SUPPLEMENTARY MATERIAL

DROPLETD2.m

I. OVERVIEW

The MATLAB function DROPLETD2.m determines the diameter of a circular object as the diameter evolves over the course of a series of consecutive images. The function has been developed for use with the images produced in the experiments performed in Cornell University's droplet combustion laboratory, but with slight modification it can be used in general for similar purposes. This document assumes that the program is being used to determine the diameter of a spherical droplet. The text below crudely describes how the function works, and describes the inputs and outputs of the function. The program listing is given in Appendix 1 below. For an understanding of the actual code, refer to the related paper.

II. HOW IT WORKS

After DROPLETD2 is called, the program prompts the user to select a point within the droplet in the first image in the series. For each image, the function performs two steps to determine the droplet's diameter. The first step is edge detection. Based on the current value of the parameter `threshold`, the function converts the input image into a black and white image (pixels with a color value greater than `threshold` are given a pixel value of 255, or white, and those equal to or below are given a pixel value of 0, or black). The pixels where the function finds the transition from black to white are considered to be on the edge of the droplet. The function creates a list of these *edge points*. The second step is to fit either a circle or an ellipse to the collected edge points. This fitting is done iteratively: the function continually recalculates the fit until two consecutive fits produce a sufficiently similar diameter for the droplet. The function runs this loop for each image, and the analysis of one image on a typical personal computer takes about 1 second.

III. INPUT

The function call is `D = DROPLETD2(FIT, DCT, IMGDIR, FIRST, LAST)`. `FIT` and `DCT` are discussed below. `IMGDIR` is the directory containing the image files (*e.g.* `'C:\20100228\02V'`). `FIRST` is the index in the image's filename for the first image to be used for the analysis. `LAST` is the index of last image to be used for the analysis (to analyze images from File0015.jpg to File0030.jpg, `FIRST` is 15 and `LAST` is 30). In the current version of the function, it is assumed that the images are JPG files.

The format for the images' filenames is defined by the value for the parameter **`filename`**. As an example, if `filename = 'File0000'`, `first` = 15, `last` = 30, then the images should have filenames from File0015.jpg to File0030.jpg).

Once DROPLETD2 is called, it creates a figure window containing the first image in the series. The function prompts the user to select approximately the droplet's center. For subsequent images the function assumes that the center of the droplet in the next images is close to the center of the droplet in the previous image.

There are many ways to tune the operation of the function by either varying the function inputs or the values of internal function parameters.

1. Function Inputs

`FIT`: For a value of 1, the function attempts to fit a *circle* to the collected edge points. For a value of 2, the function attempts to fit an *ellipse* to the collected edge points. `FIT` = 2 usually provides more precise results, but may be more finicky for noisy images.

DCT: Dynamic color threshold. For a value of 0, the function uses `threshold1` as the value of `threshold` for all images (see below). For a value of 1, the value of `threshold` is updated for each image based on average color values in and around the droplet in the previous image (for the first image, `DCT` is achieved by analyzing the first image twice). `DCT` can be set as 0 for images in which the color values around the droplet remain the same throughout the series of images. The runtime of the program does not differ significantly for the two different values of `DCT`.

2. Function Parameters

increment: For a value of 1, the function analyzes each image between `FIRST` and `LAST`. For a value of 2, the function analyzes only every other image between `FIRST` and `LAST`, etc. The default value is 1.

searchlimit1: This is the initial value for the variable `searchlimit`. This parameter determines the number of pixels the function should search in the horizontal and vertical directions from the initial mouse-clicked input for the "center" of the droplet. A value that is too large might deceive the function into thinking that extraneous objects in the images are part of the droplet (the function looks too far out). A value that is too small will prevent the function from finding the edge of the droplet. The user must only select this parameter for the first image analyzed, as the function determines an appropriate value to use for `searchlimit` for subsequent images in the series.

threshold1: This is the initial value for the variable `threshold`, which separates black pixels from white pixels. A `threshold` value of 100 means that a pixel with a color value between 101 and 255 is converted into a white pixel (255) and a pixel with a color value between 0 and 100 is converted into a black pixel (0). This parameter is only significant when `DCT` = 0, as the function determines the value to use for `threshold` on its own when `DCT` = 1. When `DCT` = 0, `threshold1` is used as the value of `threshold` for all images. The default value is 100.

