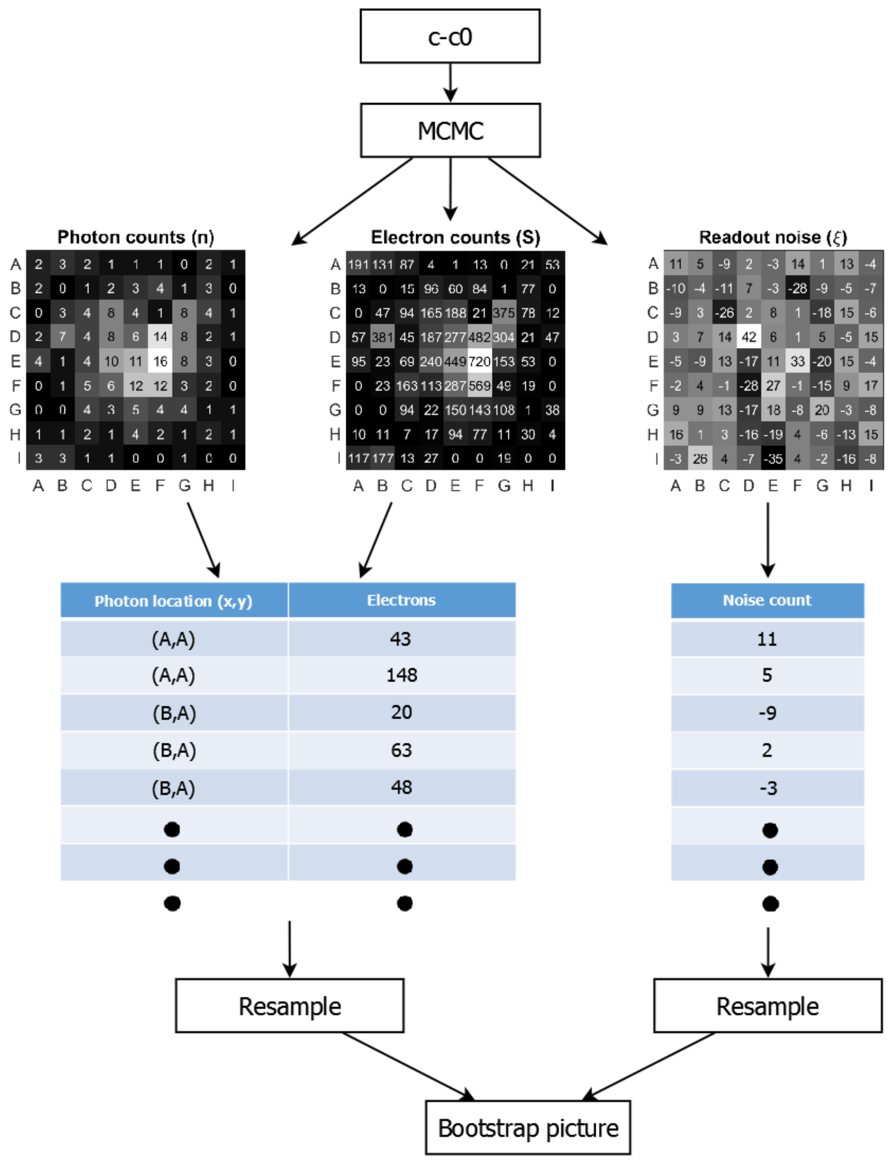
Method 3: Bootstrap n,S and ξ

The last method is identical to the second method with the exception that the noise added to S_{bs} is here not generated but rather bootstrapped from the noise in the MCMC sample. Figure 9 shows how a single bootstrap samples is generated using this method.

9 Results

To examine how well the different bootstrap methods work some numerical tests were performed where the mean value of the bootstrap variance was compared to the sample variance for a series of generated EMCCD pictures. This was then also done for a few cases where the camera values used for the maximum likelihood estimation and bootstrap sampling deviated from the vales used to create the images in order to see how sensitive the bootstrap estimations are to calibration errors.

