Automated cell migration analysis is accomplished using MATLAB. Raw images are used to track moving cells and create videos of their migration. The speed, mean squared displacement (MSD), and directionality ratio (DR) of tracked cells are compared within each condition, as well as among the conditions. To accomplish this, three codes are necessary. All codes are optimized to work with different data sets using minimal editing. Each program is described in this manual, including what the code does, how it works, how to edit it, and what it outputs.

To begin, all the necessary codes need to be placed in the same folder. It is recommended that this folder is separate from raw cell images. Outputs from the cell tracking program (BATCH_tracking_2) will be saved in the same folder as the code. The final two codes allow the user save outputs in a chosen destination. It is again suggested that all outputs are kept separate from raw images. Raw phase images and SiR DNA images can be kept in the same folder or separate folders.

The terms “stage” and “frame” are used often in the manual. In this context, a “stage” is a specific location where cells were imaged. If the user took a sequence of images in three different locations total, each location would be a different stage number: the first is stage 1, the second is stage 2, and the third is stage 3. Stage numbers should increase sequentially for each new location that is imaged. A “frame” is the image number within each stage. If each stage contained a sequence of 100 images, the 49th image would be frame 49.

These codes require little knowledge of MATLAB, though the user will need to know some basics such as writing file paths and running programs. For additional help, please see the “additional information” section at the end of the manual.

Cell Tracking and Video Creation

The first code, BATCH_tracking_2, creates videos using raw images of cells. It works one stage at a time, loading in all of the images for the current stage. From these images it finds the best black and white and display thresholds. The cells are then tracked and filtered, excluding any cells that migrated for fewer than 12 frames or cells that were considered non-migratory (for example, dead cells that were detected). After this, the movies are created using a built-in MATLAB function. Movies will include both the SiR DNA and phase image sequences side-by-side, allowing for easy comparison of the two videos. The frame rate is 7 frames/second. It also outputs other cell information that can be used in further analysis.

A list of parameters that require editing can be seen below.

- "pixpermic" is the pixel to micron calibration. This will need to be edited depending on the microscope magnification used. For 10x, the `pixpermic = 1/1.13`. If 20x was used, the denominator would be multiplied by 2 (`pixpermic = 1/(2*1.13)`).

- "time_step" is the duration of time between consecutive frames, in minutes. If images were taken every 30 minutes, time_step = 30.

- "main_dir" is the file path of the folder where the SiR DNA images are kept. It is important that this ends with a slash (either / or \). Ex. `L:\Image Data\Aslan\Summer 2018\Exosome Migration\Second Wound Healing\'`.

- "phase_dir" is similar to main_dir, except this will be the file path to the folder where the phase images are kept. In many cases, these two variables will be the same.

- "frame_range" is the range of frames that should be included in the video. For `frame_range = [1 48]`, the video would include all the images from frames 1 to 48. This allows data to be excluded if necessary.

- "bw_thresh_correction" adds a correction to the black and white threshold that is automatically calculated by the code. If a significant amount of cells are not being detected, a correction could fix the issue. If this is the case, the correction should be set to a decimal between 0 and 1. Usually a correction will not be needed, so bw_thresh_correction will equal 1. It is suggested that the user starts with this value and adjusts if needed.

- "area_range" allows the user to set the minimum and maximum number of pixels that the code will detect as a cell. The minimum and maximum are written within brackets and separated by a space. If area_range = [50 1000], only objects between 50 and 1000 pixels will be considered a cell. This can be useful if the video is picking up junk, as the user can either increase the minimum area or decrease the maximum area. If too few cells are detected, the minimum can be decreased and/or the maximum increased. Generally, only the minimum area will be changed, with the maximum kept constant. A good starting point is area_range = [50 1000]. The user can run the code and adjust if needed.

- "stage_range" is the range of stages that MATLAB will create videos for. Inputs are written within brackets and separated by a colon. To create movies for stages 1-15, `stage_range = [1:15]`. If the desired stages are not consecutive, stage_range can also be written as a list. For the analysis of stages 2, 17, and 24, `stage_range = [2 17 24]`. If the user would only like to create a movie for one stage, only that stage number is written within the brackets (`stage_range = [15]`).

- "Mg_disp_thresh" is the minimum length, in microns, that a cell needs to migrate in order to be considered a cell. This helps filter out immobile cells, including dead cells. A good starting point is `mg_disp_thresh = 50`. If many non-migratory cells are still being tracked, mg_disp_thresh should be increased. When several moving cells are not being tracked, it could be useful to decrease this value.

Sometimes raw images will have different naming patterns. This usually happens when imaging cells over the course of multiple days. As an example, the images might follow the naming pattern "w1SiRDNA_s1_t1" for the first 90 images and then change to "w1SiRDNA_**2**_s1_t1" for the last 90 images of a stage. In this case, a "2_" is inserted after "SiRDNA." To account for this, a few things will need to be changed between lines 70 and 108 in the code. This part of the code is defining the changes that are occurring to the name of the files, how many images these changes occur for, and where this change occurs in the file name. The instructions for editing this portion of the code are listed below. If this is not an issue, skip these instructions.

1. A "for-loop" is a section of code that tells MATLAB to repeat a sequence a specified number of times. This looks like the following.

   ```
   for i = 1:5
    x = x + i
   end
   ```

   Between lines 70 and 108 are two sections of for-loops: one that is labelled "first for-loop section" and the other labelled "second for-loop section." Each section has four for-loops that look similar to the example given above. Most of these will show up in green text, meaning that they are "commented out." For-loops from each section need to be uncommented depending on the number of name changes the images experience. In the example previously described, the user would need to uncomment the first two for-loops of each section because there are two different image names. If there were three different image file names, three for-loops from each section would be uncommented, and so on. To uncomment a section of code, highlight the desired region and press CTRL-T. Press CTRL-R to add a comment, if needed.

2. The user needs to specify how many images occur for each file name. This is changed in line 70 through the variable “num_frames_day.” Since, in the current example, the image names change every 90 images, the line would read `num_frames_day = 90`.

3. The user needs to specify what is changing within the file names. This is only done in the first for-loop section by editing day{i}. The first for-loop will always have day{i} set equal to an empty set of apostrophes (`day{i} = ''`) because the first image type is used as the baseline of comparison for the other name changes. Inside the second for-loop would be `day{i} = '2_'`, as this is what differs from the original image file name. This same pattern is followed for as many day{i} values as needed.

