Table of Contents

Initialization

```
% Image filename and extension
filename = 'Tgfbr1r2';
ext = '.tif';

% Unit conversion
nanometers_over_pixels = 500/626; % 500-nm scale bar is 626 pix long

% Scale bar region dimensions (assumed to be at the bottom-right of image),
% to be automatically excluded from analysis. Leave as empty vector [] if
% this does not apply.
scale_bar_dim = [80, 650]; % [height, width], both in pixels
```

Import and preprocessing

```
fprintf('Importing data and preprocessing...\n')

% Read image
I = imread([filename, ext]);

% Convert image to double
I = double(I)/255;

% Set scale bar region to NaN
if ~isempty(scale_bar_dim)
    I(end-scale_bar_dim(1):end, end-scale_bar_dim(2):end) = NaN;
end

% Downsize image by half until largest dimension is less than 1000 pixels
scale = 1; % Scale factor (how much to multiply dimensions by later to restore
 original dimensions)
while any(size(I) > 1e3)
```

```
    I = I(1:2:end, 1:2:end);
    scale = 2*scale;
end

% Pixel coordinates
[x, y] = meshgrid(1:size(I, 2), 1:size(I, 1));
```

Edit the image (by selecting regions to exclude)

```
fprintf('Interactive image editing:\n')

% Turn off warning that figure JavaFrame property will be obsoleted in a
% future release
warning('off', 'MATLAB:HandleGraphics:ObsoletedProperty:JavaFrame')

% Create "extended" image with 50-pixel white border, so that polygons can
% be drawn including the image edges
I_ext = ones(size(I) + [100, 100]); % All white
I_ext(50+(1:size(I,1)), 50+(1:size(I,2))) = I; % Image centered in all-white
 template

% Show image
h_fig = figure;
h_image = imshow(I_ext);
axis tight equal off

% Maximize figure
jf = get(h_fig, 'JavaFrame');
pause(0.1)
set(jf, 'Maximized', 1);

% Adjust axes location downward
hax = gca;
hax.Position(2) = hax.Position(2)/2;

% Set title
title({['\bf{Image ', strrep(filename, '_', '\_'), '}'], ...
    '\rm{Click to create a polygon around a region to exclude.}', ...
    '\rm{Create an empty polygon (by clicking the same point twice) to end
 editing.}'})

% Logical variable, indicating whether the last user-drawn polygon was
% empty
poly_is_empty = false;
while ~poly_is_empty
    % Get polygon drawn by user
    fprintf('\tWaiting for user to draw polygon...\n')
    h_poly = impoly;

    % Mask of the polygon region
    M = h_poly.createMask;
```

```
    % If polygon is empty (i.e., if mask is all false)
    if ~any(M(:))
        % Note that polygon was empty
        poly_is_empty = true;
    else
        % Edit image, with polygon region set to NaN
        I_ext(M) = NaN;
    end

    % Delete polygon, and refresh image
    h_poly.delete;
    h_image.CData = I_ext;
end

fprintf('\tEditing complete.\n')

% Close figure
close(h_fig)

% Overwrite I with the edited image
I = I_ext(50+(1:size(I,1)), 50+(1:size(I,2)));

% Overwrite M with all pixels where I is NaN (note: M is overwritten in the
% next cell, and is defined here just for debugging purposes)
M = isnan(I); %#ok

% Turn back on warning that figure JavaFrame property will be obsoleted in
% a future release
warning('on', 'MATLAB:HandleGraphics:ObsoletedProperty:JavaFrame')
```

Filtering, thresholding, and binarization

```
fprintf('Filtering, thresholding, and binarizing...\n')

% Gaussian filtered image
I_filt = imgaussfilt(I, 0.003*min(size(I)));

% Mask of excluded regions in filtered image
M_filt = isnan(I_filt);

% M_filt will generally cover more area than M because of the effect of
% filtering along the NaN boundaries. Change the original image I and mask
% M so that the original and filtered images' excluded regions match.
I(M_filt) = NaN;
M = isnan(I);

% Double-check that M and M_filt are identical
assert(isequal(M, M_filt), ...
    'Excluded region masks M and M_filt should be identical, but are not.')

% Quadratic function to set the spatially heterogeneous intensity threshold
mdl = fitlm([x(:), y(:)], I_filt(:), 'poly22', 'robustopts', 'cauchy');
```

```
% Local threshold
thres = reshape(predict(mdl, [x(:), y(:)]), size(x));

% Filtered & power-transformed image (pixels with intensity = thres are
% mapped to 0.5)
I_filt_tform = I_filt.^(log(0.5)./log(thres));

% Check for non-NaN imaginary values
if any(abs(imag(I_filt_tform)) > 0)
    error('Transformed image contains imaginary values.')
else
    I_filt_tform = real(I_filt_tform);
end

% Filtered & binarized image
I_filt_bin = imbinarize(I_filt_tform, ...
    graythresh(I_filt_tform(~M_filt))); % Otsu's method

% Set excluded regions to white in binarized image
I_filt_bin(M_filt) = true;
```

