Electronic Supplementary Material (ESI) for Nanoscale Advances.

Hand-made macros and scripts used in this study

ImageJ macros (Fiji for Windows, 64-bit)

Function	Name of macro	Page
Make kymographs from row stacked images	fiji0044-19_make_kymograph_dv_c_220201.ijm	2
Merge the kymographs of Cy3 and A647 channels	fiji0045-7_kymograph_merge_GrnMgt_mstd_220201.ijm	7
Analyze motile parameters from merged kymographs	fiji0048_5_VIR_from_merged_kymograph_a_220201.ijm	10

Python scripts (Anaconda3, 64-bit; Windows 10 Home)

Function	Name of script	Page
Arrange, filter and plot data of motile parameters	py0114_6_vir_analyzer_d_vr1um_notintsel_220121.py > py0052_7_my_library_a_220122 > py0060_8_DataFrame_to_Series_conductor_b_220125 > py0115_3_filter_by_length_py0114_b_220125 > py0056_8_csv_connector_for_0053_a_220125 > py0062_7_jitter_plot_a_220125 > py0067_3_csv_concater_by_2nd_file_key_a_220126 > py0116_9_matrix_and_histgram_c_220126 > py0081_3_n_table_concater_a_220126 > py0043_13_run_length_analyzer_e_220126 > py0086_7_run_length_analyzer_bootstrap_f_220126 > py0087_8_scatter_graph_for_py0075_c_220126 (“>” means a script imported in the top script)	15 20 25 27 30 32 35 37 44 47 51 58
Fit histograms of velocity with Gaussian distribution function	py0135_0_histgram_with_Gaussian_fitting_1_220208.py	62
Plot the Gaussian-fitted parameters	py0136_0_scatter_graph_for_vel_Gaussian_d_220209.py	68

ImageJ macros

fiji0044-19_make_kymograph_dv_c_220201.ijm

```
// fiji0044
// Make kymographs from stacked images of single-molecule imaging
//
//

macro "process_tirf_intensity_data [F2]" {

	// Initialization
	output = "C:¥¥Fukumoto¥¥edited";
	suffix = ".fits";
	run("Line Width...", "line=12");
	input = getDirectory("Chose an input Directory");	// Select a directory for analysis using GUI
	write("input: " + input);
	input_sep = split(input, "¥¥");
	input_name = input_sep[input_sep.length -1];	// Name of the directory

	// Obtain a path of the upper-level directory
	input_dir = ""
	for (i=0; i<input_sep.length -1; i++) {
		input_dir = input_dir + "¥¥" + input_sep[i];
	}
	//write("input_name: " + input_name);
	//write("input_dir: " + input_dir);

	// Directories for output
	output_tif_mst = replace(input_dir, "data", "edited//90_tif");
	output_tif = replace(input, "data", "edited//90_tif");
	output_png_mst = replace(input_dir, "data", "edited//1_spi");
	output_png = replace(input, "data", "edited//1_spi");
	File.makeDirectory(output_tif_mst);
	File.makeDirectory(output_tif);
	File.makeDirectory(output_tif + "plane");
	//File.makeDirectory(output_tif + "plane_resize");
	File.makeDirectory(output_tif + "2pix-mean");
	//File.makeDirectory(output_tif + "2pix-mean_resize");
	File.makeDirectory(output_png_mst);
	File.makeDirectory(output_png);
	File.makeDirectory(output_png + "plane");
	//File.makeDirectory(output_png + "plane_resize");
	File.makeDirectory(output_png + "2pix-mean");
	//File.makeDirectory(output_png + "2pix-mean_resize");
	File.makeDirectory(output_png + "row");
	File.makeDirectory(output_png + "rois");
	//error = 1 + e;

	// Main loop
	list = getFileList(input);	// List of files in the directory selected using GUI
	list = Array.sort(list);
	for (i = 0; i < list.length; i++) {
		if(endsWith(list[i], suffix)) {
```

```
                process_tirf_int_data(input, output, list[i], input_name);
            }
        }
        write("Completed!!")
}

function process_tirf_int_data(input, output, file, input_name) {
        // Open the passed file and make and save kymographs
        file_ret = split(file, "¥¥.");
        file_core = file_ret[0];            // Core of the file name
        image_id = open_file(input, file);
        make_kymograph_dv(image_id, file_core, input, input_name);
        close_file(image_id);
}

function open_file(input, file) {
        // Initialization: directories for output
        output_tif = replace(input, "data", "edited//90_tif");
        output_png = replace(input, "data", "edited//1_spi");

        // Open the passed file
        file_pass = input + File.separator + file;
        open(file_pass);
        image_id = getImageID();
        setSlice(5);
        setMinAndMax(80, 200);
        saveAs("PNG", output_png + "row¥¥" + file_core + "_cnt1000_frame5_row.png");
        return image_id;
}

function make_kymograph_dv(image_id, file_core, input, input_name) {
        // Make kymograph, selecting an axoneme on the red channel using GUI

        // Initialization
        if(roiManager('count') != 0){                   // The ROI manager is not empty
            roiManager("Deselect");                         // make it empty
            roiManager("Delete");
        }

        // Directories for output
        output_tif = replace(input, "data", "edited//90_tif");
        output_png = replace(input, "data", "edited//1_spi");

        // Z-project
        run("Z Project...", "projection=[Average Intensity]");
        zpro_id = getImageID();
        setMinAndMax(80, 130);

        // Select a ROI of an axoneme on the RED channel using the "Polyline" tool.
