

BGSUBFIT

User Guide

Benjamin P. Isaacoff
last update: October 5, 2016

Contents

Introduction.....	1
Using <i>BGSUBFit</i>	2
Average Subtraction.....	4
Guessing.....	5
Constructing the off_frames list.....	6
Subtracting and Fitting.....	7
Tracking	8
ViewFits.....	9
Coordinate System Definition.....	10

Introduction

BGSUBFIT is code to do real background subtraction of single molecule imaging movies so that the molecules can be fit and their intensity accurately measured even in the presence of an arbitrarily complex low frequency fluorescent background.

BGSUBFIT currently only takes input data as .tif stacks (using TIFFStack, see the note at the end of this section on how to install this). If your data is in another format the easiest thing to do would be to convert it to a .tif stack. In Matlab, this can be accomplished by importing your data to a 3D array (first two dimensions are spatial and the third dimension is frame number), then using saveastiff to save it as a .tif stack.

The function *BGSUBFit* is a wrapper for the other code to perform all of the steps in the correct order. Simply run *BGSUBFit* by specifying the directory containing your .tif stack movies, specify the three required parameters, and any optional parameters, then run it and click to choose the movies you want to fit. The various steps can also all be done independently, either using their individual standalone functions, or by using *BGSUBFit* to them individually.

The basic workflow in BGSUBFIT is:

1. Average subtraction (make avgsub movie) using *AVGSUB_tiffs.m*
 - a. To remove low frequency background
 2. Guess molecule locations in the avgsub movie using *Guessing.m*
 3. Identify frames around each guess where there isn't another guess nearby *Mol_off_frames.m*
 - a. Hasn't turned on yet, it turned off, or it blinked.
 - b. This gives an off_frames list for each guess
 4. For each guess, time average a small area around the guess using the off_frames list, occurs in *Subtract_then_fit.m*
 - a. This is the True Background
 5. Subtract the True Background, and fit using *Subtract_then_fit.m*
- OPTIONAL STEPS
6. Track the fits to filter out fits which aren't organized into a track, and remove the first and last frame of the track using *Track_filter.m*
 7. Make a ViewFits movie to help check the fitting using *ViewFits.m*

There will be several files written to the working directory containing the results from the various steps. The fits will be collected in a .mat file called *moviename_AccBGSUB_fits.m* which contains the *fits* array each individual guess will have a row in *fits* and the columns correspond to

1. Frame number
2. row position (pixels)
3. column position (pixels)
4. Gaussian fit standard deviation (pixels)
5. Offset (intensity)
6. Amplitude (intensity)
7. error (for MLE fitting this is the variance, for least squares fitting this is the mean 95% confidence interval on the position)
8. Sum of pixel intensities (intensity)
9. Goodfit Boolean

The following sections will go through each step in more detail, defining and explaining the various parameters (required and optional) for each step, and the inputs and outputs for that step.

BGSUBFIT uses the program TIFFStack to rapidly read tiff stacks. In order to use TIFFStack copy the @TIFFStack folder from this repository (leave it as a folder with that name) into your Matlab directory. The TIFFStack README has more details if you have problems.

Using *BGSUBFit*

There are only four required parameters to *BGSUBFit*, they are:

directoryname is the name of the directory where the movies will be selected, if there is an error finding the directory the program will open uigetfile in the current working directory

dfrlmsz is the size of a diffraction limited spot in pixels. It's the nominal diameter, NOT the FWHM or Gaussian standard deviation. It must be an integer. For an expected diffraction limited standard deviation, std, using the full width at 20% max, $dfrlmsz = \text{std} * (2 * \sqrt{2 * \log(5)})$

avgwin is the window size (in frames) to be used for the average subtraction. Needs to be an odd integer

moloffwin is the window size (in frames) to be checked to determine in which frames that molecule was off and are thus safe to subtract. Needs to be an even integer.