Show images

```
fig_image_original = figure;
imagesc(I)
colormap gray; colorbar; caxis([0,1]); axis tight equal
title 'Original image'

fig_image_filtered = figure;
imagesc(I_filt)
colormap gray; colorbar; caxis([0,1]); axis tight equal
title 'Filtered image'

fig_image_filtbin = figure;
imagesc(I_filt_bin)
colormap gray; colorbar; caxis([0,1]); axis tight equal
title 'Filtered & binarized image'

fig_image_overlay = figure;
imshow(I)
hold on
h = imshow(cat(3, 1-I_filt_bin, 0*I_filt_bin, 0*I_filt_bin)*255);
h.AlphaData = (1-I_filt_bin)/5;
title 'Original overlayed with filtered & binarized image'
```

Identify centroids based on distance threshold

```
fprintf('Detecting fibril centroids...\n')

% Euclidean distance to nearest light pixel
D = bwdist(I_filt_bin);
```

```
sigma = 0.5:0.5:10; % Gaussian filter bandwidths
N = nan(size(sigma)); % Number of detected fibrils
for i=1:length(sigma)
    tmp = bwconncomp(imregionalmax(imgaussfilt(D, sigma(i))));
    N(i) = tmp.NumObjects;
end

% Fit 5th-degree polynomial
p = polyfit(sigma, N, 5);

% Optimal sigma is the lowest sigma where the square of the 1st derivative
% reaches a (local) minimum. This corresponds to the lowest sigma where the
% polynomial slope is (locally) the nearest to zero, meaning the number of
% detected fibrils is minimally sensitive to bandwidth at this point.
der1_sq = conv(polyder(p), polyder(p)); % Square of 1st derivative
roots_der_der1_sq = roots(polyder(der1_sq)); % Roots of the derivative of
 der1_sq
sigma_opt = min(roots_der_der1_sq(imag(roots_der_der1_sq) == 0 & ...
    roots_der_der1_sq > 0)); % Smallest real critical point of der1_sq that is
 > 0
% Display sigma_opt
sigma_opt %#ok

% Regional maxima
regmax = imregionalmax(imgaussfilt(D, sigma_opt));
regmax_conncomp = bwconncomp(regmax);

% Centroid coordinates
x_centroid = cellfun(@(C) mean(x(C)), regmax_conncomp.PixelIdxList)';
y_centroid = cellfun(@(C) mean(y(C)), regmax_conncomp.PixelIdxList)';

% Show original image overlayed with filtered & binarized image and Voronoi
% diagram of centroids
fig_voronoi = figure;
imshow(I)
hold on
[vx, vy] = voronoi(x_centroid, y_centroid);
h_centroid = plot(x_centroid, y_centroid, 'r+', 'linewidth', 2);
h_vnoi = plot(reshape([vx; nan(1, size(vx, 2))], [], 1), ...
    reshape([vy; nan(1, size(vy, 2))], [], 1), 'b');
```