4. “Day{i}” needs to be inserted into the correct part of the file path name in lines 104 and 108, as these are the lines that read all of the images into the code. This will tell MATLAB where the name change is occurring. It will then know to switch the path name of the images after the number of frames that you specified in step 2. For line 104 of our example, assuming the original line of code is

```
im_filename{i} =  [main_dir, 'w1SiRDNA_s', int2str(si), '_t', fr_i{i},
                           '.TIF'],
```

   the new line of code would be

```
im_filename{i} =  [main_dir,'w1SiRDNA_ ,day{i}, s', int2str(si), '_t',
                        fr_i{i},'.TIF']
```

   because the file name change occurs after “w1SiRDNA.”

After editing, the code can be run. The user is able to watch the video as it is being created. It is important not to move the MATLAB figure as it is tracking, or else the code will result in an error and tracking will need to be restarted.

This code will output the following.

- The videos, titled “[tracking] stage (it will insert stage here)” (fig. 1).

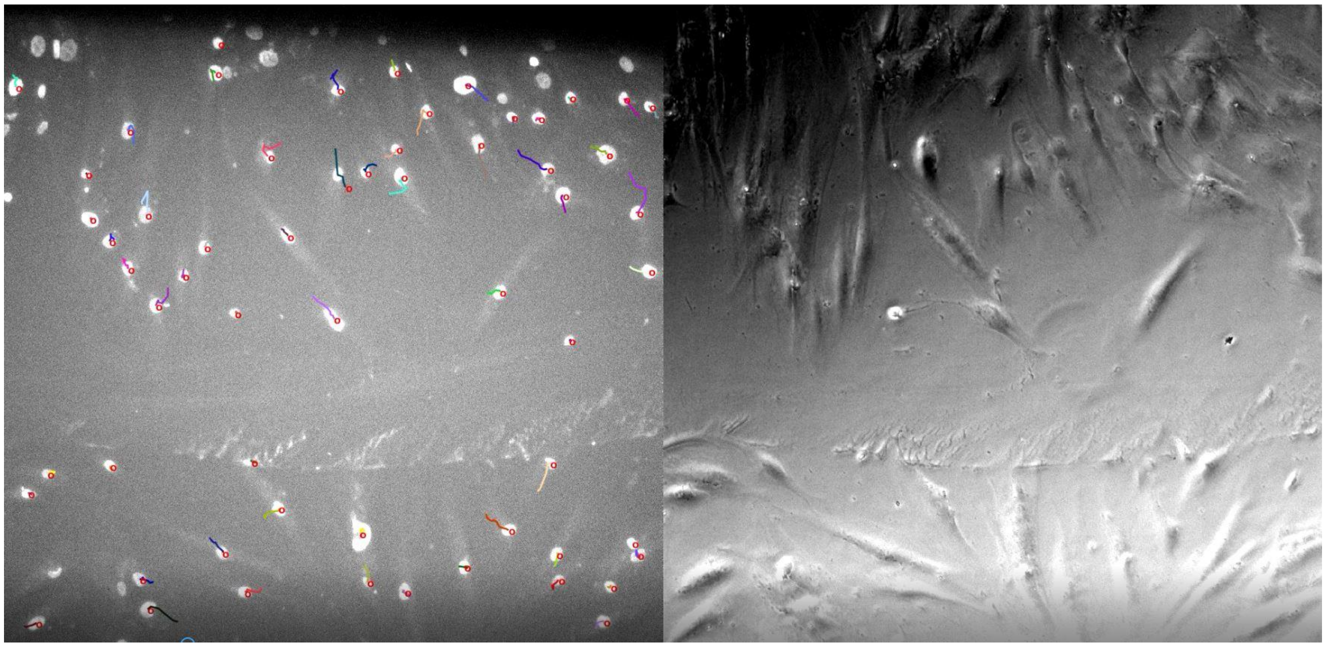


Figure 1. A screenshot of the side-by-side SiR DNA and phase image movies. The left side contains SiR DNA images and the right contains phase images.

- A screenshot of the video with the ID number associated with each individual cell. This is saved as "[cell ID] analyzed cell stage (insert stage)" (fig. 2).

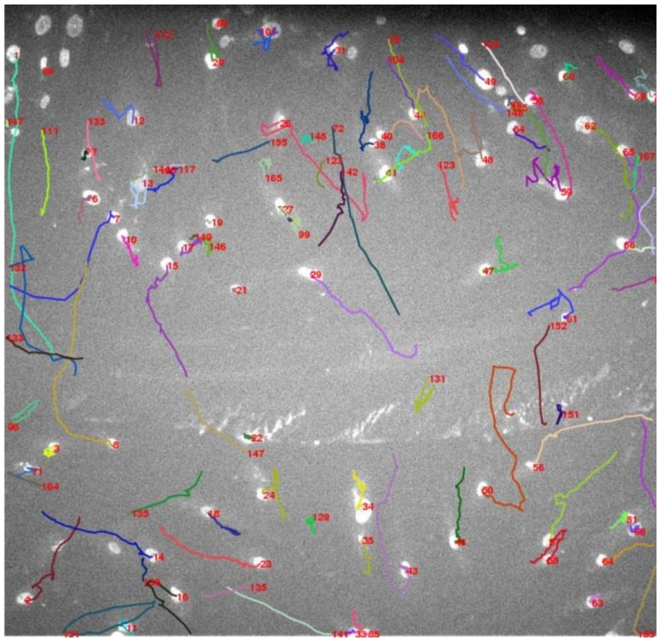


Figure 2. The cell ID image output. Each cell's trajectory is shown by the multi-colored paths. The red numbers indicate the ID of the nearby cell.

- A plot of the cell count over time, in frames. This is saved as “[cell count] stage (insert stage)” (fig. 3).

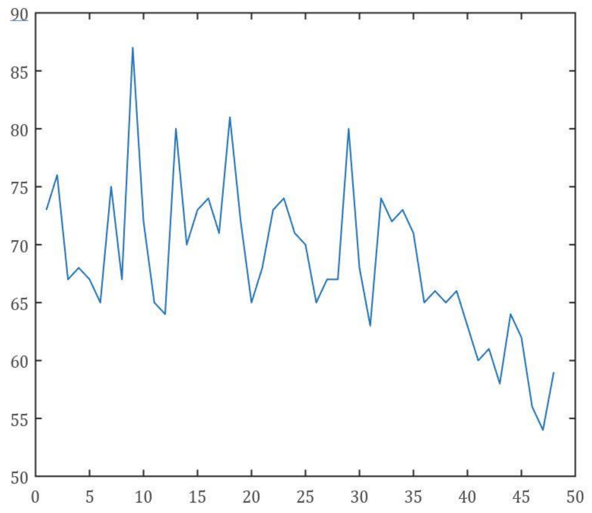
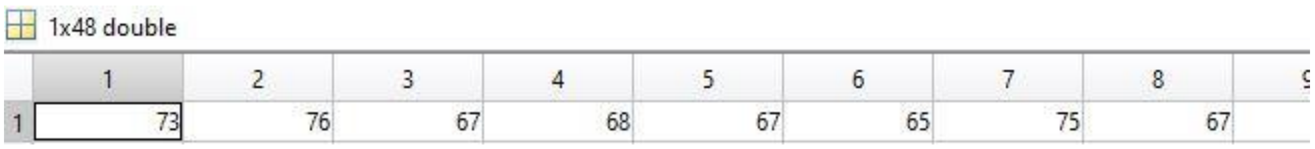


Figure 3. The cell count plot. The y-axis represents the number of cells and the x-axis represents the time, in frames.