tolerance: For a given image, the function iteratively fits a circle or ellipse to the collected edge points. After calculating a fit, the function discards points that are far from the fit. Then the function performs the fit again using the edge points that were not discarded. The iterations stop when two consecutive fits provide diameters that are within the tolerance (relative error) given by the parameter `tolerance`. This convergence usually requires about 10 iterations per image. The default value is 10^{-4}.

eliminator: This parameter determines how far a collected edge point must be from a circle or ellipse fit to be to be discarded. Points beyond a distance of `eliminator×r` from the fit are discarded, where `r` is the radius of the fit). The default value is 0.1.

f: This parameter is only significant when `DCT` = 1. It is a scaling factor for the value of `threshold` (see Eq. 1) which is determined by an average of the color values across the droplet and the region around the droplet. When `DCT=1`, `threshold` is set as exactly the average of the color values picked across the droplet boundary. For `f` = 0.67, `threshold` is set as 0.67 of the average of those color values. The value of `f` that provides the most precise results varies with other parameters and with the nature of the images being analyzed.

IV. OUTPUT

The function produces three outputs. The first is a diameter matrix **D**. For a circle fit with `DCT` = 0, the matrix has only one column. This column holds the diameter of the droplet in each image. For a circle fit with `DCT` = 1, the matrix has two columns, where the second column holds the value of `threshold` used for each image. For an ellipse fit with `DCT` = 0, the matrix has three columns, where the second two hold the major and minor axis lengths of the ellipse used for each image. For an ellipse fit with `DCT` = 1, the matrix has four columns, where the fourth column holds the value of `threshold` used for each image.

The other two outputs are files that are placed in a folder in the directory `IMGDIR`. The first of these is a series of JPG images (one for each input) showing, on the input images, the pixels detected on the border of

the droplet as green dots, the pixels discarded as red dots, and and the value of the diameter determined for that image. The third output is a data file (.TXT) that contains the parameters used for the run as well as the diameter matrix **D**. This file can be imported to both MATLAB and Excel.