Interactively add and delete centroids

```
fprintf('Waiting for user to adjust fibril centroids...\n')

% Axis limits
axlim = 0.5 + [0, size(I,2)/2, 0, size(I,1)/2;
    size(I,2)/2, size(I,2), 0, size(I,1)/2;
    0, size(I,2)/2, size(I,1)/2, size(I,1);
    size(I,2)/2, size(I,2), size(I,1)/2, size(I,1);
    0, size(I,2), 0, size(I,1)];

% Names for each view that will be shown to the user
```

```
view_name = {'(1/5) Top-left quadrant', '(2/5) Top-right quadrant', ...
    '(3/5) Bottom-left quadrant', '(4/5) Bottom-right quadrant', ...
    '(5/5) Entire image'};

% Adjust axes location downward
hax = gca;
hax.Position(2) = hax.Position(2)/2;

for i=1:size(axlim, 1) % For each set of axis limits
    % Set axis limits
    axis(axlim(i,:))

    % Set title
    title({['\bf{', view_name{i}, '}'], ...
        ['\rm{Left-click to add centroid. ', ...
        'Right-click to remove centroid. ', ...
        'Press Enter to continue.}']})

    button = 1;
    while ~isempty(button)
        [x_click, y_click, button] = ginput(1);
        if button == 1 % Left click: add centroid
            x_centroid(end+1) = x_click;
            y_centroid(end+1) = y_click;
            [vx, vy] = voronoi(x_centroid, y_centroid);
            delete(h_centroid)
            delete(h_vnoi)
            h_centroid = plot(x_centroid, y_centroid, 'r+', ...
                'linewidth', 2);
            h_vnoi = plot(reshape([vx; nan(1, size(vx, 2))], [], 1), ...
                reshape([vy; nan(1, size(vy, 2))], [], 1), 'b');
        elseif button == 3 % Right click: delete nearest centroid
            nearest_centroid_idx = knnsearch([x_centroid, y_centroid], ...
                [x_click, y_click]);
            x_centroid(nearest_centroid_idx) = [];
            y_centroid(nearest_centroid_idx) = [];
            [vx, vy] = voronoi(x_centroid, y_centroid);
            delete(h_centroid)
            delete(h_vnoi)
            h_centroid = plot(x_centroid, y_centroid, 'r+', ...
                'linewidth', 2);
            h_vnoi = plot(vx, vy, 'b');
        end
    end
end

title 'Original overlayed with filt./bin. image and Voronoi diagram'

% Adjust axes location upward (back to default position)
hax.Position(2) = hax.Position(2)*2;
```

Identify fibrils on the outer boundary of the image

```
fprintf('Identifying boundary fibrils...\n')

% Voronoi vertices and cells
[VORVERT, VORCELL] = voronoin([x_centroid, y_centroid]);

% Nearest pixel indices
nearest_pixel = knnsearch([x(:), y(:)], VORVERT);

% Logical vector (for each Voronoi cell, are any Voronoi vertices either
% outside of the image or within the excluded regions?)
is_bdy_centroid = cellfun(@(idx) any(any(VORVERT(idx,:) < 0)) || ...
    any(VORVERT(idx,1) > size(I,2)) || any(VORVERT(idx,2) > size(I,1)) || ...
    any(M_filt(nearest_pixel(idx))), ...
    VORCELL);
```

Use selected centroids to create a better binarized image

```
fprintf('Improving binarized image...\n')

% Nearest centroid for every image pixel
[nearest_centroid_idx_all, nearest_centroid_dist_all] = knnsearch( ...
    [x_centroid, y_centroid], [x(:), y(:)]);

% Set excluded regions to NaN
nearest_centroid_idx_all(M_filt) = NaN;

% Characteristic Voronoi cell intensity: For centroid i (and thus Voronoi
% cell i), what is the mean intensity (I_filt_tform) of pixels in cell i
% whose distance to centroid i is less than the 5th percentile distance to
% centroid i within that cell?
char_cell_intensity = nan(size(x_centroid));
for i=1:length(char_cell_intensity)
    char_cell_intensity(i) = nanmean( ...
        I_filt_tform(nearest_centroid_idx_all == i & ...
        nearest_centroid_dist_all < ...
        quantile(nearest_centroid_dist_all(nearest_centroid_idx_all == i),
 0.05)));
end

% Scattered (natural) interpolant of characteristic cell intensities
F = scatteredInterpolant(x_centroid, y_centroid, char_cell_intensity, ...
    'natural', 'nearest');

% Transform the (already-transformed) image again, the same way as above
% (pixels with intensity = F(x,y) are mapped to 0.5)
I_filt_tform_2 = I_filt_tform.^(log(0.5)./log(F(x,y)));
```

```
% Check for non-NaN imaginary values
if any(abs(imag(I_filt_tform_2)) > 0)
    error('Transformed image contains imaginary values.')
else
    I_filt_tform_2 = real(I_filt_tform_2);
end

% Binarize using Otsu's method
I_filt_bin_2 = imbinarize(I_filt_tform_2, ...
    graythresh(I_filt_tform_2(~M_filt)));

% Set excluded regions to white in binarized image
I_filt_bin_2(M_filt) = true;

% Show pair of old/new binarized images (newly added fibril pixels are
% green; removed fibril pixels are magenta)
fig_filtbin_pair = figure;
imshowpair(I_filt_bin, I_filt_bin_2)
title({'\bf{Pair of old/new binarized images}', ...
    ['\rm{(newly added fibril pixels are green; ', ...
    'removed fibril pixels are magenta)}']})
% Adjust axes location downward
hax = gca;
hax.Position(2) = hax.Position(2)/2;
```

