Biophysical Journal, Volume *110*

Supplemental Information

Resolving Fast, Confined Diffusion in Bacteria with Image Correlation Spectroscopy

David J. Rowland, Hannah H. Tuson, and Julie S. Biteen

Contents

- SI Figures S1 – S4
- SI Table S1
- SI Movie S1 Caption

Matlab codes for generating simulation data (dataGen.m) and for the single-particle tracking (SPT) and spatio-temporal image correlation spectroscopy (STICS) analyses (dataParse.m) and accessory code (bpassDJR.m, gaussFit.m)

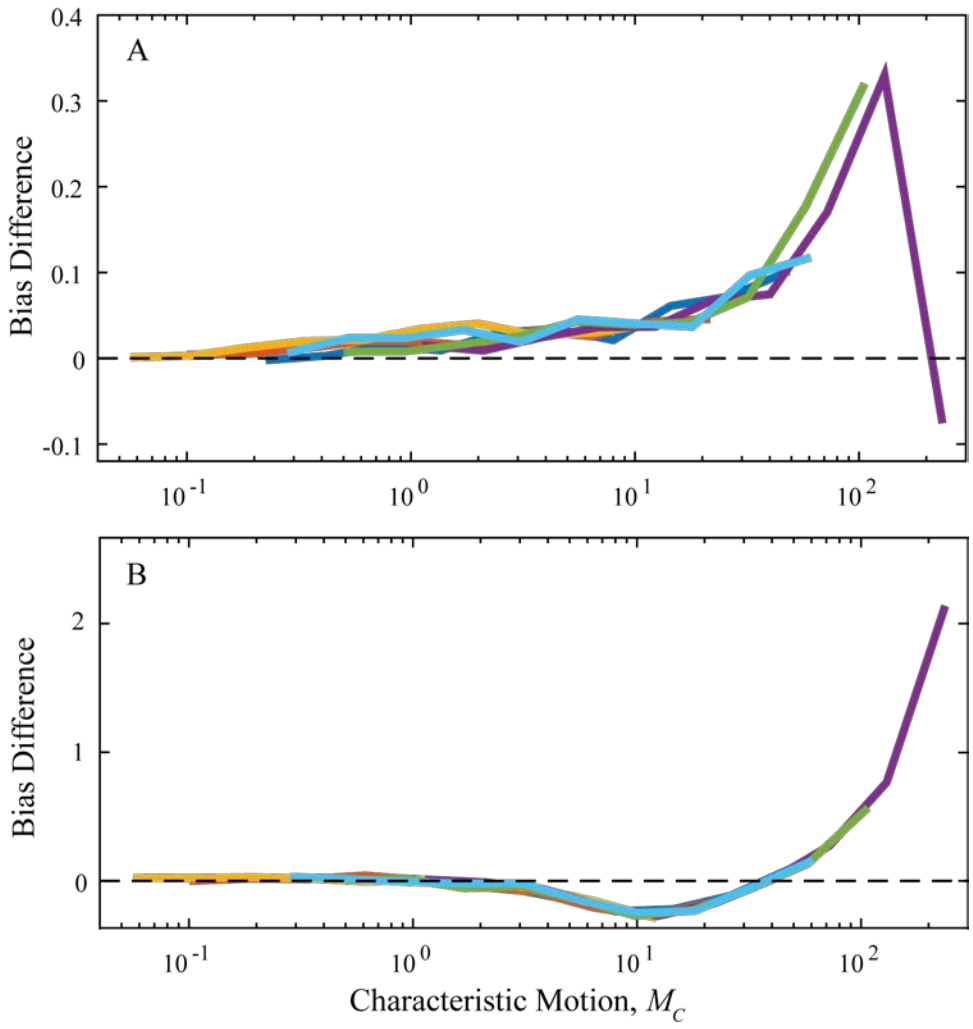

Figure S1. (A) Figures 3A and 3B showed the bias in determination by Gaussian fitting STICS of the diffusion coefficient, *D*, in the presence and absence of in-frame motion, respectively. Here, the difference between these two biases shows that removing in-frame motion does decrease the bias, especially at $M_C > 10$. (B) Figures 3A and 3C showed the bias in determination of the diffusion coefficient, *D*, by Gaussian fitting STICS and by STICS with direct computation of the variance, respectively. Here, the difference between these two biases highlights the large magnitude of the confinement artifact in Gaussian fitting STICS, which overwhelms the in-frame motion artifact in magnitude. Colors indicate the set of parameters (t_{frame}, σ_{PSF}) (SI Table S1) and M_C is defined according to Eq. 3B.