        waitForUser("Select Axoname in RED channel");
        roiManager("add");
        roiManager("deselect");
        roiManager("Save", output_png + "rois¥¥" + file_core + "_axoneme.roi");
        roiManager("delete");
```

```
	//
	// Calculate the coorinate of the axoneme on the green channel
	//
	// Obrain information of the ROI on the red channel
	getSelectionCoordinates(xpoints, ypoints);	// Coordinate of the polyline ROI on the RED channel
	n_points = lengthOf(xpoints);	// Number of point in the polyline ROI

	// Green channel
	new_xpoints = set_roi_to_another_x(xpoints, ypoints, n_points);
	new_ypoints = set_roi_to_another_y(xpoints, ypoints, n_points);

	//setBatchMode(true);
	selectImage(image_id);
	run("Restore Selection");
	setBatchMode(true);
	selectImage(zpro_id);
	close();

	// Make kymographs
	make_kymograph_wzpro(image_id, file_core, input, input_name, "plane", xpoints, ypoints, new_xpoints, new_ypoints, n_points);

	// Same process with mean filter
	selectImage(image_id);
	run("Duplicate...", "duplicate");
	dup_image_id = getImageID();
	run("Mean...", "radius=2 stack");	// Mean filter (2 pixcel)
	run("Restore Selection");		// Restore of ROI of the axoname
	//setSelectionLocation(x-dx, y-dy);	// Parrallel shift of ROI
	make_kymograph_wzpro(image_id, file_core, input, input_name, "2pix-mean", xpoints, ypoints, new_xpoints, new_ypoints, n_points);

	selectImage(dup_image_id);
	setBatchMode(false);
	close();

}


function make_kymograph_wzpro(image_id, file_core, input, input_name, sub_id, xpoints, ypoints, new_xpoints, new_ypoints, n_points) {
	// Make kymographs following the passed ROI
	// Initialization
	ratio=0.1;
	output_tif = replace(input, "data", "edited//90_tif");
	output_png = replace(input, "data", "edited//1_spi");

	// Red channel
	selectImage(image_id);
	makeSelection("polyline", xpoints, ypoints, n_points);	// ROI for an axoneme on the RED channel
	//setMinAndMax(80, 900);
	//setMinAndMax(80, 500);
	run("Reslice [/]...", "output=1.000 slice_count=1 avoid");
	getDimensions(width, height, channels, slices, frames);
	saveAs("TIF", output_tif + sub_id + "¥¥" + file_core + "_red_" + sub_id + "_kymograph.tif");
	saveAs("PNG", output_png + sub_id + "¥¥" + file_core + "_red_" + sub_id + "_kymograph.png");
```

```
	close();

	// Green channel
	selectImage(image_id);
	makeSelection("polyline", new_xpoints, new_ypoints, n_points);		// ROI for an axoneme on the Green channel
	//setMinAndMax(80, 900);
	//setMinAndMax(80, 1000);
	run("Reslice [/]...", "output=1.000 slice_count=1 avoid");
	id_plane = getImageID();

	// Resize in ordet to match the size of both of kymographs
	run("Scale...", "x=- y=- width=" + width + " height=" + height +" interpolation=Bilinear average create title=resized");
	id_resize = getImageID();

	// Save
	saveAs("TIF", output_tif + sub_id + "¥¥" + file_core + "_green_" + sub_id + "_kymograph.tif");
	saveAs("PNG", output_png + sub_id + "¥¥" + file_core + "_green_" + sub_id + "_kymograph.png");

	// Post-processing
	selectImage(id_resize);
	close();
	selectImage(id_plane);
	close();
}

function set_roi_to_another_x(xpoints, ypoints, n_points) {
	// Translate the passed coordinate of a ROI
	// FOR x (Output x and y separately, since Array in Array is not supported.)

	// Initialization
	new_xpoints = newArray(n_points);

	// Calculate a coordinate of each point in the Polyline
	for (i = 0; i < n_points; i++) {
		x = xpoints[i];
		y = ypoints[i];
		if (y<300) {		// Constant (x=109) within 0<y<300
			dx = 109;
		} else {		// linear function depending on the coorinate of y for y>=300.
			dx=-0.012*y +112.5;
		}
		new_xpoints[i] = x +dx;
	}
	return new_xpoints;
}

function set_roi_to_another_y(xpoints, ypoints, n_points) {
	// Translate the passed coordinate of a ROI
	// FOR y (Output x and y separately, since Array in Array is not supported.)

	// Initialization
	new_ypoints = newArray(n_points);

	// Calculate a coordinate of each point in the Polyline
```

```
	for (i = 0; i < n_points; i++) {
		y = ypoints[i];
		dy = -0.0092*y -9.0;		// linear function depending on the coorinate of y
		new_ypoints[i] = y +dy;
	}
	return new_ypoints;
}

function close_file(image_id) {
	selectImage(image_id);
	close();
}
```

fiji0045-7_kymograph_merge_GrnMgt_mstd_220201.ijm

```
// fiji0045
// Merge kymographs of red and green channels
//
macro "process_images_3 [F3]" {
	print("¥¥Clear");
	input = getDirectory("Chose an input Directory");	// Choose a directory for analysis using GUI
	list = getFileList(input);
	list = Array.sort(list);
	// Analyze directories in the designated directory
	for (i=0; i<list.length; i++) {
		write(list[i]);
		subscript(input, list[i]);	// Main loop
	}
}

function subscript(dir_master, dir_sel) {
	// Initialization
	suffix = ".tif";
	//input_sep = split(input, "¥¥");
	//wdir_name = input_sep[input_sep.length -1];	// Name of the passed directory
	input = dir_master + dir_sel;
	dir_work = input + "2pix-mean";
	output_tif = input + "¥¥merge_recontrast";
	output_jpg = replace(output_tif, "90_tif", "1_spi");
	output_color_jpg = replace(output_jpg, "merge", "color");
	//File.makeDirectory(replace(output_jpg, "¥¥merge", ""));
	File.makeDirectory(output_jpg);
	File.makeDirectory(output_color_jpg);
	File.makeDirectory(output_tif);

	// Analyze files in the passed directory
	list = getFileList(dir_work);	// List of files in the directory
	list = Array.sort(list);
	for (i = 0; i < list.length; i+=2) {
		write(list[i]);
		if(endsWith(list[i], suffix)) {
			file = list[i];	// file name
			sep = split(file, "_");
			gfile = file;
			mfile = replace(file, "green", "red");
			process_file(dir_work, gfile, mfile, output_jpg, output_tif, output_color_jpg);
		}
	}
}

function process_file(input, gfile, mfile, output_jpg, output_tif, output_color_jpg) {
	// Initialization
	setBatchMode(true);
	gfile_pass = input + "¥¥" + gfile;
	mfile_pass = input + "¥¥" + mfile;
	gimage_id = open_file(input, gfile);
	mimage_id = open_file(input, mfile);
```

```
	file_ret = split(gfile, "_");
	file_core = file_ret[0] + "_" + file_ret[1];		// Core of the file name
	write("	File_core_name is, " + file_core);	// Logging
	write("	Output directory is, " + output_jpg);	// Logging

	// Main process
	merge_image_id = merge_channels(gfile, mfile, file_core, output_tif);
	process_merge_image(merge_image_id, file_core, output_jpg, output_color_jpg);

	// Post-process
	close_file(merge_image_id);
	close_file(gimage_id);
	close_file(mimage_id);
	setBatchMode(false);
}

function open_file(input, file) {
	// Open the passed file
	open(input + "¥¥" + file);
	image_id = getImageID();
	setMinAndMax(80, 300);
	// run("In [+]");
	return image_id;
}

function merge_channels(gfile, mfile, file_core, output) {
	// generate a stacked image merging two files
	run("Merge Channels...", "c2=2pix-mean¥¥" + gfile +" c6=2pix-mean¥¥" + mfile + " create keep");
	image_id = getImageID();
	fig_name = file_core + "_merge.tif";
	run("Save", "save=" + output + "¥¥" + fig_name);
	write("	Saved figure as, " + fig_name);
	return image_id
}

function process_merge_image(merge_image_id, file_core, output, output_color) {

	// green channel
	selectImage(merge_image_id);
	setSlice(1);
	//setMinAndMax(gmin, gmax);
	run("Duplicate...", "duplicate channels=1");	// Cy3 chennel
	fig_g = file_core + "_green.jpg";
	saveAs("Jpeg", output_color + "¥¥" + fig_g);
	//saveAs("PNG", output_color + "¥¥" + fig_g);
	write("	Saved figure as, " + fig_g);
	close();

	// magenta channel
	selectImage(merge_image_id);
	setSlice(2);
	//setMinAndMax(80, 200);
	run("Duplicate...", "duplicate channels=2");	// A647 channel
	fig_m = file_core + "_magenta.jpg";
```

```
        saveAs("Jpeg", output_color + "¥¥" + fig_m);
        //saveAs("PNG", output_color + "¥¥" + fig_m);
        write("        Saved figure as, " + fig_m);
        close();

        // merge image
        selectImage(merge_image_id);
        run("Stack to RGB");        // Merge