From a PSF with $N = 200$, $b = 1$ and $\sigma_E = 1.5$pixels $1,000$ EMCCD pictures were generated with camera values; $\sigma_\xi = 15$ and $\alpha = 1/30$. Using the different bootstrap methods on different accurate and faulty camera calibrations gave the results presented in figure 10. Method 1 had such poor results in the case where the camera was properly calibrated so no further simulations were done with this method. The other two methods where both n and S is bootstrapped but ξ is either generated from scratch or bootstrapped along with the other values seem to perform quite similar to each other in all cases. As the numerical estimates calculated from the EMCCD pictures seem to deviate marginally from faulty calibrations the bootstrap methods seem to be effected to a larger extent. The reason for the increased sensitivity to the calibration of the camera values in the bootstrap methods is most likely due to the fact that these faulty values effect both the



Photon counts (n)

	A	B	C	D	E	F	G	H	I
A	2	3	2	1	1	1	0	2	1
B	2	0	1	2	3	4	1	3	0
C	0	3	4	8	4	1	8	4	1
D	2	7	4	8	6	14	8	2	1
E	4	1	4	10	11	16	8	3	0
F	0	1	5	6	12	12	3	2	0
G	0	0	4	3	5	4	4	1	1
H	1	1	2	1	4	2	1	2	1
I	3	3	1	1	0	0	1	0	0

Electron counts (S)

	A	B	C	D	E	F	G	H	I
A	191	131	87	4	1	13	0	21	53
B	13	0	15	96	60	84	1	77	0
C	0	47	94	165	188	21	375	78	12
D	57	381	45	187	277	482	304	21	47
E	95	23	69	240	449	720	153	53	0
F	0	23	163	113	287	569	49	19	0
G	0	0	94	22	150	143	108	1	38
H	10	11	7	17	94	77	11	30	4
I	117	177	13	27	0	0	19	0	0

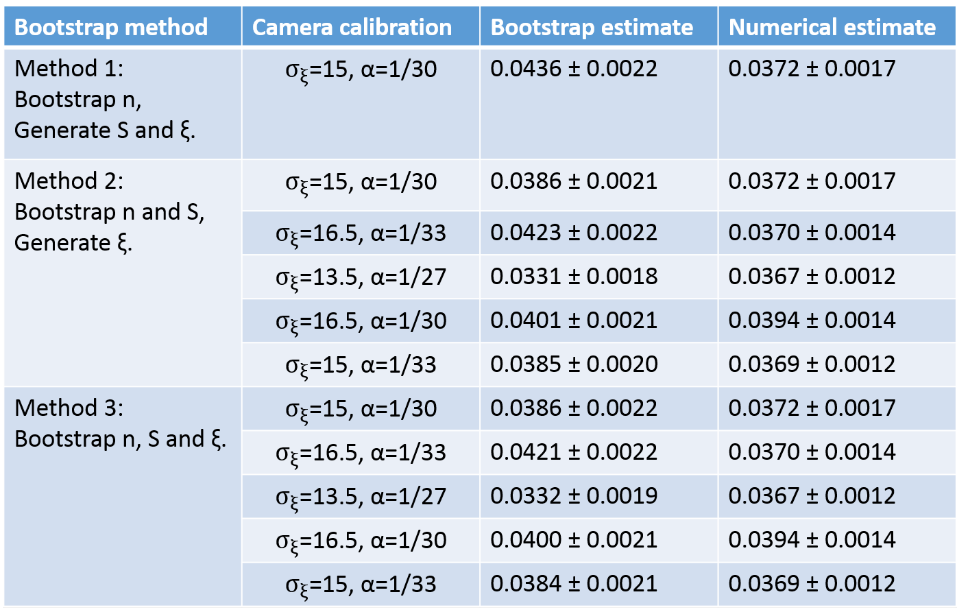
Readout noise (ξ)