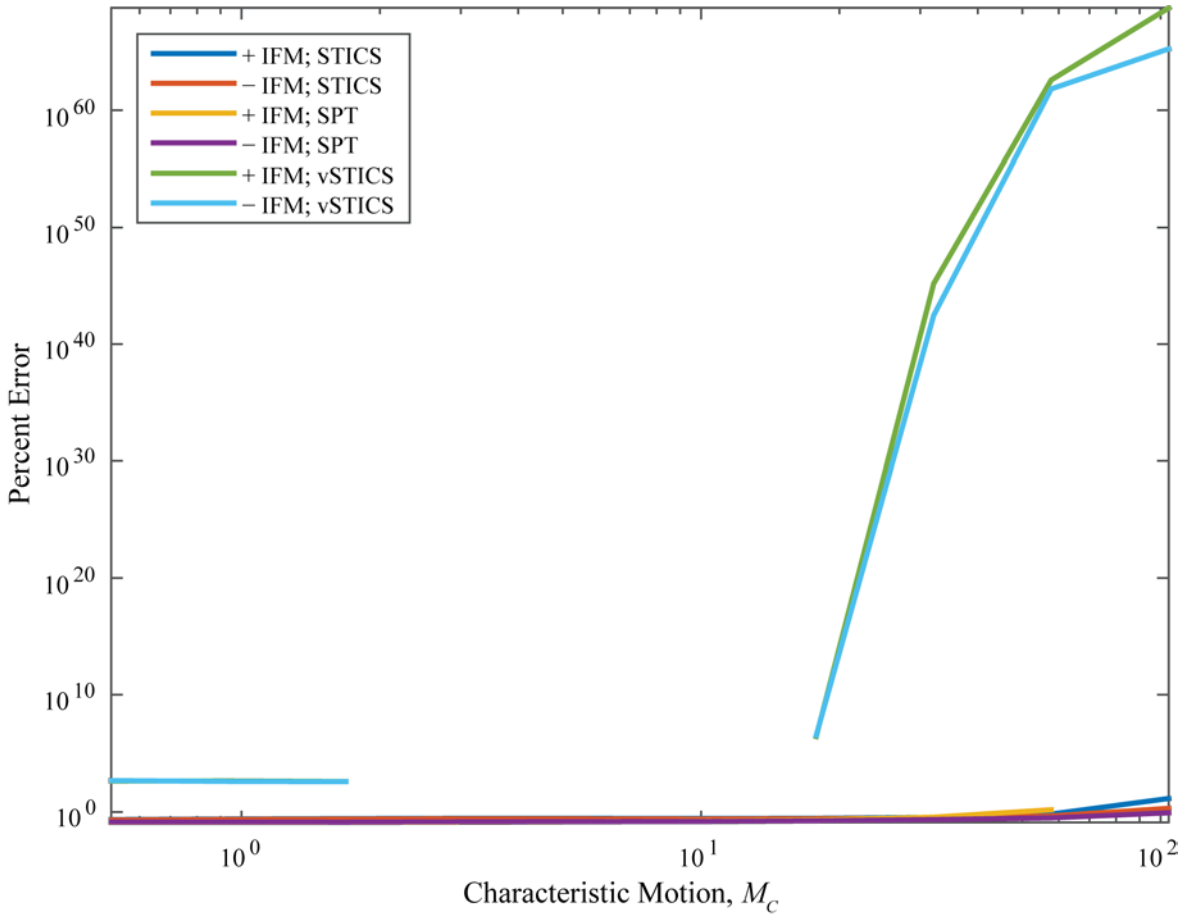

Figure S2. Measurement variance of STICS with the two methods of variance computation. Dark blue, red, yellow, and purple curves as in Figure 4. The green and light blue lines give the percent error when STICS is used to analyze data with and without in-frame motion (IFM), when the Gaussian-fitting step is replaced by direct variance computation. The missing points in the light blue and green curves (direct variance method, vSTICS) represent a complete failure to estimate the value, and even when the analysis algorithm had sufficient data to succeed the percent error was never below 1000%. The M_C is defined according to Eq. 3B.

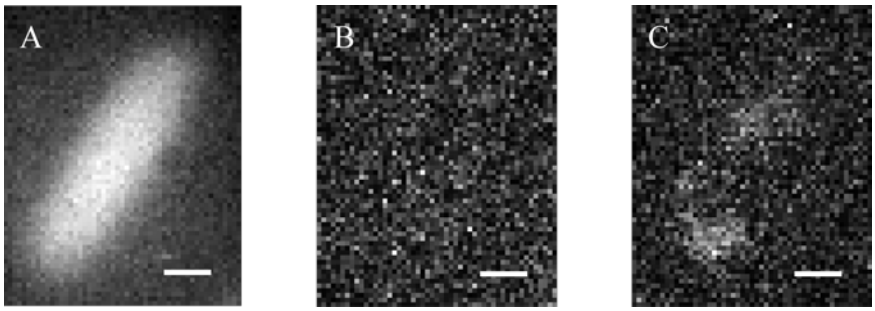

Figure S3. (A) Time-averaged fluorescence image obtained from summing a time series of fluorescence images of mMaple3 diffusion inside an *E. coli* cell (full time series in Movie S1). (B) A single image (t_{frame} = 40 ms) from that time series shows a cell after the mMaple3 molecule has bleached. (C) An image of the same *E. coli* cell in 'B' two frames earlier, showing a typical mMaple3 molecule diffusing so rapidly that it is diffuse over nearly the entire bacterium. Scale bars: 0.5 µm.