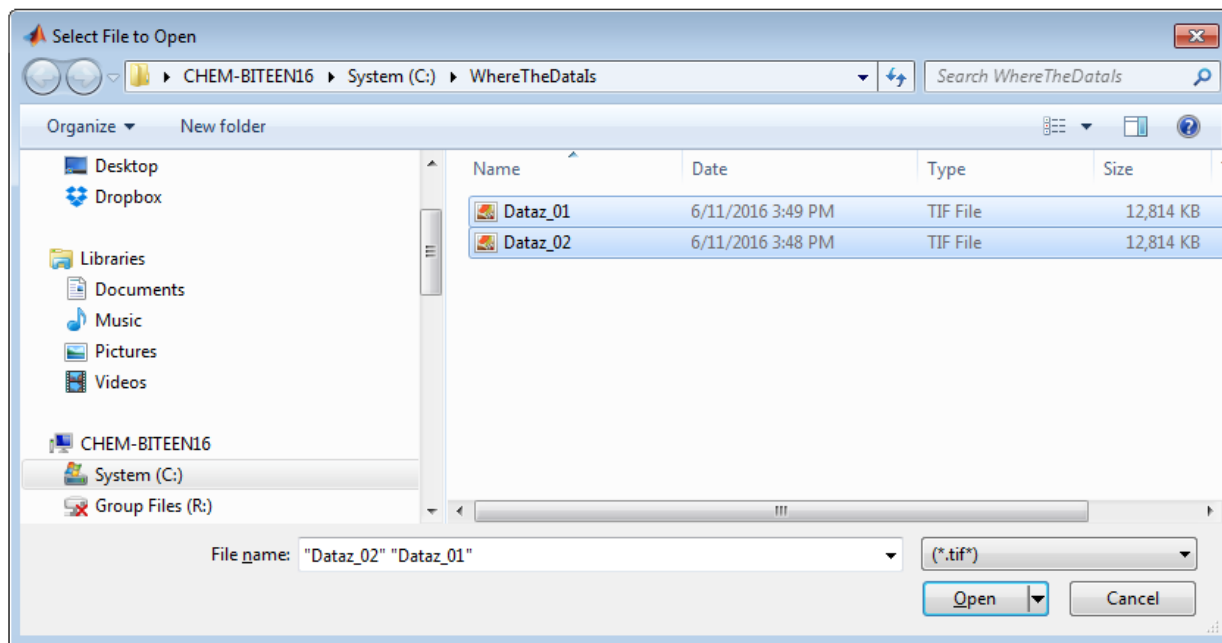
To call *BGSUBFit* in the Matlab workspace enter

```
BGSUBFit(directoryname,dfrlmsz,avgwin,moloffwin)
```

For example if you set *dfrlmsz* = 7, *avgwin* = 501, and *moloffwin* = 150, and your data is stored in the directory C:\WhereTheDataIs, and you are setting the optional parameters *bpthrsh* = 90 and *orig_movie* = 0, you would enter:

```
BGSUBFit('C:\WhereTheDataIs',7,501,150,'bpthrsh',90,'orig_movie',0)
```

This brings up uigetfile interface for you to select the .tif stack movies, which on my system, after selecting the two movies in the directory looks like



BGSUBFit will then proceed and write out the various results without any further input from the user. Many of the steps will display a progress bar to show how far along that particular step is. In general *BGSUBFit* is a fairly fast and memory efficient program. The slowest step is fitting, where least squares fitting is much

faster than MLE fitting. If speed is an issue, reducing the number of guesses will reduce the total time required for *BGSUBFit* to run.

There are many optional parameters to send to *BGSUBFit* which will be covered in this document. You can control which steps *BGSUBFit* does by changing the following Booleans all of which are default set to 1.

makeGuesses determines whether the avgsub movie will be made (if one doesn't already exist) and if guessing will be done.

makeOffFrames determines whether the off frames list will be constructed.

fitting determines if the subtract & fitting program will be run

tracking determines if tracking will be run.

makeViewFits determines if the ViewFits program will be called.

Average Subtraction

The first step in *BGSUBFIT* is making the average subtracted (avgsub) movie. Because this step can be somewhat slow *BGSUBFit* checks if *movienam_avgsub.tif* already exists in the working directory. If it does then it skips making a new avgsub movie. If you want to make a new avgsub movie, simply delete the *movienam_avgsub.tif* file from that directory.

This step utilizes *AVGSUB_tiffs.m* which is called like

```
AVGSUB_tiffs(movie_filename,runningavg,avgwin,offset);
```

BGSUBFit required parameters

avgwin is the width of temporal window that will be averaged. Needs to be an odd integer.

BGSUBFit optional parameters

movie_filename is the filename of the original .tif stack movie.

runningavg is a Boolean, set to 1 to do a running average or set to 0 to do a static window background subtraction. Default is 1.

offset is the intensity offset to deal with negative pixels. Default is 1000.