- A matrix of the number of cells for each frame. Saving matrices such as this one can be useful for loading back into MATLAB in later codes. This is saved as “n stage (insert stage)” (fig. 4).

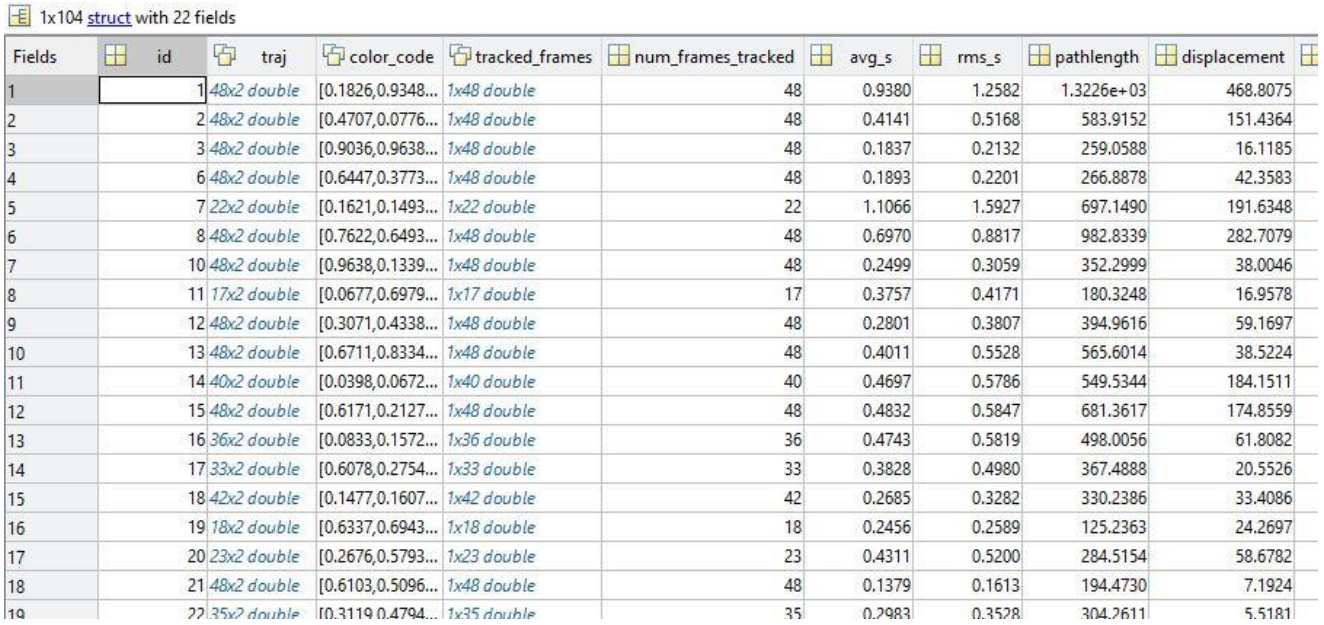


1x48 double

	1	2	3	4	5	6	7	8	9
1	73	76	67	68	67	65	75	67	

Figure 4. A section of the "n stage" vector. The number of columns represent the number of frames that were tracked. Each value inside the vector is the number of cells that were detected for that specific frame.

- A matrix containing different information about each individual cell, including ID number, trajectory, etc. Saved as “analyzed cell stage (insert stage)” (fig. 5).

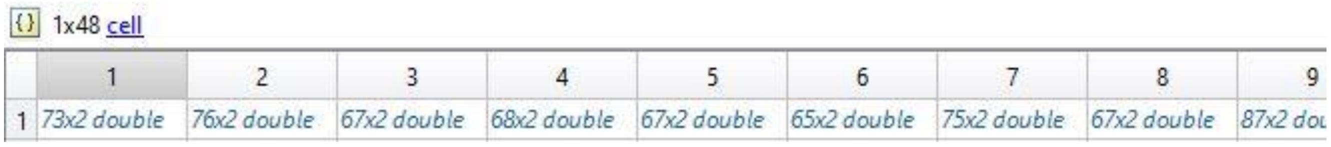


1x104 struct with 22 fields

Fields	id	traj	color_code	tracked_frames	num_frames_tracked	avg_s	rms_s	pathlength	displacement
1	1	48x2 double	[0.1826,0.9348...	1x48 double	48	0.9380	1.2582	1.3226e+03	468.8075
2	2	48x2 double	[0.4707,0.0776...	1x48 double	48	0.4141	0.5168	583.9152	151.4364
3	3	48x2 double	[0.9036,0.9638...	1x48 double	48	0.1837	0.2132	259.0588	16.1185
4	6	48x2 double	[0.6447,0.3773...	1x48 double	48	0.1893	0.2201	266.8878	42.3583
5	7	22x2 double	[0.1621,0.1493...	1x22 double	22	1.1066	1.5927	697.1490	191.6348
6	8	48x2 double	[0.7622,0.6493...	1x48 double	48	0.6970	0.8817	982.8339	282.7079
7	10	48x2 double	[0.9638,0.1339...	1x48 double	48	0.2499	0.3059	352.2999	38.0046
8	11	17x2 double	[0.0677,0.6979...	1x17 double	17	0.3757	0.4171	180.3248	16.9578
9	12	48x2 double	[0.3071,0.4338...	1x48 double	48	0.2801	0.3807	394.9616	59.1697
10	13	48x2 double	[0.6711,0.8334...	1x48 double	48	0.4011	0.5528	565.6014	38.5224
11	14	40x2 double	[0.0398,0.0672...	1x40 double	40	0.4697	0.5786	549.5344	184.1511
12	15	48x2 double	[0.6171,0.2127...	1x48 double	48	0.4832	0.5847	681.3617	174.8559
13	16	36x2 double	[0.0833,0.1572...	1x36 double	36	0.4743	0.5819	498.0056	61.8082
14	17	33x2 double	[0.6078,0.2754...	1x33 double	33	0.3828	0.4980	367.4888	20.5526
15	18	42x2 double	[0.1477,0.1607...	1x42 double	42	0.2685	0.3282	330.2386	33.4086
16	19	18x2 double	[0.6337,0.6943...	1x18 double	18	0.2456	0.2589	125.2363	24.2697
17	20	23x2 double	[0.2676,0.5793...	1x23 double	23	0.4311	0.5200	284.5154	58.6782
18	21	48x2 double	[0.6103,0.5096...	1x48 double	48	0.1379	0.1613	194.4730	7.1924
19	22	35x2 double	[0.3119,0.4794...	1x35 double	35	0.2983	0.3528	304.2611	5.5181