	A	B	C	D	E	F	G	H	I
A	11	5	-9	2	-3	14	1	13	-4
B	-10	-4	-11	7	-3	-28	-9	-5	-7
C	-9	3	-26	2	8	1	-18	15	-6
D	3	7	14	42	6	1	5	-5	15
E	-5	-9	13	-17	11	33	-20	15	-4
F	-2	4	-1	-28	27	-1	-15	9	17
G	9	9	13	-17	18	-8	20	-3	-8
H	16	1	3	-16	-19	4	-6	-13	15
I	-3	26	4	-7	-35	4	-2	-16	-8

Photon location (x,y)	Electrons
(A,A)	43
(A,A)	148
(B,A)	20
(B,A)	63
(B,A)	48
●	●
●	●
●	●

Noise count
11
5
-9
2
-3
●
●
●

Figure 9: Flowchart on how the 3rd bootstrap method which re-samples n, S and ξ from the MCMC algorithm is implemented. It should be noted that the depictions of n,S and ξ and the resulting bootstrap picture are the results of one single sample from the MCMC algorithm. A new set of n,S and ξ values used to generate a new bootstrap picture are thus created at each step in the Markov chain.



Bootstrap method	Camera calibration	Bootstrap estimate	Numerical estimate
Method 1: Bootstrap n, Generate S and ξ.	$\sigma_\xi=15$, $\alpha=1/30$	0.0436 ± 0.0022	0.0372 ± 0.0017
Method 2: Bootstrap n and S, Generate ξ.	$\sigma_\xi=15$, $\alpha=1/30$	0.0386 ± 0.0021	0.0372 ± 0.0017
	$\sigma_\xi=16.5$, $\alpha=1/33$	0.0423 ± 0.0022	0.0370 ± 0.0014
	$\sigma_\xi=13.5$, $\alpha=1/27$	0.0331 ± 0.0018	0.0367 ± 0.0012
	$\sigma_\xi=16.5$, $\alpha=1/30$	0.0401 ± 0.0021	0.0394 ± 0.0014
	$\sigma_\xi=15$, $\alpha=1/33$	0.0385 ± 0.0020	0.0369 ± 0.0012
Method 3: Bootstrap n, S and ξ.	$\sigma_\xi=15$, $\alpha=1/30$	0.0386 ± 0.0022	0.0372 ± 0.0017
	$\sigma_\xi=16.5$, $\alpha=1/33$	0.0421 ± 0.0022	0.0370 ± 0.0014
	$\sigma_\xi=13.5$, $\alpha=1/27$	0.0332 ± 0.0019	0.0367 ± 0.0012
	$\sigma_\xi=16.5$, $\alpha=1/30$	0.0400 ± 0.0021	0.0394 ± 0.0014
	$\sigma_\xi=15$, $\alpha=1/33$	0.0384 ± 0.0021	0.0369 ± 0.0012

Figure 10: Results of bootstrap and numerical estimates for the variance with different camera calibrations.

MCMC method which is used to generate the bootstrap samples as well as the likelihood method used to estimate μ. While the numerical estimates is only effected in the form of its maximum likelihood estimates the bootstrap estimates thus suffer from the effect of the calibration an additional time when the bootstrap samples are generated.

10 Conclusion and outlook

By implementing and testing three different bootstrap algorithms which utilizes a MCMC algorithm for its sampling two algorithms which show promise on simulated data was found. The two methods which seem to work quite well are those for which both n and S are bootstrapped and where ξ is either bootstrapped as well or generated from the camera model. From the results presented in figure 10 these two methods both somewhat overestimate the variance within the maximum likelihood estimate but their results still lie within one standard deviation of the numerical estimate. The results also seem to suggest that generating or bootstrapping the readout noise will produce similar results but this might be an effect of the particular intensity and camera parameters used in the generated images. It does seem reasonable that if there is a significant difference between these two methods it should become more distinguishable in low intensity pictures where the readout noise has a more profound effect on the image counts of the pixels.