        fig_merge = file_core + "_merge.jpg";
        saveAs("Jpeg", output + "¥¥" + fig_merge);
        //saveAs("PNG", output + "¥¥" + fig_merge);
        write("        Saved figure as, " + fig_merge);
        close();
        //selectImage(merge_image_id);
}

function close_file(image_id) {
        selectImage(image_id);
        close();
}
```

fiji0048_5_VIR_from_merged_kymograph_a_220201.ijm

```
// fiji0048:
// Get values of velocities, intensities (slice 1 and 2) and run lengths
// of motile particles from kymographs made by Fiji0045 etc
//

macro "process_tirf_intensity_data [F4]" {

	// Initialization
	suffix = ".tif";
	cnt = 0;
	run("Line Width...", "line=3.5");
	run("Clear Results");
	print("¥¥Clear");

	// Select a directory for analysis
	input = getDirectory("Chose an input Directory");
	input_sep = split(input, "¥¥");
	input_name = input_sep[input_sep.length -1];	// Name of the input directory
		//dir_work = input + "¥¥mean";
		//dir_work = input + "¥¥merge"
		dir_work = input + "¥¥merge";

	// Make a directory for output
	output_tif = input + "vir_analysis";	// Directory to output tiff files
	output_png = replace(output_tif, "90_tif", "1_spi");	// Directory to output png files
	//File.makeDirectory(output_tif);
	File.makeDirectory(output_png);

	// Analyze each file in the input directory
	list = getFileList(dir_work);	// List of files in dir_work directory
	list = Array.sort(list);
	for (i = 0; i < list.length; i++) {
		write(list[i]);
		if(endsWith(list[i], suffix)) {	// files including a designated suffix
			cnt = sub_script(dir_work, output_png, list[i], input_name, cnt);
		}
	}
	updateResults();
	saveAs("Results", output_png + "¥¥" + input_name + "_vir.csv")
	write("Completed!!")	// Logging
}

function sub_script(input, output, file, input_name, cnt) {
	file_ret = split(file, "_");
	file_core = file_ret[0] + "_" + file_ret[1];	// Core of the file name
	//write(file_core);
	image_id = open_file(input, file);
	cropped_id = crop_color(file, image_id);
	cnt = analyze_velocity_with_kymograph(image_id, file_core, output, cnt);
	close_file(image_id);
	close_file(cropped_id[0]);
	close_file(cropped_id[1]);
```

```
	return cnt;
}

function open_file(input, file) {
	// Open a passed file
	file_pass = input + File.separator + file;
	open(file_pass);
	image_id = getImageID();
	//run("In [+]");
	setSlice(1);
	setMinAndMax(80, 200);
	setSlice(2);
	setMinAndMax(80, 200);
	return image_id;
}

function crop_color(file, image_id) {
	// Crop each slice from a marged (stacked) image
	cropped_id = newArray(2);
	for (i = 0; i < 2; i++) {
		selectImage(image_id);
		slice = i +1;
		setSlice(slice);
		run("Duplicate...", "title=" + replace(file, ".tif", "_channel_" + slice +".tif") + " duplicate channels=" + slice);
		id_tmp = getImageID();
		cropped_id[i] = id_tmp;
	}
	return cropped_id;
}

function analyze_velocity_with_kymograph(image_id, file_core, output, cnt) {
	// Select each motile particle using GUI and obtain the motile parameters

	// Initialization
	selectImage(image_id);
	setSlice(1);
	bg_tmp = newArray(2);
	bg_sum = newArray(2);
	bg_mean = newArray(2);
	if(roiManager('count') != 0){			// If ROI manager is not empty
		roiManager("Deselect");				// make ROI manager empty
		roiManager("Delete");
	}

	// **** Set background intensity ****
	waitForUser("Set BUCKGROUNDS");		// Select 3 ROIs for background using "Straight" tool
	n = roiManager("count");
	selectImage(image_id);
	if(roiManager('count') != 0){		// If one more ROI(s) are selected for background
		// Initialization
		setBatchMode(true);
		roiManager("Deselect");
		roiManager("Save", output + "¥¥" + file_core + "_bg_roiset.zip");
		sum = 0;
```

```
        // Calculate the values of backgound as mean of selected ROI(s)
        for(index=0; index<n; index++) {        // index = ROI #
            bg_tmp = bg_loop(file, index);
            bg_sum[0] += bg_tmp[0];        // Sum on slice 1
            bg_sum[1] += bg_tmp[1];        // Sum on slice 2
            //print(sum);
        }
        bg_mean[0] = bg_sum[0]/n;        // Mean on slice 1
        bg_mean[1] = bg_sum[1]/n;        // Mean on slice 2
        print("bg_signal = (" + bg_mean[0] + "," + bg_mean[1] + ")");

        // Post-processing
        roiManager("Deselect");                    // Empty the ROI manager
        roiManager("Delete");
        setBatchMode(false);
    }

    // **** Set signals (main process) ****
    getDimensions(width, height, channels, slices, frames);
    thr_w50 = width -50;
    thr_w100 = width -100;
    thr_h50 = height -50;
    thr_h100 = height -100;
    write("Threshold of Width: (fifty, " + thr_w50 + "), (a hundred, " + thr_w100 + ")");
    write("Threshold of Height: (fifty, " + thr_h50 + "), (a hundred, " + thr_h100 + ")");
    waitForUser("Select Particles (Vel/Int -> Run Length).");
    // NOTE
    // Add two ROIs per one motile particle
    // Primary: select a region which is not stacked with other any partice(s), determining the velocity and intensities
    // Seconcary: select the region that connect the appearance and disappearence points, determining the run length。
    n = roiManager("count");
    selectImage(image_id);

    // Collect data from the selected ROI(s)
    if(roiManager('count') != 0){
        // Save the ROI(s)
        setBatchMode(true);
        roiManager("Deselect");
        roiManager("Save", output + "¥¥" + file_core + "_signal_roiset.zip");

        // Collect values
        for(index = 0; index < n; index += 2) {        // Load even ROI(s) since 2 ROIs are selected per one particle
            cnt = main_loop(file_core, index, cnt, bg_mean);
        }

        // Post-processing
        roiManager("Deselect");
        roiManager("Delete");
        setBatchMode(false);
    }

    return cnt;
}
```

```
function bg_loop(file, index) {
	// Obtain mean intensities of the passed ROI on two channels
	// Initialization
	bg = newArray(2);
	dx = 0;		// 190625
	roiManager("select", index);
	//getSelectionBounds(x, y, w, h);

	// Obtain mean intensities of the ROI
	// Slice 1
	setSlice(1);
	getStatistics(area, intensity_slice1);
	bg[0] = intensity_slice1;
	print(file + "," + index + ",slice 1," + area + "," + intensity_slice1);		// Logging

	// Slice 2
	setSlice(2);
	getStatistics(area, intensity_slice2);
	bg[1] = intensity_slice2;
	print(file + "," + index + ",slice 2," + area + "," + intensity_slice2);		// Logging

	//setSlice(1);
	return bg;
}

function main_loop(file, index, cnt, bg) {
	// Obtain velocities, intensities and run lengths using information of the ROI manager

	// Obtain velocity and intensity (even number)
	roiManager("select", index);
	getSelectionBounds(x, y, w, h);
	l = w*0.07;
	t = h*0.2;		// exposure time (s)

	// Slice 1
	setSlice(1);
	getStatistics(area, intensity_slice1, min, max, sd_slice1);
	intensity_slice1_delta = intensity_slice1 -bg[0];

	// Slice 2
	setSlice(2);
	getStatistics(area, intensity_slice2, min, max, sd_slice2);
	intensity_slice2_delta = intensity_slice2 -bg[1];

	// Obtain run length (odd number)
	roiManager("select", index+1);
	getSelectionBounds(x_run, y_run, w_run, h_run);
	l_run = w_run*0.07;
	t_run = h_run*0.2;

	// Whether the particle reached an edge of the kymograph
	getDimensions(width, height, channels, slices, frames);		// Size of the kymograph
	getSelectionCoordinates(xpoints, ypoints);		//	(xpoints[1], ypoints[1]) is the coorinate of the disappeared point
```

```
	// x (displacement axis)
	full_x = 1;	// Initialization
	if(xpoints[1]>width-5 || xpoints[1]<5) {	// within 5 pixels from either edge
		full_x = 0;
	}
	// y (time axis)
	full_y = 1;	// Initialization
	if(ypoints[1]>height-5) {	// within 5 pixcels from the bottom
		full_y = 0;
	}

	// Write down ROI information and motile parameters into Results table
	setResult("File", cnt, file);		//	File	Index	x	y	Width	Height
	setResult("Index", cnt, index);		//	0	file_0	index_0	x_0	y_0	w_0	h_0
	setResult("x", cnt, x);			//	1	...