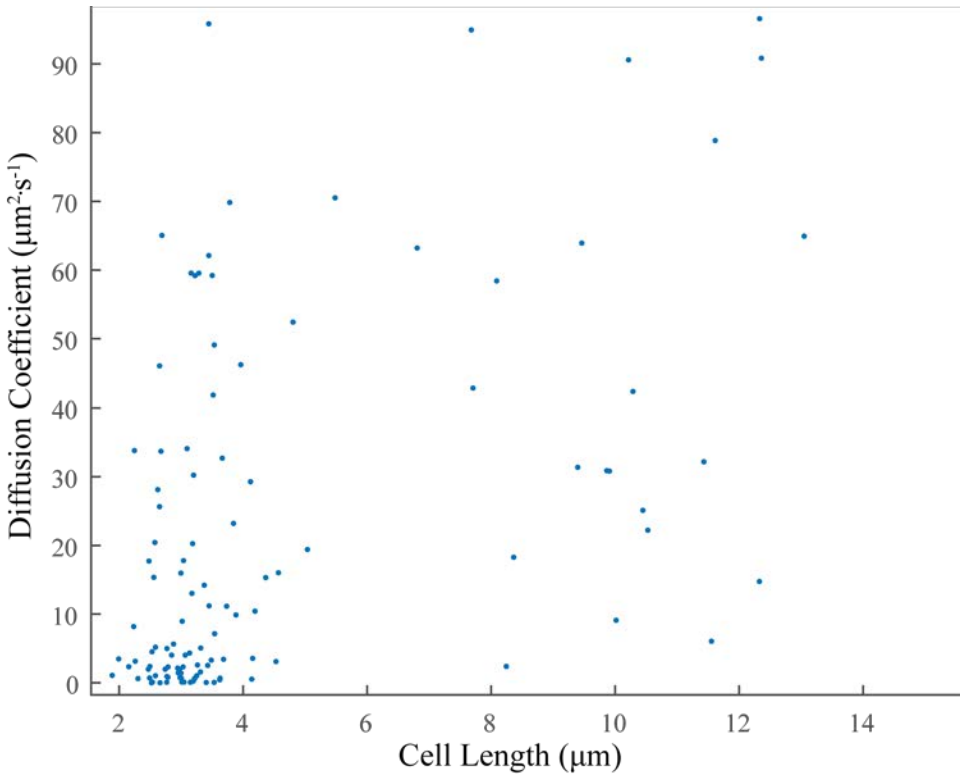

Figure S4. The measured diffusion coefficient for mMaple3 molecules in each *E. coli* cell did not strongly depend on the cell length calculated from the phase contrast image of that cell.

Table S1. Line colors for the simulated data in Figures 2, 3, 6 and S1.

	t_{frame} (ms)	σ_{PSF} (nm)	L (μm)	Line Color
Figure 2	10	49	NA	
	10	98	NA	
	10	146	NA	
	50	49	NA	
	50	98	NA	
	50	146	NA	
Figures 3, 6, and S1	10	98	2	
	10	98	3	
	10	98	4	
	50	98	2	
	50	98	3	
	50	98	4	

Movie S1. Photoswitching and imaging of mMaple3 in an *E. coli* cell. To visualize single molecules, a low-power 406-nm laser (shown in the movie as a flash in the background) was used to photoswitch 0 – 1 copies of mMaple3 per cell at a time. Two frames from this movie are presented as Figures S3B and S3C. Imaging rate: 25 frames per second. The movie area is 2.50 μm × 3.14 μm.

MATLAB CODE: bpassDJR.m

```
function res = bpassDJR(image_array,lnoise,lobject,threshold,lzero)
%
% NAME:
%       bpassDJR
% PURPOSE:
%       Implements a real-space bandpass filter that suppresses
%       pixel noise and long-wavelength image variations while
%       retaining information of a characteristic size.
%
% CATEGORY:
%       Image Processing
% CALLING SEQUENCE:
%       res = bpass( image_array, lnoise, lobject, threshold, lzero )
% INPUTS:
%       image:  The two-dimensional array to be filtered.
%       lnoise: Characteristic lengthscale of noise in pixels.
%           Additive noise averaged over this length should
%           vanish. May assume any positive floating value.
%           May be set to 0 or false, in which case only the
%           highpass "background subtraction" operation is
%           performed.
%       lobject: (optional) Integer length in pixels somewhat
%           larger than a typical object. Can also be set to
%           0 or false, in which case only the lowpass
%           "blurring" operation defined by lnoise is done,
%           without the background subtraction defined by
%           lobject.  Defaults to false.
%       threshold: (optional) By default, after the convolution,
%           any negative pixels are reset to 0.  Threshold
%           changes the threshhold for setting pixels to
%           0.  Positive values may be useful for removing
%           stray noise or small particles.  Alternatively, can
%           be set to -Inf so that no threshholding is
%           performed at all.
%
% OUTPUTS:
%       res:    filtered image.
% PROCEDURE:
%       simple convolution yields spatial bandpass filtering.
% NOTES:
% Performs a bandpass by convolving with an appropriate kernel.  You can
% think of this as a two part process.  First, a lowpassed image is
% produced by convolving the original with a gaussian.  Next, a second
% lowpassed image is produced by convolving the original with a boxcar
% function. By subtracting the boxcar version from the gaussian version, we
% are using the boxcar version to perform a highpass.
%
% original - lowpassed version of original => highpassed version of the
% original
%
% Performing a lowpass and a highpass results in a bandpassed image.
%
% Converts input to double.  Be advised that commands like 'image' display
% double precision arrays differently from UINT8 arrays.
```

```
% MODIFICATION HISTORY:
%       Written by David G. Grier, The University of Chicago, 2/93.
%
%       Greatly revised version DGG 5/95.
%
%       Added /field keyword JCC 12/95.
%