Program Listing

```
function D = dropletd2(fit,dct,imgdir,first,last)
%DROPLETD Determine diameter evolution for spherical droplet combustion.
%   [D] = DROPLETD2(FIT,DCT,IMGDIR,FIRST,LAST) analyzes all photos in
directory
%   IMGDIR with filenames of the format File0000000.jpg between indices
%   FIRST and LAST (inclusive), where FIRST and LAST are integer
scalars,
%   according to a circular (FIT = 1) or elliptical (FIT = 2) fit.
%   For FIT = 1, D is a column vector containing the diameter (in
pixels)
%   for the droplet in each photo between FIRST and LAST. For FIT = 2,
D is
%   a 3-column array containing the geometric mean of major and mminor
axis
%   lengths, the major axis lengths, and minor axis lengths. DCT
toggles
%   dynamic color thresholding, and is only needed for sooty data. If
%   DCT = 0, the same color threshold value will be used for edge
detection
%   for all images. If DCT = 1, the program will use an average color
%   across the droplet boundary to constantly update the color
threshold.
%
% INPUTS
% fit     (int) 1 for circular least squares fit, 2 for elliptical
least
%         squares fit.
% dct     (int) 0 for constant threshold, 1 for dynamic color threshold.
% imgdir  (string) the full pathname containing JPG images for
consideration.
% first   (int) index of the first image of interest.
%         for example: the image with filename File0000078.jpg contains
the
%         image of the droplet at the moment of ignition.
% last    (int) index of the last image of interest.
%
% OUTPUTS
% D       (array) droplet diameter in pixels, size (last-first+1,1),
for
%         fit = 1, dct = 0. geometric mean of major and minor axis
lengths,
%         major axis lengths, minor axis lengths (all doubled), size
%         (last-first+1,3), for fit = 2, dct = 0. additional column if
%         dct = 1.
% (JPG) for each photo analyzed, (last-first+1) many, image indicating
the
%         detected droplet boundary and least squares fitted conic
section
%         and diameter (semiaxis) result.
% (TXT) containing a timestamp, program runtime, parameter values used
%         (searchlimit1, threshold, tolerance, eliminator) and matrix D
%         (comma delimited for fit = 2 or dct ~= 0).
%
% SAMPLE FUNCTION CALL
%  D = dropletd2(2,1,'C:\20100228\02v',72,105)
```

```
%
% LIMITATIONS
% The position of the droplet must overlap between consecutive images
or
% the function will have trouble determining where the droplet is.
% At least half of a droplet must be visible in an image for the image
to
% be analyzed.
%
% CALIBRATION
% For the output of this program to mean anything useful, a calibration
% image must be run through the program to obtain a conversion between
% pixel units and physical units. The calibration image is specific to
the
% experimental setup. The calibration run should use the same settings
% (fit, dct, and all parameter values) as those used to analyze data
from
% an experiment.
%
% PARAMETERS
% increment     interval of images to skip between first and last image
(5
%               if only every 5th image between first and last are to
be
%               analyzed).
% searchlimit1  how far to look for a droplet boundary for the first
image.
% threshold1    the approximate value of the color (0 to 255) at the
%               droplet's edge in the first image.
% tolerance     decides when a radius (diameter) estimate is sufficient.
% eliminator    decides which points in vectors X, Y are discarded.
% f             affects how the dynamic threshold
%               is chosen.
%
% This function has been developed by Chris Dembia (cld72@cornell.edu)
and
% Frank Liu (yl677@cornell.edu).

increment = 1;
searchlimit1 = 300;
searchlimit = searchlimit1;
threshold1 = 100;
threshold = threshold1; %0 is black, 255 is white
tolerance = 10^(-4);
eliminator = .1;
f = 0.67;

%% CHECK FOR INPUT ERRORS

if fit ~= 1 && fit ~= 2
    error('Input FIT must be either 1 for circle fit or 2 for ellipse
fit.')