	setResult("y", cnt, y);
	setResult("Width (pixel)", cnt, w);
	setResult("Height (pixel)", cnt, h);
	setResult("Length (um)", cnt, l);
	setResult("Time (s)", cnt, t);
	setResult("Velocity (um/s)", cnt, l/t);
	setResult("Slice_1", cnt, intensity_slice1_delta);
	setResult("Slice_2", cnt, intensity_slice2_delta);
	setResult("SD_Slice_1", cnt, sd_slice1);
	setResult("SD_Slice_2", cnt, sd_slice2);
	setResult("x_run", cnt, x_run);
	setResult("y_run", cnt, y_run);
	setResult("Width_run (pixel)", cnt, w_run);
	setResult("Height_run (pixel)", cnt, h_run);
	setResult("Run Length (um)", cnt, l_run);
	setResult("Run Time (s)", cnt, t_run);
	setResult("Full_x", cnt, full_x);
	setResult("Full_y", cnt, full_y);
	setResult("Full", cnt, full_x*full_y);

	// Check
	if(w>w_run+5) {		// Raise an error if the width of ROI #1 is longer than one of ROI #2. Five-pixel margin is considered as human error of ROI selection.
		setResult("ERROR", cnt, 1);
	}

	cnt += 1;
	return cnt;
}


function close_file(image_id) {
	selectImage(image_id);
	close();
}
```

Python scripts

py0114_6_vir_analyzer_d_vr1um_notintsel_220121.py

```
#
# py0114:
# Analyze and visualize data including velocity, intensities and run length (VIR) obtained using
Fiji0048 or related script
#
# Criteria
# Segment legnth < 0.5 μm
# No selection with intensity
#
# -*- coding: utf8 -*-

# Import standard modules
import sys
import importlib
import pandas as pd


# Import hand-made modules/scripts
import py0052_7_my_library_a_220122 as ml  # OK
import py0060_8_DataFrame_to_Series_conductor_b_220125 as py0060  # OK
import py0115_3_filter_by_length_py0114_b_220125 as py0066  # OK
import py0056_8_csv_connector_for_0053_a_220125 as py0056  # OK
import py0062_7_jitter_plot_a_220125 as py0062  # OK
import py0067_3_csv_concater_by_2nd_file_key_a_220126 as py0067  # OK
#import py0070_2_csv_concater_by_1st_prior_key_a_220126 as py0070
import py0116_9_matrix_and_histgram_c_220126 as py0065  # OK
import py0081_3_n_table_concater_a_220126 as py0081  # OK
import py0043_13_run_length_analyzer_e_220126 as py0043  # OK
import py0086_7_run_length_analyzer_bootstrap_f_220126 as py0086  # OK
import py0087_8_scatter_graph_for_py0075_c_220126 as py0087  # OK

# Reload the hand-made modules
#modules = [ml, py0060, py0066, py0056, py0062, py0067, py0070, py0065, py0081, py0043, py0086,
py0087]
modules = [ml, py0060, py0066, py0056, py0062, py0067, py0065, py0081, py0043, py0086, py0087]
for module in modules:
    importlib.reload(module)

#
# Main process
#
def main():
    # Initialization
    args = sys.argv
    script_num = args[0].split("_")[0]
    print("¥n¥n** ** ** ** ** **  {} ** ** ** ** ** **".format(args[0]))
    context_small = "notebook"
    context = "poster"
    config = "GUI"
    #config = "test"
```

```
    path_log = "C:¥¥fukumoto¥¥script_py¥¥py_log.txt"
    id, dir_log, dir_in, dir_out = ml.initialize(path_log, label="VIR_vr0p5um_notintsel",
config=config)

    # Setting Values
    dir_in_ret = dir_in.split("/")
    dir_in_name = dir_in_ret[-1]    # Name of the directory
    print("¥nDir_in: {}".format(dir_in_name))    # log
    threshold = {"Length (um)": 0.5, "Run Length (um)": 0.5, "Slice_1": 10000, "Slice_2": -10000}
# threshold of analysis
    if dir_in_name.startswith("Layout_Sx2"):
        ylim = (0.8, 8)    # (ylim of velocity, ylim of run length) at summarized graphs
        xrange = [(-30, 300), (0, 1.5), (-40, 500), (-20, 400), (0, 20)]  # Intensity, Velocity,
Slice1, Slice2, Run length
    elif dir_in_name.startswith("Layout_Sx4"):
        ylim = (0.8, 5)
        xrange = [(-30, 300), (0, 1), (-40, 700), (-20, 500), (0, 20)]
    elif dir_in_name.startswith("Number"):
        ylim = (1.3, 8)
        xrange = [(-30, 300), (0, 2.5), (-40, 500), (-20, 300), (0, 20)]
    elif dir_in_name.startswith("free"):
        ylim = (1.3, 2)
        xrange = [(-30, 300), (0, 2.6), (-40, 500), (-20, 400), (0, 20)]
    elif dir_in_name.startswith("monomer_layout_Sx4"):
        ylim = (0.3, 5)
        xrange = [(-30, 300), (0, 2.6), (-40, 200), (-20, 100), (0, 20)]
    elif dir_in_name.startswith("monomer_layout_Sx2"):
        ylim = (0.3, 5)
        xrange = [(-30, 300), (0, 0.8), (-40, 200), (-20, 100), (0, 20)]
    elif dir_in_name.startswith("monomer_number"):
        ylim = (0.3, 5)
        xrange = [(-30, 300), (0, 0.8), (-40, 200), (-20, 100), (0, 20)]
    else:
        print("** ** OTHER SAMPLE: {} ** **".format(dir_in_name))
        ylim = (1.2, 8)
        xrange = [(-30, 300), (0, 1.5), (-40, 500), (-20, 400), (0, 20)]

    # Preprocessing: Combine data from several directories (Filter, F)
    dir_tmp_1 = py0060.main(dir_in=dir_in, dir_out_master=dir_out, label="M_arrange",
config="pre_set",
                            pre_set_id=id, path_log=path_log)

    #
    # Analyze the data combining each chamber (4 axonemes) (F)
    #
    # Remove data which does not suit the threshold
    dir_tmp_f2_r, dir_n = py0066.main(dir_in=dir_tmp_1, dir_out_master=dir_out, label="F_filter",
config="pre_set",
                                  pre_set_id=id, path_log=path_log, foo_threshold=foo_threshold,
threshold=threshold)
    # Jitter plot
    dir_tmp_f4_r = py0056.main(dir_in=dir_tmp_f2_r, dir_out_master=dir_out, label="F_organize_R",
config="pre_set",
                               pre_set_id=id, path_log=path_log)  # Organize the data
```

```
    dir_dst_f_r = py0062.main(dir_in=dir_tmp_f4_r, dir_out_master=dir_out, label="F_jitter_R",
config="pre_set",
                              pre_set_id=id, path_log=path_log, context=context_small)  # Jitter plot
    # Fit the data of run lengths using CDF
    dir_fit_f = py0043.main(dir_in=dir_tmp_f2_r, dir_out_master=dir_out, label="F_fitting_R",
config="pre_set",
                            pre_set_id=id, path_log=path_log, markersize=10)

    #
    # Analyze the data combining each sample (4 chambers x 4 axonemes) (FC)
    #
    # Table showing n
    dir_n2 = py0081.main(dir_in=dir_n, dir_out_master=dir_out, label="FC_concat_n",
config="pre_set",
                             pre_set_id=id, path_log=path_log)
    # Concat the data of columns (R)
    dir_tmp_fc3_r = py0067.main(dir_in=dir_tmp_f2_r, dir_out_master=dir_out, label="FC_concat_R",
config="pre_set",
                             pre_set_id=id, path_log=path_log)