%       Memory optimizations and fixed normalization, DGG 8/99.
%       Converted to Matlab by D.Blair 4/2004-ish
%
%       Fixed some bugs with conv2 to make sure the edges are
%       removed D.B. 6/05
%
%       Removed inadvertent image shift ERD 6/05
%
%       Added threshold to output.  Now sets all pixels with
%       negative values equal to zero.  Gets rid of ringing which
%       was destroying sub-pixel accuracy, unless window size in
%       cntrd was picked perfectly.  Now centrd gets sub-pixel
%       accuracy much more robustly ERD 8/24/05
%
%       Refactored for clarity and converted all convolutions to
%       use column vector kernels for speed.  Running on my
%       macbook, the old version took ~1.3 seconds to do
%       bpass(image_array,1,19) on a 1024 x 1024 image; this
%       version takes roughly half that. JWM 6/07
%
%   This code 'bpass.pro' is copyright 1997, John C. Crocker and
%   David G. Grier.  It should be considered 'freeware'- and may be
%   distributed freely in its original form when properly attributed.
%
%
% 2016 edits by David J. Rowland, The University of Michigan:
% added lzero to the input list instead of just declaring it inside the function.

if nargin < 3, lobject = false; end
if nargin < 4, threshold = 0; end

normalize = @(x) x/sum(x);

image_array = double(image_array);

if lnoise == 0
  gaussian_kernel = 1;
else
  gaussian_kernel = normalize(...
    exp(-((-ceil(5*lnoise):ceil(5*lnoise))/(2*lnoise)).^2));
end

if lobject
  boxcar_kernel = normalize(...
     ones(1,length(-round(lobject):round(lobject))));
end

% JWM: Do a 2D convolution with the kernels in two steps each.  It is
% possible to do the convolution in only one step per kernel with
```

```
%
  % gconv = conv2(gaussian_kernel',gaussian_kernel,image_array,'same');
  % bconv = conv2(boxcar_kernel', boxcar_kernel,image_array,'same');
%
% but for some reason, this is slow.  The whole operation could be reduced
% to a single step using the associative and distributive properties of
% convolution:
%
  % filtered = conv2(image_array,...
  %   gaussian_kernel'*gaussian_kernel - boxcar_kernel'*boxcar_kernel,...
  %   'same');
%
% But this is also comparatively slow (though inexplicably faster than the
% above).  It turns out that convolving with a column vector is faster than
% convolving with a row vector, so instead of transposing the kernel, the
% image is transposed twice.

gconv = conv2(image_array',gaussian_kernel','same');
gconv = conv2(gconv',gaussian_kernel','same');

if lobject
  bconv = conv2(image_array',boxcar_kernel','same');
  bconv = conv2(bconv',boxcar_kernel','same');

  filtered = gconv - bconv;
else
  filtered = gconv;
end

% commented out because why do it in the first place?
% % Zero out the values on the edges to signal that they're not useful.
% lzero = max(lobject,ceil(5*lnoise));
% lzero=0;

filtered(1:(round(lzero)),:) = 0;
filtered((end - lzero + 1):end,:) = 0;
filtered(:,1:(round(lzero))) = 0;
filtered(:,(end - lzero + 1):end) = 0;

% JWM: I question the value of zeroing out negative pixels.  It's a
% nonlinear operation which could potentially mess up our expectations
% about statistics.  Is there data on 'Now centroid gets subpixel accuracy
% much more robustly'?  To choose which approach to take, uncomment one of
% the following two lines.
% ERD: The negative values shift the peak if the center of the cntrd mask
% is not centered on the particle.

% res = filtered;
filtered(filtered < threshold) = 0;
res = filtered;
end
```