end
if dct ~=0 && dct ~= 1
    error('Input DCT must either be 0 for static threshold or 1 for
DCT.')
end
```

```
if round(first) ~= first
    error('Input FIRST must be an integer.')
end
if round(last) ~= last
    error('Input LAST must be an integer.')
end

%% INITIALIZE

% If dir does not end in a backslash, add a backslash.
% but don't do it if the imgdir string is null ('')
if imgdir(end) ~= '\' &&  ~isempty(imgdir)
    imgdir = [imgdir '\'];
end

% Set up the filename of the images to be analyzed.
filename = 'File0000000';
filename(end-length(num2str(first))+1:end) = num2str(first);

% Obtain the first image.
imgrgb = imread([imgdir filename '.jpg']);

[N, M, P] = size(imgrgb); % We assume all images have the same
dimensions.

% Determine if input image is 1-D (grayscale) or 3-D.
% Typically the input images are 3-D despite being grayscale.
if P == 3
    img = rgb2gray(imgrgb);
else % P = 1
    img = imgrgb;
end

%% OUTPUT PREPARATION
% Create output directory.
outputdir = ['output' datestr(now,30) '_' num2str(first) '-'
num2str(last)];
mkdir(imgdir,outputdir) % create dir where outputs will go.

% Output run-identifying information, parameters, and diameter array.
% File is saved in outputdir as a .txt, can be imported by at least
Excel
% and MATLAB.
fid = fopen([imgdir outputdir '\output.txt'],'w+');
if fit == 1
    fprintf(fid,'%s\n',['Droplet Circle Fit (least squares): '
datestr(now,31)]);
else
    fprintf(fid,'%s\n',['Droplet Ellipse Fit (least squares): '
datestr(now,31)]);
end
fprintf(fid,'%s\n',[imgdir ' files ' num2str(first) ' to '
num2str(last)]);
fprintf(fid,'%s\n',sprintf('searchlimit1: %d',searchlimit1));
fprintf(fid,'%s\n',sprintf('threshold1: %d',threshold));
fprintf(fid,'%s\n',sprintf('tolerance: %1f',tolerance));
fprintf(fid,'%s\n',sprintf('eliminator: %1f',eliminator));
```

```
fprintf(fid,'%s\n',sprintf('f: %1f',f));
if fit == 1 % Put a header thing in the output file.
    if dct == 0
        fprintf(fid,'%s\n',sprintf('D'));
    else
        fprintf(fid,'%s\n',sprintf('D,threshold'));
    end
else
    if dct == 0
        fprintf(fid,'%s\n',sprintf('D,D1,D2'));
    else
        fprintf(fid,'%s\n',sprintf('D,D1,D2,threshold'));
    end
end
% Diameters are added to the file as they are calculated.

% Initialize a little bit more.
theta = 0:.1:2.01*pi; % This will help plot a circle/ellipse on each
output jpg.

% Output matrix. If fit = 1, dct = 0, then only diameters are output.
% If fit = 1, dct = 1, then diameters and threshold values are output.
% If fit = 2, dct = 0, then diameters, 2*semimajor, 2*semiminor are
output.
% If fit = 2, dct = 1, then threshold values are also output.
D = zeros(last-first+1,2*fit-1+dct);

%% PROMPT USER FOR INITIAL DROPLET LOCATION

% Set up the first figure.
h = figure('Name','DROPLETD2.m: Choose a point within the droplet');
figure(h)
image(imgrgb) % Plot image.
title([filename ', ' datestr(now,31)])
xlabel('Use the mouse to click a point inside the droplet.')
axis image
hold on

% Intake initial values of r and c, the row and column coordinates.
[xg, yg] = ginput(1); % we want to start somewhere inside the droplet.
tic % and we're off!
xc = round(xg); % these don't need to be near the center of the droplet.
yc = round(yg);

set(h,'Visible','off')
plot(xg,yg,'o') % the ginput point

% Loop through all images.
scrap = 1; % the first run-through is scrapped.
loopvector = [first first:increment:last];
for count = loopvector

    % Initialize output (unless it's the first run through).
    if ~scrap
        filename(end-length(num2str(count))+1:end) = num2str(count);
        imgrgb = imread([imgdir filename '.jpg']);
        if P == 3
```

```
            img = rgb2gray(imgrgb);
        else
            img = imgrgb;
        end
        h = figure('Visible','off','Name','DROPLETD2.m');
        image(imgrgb)
        title([filename ', ' datestr(now,31)])
        axis image
        hold on
    end

    % Image Conditioning.