There are several different aspects to further look into when it comes to this method of estimating variance. As can be seen in figure 5 the error of the variance estimate is effected by the intensity for the case where bootstrapping was only performed on pictures of photon counts. How this effect translate to bootstrapping EMCCD pictures and also how well the bootstrap method performs for different intensity levels is worth looking into. Since this method can be used on any localization algorithm some diagnostics on its performance on localization methods other than the maximum likelihood method would be of interest. The bootstrap method is also highly dependent on how well the camera model describes reality, performing tests on different camera models with both real and simulated data is still something that needs to be done.

11 Acknowledgements

I would like to thank my supervisor Martin Lindén for his immense help and dedication throughout this project, Vladimir Curic for the code and help he provided me. I also want to thank all the other people in Elf-lab for making it a pleasant place to work and Ingemar Kaj for helping me find this project.

References

[1] Shane Stahlheber Alex Small. Fluorophore localization algorithms for super-resolution microscopy. *Nat. Methods*, 11:267–279, 2014.

[2] Kim I. Mortensen, L. Stirling Churchman, James A. Spudich, and Henrik Flyvbjerg. Optimized localization analysis for single-molecule tracking and super-resolution microscopy. 7(5):377–381, 2010. doi:10.1038/nmeth.1447.

[3] M.H. Ulbrich and E.Y. Isacoff. Subunit counting in membrane-bound proteins. nat. methods 4. pages 319–321, 2007.

[4] R.N. Tubbs. Lucky exposures: Diffraction limited astronomical imaging through the atmosphere. observatory 124. 2004.

[5] Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 186(1007):453–461, 1946. doi:10.1098/rspa.1946.0056.

[6] Wikipedia. Jeffreys prior — wikipedia, the free encyclopedia, 2015. URL `http://en.wikipedia.org/w/index.php?title=Jeffreys_prior`. [Online; accessed 15-Nov-2015].

[7] Wikipedia. Maximum likelihood— wikipedia, the free encyclopedia, 2015. URL `https://en.wikipedia.org/wiki/Jeffreys_prior`. [Online; accessed 20-Dec-2015].

[8] Bradley Efron. Bootstrap methods: Another look at the jackknife. 7(1): 1–26, 1979. doi:10.1214/aos/1176344552.