MATLAB CODE: dataGen.m

```
function [v, simProps] = dataGen(varargin)
%
% NAME:
%       dataGen
% PURPOSE:
%       Generates space and time resolved single-molecule imaging data of a
%                               single diffuser.
% CATEGORY:
%       Data Simulation
% CALLING SEQUENCE:
%       [v, simProps] = dataGen(boundaryCondition, blurFlag);
% INPUTS:
%       varargin:  use paired inputs to set the property (input 1) to the
%          value (input 2) desired.
%
%       Properties:             Descriptions:
%
%       D                   diffusion coefficient in microns^2/s
%
%       tFrame                  image frame integration time in seconds
%
%       pixSize                 Pixel size in micrometers
%
%       psfSize                 standard deviation (width) of the microscope's
%                           point spread function in micrometers. i.e.
%                           FWHM = sqrt(2*log(2)) * psfSize
%
%       celSize                 1x2 vector: [width (diameter), height
%                           (length)] of the bounding box (cylinder)
%                           depending on the confinement condition
%
%       nFrames                 number of frames to be simulated
%
%       SNR                     signal to noise ratio (ratio of maximum
%                           signal amplitude to standard deviation of
%                           background noise)
%
%       confBool                1 for confined to the interior of a
%                           cylinder, and 0 for free diffusion.
%
%       blurFlag                1 for blurry motion, 0 for 'stroboscopic
%                           illumination'
%
% OUTPUTS:
%       v:                      simulated image time sequence
%       simProps:               properties of the simulation in the Matlab structure format
% PROCEDURE:
%       1. Simulate molecular trajectory
%       2. Evaluate pixel intensities
%                       3. Add noise
% MODIFICATION HISTORY:
%       Written by David J. Rowland, The University of Michigan, 3/16.
% NOTES:
%       This code 'dataGen.m' should be considered 'freeware'- and may be
%       distributed freely in its original form when properly attributed.
```

```
%
%       For testing purposes, this line makes a movie of confined, blurry
%       diffusion.
%
%       v=dataGen('confined', 1)


%% Default simulation parameters
D = .01;              % diffusion coefficient in microns^2/s
tFrame = .05;                       % frame integration time in seconds
pixSize = .049;                     % width of pixels in microns
psfSize = .098;                     % s.d. of the psf in microns
celSize = [1,3];       % [width, length] of confinement cylinder in microns
nFrames = 1e3;                      % number of frames in the simulated movie
SNR = 20;                                   % signal to noise ratio for added white noise (0:inf)
confBool = 1;          % 'confined' or 'unconfined'
blurFlag = 1;          % include blur subframes or not

% algorithmically-determined image size designed to disallow edge effects
imSize = ceil(celSize/pixSize+4*ceil(psfSize/pixSize));

% check variable size
if prod([imSize(1:2),nFrames])*8/1e9>1
   warning('video is over a GB. manually pass this block if you wish to continue')
end

% minimum increment of speed in units of microns^2/subframe
dtRef = 0.0001;

% initialize simulation properties structure
simProps.D = D;
simProps.tFrame = tFrame;
simProps.pixSize = pixSize;
simProps.psfSize = psfSize;
simProps.celSize = celSize;
simProps.nFrames = nFrames;
simProps.SNR = SNR;
simProps.blurFlag = blurFlag;
simProps.nSubs = [];
simProps.dtRef = dtRef;
simProps.confBool = confBool;

% if any sim parameters are included as inputs, change the simulation
% parameters mentioned
if ~rem(nargin,2)
   fNames=fieldnames(simProps);
   for ii=1:2:nargin
      whichField = strcmp(fNames,varargin{ii});

      if all(~whichField)
         warning('Check spelling. Parameter change may have not occurred')
      end

      eval([fNames{whichField} ' = varargin{ii+1}'])
      eval(['simProps.' fNames{whichField} ' = ' fNames{whichField},';'])
   end
```

```
elseif ~rem(nargin,1)
   warning('use paired inputs')
   v=[];
   return
end

% number of subframes required
nSubs = ceil(D*tFrame/dtRef);

% update the value of the diffusion coefficient since rounding may change it.
D = nSubs*dtRef/tFrame;

%% Trajectory generation
if confBool
   % confined particle trajectory
   mLocs = zeros(3, nFrames);
   for ii = 1 : nFrames*nSubs-1    % this loop can be compiled to mex64 to increase its speed
      % three 1d steps pulled from normal distribution with variance 2*dtRef
      step = sqrt(2*dtRef) * randn(3,1);

      candPos = mLocs(:,ii) + step;
      prevPos = mLocs(:,ii);

      % the ordering in the celSize vector matters because of this line:
      r = celSize(1)/2;

      % if the candidate position is outside of the cylinder in the x/z dimensions, reflect the step
      % against the inside of the cylinder. path length is preserved.
      if sqrt(sum(candPos([1,3]).^2)) > celSize(1)/2
         m = (candPos(3)-prevPos(3)) / (candPos(1)-prevPos(1));
         b = candPos(3)-m*candPos(1);

         xi=[(-m*b+sqrt(-b^2+r^2+m^2*r^2))/(1+m^2),...
            (-m*b-sqrt(-b^2+r^2+m^2*r^2))/(1+m^2)];
         yi=m*xi+b;

         % there are two solutions. the one closest to the candidate position is chosen. the farther
         % one is on the other side of the cell.
         whichone = (xi-candPos(1)).^2 + (yi-candPos(3)).^2 + (xi-prevPos(1)).^2 + (yi-prevPos(3)).^2;
         xip = [xi(find(whichone == min(whichone))),yi(find(whichone == min(whichone)))];

         normv = -xip/sqrt(sum(xip.^2));
         l = sqrt(sum((candPos([1,3])'-xip).^2));
         pf = 2*sum((prevPos([1,3])'-xip).*normv)*normv-(prevPos([1,3])'-xip);
         pf = pf/sqrt(sum(pf.^2))*l+xip;

         % replace x/z components of the position with the reflected x/z components
         out=candPos;
         out([1,3])=pf;

         candPos=out;
      end

      % if the candidate position is outside of the cylinder in the y dimension (cell's long axis)
      if candPos(2) < -celSize(2)/2
```

```
            candPos(2) = 2*-celSize(2)/2 - candPos(2);
        end
        if candPos(2) > celSize(2)/2
            candPos(2) = 2*celSize(2)/2 - candPos(2);
        end

        mLocs(:, ii+1) = candPos;
    end
else

    % unconfined particle trajectory
    mLocs=cumsum(sqrt(2*dtRef) * randn(3,nFrames*nSubs),2);
end

%% movie generation

if blurFlag
    % arrange tracks for subframe averaging
    tr_x = zeros(nSubs, nFrames);
    tr_y = zeros(nSubs, nFrames);
    for ii = 1:nFrames
        tr_x(:, ii) = mLocs(1, 1+(ii-1)*nSubs : ii*nSubs);
        tr_y(:, ii) = mLocs(2, 1+(ii-1)*nSubs : ii*nSubs);
    end
else
    % just use the first subframe from each frame
    tr_x = mLocs(1, 1:nSubs:end);
    tr_y = mLocs(2, 1:nSubs:end);
end

% shift to positive values
tr_x=tr_x+celSize(1)/2;
tr_y=tr_y+celSize(2)/2;

% noiseless pixel intensities
v=zeros([imSize(1:2),nFrames]); cx = 0; cy = 0;
for ii=-2*psfSize:pixSize:celSize(1)+2*psfSize        % x pixel locations with padding
    cx = cx+1;
    for jj=-2*psfSize:pixSize:celSize(2)+2*psfSize    % y pixel locations with padding
        cy = cy+1;

        % symmeteric gaussian function approximation of Airy Disk
        v(cx,cy,:) = mean(exp(-((ii-tr_x).^2+(jj-tr_y).^2)/2/psfSize^2),1);
    end
    cy = 0;
end

% add white noise
v = v + 1/SNR*randn(size(v));

simProps.D = D;
simProps.tFrame = tFrame;
simProps.pixSize = pixSize;
simProps.psfSize = psfSize;
simProps.celSize = celSize;
simProps.nFrames = nFrames;
```

```
simProps.SNR = SNR;
simProps.blurFlag = blurFlag;
simProps.nSubs = nSubs;
simProps.dtRef = dtRef;
simProps.confBool = confBool;
end
```