    % Examine each "quadrant" of the droplet as specified by the
initial input.
    % Convert grayscale image to a black-and-white image according to
the
    % threshold parameter.
    img2 = 255*(img > threshold);

    % Coarse Coordinate Gathering.
    % Initialize the arrays that will house the parameterized points of
the circle.
    X = [];
    Y = [];

    if xc > M || yc > N
        error('The fit has diverged. Sorry.')
    end

    % QUADRANT I
    c = min(xc + searchlimit,M); % column index.
    while c > xc
        isEdge = 0;
        r = yc; % row index.
        while r > 1 && abs(r - yc) < searchlimit && ~isEdge
            isEdge = (img2(r,c,1) == 0) && ~(img2(r-1,c,1) == 0);
            if isEdge % concatenate these points onto our list.
                X = [X c];
                Y = [Y r];
            end
            r = r - 1; % Step upward.
        end
        c = c - 1; % Step leftward.
    end

    % QUADRANT II
    c = xc;
    while c > 1 && abs(c - xc) < searchlimit
        isEdge = 0;
        r = yc;
        while r > 1 && abs(r - yc) < searchlimit && ~isEdge
            isEdge = (img2(r,c,1) == 0) && ~(img2(r-1,c,1) == 0);
            if isEdge
                X = [X c];
                Y = [Y r];
            end
            r = r - 1; % Step upward.
```

```
        end
        c = c - 1; % Step leftward.
    end

    % QUADRANT III
    c = max(xc - searchlimit,0);
    while c < xc
        isEdge = 0;
        r = yc;
        while r < N && abs(r - yc) < searchlimit && ~isEdge
            isEdge = (img2(r,c,1) == 0) && ~(img2(r+1,c,1) == 0);
            if isEdge
                X = [X c];
                Y = [Y r];
            end
            r = r + 1; % Step downward.
        end
        c = c + 1; % Step rightward.
    end

    % QUADRANT IV
    c = xc;
    while c < M && abs(c - xc) < searchlimit
        isEdge = 0;
        r = yc;
        while r < N && abs(r - yc) < searchlimit && ~isEdge
            isEdge = (img2(r,c,1) == 0) && ~(img2(r+1,c,1) == 0);
            if isEdge
                X = [X c];
                Y = [Y r];
            end
            r = r + 1; % Step downward.
        end
        c = c + 1; % Step rightward.
    end

    %% CIRCLE OR ELLIPSE? Execute fit for current image
    if fit == 1 % Circle

        % Recalculate radius until we're below the error tolerance.
        k = 0;
        err = inf; %the loop must run at least once.
        r_last = inf; %the first err will thus always be infinity.

        while err > tolerance %tolerance is a parameter
            if length(X) < 3
                error('Data is not clean enough; need at least 3 good
points for a circle.')
            end

            % Local Iteration fit.
            L = floor(length(X)/3);
            xc = zeros(L,1);
            yc = zeros(L,1);
            r  = zeros(L,1);

            % Partition the collected edge point into 3 bins and fit a
```

```
            % circle to many sets of three points, one point taken from
each bin in
            % order.
            for i = 1:L
                xin = [X(i); X(i+L); X(i+2*L)];
                yin = [Y(i); Y(i+L); Y(i+2*L)];

                [xc(i), yc(i), r(i)] = circlelsq(xin,yin);

            end

            if isempty(r)
                error('Not enough data collected to generate a circle
fit.')
            end

            [r, I] = sort(r); % Take the circle with the median radius.
            r = r(max(ceil(L/2),1));
            I = I(max(ceil(L/2)));
            xc = xc(I); % Choose the corresponding center.
            yc = yc(I);

            plot(xc,yc,'ro') % Plot the center of the circle from this
iteration.

            % Eliminate bad boys.
            i = 1;
            while i <= length(X)
                dist = sqrt((X(i) - xc)^2 + (Y(i) - yc)^2);
                if abs(dist - r) > eliminator*r
                    plot(X(i),Y(i),'r.','MarkerSize',3)
                    X = [X(1:max(1,i-1)) X(min(i+1,end):end)];
                    Y = [Y(1:max(1,i-1)) Y(min(i+1,end):end)];