Figure 5. A section of the "analyzed_ cell stage" matrix. There are 22 columns, each representing a different variable that the code has measured (ID number, average speed, etc). Each row represents a different cell that the code has detected.

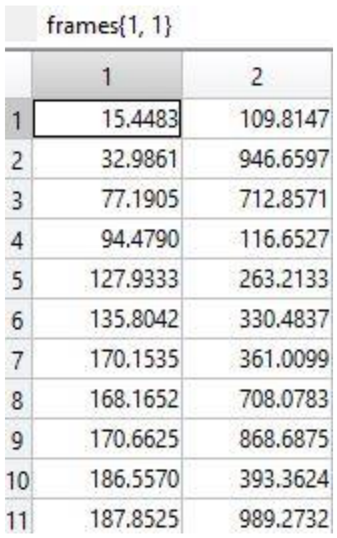
- A matrix of cell position at each time. Saved as “frames stage (insert stage)” (fig. 6-7).



1x48 cell

	1	2	3	4	5	6	7	8	9
1	73x2 double	76x2 double	67x2 double	68x2 double	67x2 double	65x2 double	75x2 double	67x2 double	87x2 dou

Figure 6. The "frames stage" matrix. Each column represents each frame that was tracked. Each cell of the vector contains another matrix.



frames{1, 1}

	1	2
1	15.4483	109.8147
2	32.9861	946.6597
3	77.1905	712.8571
4	94.4790	116.6527
5	127.9333	263.2133
6	135.8042	330.4837
7	170.1535	361.0099
8	168.1652	708.0783
9	170.6625	868.6875
10	186.5570	393.3624
11	187.8525	989.2732

Figure 7. The matrix inside of the first cell of fig. 7. The first and second columns correspond to the x- and y-coordinates of a cell's position, respectively. Each row is a different cell that has been detected.

All movies should be closely analyzed by the user prior to running the remaining programs. If the movies detect too many/too few cells, the rest of the analysis will be inaccurate. This is because the data used in the rest of the code is extracted from the information from the tracked cells in this code. If something is incorrect, the inputs should be edited again. Typically, issues are fixed by adding a correction to the black and white threshold or by changing the minimum cell area.

Intragroup and Intergroup Analysis

Two additional MATLAB codes are used to determine the speed, MSD, and DR of each stage. The first code, a program used to compare the stages within each condition, is referred to as the intragroup analysis code. In this program, the user sorts each stage into its respective condition and specifies which time ranges the data should be analyzed at. A function performs all of the necessary calculations and plots all of the data in one figure for easy comparison. All calculations are saved in a variable that is called to in the third program.

After opening the code, the following variables will need to be edited.

- “num_frames” is the total number of frames that were imaged for each frame overall. If you took 100 images at each stage but only wanted to analyze the first 50, the value of num_frames would still be 100.

- “condition_list” is a list of condition names of the data set, saved as strings. To do this, all of the conditions need to be listed within curly brackets, separated by commas. Each condition also needs to be inside of apostrophes. A condition_list could look like this, for example: `condition_list = {'CM','RPMI','RPMI_DMEM','Co culture (mem)','Co culture (no mem)'}`. These should be in the same order as the conditions of the time master.

- “main_dir_analyzed_cell” is the path name of the folder containing “analyzed cell stage (insert stage)” from BATCH_tracking_2. Make sure to include a slash at the end of the pathname.

- “cell_stage_filename” is the name of the “analyzed cell stage “ data. This will only need to be changed if the name of the output in BATCH_tracking_2 was altered.

- “dest_dir” is the path name to the folder that you would like all outputs to save to after the code has been run. Make sure to include a slash at the end of the pathname.

- “timelapse” is the duration of time between consecutive frames divided by 60. For example, if 30 minutes passed between each image, time_lapse would be 30/60, which is 0.5.

- “num_bootstrapping” is used for statistics. It should not need to be edited.

- “dest_file” is the name that will be given to the output file containing the speed, MSD, and DR plots. Originally, the file will be named “[intra stats] stage (insert stage).” This can be changed if a different file name is desired.

- "file_name" is the file path to the analyzed cell stage file. This will most likely not need to be altered if the previous edits were done correctly.

- The final line, which reads "`save('needed_files.mat', 'file_name', 'condition_list','dest_dir', 'num_bootstrapping')`," should not be edited.

The user will also need to organize each stage into its respective condition. This is accomplished using the stage master. There is one stage master for each condition. The first condition is `stage_master{1}`, the second is `stage_master{2}`, and so on. These are set equal to a set of brackets containing all of the wound regions for that stage, separated by a space. For example, our third condition consists of wound regions at stages 13, 16, 49, and 52. Therefore, the stage master for this condition is `stage_master{3} = [13 16 49 52]`. These will need to be created for each condition. A potential stage master list could look like the following.

```
stage_master{1} = [1 4 5 7];
stage_master{2} = [8 10 11];
stage_master{3} = [13 16 49 52];
```

Finally, the user will need to create a time master, a list similar to the stage master. This variable will specify what time ranges the intragroup/intergroup codes will analyze the data at. If the user would like to analyze the data between 0-24 frames and 24-48 frames, the time master would be edited as seen below (NOTE: since "frame 0" does not exist, values within the time master can never be zero. For time ranges starting at time zero, use the number 1 instead, as this is the earliest frame number that will exist).

```
time_master{1} = [1 24];
time_master{2} = [24 48];
```

Sometimes the user will know the time ranges they would like to analyze in hours, but not in frames. In this case, simple math will need to be carried out to determine how many frames occurred for this time range. For example, if a user imaged every 15 minutes and wants to analyze their data for 0 – 6 and 6 – 12 hours, the following calculation would be needed to convert hours to frames.