MATLAB CODE: dataParse.m

```
function resStruct = dataParse(data, mask)
%
% NAME:
%       dataParse
% PURPOSE:
%       Analyze single-molecule imaging data with tracking or STICS
% CATEGORY:
%       Image Processing
% CALLING SEQUENCE:
%       resStruct = dataParse(data,flag);
% INPUTS:
%       data:               x by y by t movie of fluorescence data
%       mask:               (optional) 2d binary roi selection mask
% OUTPUTS:
%       resStruct:          fitting result 'structure'
% PROCEDURE:
%       1. Peak fitting, then MSD calculation a la single molecule analysis
%       2. Correlation function calculation and width estimation
%       3. Diffusion coefficient estimation by MSD fitting
% MODIFICATION HISTORY:
%       Written by David J. Rowland, The University of Michigan, 3/16.
% NOTES:
%       This code 'dataParse.m' should be considered 'freeware'- and may be
%       distributed freely in its original form when properly attributed.
%
%       For testing purposes, run this script:
%
%       [v,sP]=dataGen('D',.1,'tFrame',.05,'nFrames',500);
%       r = dataParse(v)
%       plot(r.iMSDs); hold all
%       plot(r.MSDs)
%       plot(r.MSDd); hold off


%% analysis parameters
tFrame = 0.05;          % camera integration time in seconds
nTau = 5;               % number of smallest time lag values to use (excluding 0)
pixelSize = .049;       % pixel size in microns
nFrames = size(data,3); % number of frames in the data
yesOverlap = 1;         % use overlapping time lags?
isConfined = 1;         % use confined MSD curve fit?
padStyle = 1;           % padding style for STICS analysis
                    % 1 : pad zeros, 2: pad mean, 3: pad mean outside mask

%% single molecule tracking analysis

% spot fitting.
fitP = zeros(nFrames, 6);
parfor ii = 1:nFrames
   % this loop may be parallelized by simply replaceing 'for' with 'parfor'
   % and starting a parallel pool before running the code.
   fitP(ii,:) = gaussFit(data(:,:,ii));
end

% 'tracking'. missed spots will register as nans.
```

```
tr = fitP(:,1:2) * pixelSize;

% calculate mean squared displacements, one for each time lag value
MSDs=zeros(nTau,2);
for ii=1:nTau
   if yesOverlap                    % overlapping frame pairs
      indvec1=ii+1:nFrames;
      indvec2=1:nFrames-ii;
   elseif ~yesOverlap               % nonoverlapping frame pairs
      indvec2=1:ii:nFrames;
      indvec1=indvec2(1:end-1);
      indvec2=indvec2(2:end);
   end

   % mean squared displacements vs time lag
   MSDs(ii,:)=nanmean((tr(indvec2,:)-tr(indvec1,:)).^2,1);
end

%% STICS analysis

% pad images
switch padStyle
   case 1
      data = padZeros(data);
   case 2
      data = padMean(data);
   case 3
      data = padMask(data,mask);
end

% time-space correlation function calculation
if yesOverlap                       % overlapping frame pairs
   famps=abs(fft(fft(fft(data,[],1),[],2),[],3)).^2;
   STCorr = fftshift(fftshift(real(ifft(ifft(ifft(famps...
      ,[],1),[],2),[],3)),1),2)/numel(famps)/mean(data(:))^2-1;
   STCorr = STCorr(:,:,2:nTau+1);

elseif ~yesOverlap                  % nonoverlapping frame pairs
   vFft=fft2(data);
   STCorr=zeros(size(vFft,1),size(vFft,2),nTau+1);

   STCorr(:,:,1)=mean(bsxfun(@times,real(fftshift(fftshift(...