                end
                i = i + 1;
            end
            k = k + 1;

            err = abs(r-r_last)/r;
            r_last = r;

        end

        % Diameter achieved, clean up & prepare for tomorrow.

        % the center of the current circle become the interior starting
point
        % for the next image.
        xc = round(xc);
        yc = round(yc);

        idx = count - first + 1;
        D(idx,1) = 2*r;

        % Plot all collected datapoints that survived.
        plot(X,Y,'g.','MarkerSize',3)
        x = xc + r*cos(theta);
```

```
        y = yc + r*sin(theta);
        plot(x,y,'b-')

        if ~scrap % Don't execute this for the first run-through.
            if dct == 0
                % Send the diameter information to the output file.
                fprintf(fid,'%s\n',sprintf('%f',D(idx,1)));
                xlabel(sprintf('droplet diameter (D) = %.2f
pixels',D(idx,1)))
            else % Also output threshold information.
                D(idx,2) = threshold;
                fprintf(fid,'%s\n',sprintf('%f,%f',D(idx,1),D(idx,2)));
                xlabel({sprintf('threshold = %.2f',D(idx,2)),...
                    sprintf('droplet diameter (D) = %.2f
pixels',D(idx,1))})
            end
        end

        searchlimit = round(1.2*r); % Don't look too far out.

        if dct == 1
            % Extract the color values (0 to 255) along a 45, 90 and
135
            % degree line through the center of the droplet.
            [Section1, xg1, yg1] = ddsection2(img,xc,yc,D(idx,1),pi/4);
            [Section2, xg2, yg2] = ddsection2(img,xc,yc,D(idx,1),pi/2);
            [Section3, xg3, yg3] =
ddsection2(img,xc,yc,D(idx,1),pi/2+pi/4);
            threshold = f*mean([Section1; Section2; Section3]);

            % Plot the lines along which color values are sampled.
            plot([xg1 xg2 xg3],[yg1 yg2 yg3],'.-')
            % Make a cute inset plot of the color values along the
plotted lines.
            % 'Position' determines inset plot's location in the figure.
            axes('Position',[.2 .65 .2 .2],'Layer','top','Box','on',...
                 'XTick',[],'YTick',[0 255]);
            hold on
            plot(1:length(Section1),Section1,...

(1:length(Section2))*length(Section1)/length(Section2),Section2,...

(1:length(Section3))*length(Section1)/length(Section3),Section3)
            plot([1 length(Section1)],threshold*[1 1])
            axis([-inf inf 0 255])
        end

        if ~scrap % The first iteration is scrapped.
            saveas(h,[imgdir outputdir '\' filename '_1.jpg'])
        end
        set(h,'Visible','on')
        close(h)

    elseif fit == 2 % Ellipse

        k = 0;
        err = inf;
```

```
        r_last = inf;

        while err > tolerance
            if length(X) < 5
                error('Data is not clean enough; need at least 5 good
points for an ellipse.')
            end

            % Calculate ellipse fit using a subfuction with least
squares
            % stuff.
            [a,b,xc,yc,phi] = ellipselsq(X,Y);
            r = sqrt(a*b);

            plot(xc,yc,'ro')

            % Eliminate bad boys.
            i = 1;
            while i <= length(X)
                dist = sqrt((X(i) - xc)^2 + (Y(i) - yc)^2);
                if dist > (1 + eliminator)*a || dist < (1 -
eliminator)*b
                    plot(X(i),Y(i),'r.','MarkerSize',3)
                    X = [X(1:max(1,i-1)) X(min(i+1,end):end)];
                    Y = [Y(1:max(1,i-1)) Y(min(i+1,end):end)];
                end
                i = i + 1;
            end
            k = k + 1;

            err = abs(r-r_last)/r;
            r_last = r;

        end

        % Diameter achieved, clean up & prepare for tomorrow.

        idx = count - first + 1;
        xc = round(xc);
        yc = round(yc);
        D(idx,1) = 2*r;
        D(idx,2) = 2*a;
        D(idx,3) = 2*b;

        % Plot all collected datapoints that survived.