    # Jitter plot
    dir_tmp_fc4_r = py0056.main(dir_in=dir_tmp_fc3_r, dir_out_master=dir_out,
label="FC_organize_R", config="pre_set",
                             pre_set_id=id, path_log=path_log)  # organize the data
    dir_dst_fc_r = py0062.main(dir_in=dir_tmp_fc4_r, dir_out_master=dir_out, label="FC_jitter_R",
config="pre_set",
                            pre_set_id=id, path_log=path_log, context=context)  # Jitter plot

    # Fit the data of run lengths using CDF (bootstrap)
    dir_fit_fc = py0086.main(dir_in=dir_tmp_fc3_r, dir_out_master=dir_out,
label="FC_bootstrap_fitting_R", config="pre_set",
                             pre_set_id=id, path_log=path_log, markersize=10)

    #
    # Make graph of Mean ±SD (±SE)
    #
    dir_sct_v = py0087.main(dir_in=dir_tmp_fc3_r, dir_out_master=dir_out, label="V_graph",
config="pre_set",
                            pre_set_id=id, path_log=path_log, mode="v", ylim=ylim)
    dir_sct_r = py0087.main(dir_in=dir_fit_fc, dir_out_master=dir_out, label="R_graph",
config="pre_set",
                            pre_set_id=id, path_log=path_log, mode="r", ylim=ylim)

    #
    # Make a matrix (FC)
    #
    dir_dst_m3_r = py0065.main(dir_in=dir_tmp_fc3_r, dir_out_master=dir_out, label="FC_matrix_R",
config="pre_set",
                            pre_set_id=id, path_log=path_log, fontsize=20, xrange=xrange)  # FC

    # Logging
    print("¥n¥n********** {} Script Finished. **********¥n".format(script_num))


def foo_threshold(data_src, setting, file, threshold):
```

```
    '''
    A function to select data meeting the designated criteria (length, intensities)
    :param data_src: data before selection (pd.DataFrame)
    :param setting: "vi" (velocity and intensity, ROI #1), "r" (run length, ROI #2) or "vir" (vi
and r)
    :param file: file name
    :return: data after selection (pd.DataFrame), list of n before and after selection
(pd.series)
    '''

    # ROI 1: velocity and intensity
    if setting == "vi":
        data_tmp1 = data_src[data_src["Length (um)"] > threshold["Length (um)"]]    # Remove data
with run length LESS than a threshold
        drop_columns = [" ", "x_run", "y_run", "Width_run (pixel)", "Height_run (pixel)",
                        "Run Length (um)", "Run Time (s)", "Full_x", "Full_y", "Full"]
        for column in drop_columns:
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
            except:
                pass
        index = ["n (source)", "n (sel length)"]

    # ROI 2: run length
    elif setting == "r":
        data_tmp1 = data_src[data_src["Length (um)"] > threshold["Run Length (um)"]]
        drop_columns = [" ", "x", "y", "Width (pixel)", "Height (pixel)", "Time (s)",
                        "Velocity (um/s)", "SD_Slice_1", "SD_Slice_2"]
        for column in drop_columns:
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
            except:
                pass
        index = ["n (source)", "n (sel run length)"]

    # Velocity and run length
    elif setting == "vir":
        data_tmp1 = data_src[data_src["Run Length (um)"] > threshold["Run Length (um)"]]
        drop_columns = ["Unnamed: 0", "x", "y", "Length (um)", "Time (s)",
                        "x_run", "y_run", "SD_Slice_1", "SD_Slice_2"]
        for column in drop_columns:
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
            except:
                pass
        index = ["n (source)", "n (sel run length)"]

    list_n = [len(data_src), len(data_tmp1)]

    # Common process: selection by threshold of fluorescence intensity
    try:
        data_tmp2 = data_tmp1[data_tmp1["Slice_1"] < threshold["Slice_1"]]  # Remove data with Cy3
intensity is MORE than a threshold
        data_dst = data_tmp2[data_tmp2["Slice_2"] > threshold["Slice_2"]]  # Remove data with A647
```

```
intensity is LESS than a threshold
        index += ["n (sel Slice 1)", "n (sel Slice 2)"]
        list_n += [len(data_tmp2), len(data_dst)]
    except:
        data_dst = data_tmp1

    # ROI 2: Record particles reached the edge of the kymograph
    try:
        if setting == "r" or setting == "vir":
            data_sup1 = data_dst[data_dst["Full_x"] > 0]  # 軸糸の変位方向で、kymograph の末端まで到達せずに解離したもの
            data_sup2 = data_sup1[data_sup1["Full_y"] > 0]  # 時間方向で、kymograph の末端まに解離したもの
            index += ["* n (totally_run_x))", "* n (totally_run_t)"]
            list_n += [len(data_sup1), len(data_sup2)]
    except:
        pass

    # Output
    sr_n = pd.Series(list_n, index=index)
    sr_n.name = file
    return data_dst, sr_n

if __name__ == "__main__":
    main()
```

```
#
# py0052_my_library
# Module
#
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import pprint

class Log_File():
    '''
    A class contains information about path, contents, etc of a log file
    '''
    def __init__(self, filepath):
        '''
        Initialization
        :param filepath: Absolute path of a log file
        '''
        self.file_path = filepath
        self.file_name = os.path.basename(filepath)     # Name of a log file
        with open(self.file_path, encoding='UTF-8') as f:
            self.content = f.read()                  # Content of the log file

    def save(self, s):
        '''
        The method to edit the content of the log file
        :param s: str which you would like to wite in the log file
        '''
        with open(self.file_path, mode='w', encoding='UTF-8') as f:  # save the parameter in the
log file
            f.write(s)

    def load(self):
        '''
        The method to load the content of the log file again
        :return: str written in the log file
        '''
        with open(self.file_path, encoding='UTF-8') as f:    # Load the log file
            self.content = f.read()                      # Write str in the log file
        print("¥n¥nLog file was load,¥n",self.content)
        return self.content


def get_id_logdir(label="", config="GUI", pre_set_id=""):
    '''
    Make a directory whose name is "date_##" (ID)
    :return: id (str, today_#), path (str) of a directory for a log file
```

```
(¥¥log¥¥today¥¥today_time¥¥)
    '''
    path_log = "C:¥¥fukumoto¥¥script_py¥¥log_files¥¥logfile_id_for_analysis.txt"
    if config=="GUI" or config=="test":
        # Today
        now = datetime.datetime.now()