Gaussian mixture model-based fibril clustering

```
fprintf('Fitting Gaussian mixture model...\n')

% Coordinates of fibril pixels
x_fibril = x(~I_filt_bin_2);
y_fibril = y(~I_filt_bin_2);

% Nearest centroid index for every fibril pixel
nearest_centroid_idx = knnsearch([x_centroid, y_centroid], ...
    [x_fibril, y_fibril]);

% Data thinning increment
thin_inc = 10;

% Fit Gaussian mixture model using nearest centroid indices as initial
% cluster indices
warning('off', 'stats:gmdistribution:FailedToConverge')
% Initial fit (1 iteration) to define GMM and relevant variables
GMM = fitgmdist([x_fibril(1:thin_inc:end), ...
    y_fibril(1:thin_inc:end)], length(x_centroid), ...
    'start', nearest_centroid_idx(1:thin_inc:end), ...
    'options', statset('maxiter', 1), ...
    'regularizationvalue', 0.1);
mu = GMM.mu;
Sigma_0 = GMM.Sigma;
Sigma = Sigma_0;
```

```
ComponentProportion_0 = GMM.ComponentProportion;
ComponentProportion = ComponentProportion_0;
GMM = gmdistribution(mu, Sigma, ComponentProportion);
cluster_idx = cluster(GMM, [x_fibril(1:thin_inc:end), ...
    y_fibril(1:thin_inc:end)]);

% Adjust mu and ComponentProportion, keeping Sigma fixed
for i=1:25 % 25 iterations
    GMM = fitgmdist([x_fibril(1:thin_inc:end), ...
        y_fibril(1:thin_inc:end)], ...
        length(x_centroid), ...
        'start', cluster_idx, 'options', statset('maxiter', 1), ...
        'regularizationvalue', 0.1);
    mu = GMM.mu;
    ComponentProportion = GMM.ComponentProportion;
    GMM = gmdistribution(mu, Sigma, ComponentProportion);
    cluster_idx = cluster(GMM, [x_fibril(1:thin_inc:end), ...
        y_fibril(1:thin_inc:end)]);

    % Display progress
    if mod(2*i, 10) == 0
        fprintf('\t%i%% complete...\n', 2*i)
    end
end
% Adjust Sigma and ComponentProportion, keeping mu fixed
for i=26:50 % 25 iterations
    GMM = fitgmdist([x_fibril(1:thin_inc:end), ...
        y_fibril(1:thin_inc:end)], ...
        length(x_centroid), ...
        'start', cluster_idx, 'options', statset('maxiter', 1), ...
        'regularizationvalue', 0.1);
    Sigma = GMM.Sigma;
    ComponentProportion = GMM.ComponentProportion;
    GMM = gmdistribution(mu, Sigma, ComponentProportion);
    cluster_idx = cluster(GMM, [x_fibril(1:thin_inc:end), ...
        y_fibril(1:thin_inc:end)]);

    % Display progress
    if 2*i == 100
        fprintf('\t%i%% complete.\n', 2*i)
    elseif mod(2*i, 10) == 0
        fprintf('\t%i%% complete...\n', 2*i)
    end
end
warning('on', 'stats:gmdistribution:FailedToConverge')

fprintf('Computing posterior probabilities...\n')

% Posterior probability array for all fibril pixels
post_fibril = posterior(GMM, [x_fibril, y_fibril]);
% Convert to sparse matrix to save memory
post_fibril = sparse(post_fibril.*(post_fibril > 0.01));

% Posterior probability array for all image pixels
```

```
post_all = posterior(GMM, [x(:), y(:)]);
% Convert to sparse matrix to save memory
post_all = sparse(post_all.*(post_all > 0.01));

fprintf('Clustering fibril pixels...\n')

% Cluster all fibril pixels
% cluster_idx = cluster(GMM, [x_fibril, y_fibril]);
[~, cluster_idx] = max(post_fibril, [], 2); % Faster than cluster()

% Contours of posterior probability == 0.5 for non-boundary fibrils
contour_matrix = [];
for i=1:length(x_centroid)
    if ~is_bdy_centroid(i)
        % Get contour matrix for this fibril
        tmp = contourc(x(1,:), y(:,1), ...
            reshape(full(post_all(:,i)), size(I)), [0.5, 0.5]);
        % Convert contour matrix to cell arrays of contour coordinates
        [x_contour, y_contour] = C2xyz(tmp);
        % Sometimes multiple contour curves will be produced for one
        % fibril. Extract the correct contour curve, which is the contour
        % curve that has the smallest mean distance to centroid i.
        mean_dist_to_centroid = nan(length(x_contour), 1);
        for j=1:length(x_contour) % For each contour
            mean_dist_to_centroid(j) = mean(sqrt( ...
                (x_contour{j} - GMM.mu(i,1)).^2 + ...
                (y_contour{j} - GMM.mu(i,2)).^2));
        end
        [~, correct_contour_idx] = min(mean_dist_to_centroid);
        contour_matrix = [contour_matrix, nan(2,1), ...
            [x_contour{correct_contour_idx}; ...
            y_contour{correct_contour_idx}]];
    end
end
contour_matrix = [contour_matrix, nan(2,1)];

% Show image and binarized overlay and (for non-boundary fibrils) contours
% of the posterior probability == 0.5
fig_GMM_boundaries = figure;
imshow(I_filt)
hold on
h = imshow(cat(3, 1-I_filt_bin_2, 0*I_filt_bin_2, 0*I_filt_bin_2)*255);
h.AlphaData = (1-I_filt_bin_2)/5;
plot(GMM.mu(~is_bdy_centroid, 1), GMM.mu(~is_bdy_centroid, 2), 'r+', ...
    'linewidth', 2)
plot(contour_matrix(1,:), contour_matrix(2,:), 'b');
h = imshow(~M_filt);
h.AlphaData = M_filt;
```