      ifft2(vFft.*conj(vFft)),1),2)),1./(mean(mean(data)).^2)),3);

   for kk=1:nTau
      ind1 = 1:kk:vidsize(3);
      ind2 = ind1(2:end);
      ind1 = ind1(1:end-1);
      STCorr(:,:,kk+1) = mean(bsxfun(@times,real(fftshift(fftshift( ...
         ifft2(vFft(:,:,ind2).*conj(vFft(:,:,ind1))),1),2)), ...
         1./mean(mean(data(:,:,ind1))./mean(mean(data(:,:,ind2)))),3);
   end
   STCorr=STCorr/numel(vFft(:,:,1))-1;
end

% estimate the widths of the correlation function
```

```
iMSDs=zeros(nTau,2); MSDd=iMSDs;
[x,y]=ndgrid(1:size(STCorr,1),1:size(STCorr,2));
for ii = 1:nTau
    fitP = gaussFit(STCorr(:,:,ii),'widthGuess',5,'nPixels',...
        min(size(STCorr(:,:,1))),'findTheSpot',0);
    iMSDs(ii,:) = fitP(3:4).^2 * pixelSize^2;

    % discrete variance calculation for x dimension
    pmf=sum(STCorr(:,:,ii)/sum(sum(STCorr(:,:,ii))),2);
    MSDd(ii,1) = sum(pmf.*(x(:,1)-mean(x(:))).^2) * pixelSize^2;

    % discrete variance calculation for x dimension
    pmf=sum(STCorr(:,:,ii)/sum(sum(STCorr(:,:,ii))),1);
    MSDd(ii,2) = sum(pmf.*(y(1,:)-mean(y(:))).^2) * pixelSize^2;
end

%% MSD Fitting

% choose fitting function
if isConfined
    f=@(p,X)sqconfMSD1D(p,X);
    pStart = [.1, .1, 0];
    lb = [0, 0, -inf];
    ub = [inf, inf, inf];
else
    f=@(p,X) 2*p(2)*X+p(1);
    pStart = [0, .1];
    lb = [-inf, 0];
    ub = [inf, inf];
end

% time lag domain vector
tau = (1:nTau)'*tFrame;

% D from tracking
pT(:,1)=lsqcurvefit(f,pStart,tau,MSDs(:,1),lb,ub);
pT(:,2)=lsqcurvefit(f,pStart,tau,MSDs(:,2),lb,ub);

% D from STICS
pS(:,1)=lsqcurvefit(f,pStart,tau,iMSDs(:,1),lb,ub);
pS(:,2)=lsqcurvefit(f,pStart,tau,iMSDs(:,2),lb,ub);

% D from 'discrete variance'
pD(:,1)=lsqcurvefit(f,pStart,tau,MSDd(:,1),lb,ub);
pD(:,2)=lsqcurvefit(f,pStart,tau,MSDd(:,2),lb,ub);

resStruct.Dtracking = pT(2,:);
resStruct.Dstics =  pS(2,:);
resStruct.Dvar =  pD(2,:);
resStruct.MSDs = MSDs;
resStruct.iMSDs = iMSDs;
resStruct.MSDd = MSDd;

end

function zi=sqconfMSD1D(p,X)
```

```
l=p(1);
d=exp(p(2));
ns=p(3);
tau=X;

summedTerm=@(t,d,l,n)1/n^4*exp(-(n*pi/l).^2*d*t);

temp=eps*ones(size(tau));
for ii=1:2:2*400-1
   s=summedTerm(tau,d,l,ii);
   if sum(s./temp)<1e-10
      break
   end
   temp=temp+s;
end
zi=l^2/6*(1-96/pi^4*temp)+ns;
zi(isnan(zi))=eps;
zi(isinf(zi))=eps;
end

function imStack=padMask(imStack,mask)
% replace pixels outside the mask with the average value inside the mask in
% each frame

imsize=size(imStack);
mMean=mean(reshape(imStack(mask(:,:,ones(1,imsize(3)))),[],imsize(3)));
imStack(~mask(:,:,ones(1,imsize(3))))=mMean(ones(1,sum(~mask(:))),:);
end

function imstack=padMean(imstack)
% pad the first two dimensions to double size with the mean of each image
imstack=mat2cell(imstack,size(imstack,1),size(imstack,2),ones(1,size(imstack,3)));
imm=cellfun(@(x)mean(mean(x(mask))),imstack,'uniformoutput',false);

imstack=cellfun(@(x,y)padarray(x,floor(size(x)/2),y),imstack,imm,...
   'uniformoutput',false);
imstack=cat(3,imstack{:});
end

function imstack=padZeros(imstack)
% pad the first two dimensions to double size with zeros
imstack=mat2cell(imstack,size(imstack,1),size(imstack,2),ones(1,size(imstack,3)));

imstack=cellfun(@(x,y)padarray(x,floor(size(x)/2),y),...
   imstack,repmat({0},1,1,size(imstack,3)),'uniformoutput',false);
imstack=cat(3,imstack{:});
end
```