        today = "{0:%y%m%d}".format(now)

        # Load a previous analysis ID in order to determine this analysis ID
        log_file = Log_File(path_log)
        s = log_file.content

        # Determine analysis ID (when the analysis is the first one on the day, "00"; otherwise,
add number)
        ref = s.split(",")
        #print("log_file.content:{}".format(s))
        if ref[0] != today:  # When the date loaded from the log file is the same as today (the
first analysis on the day)
            sub_id = "0"  # set as "0"
        elif ref[0] == today:  # When the date loaded from the log file is the same as today
(second or more analysis on the day)
            tmp = int(ref[1]) + 1  # add number
            sub_id = str(tmp)
        log_file.save(today + "," + sub_id +",0")  # Log the new analysis ID (today,#)
        if int(sub_id) < 10:  # When the analysis id number is single-digit
            sub_id = "0" + sub_id  # Add "0"
        id = today + "_" + sub_id

    elif config=="pre_set":
        # Load a previous sub-sub-ID in order to determine the current one
        log_file = Log_File(path_log)
        s = log_file.content    # Date,sub-id,sub_sub_id
        s_ret = s.split(",")    # [Date, sub-id, sub_sub_id]
        today = s_ret[0]    # Date
        sub_id = s_ret[1]    # sub-id
        sub_sub_id = s_ret[2]
        #print("sub_sub_id: {}".format(sub_sub_id))

        # Determine sub-sub-ID (First analysis with the main-ID, "00"; others, add 1)
        main_id = today + "_" + sub_id
        #ref = main_id.split("_")    # [Date, sub-id]
        #today = ref[0]    # Date
        if main_id != pre_set_id:    # NEW SCRIPT (main-ID)
            main_id = pre_set_id
            today = main_id.split("_")[0]
            sub_id = main_id.split("_")[1]
            sub_sub_id = "0"       # sub_sub_id = 0
        else:    # SAME SCRIPT
            main_id = main_id    # SKIP
            tmp = int(sub_sub_id) + 1    # increase the number of sub_sub_id
            sub_sub_id = str(tmp)
        log_file.save("{},{},{}".format(today, sub_id, sub_sub_id))  # 今の解析の ID を書き出す
(today,#,#)
        id = main_id + "-" + sub_sub_id  # Determine ID (Date + ##)
```

```
    # Make a directory for a log file and return ID and the path
    print("¥nID: {}".format(id))    # Log
    dir_out = "C:¥¥fukumoto¥¥log¥¥" + today + "¥¥" + label + "_" + id +"¥¥"    # Name of a
directory for a log file
    os.makedirs(dir_out)
    return id, dir_out    # Return ID and directory path


def select_directory(first_path):
    '''
    A function to let a user select a directory using GUI and return the path
    :return: the absolute path of the selected file
    '''
    # Display a dialog to select a file
    root_tmp = tk.Tk()
    root_tmp.withdraw()
    # fType = [("", "*" + suffix)]
    # iDir = os.path.abspath(os.path.dirname(__file__))
    first = first_path
    dir_path = tkinter.filedialog.askdirectory(initialdir = first)
    root_tmp.destroy()      # destroy() method avoids an error
    print("¥nThe selected directry: {}".format(dir_path))
    return dir_path


def select_file(first_path, suffix="csv"):
    '''
    A function to let a user select a file using GUI and return the path
    :param suffix: displayed file(s); when not given, all files are displayed
    :return: the absolute path of the selected file
    '''
    # Display a dialog to select a file
    root_tmp = tk.Tk()
    root_tmp.withdraw()
    fType = [("", "*" + suffix)]
    # iDir = os.path.abspath(os.path.dirname(__file__))
    first = first_path
    file_path = tkinter.filedialog.askopenfilename(filetypes=fType, initialdir = first)
    root_tmp.destroy()      # destroy() method avoids an error
    print("¥nThe selected directry: {}".format(file_path))
    return file_path


def get_sublayer_items(path, suffix, visualize=True):
    '''
    Pick a file endswith(suffix) from a designated directory
     (When you want to select a directory, pass "" for suffix)
    :param path: an absolute path of a directory containing files you want to select
    :param suffix: str which contains files you want to select at the end
    :param visualize: when true, log messages are printed
    :return: an list of file name you selected from the directory
    '''
    l = []    # List for files meeting a criteria
```

```
    for x in os.listdir(path):
        if x.endswith(suffix):
            l.append(x)
    if visualize:    # log
        print('These are selected,')
        pprint.pprint(l)
    return l


def initialize(path_log, label="", post_label="", config="GUI", dir_in="", pre_set_id="",
mkdir=True, opn="dir"):
    '''
    A method for starting an analysis
    1. Get an anallysis iD
    2. Get a directory for a log
    3. Select a directory for analysis using GUI
    4. Get a directory for result(s)
    :param path_log: an absolute path of the log file which contains a path of directory/file
which was previously analyzed
    :param test: False as default. When True was passed, the last-minute path was passed (useful
for testing the script)
    :return: analysis ID, path of the directory for analysis log, one for analysis (input) and
one for analysis results (outpu),
    '''
    id, dir_log = get_id_logdir(label=label, config=config, pre_set_id=pre_set_id)      # Get
analysis ID

    # Set an analysis ID (label + id + post_label)
    id_label = label + "_" + id
    if config == "pre_set":
        id_label = id + "_" + label
    if post_label:
        id_label += "_" + post_label


    # Function depending on the parameter of "config" ("test", "GUI", "pre-set")
    log = Log_File(path_log)
    if config == "test":
        input = log.content
        if opn == "dir":
            dir_in = input    # Same as one of the previous analysis
        elif opn == "file":
            dir_in = os.path.dirname(input)
    elif config == "GUI":
        if opn == "dir":
            dir_in = select_directory(log.content)    # Select a directory for analysis using GUI
            input = dir_in
        elif opn == "file":
            input = select_file(log.content, ".csv")
            dir_in = os.path.dirname(input)
        log.save(input)    # 選択したパスをログファイルに保存する
    elif config == "pre_set":
        #input = log.content
        input = ""
```