        plot(X,Y,'g.','MarkerSize',3)
        x = xc + a*cos(theta)*cos(phi) - b*sin(theta)*sin(phi);
        y = yc + a*cos(theta)*sin(phi) + b*sin(theta)*cos(phi);
        plot(x,y,'b-') % the fitted conic section.

        if ~scrap
            if dct == 0
                % Send the diameter information to the output file.

fprintf(fid,'%s\n',sprintf('%f,%f,%f',D(idx,1),D(idx,2),D(idx,3)));
                xlabel({sprintf('major axis 2a (D1) = %.2f
pixels',D(idx,2)),...
```

```
                    sprintf('minor axis 2b (D2) = %.2f
pixels',D(idx,3)),...
                    sprintf('droplet diameter (D) = %.2f
pixels',D(idx,1))})
            else
                D(idx,4) = threshold;
                % Send the diameter information to the output file.

fprintf(fid,'%s\n',sprintf('%f,%f,%f,%f',D(idx,1),D(idx,2),D(idx,3),D(i
dx,4)));
                xlabel({sprintf('threshold = %.2f',D(idx,4)),...
                    sprintf('semimajor axis 2a (D1) = %.2f
pixels',D(idx,2)),...
                    sprintf('semiminor axis 2b (D2) = %.2f
pixels',D(idx,3)),...
                    sprintf('droplet diameter (D) = %.2f
pixels',D(idx,1))})
            end
        end

        searchlimit = round(1.2*a); %don't look too far out.

        if dct == 1
            [Section1, xg1, yg1] = ddsection2(img,xc,yc,D(idx,1),pi/4);
            [Section2, xg2, yg2] = ddsection2(img,xc,yc,D(idx,1),pi/2);
            [Section3, xg3, yg3] =
ddsection2(img,xc,yc,D(idx,1),pi/2+pi/4);
            threshold = f*mean([Section1; Section2; Section3]);

            plot([xg1 xg2 xg3],[yg1 yg2 yg3],'.-')
            axes('Position',[.2 .65 .2 .2],'Layer','top','Box','on',...
                 'XTick',[],'YTick',[0 255]);
            hold on
            plot(1:length(Section1),Section1,...

(1:length(Section2))*length(Section1)/length(Section2),Section2,...

(1:length(Section3))*length(Section1)/length(Section3),Section3)
            plot([1 length(Section1)],threshold*[1 1])
            axis([-inf inf 0 256])
        end

        if ~scrap
            saveas(h,[imgdir outputdir '\' filename '_1.jpg'])
        end
        set(h,'Visible','on')
        close(h)

    end

    fprintf([filename ' processed; converged in %d iterations.\n'],k)

    scrap = 0; % Done with the first loop.
end % Have gone through all images.

%% OUTPUT
```

```
% Display total runtime of the program & close out of output file.
runtime = toc;
if runtime < 60
    fprintf('EXECUTION COMPLETE. Runtime: %.2f seconds\n',runtime)
else
    fprintf('EXECUTION COMPLETE. Runtime: %d min and %.2f seconds\n',...
        floor(runtime/60),mod(runtime,60))
end
fprintf(fid,'\n%s\n',sprintf('Runtime: %.2f seconds',runtime));
fclose(fid);

end

function [xc, yc, r] = circlelsq(X,Y)

% This least squares solution for a circle comes from the internet.
% http://www.infogoaround.org/JBook/LSQ_Circle.html

L = length(X);

A = [2*sum(X.^2) 2*sum(X.*Y) sum(X);
    2*sum(X.*Y) 2*sum(Y.^2) sum(Y);
    2*sum(X)    2*sum(Y)    L    ];
b = -[sum((X.^2 + Y.^2).*X);
    sum((X.^2 + Y.^2).*Y);
    sum( X.^2 + Y.^2)   ];

parameters = A\b;
xc = -parameters(1);
yc = -parameters(2);
r = sqrt(xc^2 + yc^2 - parameters(3));

end

function [semimajor_axis, semiminor_axis, x0, y0, phi] = ellipselsq(x,
y)
%
% This least squares solution for a circle comes from the Mathworks
website.
% http://www.mathworks.com
% ellipse_fit - Given a set of points (x,y), ellipse_fit returns the
% best-fit ellipse (in the Least Squares sense)
%
% Input:
%                       x - a vector of x measurements
%                       y - a vector of y measurements
%
% Output:
%
%                   semimajor_axis - Magnitude of ellipse longer axis
%                   semiminor_axis - Magnitude of ellipse shorter axis
%                   x0 - x coordinate of ellipse center
%                   y0-  y coordinate of ellipse center
%                   phi - Angle of rotation in radians with respect to
%                   the x-axis
%
% Algorithm used:
```

```
%
% Given the quadratic form of an ellipse:
%
%       a*x^2 + 2*b*x*y + c*y^2  + 2*d*x + 2*f*y + g = 0   (1)
%
%  we need to find the best (in the Least Square sense) parameters
a,b,c,d,f,g.
%  To transform this into the usual way in which such estimation
problems are presented,
%  divide both sides of equation (1) by a and then move x^2 to the
% other side. This gives us:
%
%       2*b'*x*y + c'*y^2  + 2*d'*x + 2*f'*y + g' = -x^2            (2)
%
%   where the primed parametes are the original ones divided by a.
%  Now the usual estimation technique is used where the problem is
%  presented as:
%
%    M * p = b,  where M = [2*x*y y^2 2*x 2*y ones(size(x))],
%    p = [b c d e f g], and b = -x^2. We seek the vector p, given by:
%
%    p = pseudoinverse(M) * b.
%
%    From here on I used formulas (19) - (24) in Wolfram Mathworld:
%    http://mathworld.wolfram.com/Ellipse.html
%
%
% Programmed by: Tal Hendel <thendel@tx.technion.ac.il>
% Faculty of Biomedical Engineering, Technion- Israel Institute of
Technology
% 12-Dec-2008
%
%-------------------------------------------------------------------------
----

x = x(:);
y = y(:);

%Construct M
M = [2*x.*y y.^2 2*x 2*y ones(size(x))];

% Multiply (-X.^2) by pseudoinverse(M)
e = M\(-x.^2);

%Extract parameters from vector e
a = 1;
b = e(1);
c = e(2);
d = e(3);
f = e(4);
g = e(5);

%Use Formulas from Mathworld to find semimajor_axis, semiminor_axis, x0,
y0
%, and phi

delta = b^2-a*c;
```

```
x0 = (c*d - b*f)/delta;
y0 = (a*f - b*d)/delta;

phi = 0.5 * acot((c - a)/(2*b));

nom = 2 * (a*f^2 + c*d^2 + g*b^2 - 2*b*d*f - a*c*g);
s = sqrt(1 + (4*b^2)/(a-c)^2);

a_prime = sqrt(nom/(delta* ( (c-a)*s -(c+a))));

b_prime = sqrt(nom/(delta* ( (a-c)*s -(c+a))));

semimajor_axis = max(a_prime, b_prime);
semiminor_axis = min(a_prime, b_prime);

if (a_prime < b_prime)
    phi = pi/2 - phi;
end


if abs(phi) < pi/4
    phi = -phi;
end

end

function [Section,xg,yg] = ddsection2(img,xc,yc,D,theta)

xg = zeros(2,1);
yg = zeros(2,1);

xg(1) = xc - D*cos(theta);
xg(2) = xc + D*cos(theta);
yg(1) = yc + D*sin(theta);
yg(2) = yc - D*sin(theta);

height = abs(yg(2)-yg(1));
width = abs(xg(2)-xg(1));

if sqrt(width^2 + height^2) < 5
    error('points are too close!')
end

xg = round(xg);
yg = round(yg);

if xg(2) < xg(1)
    xg = [xg(2); xg(1)];
    yg = [yg(2); yg(1)];
end

m = (yg(2) - yg(1))/(xg(2) - xg(1));
b = -xg(1)*m + yg(1);

% maximus = max(height,width);
% Section = zeros([maximus,1]);
```

```
if m == Inf

    Section = img(yg(1):yg(2),xg(1));

elseif m == 0

    Section = img(yg(1),xg(1):xg(2));

elseif m > 1/2 %more vertical than horizontal, positive
    %each row only has one img point extracted
    Section = img(yg(1),xg(1));

    xC = xg(1);

    for yC = yg(1) + 1:yg(2)

        xR = xC + .5;
        yR = m*xR + b;
        if yR < yC
            xC = xC + 1;
        end
        Section = [Section; img(yC,xC)];
    end

elseif m > 0 %more horizontal than vertical, positive
    %each column only has one img point extracted
    Section = img(yg(1),xg(1));

    yC = yg(1);

    for xC = xg(1) + 1:xg(2)

        yT = yC + .5;
        xT = (yT - b)/m;
        if xT < xC %cell (xC,yC+1) is chosen.
            yC = yC + 1;
        end
        Section = [Section; img(yC,xC)];
    end

elseif m >= -1/2 %more horizontal than vertical, negative
    %each column only has one img point extracted
    Section = img(yg(1),xg(1));

    yC = yg(1);

    for xC = xg(1) + 1:xg(2)
        yB = yC - .5;
        xB = (yB - b)/m;

        if xB < xC
            yC = yC - 1;
        end
        Section = [Section; img(yC,xC)];
    end
```

```
elseif m < -1/2 %more vertical than horizontal, negative
    %each row only has one img point extracted
    Section = img(yg(1),xg(1));

    xC = xg(1);

    for yC = yg(1) + 1:-1:yg(2)

        xR = xC + .5;
        yR = m*xR + b;
        if yR > yC
            xC = xC + 1;
        end
        Section = [Section; img(yC,xC)];
    end

end
end
```