60 minutes per hour / 15 minutes per image = 4 images per hour

6 hours * 4 images / hour = 24 images = 24 frames
12 hours * 4 images / hour = 48 images = 48 frames

```
time_master{1} = [1 24]
time_master{2} = [24 48]
```

Since time_master is a range, there can only be two numbers within the brackets of each time master (as opposed to a stage master, which can have as many numbers as the user would like). While the speed, MSD, and DR are calculated and saved for all time points, only the plots created for the last time master will be saved. This prevents the user from obtaining excess plots to sort through. For this reason, it is recommended that the final time master includes the entire range of data. This way, the plots that are saved into the dest_dir folder include stats for the whole range of data rather than a small range within the dataset. In the example above, the overall range of time would be from frames 1 to 48, and therefore `time_master{end} = [1 48]`. The overall time master list would then look like the following.

```
time_master{1} = [1 24]
time_master{2} = [24 48]
time_master{3} = [1 48]
```

Run the code. Each condition will output one figure containing six different plots, saved as “[intra stats] (insert condition)” (fig. 8). The top three bar plots display the average values of speed, MSD, and DR at each stage. Beneath these are line plots representing the instantaneous values over time, in hours, for each stage. This figure is useful for detecting potential issues with a dataset. Visually, users can quickly detect issues with specific stages. Such stages can potentially be excluded from the dataset, preventing data from being skewed.

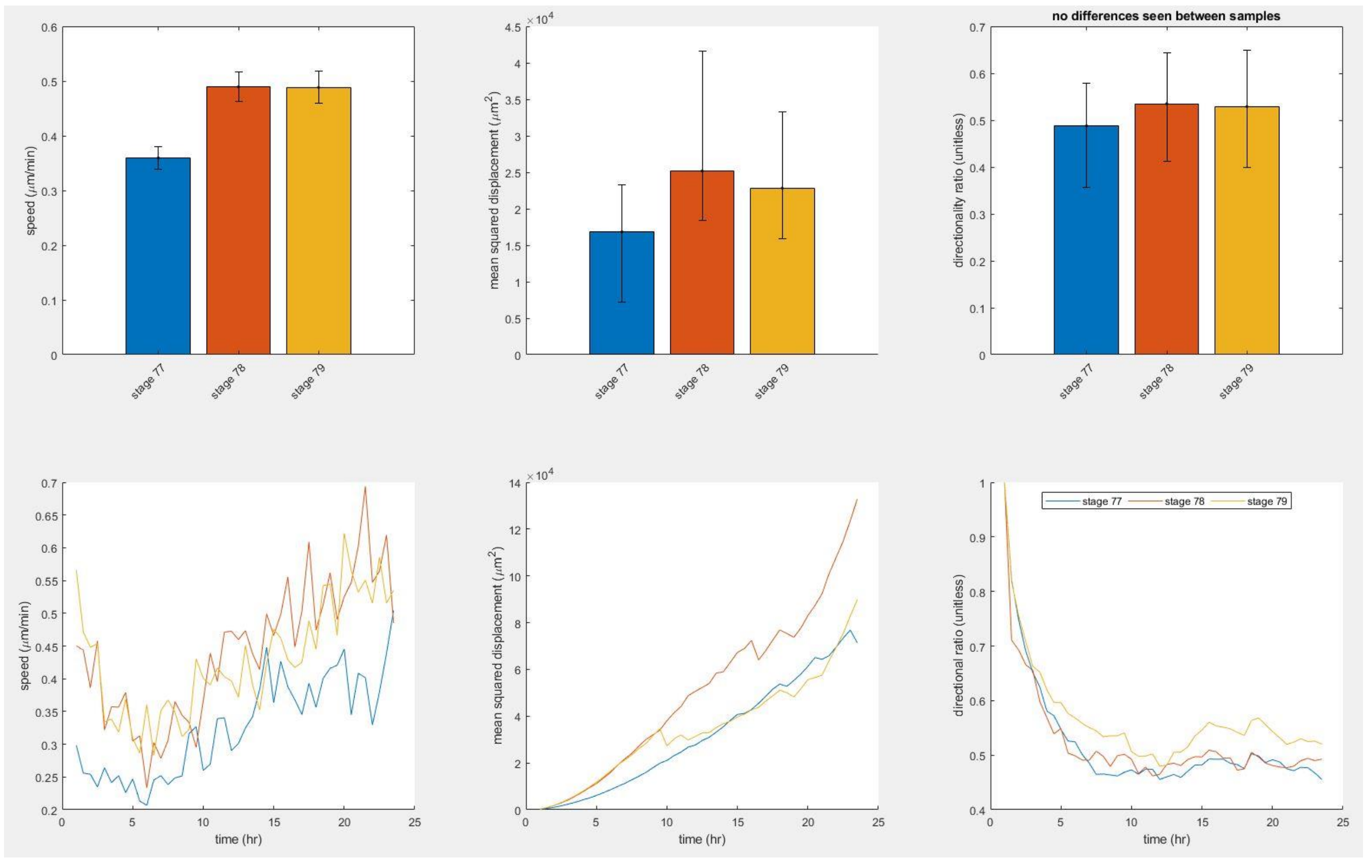
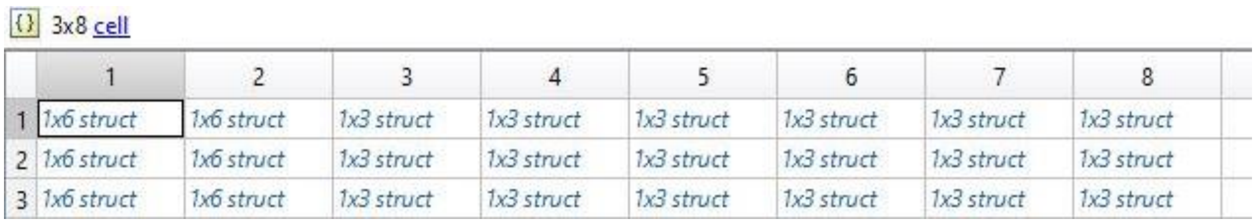


Figure 8. Plots of speed, MSD, and DR as output by the intragroup analysis code. The top three graphs display the average speed for each stage of the specified condition. The bottom three graphs display the instantaneous speed over time (in hours) for each stage.