Fit ellipse to each fibril

```
theta = (0:360)';
```

```
n_centroid = length(x_centroid); % Number of centroids (including boundary
 centroids)
MU = nan(n_centroid, 2);
SIGMA = nan(2, 2, n_centroid);
radius_pix = nan(n_centroid, 2);
x_ellipse = nan(length(theta)+1, n_centroid);
y_ellipse = nan(length(theta)+1, n_centroid);
for i = find(~is_bdy_centroid)' % For each non-boundary centroid
    % Mean and covariance of fibril pixel coordinates
    MU(i,:) = mean([x_fibril(cluster_idx==i), y_fibril(cluster_idx==i)]);
    SIGMA(:,:,i) = cov([x_fibril(cluster_idx==i), y_fibril(cluster_idx==i)]);

    % Compute ellipse from the eigenvalues/eignvectors of SIGMA, as in
    % Rego's PhD dissertation (2019), page 164
    [V, D] = eig(SIGMA(:,:,i));
    radius_pix(i,:) = 2*sqrt(diag(D)); % Ellipse radii
    ellipse_i = [radius_pix(i,1)*cosd(theta), ...
        radius_pix(i,2)*sind(theta)]*V' + MU(i,:); % Ellipse point coordinates
    x_ellipse(1:end-1,i) = ellipse_i(:,1);
    y_ellipse(1:end-1,i) = ellipse_i(:,2);
end

% Sort ellipse radii: major radius in 1st column, minor radius in 2nd
% column
radius_pix = sort(radius_pix, 2, 'descend');

% Show image and ellipses
fig_ellipses = figure;
imshow(I_filt)
hold on
h_centroid = plot(MU(:,1), MU(:,2), 'r+', 'linewidth', 2);
h_ellipse = plot(x_ellipse(:), y_ellipse(:), 'b');
```