```
# Prepare a path of a directory for analysis results. If the directory is not exist, make it.
dir_out = dir_in + "¥¥" + id_label
if mkdir:
    try:
        os.makedirs(dir_out)
    except:
        pass
return id, dir_log, input, dir_out
```

```
#
# py0060: Arrange files generated using Fiji0048
# Concat data DataFrame in defiance of the columns, removing "0" data.
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from IPython.display import display
import pprint
#import importlib


def main(dir_in="", dir_out_master="", label="", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt"):
    # Initialize and select a directory for analysis
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, "", visualize=False)  # Get files from the designated
directory
    dir_processed = []
    dir_unprocessed = []
    for x in items:
        try:
            sub_script(dir_in + "¥¥" + x, id, dir_log, dir_out)
            dir_processed.append(x)
        except FileNotFoundError:
            dir_unprocessed.append(x)
    print("¥n* Processed directories:")  # -- log --
    pprint.pprint(dir_processed)
    print("¥n* Unprocessed directories:")
    pprint.pprint(dir_unprocessed)
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out


def sub_script(input, id, dir_log, dir_out):
    # Set the sub-layer directory
    dir_work = os.path.join(input, "vir_analysis")
    dir_name_input = os.path.basename(input)

    # Process the sub-layer directories in the super-layer directory
    items = ml.get_sublayer_items(dir_work, ".csv", visualize=False)  # Get files from the
designated directory
```

```
    for x in items:  # Process all directories in the super-layer directory
        data_src = pd.read_csv(dir_work + "¥¥" + x, index_col=" ", engine="python")    #
"engine=python" avoids an error.
        data_tmp = data_src
        data_tmp.to_csv(dir_out + "¥¥" + dir_name_input + "_" + id + "_connected.csv")


if __name__ == "__main__":
    main()
```

```
#
# py0115:
# Remove data which meets criteria (e.g. length (um) <= 0.5) from a file generated using
Fiji0048 and py0060 (Filter)
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from IPython.display import display
#import importlib

def foo_threshold_default(data_src, setting, file, threshold):
    '''
    A function to select data meeting the designated criteria (length, intensities)
    :param data_src: data before selection (pd.DataFrame)
    :param setting: "vi" (velocity and intensity, ROI #1), "r" (run length, ROI #2) or "vir" (vi
and r)
    :param file: file name
    :return: data after selection (pd.DataFrame), list of n before and after selection
(pd.series)
    '''

    # ROI 1: velocity and intensity
    if setting == "vi":
        data_tmp1 = data_src[data_src["Length (um)"] > threshold["Length (um)"]]    # Remove data
with run length LESS than a threshold
        drop_columns = ["Unnamed: 0", "x_run", "y_run", "Width_run (pixel)", "Height_run (pixel)",
                        "Run Length (um)", "Run Time (s)", "Full_x", "Full_y", "Full"]
        display(data_tmp1)
        for column in drop_columns:
            print("Column: {}".format(column))
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
            except:
                print("Failure drop")
                pass
        index = ["n (source)", "n (sel length)"]

    # ROI 2: run length
    elif setting == "r":
        data_tmp1 = data_src[data_src["Run Length (um)"] > threshold["Run Length (um)"]]
        drop_columns = ["Unnamed: 0", "x", "y", "Width (pixel)", "Height (pixel)", "Length (um)",
"Time (s)",
                        "Velocity (um/s)", "SD_Slice_1", "SD_Slice_2"]
        for column in drop_columns:
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
```

```
            except:
                pass
        index = ["n (source)", "n (sel run length)"]

    # Velocity, intensity and run length
    elif setting == "vir":
        data_tmp1 = data_src[data_src["Run Length (um)"] > threshold["Run Length (um)"]]
        drop_columns = ["Unnamed: 0", "SD_Slice_1", "SD_Slice_2"]
        for column in drop_columns:
            try:
                data_tmp1 = data_tmp1.drop(column, axis=1)
            except:
                pass
        index = ["n (source)", "n (sel run length)"]

    #list_n = [setting, len(data_src), len(data_tmp1)]
    list_n = [len(data_src), len(data_tmp1)]

    # Common process: selection by fluorescence intensity
    try:
        data_tmp2 = data_tmp1[data_tmp1["Slice_1"] < threshold["Slice_1"]]  # Remove data with Cy3
intensity is MORE than a threshold
        data_dst = data_tmp2[data_tmp2["Slice_2"] > threshold["Slice_2"]]  # Remove data with A647
intensity is LESS than a threshold
        index += ["n (sel Slice 1)", "n (sel Slice 2)"]
        list_n += [len(data_tmp2), len(data_dst)]
    except:
        data_dst = data_tmp1
    sr_n = pd.Series(list_n, index=index)
    sr_n.name = file

    return data_dst, sr_n

threshold_dafault = {"Length (um)": 1, "Run Length (um)": 0.5, "Slice_1": 220, "Slice_2": 35}
def main(dir_in="", dir_out_master="", label="", config="GUI", pre_set_id="",
         path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt",
         foo_threshold=foo_threshold_default, threshold=threshold_dafault):
    # Initialization
    first = True
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    # VI
    #id, dir_log, mock, dir_out = ml.initialize(path_log, label=label + "_VI", config=config,
    #                                           dir_in=dir_out_master, pre_set_id=pre_set_id)
    # R
    id2, dir_log2, mock2, dir_out2 = ml.initialize(path_log, label=label +"_R", config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    # Directory for the table of n
    id3, dir_log3, mock3, dir_out3 = ml.initialize(path_log, label=label + "_n", config=config,
                                            dir_in=dir_out_master, pre_set_id=pre_set_id)
    dir_name_input = os.path.basename(dir_in)
    dir_in_sep = dir_name_input.split("_")
    dir_in_abb = dir_in_sep[0] + "_" + dir_in_sep[1]
```

```
    # Print the designated thresholds and save them as a file in the analysis directory
    index = ["Length (um)", "Run Length (um)", "Slice_1", "Slice_2"]
    print("¥n** Selection by **")
    print("{} > {} (vi)   or   {} > {} (r)".format(index[0], threshold[index[0]], index[1],
                                                  threshold[index[1]]))  # Length
    print("Intensity ({}) < {}".format(index[2], threshold[index[2]]))  # Int Slice 1
    print("Intensity ({}) > {}".format(index[3], threshold[index[3]]))  # Int Slice 2
    print("** ** ** ** ** **")
    sr_threshold = pd.Series(threshold, index=index)
    sr_threshold.to_csv("{}¥¥{}_threshold.csv".format(dir_out_master, label))

    # Process each file in the directory selected using GUI
    items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)  # Get files from the
designated directory
    for file in items:
        data_src = pd.read_csv(dir_in + "¥¥" + file,
                               engine="python")  # "engine=python" avoids an error.