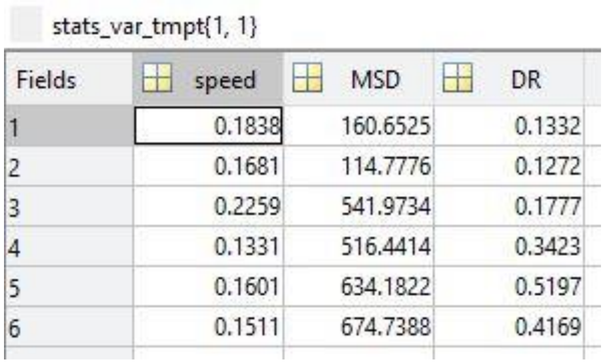
Though only one time master is plotted, the speed, MSD, and DR are calculated for all stages at each specified time range. This data is saved in a matrix called "stats_var_tmpt" (fig. 9-10).



3x8 cell

	1	2	3	4	5	6	7	8
1	1x6 struct	1x6 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct
2	1x6 struct	1x6 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct
3	1x6 struct	1x6 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct	1x3 struct

Figure 9. The "stats_var_tmpt" matrix. Each column represents a different condition, as specified in the stage master list. The rows each represent a different time range as specified in the time master. Within each cell of the matrix is another matrix.



stats_var_tmpt{1, 1}

Fields	speed	MSD	DR
1	0.1838	160.6525	0.1332
2	0.1681	114.7776	0.1272
3	0.2259	541.9734	0.1777
4	0.1331	516.4414	0.3423
5	0.1601	634.1822	0.5197
6	0.1511	674.7388	0.4169

Figure 10. The matrix inside of the first cell of "stats_var_tmpt," shown in fig. 7. This cell contains data for the first time range of the first condition, as it was in the first row and column of the broader matrix. Each value represents the speed, MSD, and DR at each stage of the condition.

The final code, referred to as intergroup analysis, uses the data from "stats_var_tmpt" to plot the average speed, MSD, and DR of each condition in single plot. This allows for comparison between the conditions. While the intragroup code only plotted a single time range, the intergroup code plots the average values for *all* time master ranges.

This program requires minimal editing, as most of the data is loaded from the intragroup code. Only "timelapse" and "condition_list" will need to be edited. Descriptions of these variables can be found in the intragroup code. The stage master and time master are also needed. These lists can be copied from the previous code, as well.

Once the program is run, a single figure with three plots will appear (fig. 11). Each plot represents speed, MSD, and DR, respectively. The conditions are specified along the x-axis. Each condition will have one bar for each time master. There are no other outputs for the intergroup analysis code.

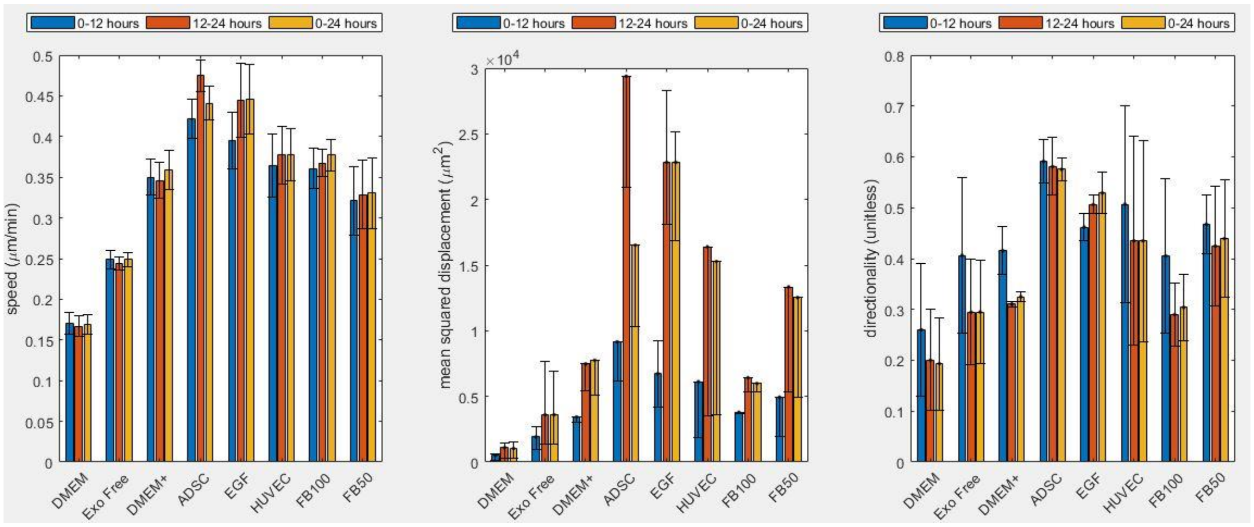


Figure 11. The figure output of the intergroup analysis code. There are three plots in the figure: speed, MSD, and DR. The average value of each condition is shown at each time point that was analyzed

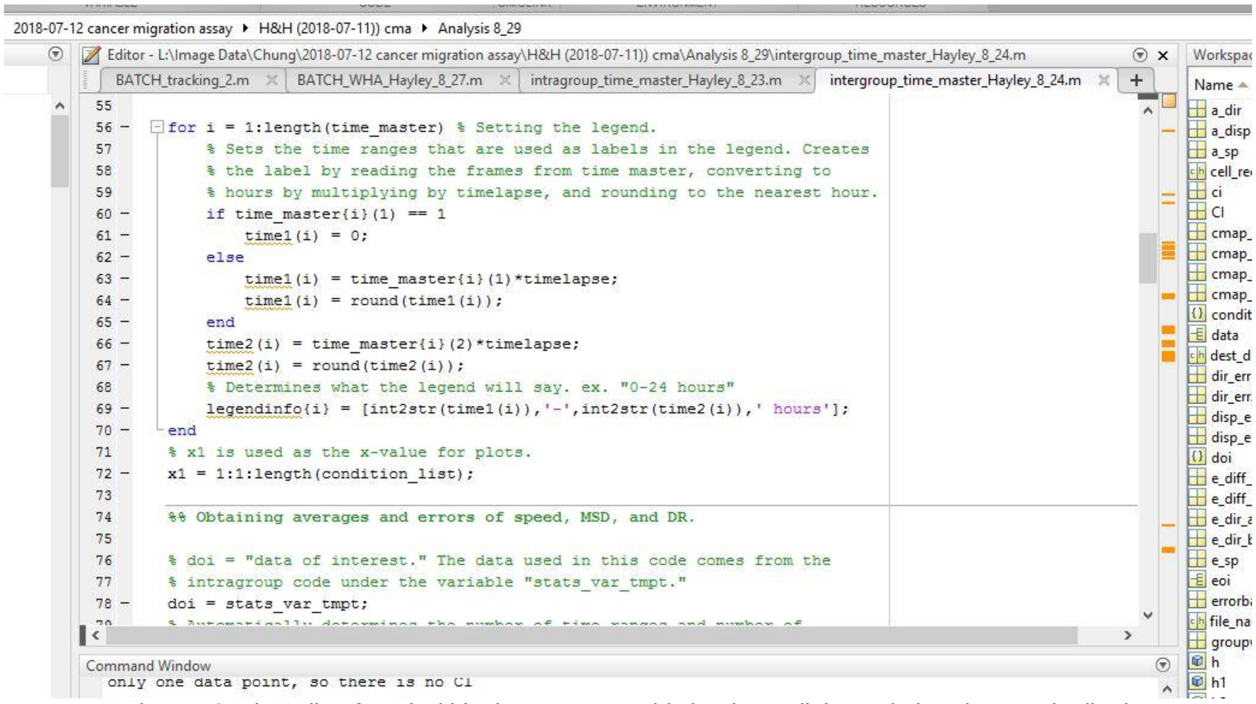
Outputs from all codes should be analyzed by the user. All results can be compared to what is visually perceived in the cell tracking videos. Any mistakes that are discerned may be due to inaccurate inputs, indicating that users should double-check any edits previously made. Inaccurate results could also be due to a specific quality held by of the raw data. For example, when cells frequently collide during cell tracking, the MSD tends to be under-reported. Since this is a flaw with the data itself, this type of issue is unavoidable.