This functions saves the AVGSUB movie as a .tif stack at the original file location with the original file name, but with `_avgsub` appended to the name. Also outputs a short text file with the parameters used called `moviename_avgsub_info.txt`

Choosing *avgwin* should balance the characteristic on time for the molecules and the characteristic timescale for background changes. *avgwin* should be much longer (approximately at least one order of magnitude is ideal) than the characteristic on time of the molecules so as to minimize molecule intensity being subtracted and thus affecting the guessing. Conversely *avgwin* should be smaller than the characteristic time of the low frequency background changes, so that over the course a window the background is essentially constant. You can check if you chose an appropriate *avgwin* value qualitatively by watching the avgsub movie, if you see that during part of a molecule's track it appears as a dark spot instead of a bright spot, then you know that you need to increase *avgwin*. Conversely if you see that the background features appear and disappear, instead of never being present then you know that you need to reduce *avgwin*. If large portions of the movie look saturated then you need to increase *offset*.

Guessing

Guessing is accomplished in the function *Guessing.m* which is called like

```
Guessing(movie_filename,dfrlmsz,bpthrsh,edgesz,pctile_frame,check_guesses);
```

movie_filename is the filename of the movie

BGSUBFit required parameters

dfrlmsz is the nominal size of a diffraction limited spot for your system, the full width at 20% max is a good approximation

BGSUBFit optional parameters

bpthrsh is the is the percentile of brightnesses of the bandpassed image below which those pixels will be ignored. Default is 90

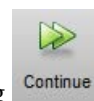
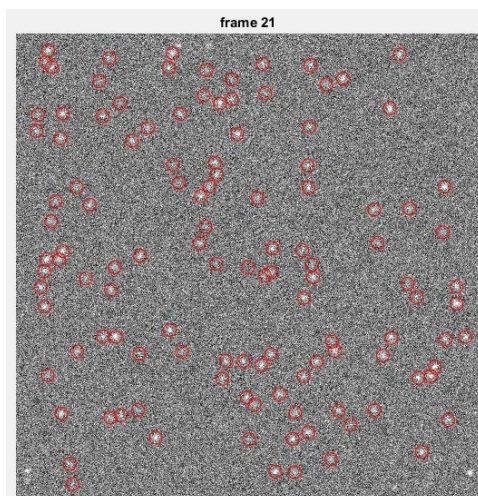
edgesz is the number of pixels on the edge of the image that will be ignored. Default is *edgesz* = *dfrlmsz*

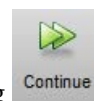
pctile_frame is a Boolean determining whether *bpthrsh* will be applied frame by frame, or to the entire movie. Using the entire movie (setting to 0) is more sensitive to low frequency noise and background changes, but is a more robust guessing method. Using each frame tends to produce a constant number of guesses per frame, regardless of their absolute brightness.

check_guesses is a Boolean to determine if you want to go through and look at the guesses. Default is 0

Guessing.m saves a .mat file called *moviename_guesses.mat* which contains the *guesses* array. The columns in *guesses* are simply 1. frame #, 2. x position (pixels), 3. y position (pixels).

You should verify that the guessing is working the first time you run *BGSUBFIT*, simply set `check_guesses` to 1 when you call the function and when the program gets to guessing you will see the first frame of the movie with the guesses indicated with red circles, for example something like



You can click through the frames by continuing the debug process either by pressing  or by typing `dbcont` into the workspace. If the parameters don't look good, simply stop the debugging and change the parameters when you run it next. Repeat this process until you find a set of parameters that give good guessing results.

The first parameter to try changing is *bpthrsh*, if *Guessing* is missing a lot of molecules (false negatives), then try reducing *bpthrsh*, and conversely if *Guessing* is producing a lot of false positives, then try increasing *bpthrsh*. For *BGSUBFIT* it is **much better** to have false positives, than to have false negatives, however too many unneeded guesses will make the fitting step last much longer than it needs to, and can reduce the precision of the fitting result by having a very noisy image subtracted.

The other parameters to consider changing to help the guessing are *dfrlmsz* and *pctile_frame*. If *dfrlmsz* is very off from its true value then *Guessing* will not be able to find the molecules well. If there are large low frequency changes occurring over the course of a movie then it would probably be best to turn *pctile_frame* on.

Note that if the movies being analyzed are very large files, you may have a memory issue in this step if *pctile_frame* is turned off. If you do have this issue then simply turn *pctile_frame* on.