Interactively remove ellipses where code didn't perform well

```
fprintf('Waiting for user to remove bad ellipses...\n')

button = 1;
while ~isempty(button)
    title(['\rm{Left-click centroids to remove from final results. ', ...
        'Press Enter to continue.}'])
    [x_click, y_click, button] = ginput(1);
    if button == 1 % Left click: delete nearest centroid and ellipse
        nearest_centroid_idx = knnsearch(MU, [x_click, y_click]);
        MU(nearest_centroid_idx, :) = NaN;
        SIGMA(:, :, nearest_centroid_idx) = NaN;
        radius_pix(nearest_centroid_idx, :) = NaN;
        x_ellipse(:, nearest_centroid_idx) = NaN;
        y_ellipse(:, nearest_centroid_idx) = NaN;

        % Re-plot centroids and ellipses
```

```
        delete(h_centroid)
        delete(h_ellipse)
        h_centroid = plot(MU(:,1), MU(:,2), 'r+', 'linewidth', 2);
        h_ellipse = plot(x_ellipse(:), y_ellipse(:), 'b');
    end
end

title('')
```

Post-processing

```
fprintf('Post-processing...\n')

% Scale all pixel-based quantities back to original image scale (i.e.,
% before image downsizing)
contour_matrix = contour_matrix*scale;
radius_pix = radius_pix*scale;
x_centroid = x_centroid*scale;
y_centroid = y_centroid*scale;
GMM = gmdistribution(GMM.mu*scale, GMM.Sigma*scale^2, ...
    GMM.ComponentProportion);
MU = MU*scale;
x_ellipse = x_ellipse*scale;
y_ellipse = y_ellipse*scale;

% Radii in nanometers
radius_nm = radius_pix*nanometers_over_pixels;

% Fibril area (computed as the sum of posterior probabilities among fibril
% pixels, for each fibril)
area_pix2 = full(sum(post_fibril)'*scale^2); % In pixels squared
area_pix2(isnan(MU(:,1))) = NaN; % Set area=NaN for boundary & bad fibrils
area_nm2 = area_pix2*nanometers_over_pixels^2; % In nanometers squared
```

Output

```
fprintf('Saving results...\n')

% Make output directory (with same name as image)
mkdir(filename)

% Save MAT file
save(fullfile(filename, filename), 'contour_matrix', 'radius_pix', ...
    'x_centroid', 'y_centroid', 'GMM', 'MU', 'x_ellipse', 'y_ellipse', ...
    'radius_nm', 'area_pix2', 'area_nm2')
```

Save figures

```
fprintf('Saving figures...\n')

% Original image
figure(fig_image_original)
```

```
savefig(fullfile(filename, '1 - Original image'))
print(fullfile(filename, '1 - Original image'), '-dpng', '-r300')

% Filtered image
figure(fig_image_filtered)
savefig(fullfile(filename, '2 - Filtered image'))
print(fullfile(filename, '2 - Filtered image'), '-dpng', '-r300')

% ("Old"/Original) filtered & binarized image
figure(fig_image_filtbin)
savefig(fullfile(filename, '3 - Original binarized image'))
print(fullfile(filename, '3 - Original binarized image'), '-dpng', '-r300')

% Overlay of original and binarized images
figure(fig_image_overlay)
savefig(fullfile(filename, '4 - Overlay, original and binarized images'))
print(fullfile(filename, '4 - Overlay, original and binarized images'), '-
dpng', '-r300')

% Overlay of original and binarized images with Voronoi diagram
figure(fig_voronoi)
savefig(fullfile(filename, '5 - Voronoi'))
print(fullfile(filename, '5 - Voronoi'), '-dpng', '-r300')

% Image pair, old and new binarized images
figure(fig_filtbin_pair)
savefig(fullfile(filename, '6 - Image pair, old and new binarized images'))
print(fullfile(filename, '6 - Image pair, old and new binarized images'), '-
dpng', '-r300')

% Fibril boundaries based on Gaussian mixture model
figure(fig_GMM_boundaries)
savefig(fullfile(filename, '7 - Fibril boundaries from GMM'))
print(fullfile(filename, '7 - Fibril boundaries from GMM'), '-dpng', '-r300')

% Best-fit ellipses for each non-boundary fibril
figure(fig_ellipses)
savefig(fullfile(filename, '8 - Ellipses'))
print(fullfile(filename, '8 - Ellipses'), '-dpng', '-r300')
```

Published with MATLAB® R2023a