Currently, the MATLAB codes in this manual are used for analysis of random migration. One flaw, however, is the lack of analysis for stages that may contain experimental wounds. MATLAB programs could be used to specify the wound region of an image, calculating the number of cells that migrate into the wound (known as “healer cells” or “leader cells”) and wound recovery. Such features would give users the ability to run wound healing experiments. The outputs would suggest the effect of different conditions on the cells’ ability to heal wounds.

Additional Information

Editing MATLAB Code

MATLAB code is edited in the editor (fig. 12). This is usually the box located in the middle (to ensure this, click on the “home” tab at the top of the screen, select “layout,” and click “default”). Along the left edge of the editor is a list of numbers, referring to the line number in the code. The top edge of the editor has tabs referring to which code the user is currently editing. Different tabs access different codes.


2018-07-12 cancer migration assay ▸ H&H (2018-07-11)) cma ▸ Analysis 8_29
Editor - L:\Image Data\Chung\2018-07-12 cancer migration assay\H&H (2018-07-11)) cma\Analysis 8_29\intergroup_time_master_Hayley_8_24.m
BATCH_tracking_2.m
BATCH_WHA_Hayley_8_27.m
intragroup_time_master_Hayley_8_23.m
intergroup_time_master_Hayley_8_24.m
Workspac
Name
a_dir
a_disp
a_sp
cell_re
ci
CI
cmap_
cmap_
cmap_
cmap_
condit
data
dest_d
dir_err
dir_err
disp_e
disp_e
doi
e_diff_
e_diff_
e_dir_a
e_dir_b
e_sp
eoi
errorb
file_na
group
h
h1
Command Window

```
for i = 1:length(time_master) % Setting the legend.
    % Sets the time ranges that are used as labels in the legend. Creates
    % the label by reading the frames from time master, converting to
    % hours by multiplying by timelapse, and rounding to the nearest hour.
    if time_master{i}(1) == 1
        time1(i) = 0;
    else
        time1(i) = time_master{i}(1)*timelapse;
        time1(i) = round(time1(i));
    end
    time2(i) = time_master{i}(2)*timelapse;
    time2(i) = round(time2(i));
    % Determines what the legend will say. ex. "0-24 hours"
    legendinfo{i} = [int2str(time1(i)),'-',int2str(time2(i)),' hours'];
end
% x1 is used as the x-value for plots.
x1 = 1:1:length(condition_list);

%% Obtaining averages and errors of speed, MSD, and DR.

% doi = "data of interest." The data used in this code comes from the
% intragroup code under the variable "stats_var_tmpt."
doi = stats_var_tmpt;
```

```
only one data point, so there is no CI
```

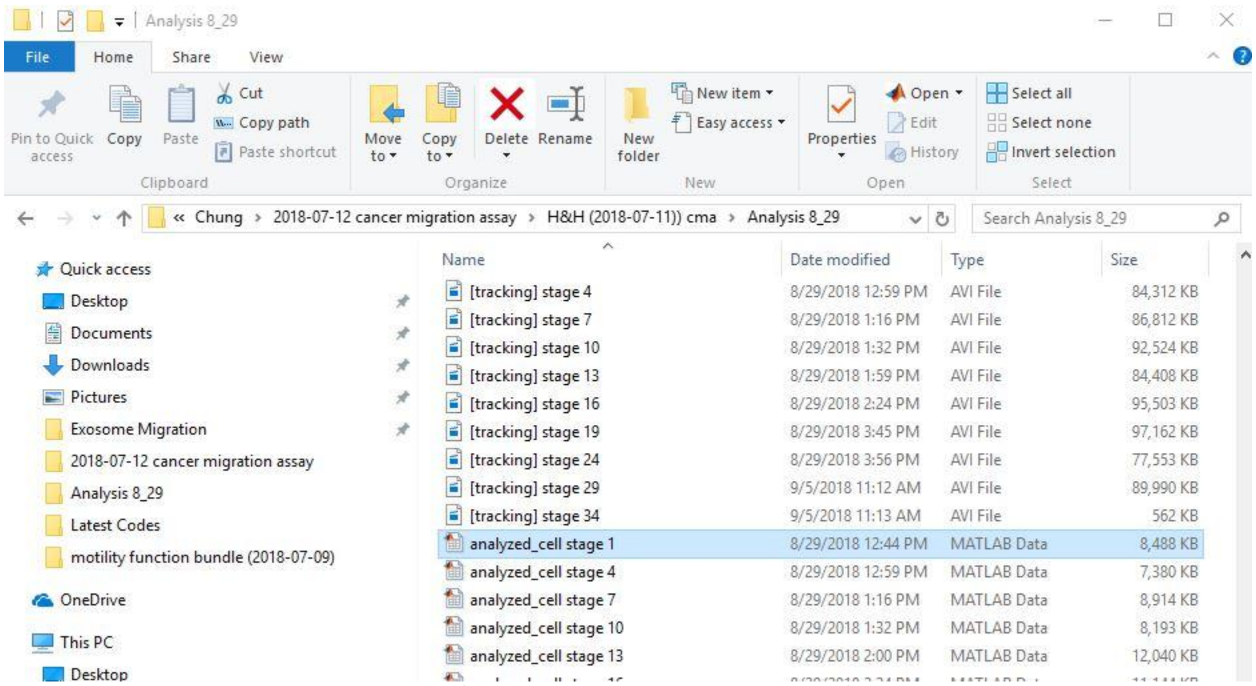
Figure 12. The editor found within the MATLAB. This is where all the code is written and edited.