Constructing the `off_frames` list

The `off_frames` list is constructed using the *Mol_off_frames.m* function, which is called like

```
Mol_off_frames(guess_filename,dfrlmsz,moloffwin);
```


guess_filename is the filename of the .mat file output from *Guessing.m* which contains the *guesses* array.

BGSUBFit required parameters

dfrlmsz is the nominal size of a diffraction limited spot for your system, the full width at 20% max is a good approximation

moloffwin is the window size (in frames) to be checked to determine in which frames that molecule was off and are thus safe to subtract. Needs to be an even integer.

Mol_off_frames.m doesn't have any optional parameters and shouldn't need to be debugged. Briefly, the way the function works is that it identifies frames in which two guesses are within a $2*dfrlmsz$ sized box of each other. There is a warning that will print out to the workspace if a particular guess has less than 5% of *moloffwin* frames off. If you see a lot of this warning you should consider increasing *moloffwin*.

The off frames list will be saved to a .mat file called *guess_filename_Mol_off_frames.mat*

Subtracting and Fitting

Subtracting and fitting (steps 4 & 5 in the intro) are accomplished in *Subtract_then_fit.m* which is called like

```
Subtract_then_fit(movie_fname,Mol_off_frames_fname,guess_fname,MLE_fit,egdesz,
maxdistfrac,stdtol,maxerr);
```

movie_fname the filename of the original tiff stack movie (NOT the avgsb movie)

Mol_off_frames_fname is the filename of the .mat file containing the off frames list

guess_fname is the filename for the guesses .mat file

BGSUBFit optional parameters

MLE_fit is a Boolean determining whether or not MLE fitting is used. Set to 1 to use MLE and to 0 to use least squares using *lsqcurvefit*. Default is 0. Note that MLE is quite slow, and so its not recommended for a large number of guesses

edgesz is the number of pixels on the edge of the image that will be ignored. Default is *edgesz* = *dfrlmsz*

maxdistfrac is max row&column distance of fit from guess, as a fraction of *dfrlmsz*, default is 0.75

stdtol is the ratio tolerance on fit Gaussian STD, default value is 5. Meaning that if the $std/stdtol \leq fit_std \leq std*stdtol$ then it is considered a goodfit

maxerr is the maximum error of the fit. For MLE fit, using variance default 0.1 (can't be above this) for LSQR fit, using the 95% confidence interval on the position, default max is 3.

Subtract_then_fit.m outputs a .mat file called *moviename_AccBGSUB_fits.m* which contains the *fits* defined in the intro. The best way to debug the goodfit parameters is to watch the ViewFits movie. If you see a lot of guesses that you think actually are molecules not passing the goodfit checks (ie a lot of red circles of good looking molecules, instead of green circles) then try relaxing the goodfit checks. Conversely if you see a lot of false positives then try tightening these parameters.

Note: As BGSUBFIT is a fairly fast program the slowest step is usually the actual fitting step, where BGSUBFit spends most of its time in either *lsqcurvefit* or *MLEwG*. Allowing molecules to be fit in parallel will likely allow the program to pass the fitting step faster. To accomplish this I've included *Subtract_then_fit_parallelized.m* in the repository. To use this function instead simply change the call in BGSUBFit to *Subtract_then_fit_parallelized* from *Subtract_then_fit*. You can either start a parallel pool before running BGSUBFit, or let Matlab start one automatically, in which case you should verify that the default settings are what you want them to be. Note that because the function is running in parallel that means it will make multiple copies of the data being used and this may cause memory issues. So if you have a large framesize, or a large *moloffwin*, then you should consider using a smaller number of cores to run in parallel.

Tracking

Tracking is currently used just to filter out fits that aren't organized into a track, this is a useful way to ensure that fitting noise in the movie doesn't make it into the final result. The result is a logical vector for each fit called *trk_fit*. Furthermore, as this was designed with fitting PAINT experiments in mind the fits that were organized in tracks and are the first and last frame in a track are excluded from being set to 1 in *trk_fit*. Tracking occurs in a few programs the topmost program is *Track_filter.m* which is called like:

```
Track_filter(fits_filename,append_vec,trackparams);
```

fits_fname the filename of the .mat file containing the *fits* array

append_vec is a Boolean determining if the *trk_fit* vector, and the *tracks* array will be appended to the fits .mat file