MATLAB CODE: gaussFit.m

```
function [fitPars, conf95]=gaussFit(img, varargin) %findTheSpot, plottingFlag)
%
% NAME:
%       gaussFit
% PURPOSE:
%       Fits a generalized gaussian function to 2d imaging data. This code
%       produces results in units of pixels for the center position and
%       widths.
% CATEGORY:
%       Image Processing
% CALLING SEQUENCE:
%       [fitPars, conf95] = gaussFit(img,findTheSpot);
% INPUTS:
%       img:            The two-dimensional array to be fit to a gaussian
%
%       varargin:       use paired inputs to set the property (input 1) to the
%          value (input 2) desired.
%
%       Properties:     Descriptions:
%
%       findTheSpot:    1 or 0. Default behavior is to fit an
%          ROI in the center of the image. If the spot is not near the
%          center or the image is very large, findTheSpot enables the code
%          to first roughly locate the spot and then use that location as
%          the ROI center.
%
%       plottingFlag:   1 or 0. show plotting output. default is 0.
%
%       widthGuess:     set the starting value for the width of the
%                       Gaussian in units of pixels.
%
%       nPixels         pixel width of ROI to be selected from img. default
%                       is 11. the value should be odd.
%
% OUTPUTS:
%       fitPars:        fitting coefficient vector, all units are pixels.
%       fitCI:          95% confidence interval of fitting coefficients at
%                       end of fitting
% PROCEDURE:
%       1. Peak guessing and/or data ROI selection of local area inside img
%       2. Non-linear least squares minimization for 7 (or 6) - parameter
%          Gaussian function on the ROI selected.
% MODIFICATION HISTORY:
%       Written by David J. Rowland, The University of Michigan, 3/16.
% NOTES:
%       This code 'gaussFit.m' should be considered 'freeware'- and may be
%       distributed freely in its original form when properly attributed.
%
%       For testing purposes, run this script:
%
%       img = exp(-x.^2/2/2^2-y.^2/2/3^2)+.02*randn(size(x));
%       p = gaussFit(img,'widthGuess',2);

% if any sim parameters are included as inputs, change the simulation
% parameters mentioned
```

```
if nargin>1
    fNames={'findTheSpot', 'plottingFlag', 'widthGuess', 'nPixels'};
    for ii=1:2:nargin-1
        whichField = strcmp(fNames,varargin{ii});

        if all(~whichField)
            warning('Check spelling. Parameter change may have not occurred.')
        end

        eval([fNames{whichField} ' = varargin{ii+1};'])
    end

elseif rem(nargin,1)
    warning('use paired inputs if using varargin.')

    % empty output. size must change if the gaussian fitting function is changed.
    fitPars = nan(1,6);
    conf95 = nan(1,6);
    return
end

%% declaring fitting predicates

if ~exist('findTheSpot','var')
    findTheSpot = 1;
end
if ~exist('plottingFlag','var')
    plottingFlag = 0;
end
if ~exist('widthGuess','var')
    widthGuess = 2;
end
if ~exist('nPixels','var')
    nPixels = 11;
end

% freely rotating bivariate gaussian function for least squares minimization
% parameters: [xCenter, yCenter, angle, xSD, ySD, amplitude, offset]
% xR=@(x,y,xc,yc,th)(x-xc)*cos(th)-(y-yc)*sin(th);
% yR=@(x,y,xc,yc,th)(x-xc)*sin(th)+(y-yc)*cos(th);
% f=@(p,X) exp( -xR(X(:,1), X(:,2), p(1), p(2), p(3)).^2/2/p(4)^2 + ...
%     -yR( X(:,1), X(:,2), p(1), p(2), p(3)).^2/2/p(5)^2 ) *p(6) + p(7);

% fixed angle fit
% parameters: [xCenter, yCenter, xSD, ySD, amplitude, offset]
th=0;
xR=@(x,y,xc,yc)(x-xc)*cos(th)-(y-yc)*sin(th);
yR=@(x,y,xc,yc)(x-xc)*sin(th)+(y-yc)*cos(th);
f=@(p,X) exp( -xR(X(:,1), X(:,2), p(1), p(2)).^2/2/p(3)^2 + ...
    -yR( X(:,1), X(:,2), p(1), p(2)).^2/2/p(4)^2 ) *p(5) + p(6);

% bounds
lb=[-inf, -inf, 0, 0, -inf, -inf];
ub=[inf, inf, inf, inf, inf, inf];
```

```
%% data selection

if findTheSpot
   % select the local area around a bright spot in a larger image

   % bandpass and threshold
   LP=1;          % low pass value
   HP=10;         % high pass value
   intThresh=0.1;  % intensity threshold. set to zero and then check by inspection
   hMax=0.1;      % larger if the dynamic range of your data is larger
   lzero=4;       % this squelches a 5 pixel boundary around the filtered image

   bIm=bpassDJR(img, LP, HP, intThresh, lzero);

   % watershed algorithm
   extImg=imextendedmax(bIm,hMax);

   % failed watershed algorithm can result in all ones
   if all(extImg(:))
      extImg=extImg-1;
   end

   % shrink to a point. this is the estimated location of the spot
   sIm=bwmorph(extImg,'shrink',inf);

   % the index of the one pixel is a good guess for the particle location
   [locInds(:,1),locInds(:,2)]=find(sIm);

   % temporally coincdident guesses are not treated with this code.
   if size(locInds(:,1))~=1
      fitPars=nan(1,6);
      conf95=nan(1,6);
      return
   end

else
   % otherwise, assume the spot is in near the center of the image
   locInds=round(size(img)/2);
end

% pad the img(s) with nans (removed later).
padsize=[nPixels,nPixels];
padVal=nan;
direction='both';
img=padarray(img,padsize,padVal,direction);
locInds=locInds+nPixels;

% find the selection domain
[sDom1,sDom2]=ndgrid(locInds(1)-(nPixels-1)/2:locInds(1)+(nPixels-1)/2, ...
   locInds(2)-(nPixels-1)/2:locInds(2)+(nPixels-1)/2);
inds=sub2ind(size(img),sDom1(:),sDom2(:));

% select the data
truImg=reshape(img(inds),[nPixels,nPixels]);

%% starting parameter selection for 6-parameter Gaussian Fit
```

```
% x, y centers starting guess
pStart(1)=0;
pStart(2)=0;

% xSD, ySD in units of pixels
pStart(3)=widthGuess;
pStart(4)=widthGuess;

% amplitude, offset
mVals=[max(truImg(:)),min(truImg(:))];
pStart(5)=mVals(1)-mVals(2);
pStart(6)=mVals(2);

%% fitting the data

[x,y]=ndgrid(1:nPixels,1:nPixels);
X=cat(2,x(:),y(:)) - nPixels/2;
[fitPars, ~, residual, ~, ~, ~,jacobian] = ...
   lsqcurvefit(f,pStart,X(~isnan(truImg(:)),:),truImg(~isnan(truImg(:))),...
   lb,ub);

% confidence intervals
conf95 = nlparci(fitPars, residual,'jacobian',jacobian);

%% plot the output
if plottingFlag
   fVals=reshape(f(fitPars,X),[nPixels,nPixels]);
   dVals=truImg;
   sVals=reshape(f(pStart,X),[nPixels,nPixels]);

   subplot(221)
   title('Data')
   pcolor(kron(dVals,ones(10)))
   shading flat; axis image; colorbar

   subplot(222)
   title('starting values')
   pcolor(kron(sVals,ones(10)))
   shading flat; axis image; colorbar

   subplot(223)
   title('fit result')
   pcolor(kron(fVals,ones(10)))
   shading flat; axis image; colorbar

   subplot(224)
   title('residuals')
   pcolor(kron(dVals - fVals,ones(10)))
   shading flat; axis image; colorbar
end

% shift center back to lab frame
fitPars([1,2])=fitPars(1:2)+locInds-nPixels;
end
```