[9] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

12 Appendix

Function for the MCMC algorithm.

```
function [nt,St,xit] = MCMC(dc, sigRead, alpha, Nsim, lag)

% dc : the image - offset
% sigRead : sigma of readout noise
% Nsim : number of samples generated
% lag : sweepsize of the algorithm

%initialize values
n0=dc*0;
S0=dc*0;
n1=dc*0;
S1=dc*0;
logP0=logP_gamma(n0,S0,dc,alpha,sigRead);

%Kappa probability. Probability of moving from (S,n) to (0,0)
k=0.5*(1+erf(1-dc/sqrt(2)/sigRead));
k=max(k,10^-5);

%matrices to store the generated values
St=zeros(size(dc,1),size(dc,2),Nsim);
nt=zeros(size(dc,1),size(dc,2),Nsim);
xit=zeros(size(dc,1),size(dc,2),Nsim);

%sigma for Sn to Sn step
Sn_sigma=sqrt(max(dc.*alpha,1));

for t=1:Nsim
    for i=1:lag
        %indicator for S,n to 0,0
        ind=(rand(size(dc))<k);
        n1(ind&n0~=0)=0;
        S1(ind&n0~=0)=0;

        %indicator for S,n to S,n Normal step
        dim=size(dc(~ind&n0~=0));

        n1(~ind&n0~=0)=round(n0(~ind&n0~=0)+Sn_sigma(~ind&n0~=0).*randn(dim));
        S1(~ind&n0~=0)=round(S0(~ind&n0~=0)+sigRead.*randn(dim));
        if n1(~ind&n0~=0)==0
            n1(~ind&n0~=0)=n0(~ind&n0~=0);
            S1(~ind&n0~=0)=S0(~ind&n0~=0);
        end

        %indicator for 0,0 to S,n Poisson step
        dim=size(dc(n0==0));
        n1(n0==0)=poissrnd(max(0.5,dc(n0==0).*alpha),dim);
        S1(n0==0)=round(dc(n0==0)+sigRead.*randn());
        %Accept probabilities
        q=ones(size(dc));
        %q-quota for steps (0,0) to (S,n)
        q(n0==0&n1~=0)=k(n0==0&n1~=0)...
            ./(0.5*erf((S1(n0==0&n1~=0)+0.5-dc(n0==0&n1~=0))/sigRead/sqrt(2))...
            -0.5*erf((S1(n0==0&n1~=0)-0.5-dc(n0==0&n1~=0))/sigRead/sqrt(2)))...
            ./poisspdf(n1(n0==0&n1~=0),max(0.5,dc(n0==0&n1~=0).*alpha));
        %q-quota for steps (S,n) to (0,0)
        q(n0~=0&n1==0)=(0.5*erf((S0(n0~=0&n1==0)+0.5-dc(n0~=0&n1==0))...
            /sigRead/sqrt(2))-0.5*erf((S0(n0~=0&n1==0)...
            -0.5-dc(n0~=0&n1==0))/sigRead/sqrt(2)))...
            .*poisspdf(n0(n0~=0&n1==0),max(0.5,dc(n0~=0&n1==0).*alpha))...
            ./k(n0~=0&n1==0);

        logP1=logP_gamma(n1,S1,dc,alpha,sigRead);

        pAcc=min(ones(size(dc)),exp(logP1-logP0).*q);
        pAcc(logP1==-inf)=0;
        ind=rand(size(dc))<pAcc;

        %update values
        n0(ind)=n1(ind);
        S0(ind)=S1(ind);
        logP0(ind)=logP1(ind);
    end
    %save values
    St(:,:,t)=S0;
    nt(:,:,t)=n0;
    xit(:,:,t)=dc-S0;
end
end
```

Function for the resampling of MCMC values.

```
function [bS, bxi] = BootstrapResample(n,S,xi)
    %Resamples the values drawn from the MCMC algorithm and outputs the
    %bootstrapped electron image with the bootstrapped noise


    %nlist is a list over all photones where their
    %first colum is electrones 2nd is the numel(n) position
    nlist=zeros(sum(sum(n)),2);

    pos=1; %position in list to keep track of where to fill in next value

    N=numel(n);
    for i = 1:N
        if(n(i)==1)
            nlist(pos,1)=S(i);
            nlist(pos,2)=i;
            pos=pos+1;
        end
        if(n(i)>1)
            %Cut up the electrones over n-1 cuts where n is the number of
            %photons
            cuts=sort(randi(S(i),1,n(i)-1));

            %update nlist with the electron value of each photon
            nlist(pos:pos+n(i)-1,1)=[cuts S(i)]-[0 cuts];

            %and its position
            nlist(pos:pos+n(i)-1,2)=repmat(i,n(i),1);
            pos=pos+n(i);
        end
    end

    %resample over nlist where "sample" is a list of indexes to be used in
    %the bootstrap sample
    bS=zeros(size(n));
    sample = randi(sum(sum(n)),1,sum(sum(n)));

    %loop over all indexes of nlist drawn from sample
    for i = 1:size(sample,2);

        %set x,y coordinates to correspond to those drawn
        xy=nlist(sample(i),2);

        %update the electrones stored in dc
        Svalue=nlist(sample(i),1);
        bS(xy)=bS(xy)+Svalue;
    end

    %bootstrap the noise
    bxi=zeros(size(n));
    sample=randi(numel(n),1,numel(n));
    for i = 1:numel(n)
        bxi(i)=xi(sample(i));
    end

end
```