BGSUBFit optional parameters

trackparams is a vector containing the tracking parameters. The elements in the vector and their default values are:

```
% minimum merit
trackparams(1)=0.01;
% Integration time (ms)
trackparams(2)=200;
% gamma
trackparams(3)=1;
% maximum step size
trackparams(4)=3;
% minimum track length
trackparams(5)=3;
% speed estimation window halfsize
trackparams(6)=1;
```

```
% time delay between consecutive frames (ms)
trackparams(7)=0;
```

ViewFits

The ViewFits movie is made with the function *ViewFits.m* which is called like

```
ViewFits(movie_fname, fits_fname, circ_D, write_mov, autoscale_on, linewidth)
```

movie_fname the filename of the tiff stack movie to put the fit indicators on

fits_fname the filename of the .mat file containing the *fits* array

BGSUBFit optional parameters

orig_movie is a Boolean determining whether or not to use the original .tif stack movie. Setting it to 0 will use the avgsub movie. Default is 0

circ_D is the diameter of the circles in the ViewFits movie. Default is *circ_D = dfrlmsz*

write_mov is a Boolean determining if the ViewFits movie will be written to an .avi movie. If set to 0 the program will go to debug mode allowing you to step through frame by frame to see the results.

autoscale_on is a Boolean determining if the movie greyscale will be set frame by frame. If set to 0 a handful of frames throughout the movie are used to set the grayscale. Default is 1

linewidth is the linewidth of the circles in the movie. Default is 1

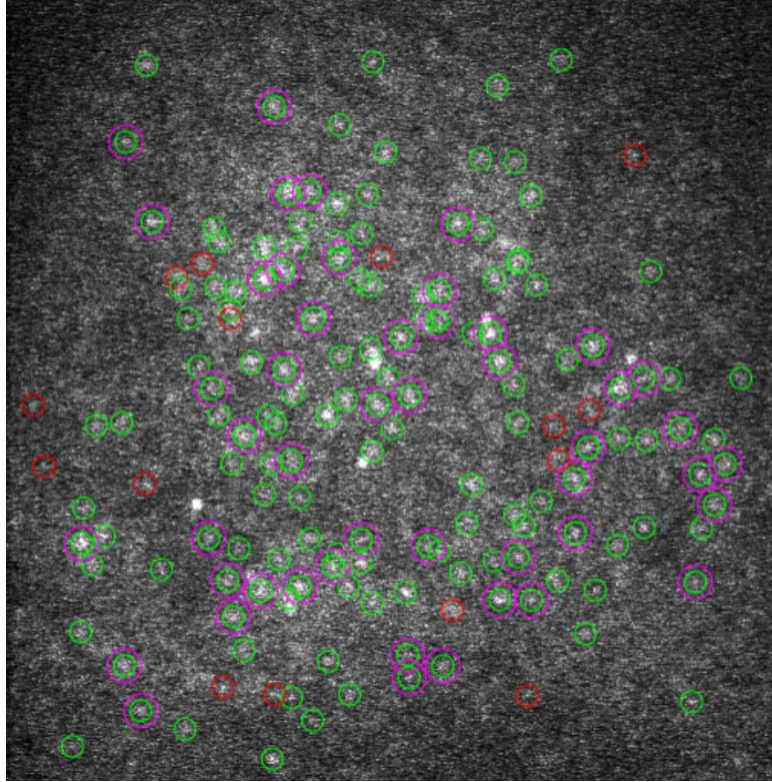
In the ViewFits movie the color scheme is

green circles are good fits

red circles are bad fits

magenta circles are fits which passed the tracking filter

An example frame is



Coordinate System Definition

BGSUBFIT runs entirely in Matlab, and as such the coordinate system used is the most meaningful for Matlab arrays. Instead of returning x & y positions in some arbitrary coordinate system, BGSUBFIT uses, and returns, coordinates in row & column position using Matlab conventions for rows & columns and the origin. Throughout most of the program the row & column number (integers) are tracked. The fitting step, which provides sub-pixel information returns a non-integer row & column position. For example, for a guess at $(r,c) = (41,153)$, after fitting the position may be saved as $(r,c) = (42.0597,153.9108)$.

For a more detailed tutorial on the coordinate system in Matlab and how it relates to the “normal” convention used in a Cartesian coordinate system, and also the coordinate system used in ImageJ, see the pdf included in this repository entitled *Image Coordinates in Matlab and ImageJ*.