Anything written in green is a comment and will not affect the code – these are simply notes that are meant to aid the user in understanding how the program works. Everything in black and blue is the actual code.

Writing File Paths

A file path is used to specify the location of a specific file. It can be written manually or copied and pasted from the computer’s file app.

Of the two options, copying and pasting the file path name is easiest. To do this, open the file app and find the file MATLAB needs to open (fig. 13). In the case below, MATLAB needs to open the file “analyzed_cell_stage 1.”



Name	Date modified	Type	Size
[tracking] stage 4	8/29/2018 12:59 PM	AVI File	84,312 KB
[tracking] stage 7	8/29/2018 1:16 PM	AVI File	86,812 KB
[tracking] stage 10	8/29/2018 1:32 PM	AVI File	92,524 KB
[tracking] stage 13	8/29/2018 1:59 PM	AVI File	84,408 KB
[tracking] stage 16	8/29/2018 2:24 PM	AVI File	95,503 KB
[tracking] stage 19	8/29/2018 3:45 PM	AVI File	97,162 KB
[tracking] stage 24	8/29/2018 3:56 PM	AVI File	77,553 KB
[tracking] stage 29	9/5/2018 11:12 AM	AVI File	89,990 KB
[tracking] stage 34	9/5/2018 11:13 AM	AVI File	562 KB
analyzed_cell stage 1	8/29/2018 12:44 PM	MATLAB Data	8,488 KB
analyzed_cell stage 4	8/29/2018 12:59 PM	MATLAB Data	7,380 KB
analyzed_cell stage 7	8/29/2018 1:16 PM	MATLAB Data	8,914 KB
analyzed_cell stage 10	8/29/2018 1:32 PM	MATLAB Data	8,193 KB
analyzed_cell stage 13	8/29/2018 2:00 PM	MATLAB Data	12,040 KB

Figure 13. The computer's file app containing the desired input file, highlighted in blue.

Between the file list and home tab is a rectangular box containing the location of the folder where the user currently is. The name of this folder is "Analysis 8_29," the last folder name in the horizontal list. To find the path to this folder, right click on the words "Analysis 8_29" and select "copy address as text." This can be pasted into MATLAB. Again, this will only direct MATLAB to the folder, not the specific file.

```
L:\Image Data\Chung\2018-07-12 cancer migration assay\H&H (2018-07-11))
cma\Analysis 8_29
```

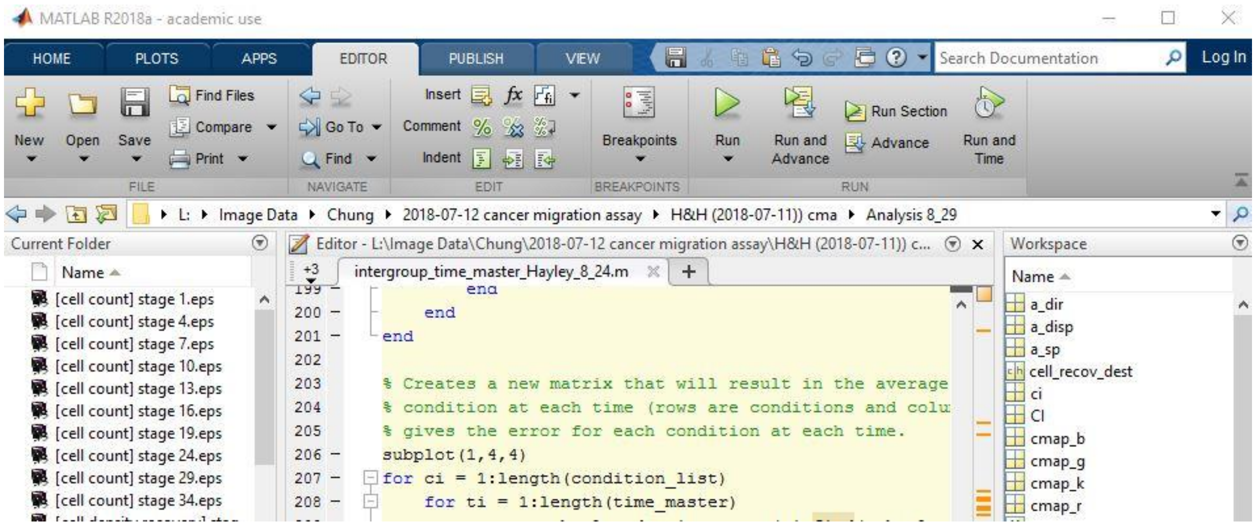
In order to find the path name to the file itself, the name of the file is added to the end of the folder's path (as previously described). Separating these should be a slash.

```
L:\Image Data\Chung\2018-07-12 cancer migration assay\H&H (2018-07-11))
cma\Analysis 8_29\analyzed_cell_stage 1
```

To write the file path manually, the user must know the sequence of folders that the file is located in. Each folder has to be written in order, separated by a slash. Again, the file name is written last.

Running MATLAB Code

To run a MATLAB code, the user must enter the "editor" tab at the top of the screen and click on the green arrow (fig. 14). If the code stops prematurely and red text is seen in the command window (below the code), there has been an error. The code needs to be edited in order to fix the error and allow the program to run correctly.


MATLAB R2018a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
L: ▸ Image Data ▸ Chung ▸ 2018-07-12 cancer migration assay ▸ H&H (2018-07-11)) cma ▸ Analysis 8_29
Current Folder
Name
[cell count] stage 1.eps
[cell count] stage 4.eps
[cell count] stage 7.eps
[cell count] stage 10.eps
[cell count] stage 13.eps
[cell count] stage 16.eps
[cell count] stage 19.eps
[cell count] stage 24.eps
[cell count] stage 29.eps
[cell count] stage 34.eps
Editor - L:\Image Data\Chung\2018-07-12 cancer migration assay\H&H (2018-07-11)) c...
intergroup_time_master_Hayley_8_24.m
Workspace
Name
a_dir
a_disp
a_sp
cell_recov_dest
ci
CI
cmap_b
cmap_g
cmap_k
cmap_r

```
            end
        end
end

% Creates a new matrix that will result in the average
% condition at each time (rows are conditions and colu
% gives the error for each condition at each time.
subplot(1,4,4)
for ci = 1:length(condition_list)
    for ti = 1:length(time_master)
```

Figure 14. The top of the MATLAB program. Inside the editor tab is a green arrow that is used to run the MATLAB code.