Function to compute the logarithm of the probability $p(S, n|\delta_c)$. Done by Martin Lindén

```
% compute the log of the posterior density for the pixel-wise posterior
% density of n photons and S electrons given an offset-subracted image dc
% and EMCCD parameters for the Gamma EM model.
%
% logP = logP_gamma(n,S,dc,alpha,sigma)
%
% n     : photons
% S     : electrons
% dc    : images-offset
% alpha : inverse gain parameters
% sigma : readout noise std
%
% ML 2015-06-12

function logP = logP_gamma(n,S,dc,alpha,sigma)

xi2=(dc-S).^2/2/sigma^2;
% build up the three cases consecutively

% base case (disallowed configurations)
% includes n=0, s>0 and all combinations of s<0 or n<0
logP=zeros(size(dc))-inf;

% now add n=0,S=0
ind = (n==0) & (S==0);
logP(ind)= -xi2(ind);
logP(ind)=logP(ind)+gammaln(0.5);

% and finally n>0,S>0
ind=(n>0)&(S>0);
logP(ind) = -xi2(ind)+n(ind)*log(alpha)+(n(ind)-1).*log(S(ind))...
    -alpha*S(ind)+gammaln(n(ind)+0.5)-gammaln(n(ind)+1)-gammaln(n(ind));
```

Function to generate the intensities E_i for given PSF parameters. Done by Martin Lindén and Vladimir Curic.

```
% This code computes PSF using a 3*3-point Gauss quadrature rule.
%
% E = psfGaussianSymmetric_9ptGQ(x,y,b,A,mu_x,mu_y,sigmaS,pxL)
%
% % Input :
%      - x : data (vector format) that corresponds to the x-variable of
%            pixel mid-points
%      - y : data (vector format) that corresponds to the y-variable of
%            pixel mid-points
%      - b : background
%      - A : expected number of photons per frame
%      - mu_x : x_position of the centre of PSF
%      - mu_y : y_position of the centre of PSF
%      - sigmaS : PSF standard deviation
%      - pxL : linear pixel size (default 1)
%
% Output:
%      - psfGaussianSymmetric : PSF function (vector format), approximately
%        integrated over every pixel.
% Author: Vladimir Curic, 2015-01-22, vladacuric@gmail.com
%         Martin Lind n , 2015-08-21, bmelinden@gmail.com, added multipoint
%         version.
%
% ref: https://en.wikipedia.org/wiki/Gaussian_quadrature
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  E = psfGaussianSymmetric_9ptGQ(x,y,b,A,mu_x,mu_y,sigmaS,pxL)
```

```
% default parameter
if(nargin<8)
    pxL=1;
end
% check
if(numel(b)>1 || numel(A)>1 || numel(mu_x)>1 || numel(mu_y)>1 || numel(sigmaS)>1)
   error(' psfGaussianSymmetric_refined cannot handle more than one set of shape paramete
end


% node points: +-1/sqrt(3) on the [-1,1] interval, translates to +-
% 1/2/sqrt(3)) on the [-0.5,0.5] interval (
dx=pxL/2*[0 -1 1 0 -1 1 0 -1 1]*sqrt(3/5);
dy=pxL/2*[0 0 0 -1 -1 -1 1 1 1]*sqrt(3/5);
w=(pxL/2/9)^2*[64;40;40;40;25;25;40;25;25];% [8 5 5  8 5 5 8 5 5]'.*[8 8 8 5 5 5 5 5 5]';

% compute refined
N = numel(x);
E = zeros(1,N);
for i = 1:N
    E(i) = (b+A*(1./(2*pi*sigmaS^2))*(exp(-((x(i)+dx-mu_x).^2+(y(i)+dy-mu_y).^2)/(2*sigma
end
```