        file_ret = file.split("_")
        file_core = file_ret[0] + "_" + file_ret[1] + "_" + file_ret[2]

        # Selection by length etc.
        data_tmp_r, sr_n_r = foo_threshold(data_src, setting="vir", file=file_core,
threshold=threshold)
        if first:  # First process
            #df_n_vi = sr_n_vi    # VI の n の table
            df_n_r = sr_n_r    # R の n の table
            #print("First: ¥n", df_n_vi)
            first = False
        else:    # Concat the dataframe
            #df_n_vi = pd.concat([df_n_vi, sr_n_vi], axis=1)
            df_n_r = pd.concat([df_n_r, sr_n_r], axis=1)
            #print(">2: ¥n", df_n_vi)

        # Save data after secelction
        #data_tmp_vi.to_csv("{}¥¥{}_Master_{}_Work_{}_vi_filter.csv".format(dir_out, file_core,
dir_in_abb, id))
        data_tmp_r.to_csv("{}¥¥{}_Master_{}_Work_{}_r_filter.csv".format(dir_out2, file_core,
dir_in_abb, id2))

    # Save n before and after selection as a csv file
    print("¥nn (R):")
    display(df_n_r)
    #df_n_vi.to_csv("{}¥¥Master_{}_Work_{}_{}_vi_n.csv".format(dir_out3, dir_in_abb, id, label))
    df_n_r.to_csv("{}¥¥Master_{}_Work_{}_{}_r_n.csv".format(dir_out3, dir_in_abb, id2, label))

    print("¥n** {} Script Finished **".format(script_num))
    return dir_out2, dir_out3


if __name__ == "__main__":
    main()
```

py0056_8_csv_connector_for_0053_a_220125.py

```
#
# py0056:
# Concat csv files as a pre-processing for py0053 (box plot) and py0062 (jitter plot)
# Adaptable for both of single and double channels
#
#
# -*- coding: utf8 -*-

import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from IPython.display import display
import numpy as np
import py0052_7_my_library_a_220122 as ml
import importlib
import pprint

#importlib.reload(ml)


def main(dir_in="", dir_out_master="", label="", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt"):
    # Initialization
    # importlib.reload(ml)
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)

    # Process each file in the directory selected using GUI
    items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)  # Get files from the
designated directory

    # Concat data which meets a parameter of the follwoing list
    #columns = ["Intensity", "Velocity (um/s)", "Intensity in Slice1", "Intensity in Slice2",
"Length (um)"]
    columns = ["Mean intensity", "Velocity (um/s)", "Slice_1", "Slice_2", "Run Length (um)"]
    col_processed = []
    col_unprocessed = []
    for i in range(len(columns)):
        try:
            subscript(dir_in, items, columns, i, dir_out, id)
            col_processed.append(columns[i])
        except:
            col_unprocessed.append(columns[i])
    print("¥n* Processed columns:")
    pprint.pprint(col_processed)
    print("¥n* Unprocessed columns:")
    pprint.pprint(col_unprocessed)
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out
```

```
def subscript(dir_in, items, columns, i, dir_out, id):
    # Initialization
    file_date = ""
    dir_in_name = os.path.basename(dir_in)

    # Load each file, take data of a designated column and concat them
    for file in items:
        # Get date of a file name (for output)
        if not file_date:
            file_date = file.split("_")[0]

        # Load row data and take data of a designated column
        data_src = pd.read_csv(dir_in + "¥¥" + file)
        data_tmp = data_src[columns[i]]
        data_tmp.name = file.split("_")[2]

        # Concat data
        try:
            data_sum = pd.concat([data_sum, data_tmp], axis=1)
        except UnboundLocalError:
            data_sum = data_tmp

    # Sort and save the concated data
    data_sum.index.name = "Index"
    data_sum = data_sum.sort_index(axis=1, ascending=True)
    column_edt = columns[i].replace("/", "%")
    try:
        data_sum.to_csv(dir_out + "¥¥" + file_date + "_" + column_edt + "_" + id + "_column.csv")
    except:
        pass


if __name__ == "__main__":
    main()
```

py0062_7_jitter_plot_a_220125.py

```
#
# py0062:
# Make jitter plots
#
# -*- coding: utf8 -*-

import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import seaborn as sns
import importlib
import py0052_7_my_library_a_220122 as ml
from IPython.display import display
import subprocess

#importlib.reload(my_lib)

def main(dir_in="", dir_out_master="", label="jitter", config="GUI", pre_set_id="",
        path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", context="notebook"):
    # Initialize and select a directory for analysis
    # importlib.reload(ml)
    #config = "test"
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock
        print("dir_in: {}".format(dir_in))

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)  # Get files from the
designated directory
    for x in items:
        print("¥nFile:{}".format(x))
        ret = x.split("_")
        if not ret[1]=="Slice":
            second_key = ret[1]
        else:    # When the key is Slice_1 or Slice_2 (separated with "_")
            second_key = ret[1] + ret[2]
        data = pd.read_csv(dir_in + "¥¥" + x, engine="python")
        jitter_plot(data, dir_out, x, second_key, context)
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out


def jitter_plot(data_src, dir_out, file_name, second_key, context):
    # Initialization
    plt.close()
    #fig, ax = plt.subplots()
```

```
    #plt.tick_params(labelsize=10)

    # Arrange passed data
    data_tmp = data_src.drop("Index", axis=1)
    #data_dst = []
    for column in data_tmp.columns:
        length = len(data_tmp[column])
        list_sample = [column for i in range(length)]
        list_data = data_tmp[column].values
        data_tmp2 = {"Sample":list_sample, "Values":list_data}
        data_tmp3 = pd.DataFrame(data_tmp2)
        try:
            data_dst = pd.concat([data_dst, data_tmp3], ignore_index=True)
        except UnboundLocalError:
            data_dst = data_tmp3

    # Get statistics of the data
    statistics = data_tmp.describe()
    statistics.to_csv(dir_out + "¥¥" + file_name.replace(".csv", second_key +
"_description.csv"))

    # Plot
    sns.set_context(context)    # Font size: paper < notebook < talk < poster
    n = len(data_tmp.columns)
    if context=="notebook":
        fig = plt.figure(figsize=(n, 4))
    else:
        fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    sns.boxplot(x='Sample', y='Values', data=data_dst, showfliers=False, ax=ax)
    sns.stripplot(x='Sample', y='Values', data=data_dst, jitter=True, color='black', ax=ax)
    #bp = ax.boxplot(data_dst, whis="range")    # whis="range": outlier
    #ax.set_xticklabels(data_tmp.columns)

    # Arrange appearance of the graph
    if second_key=="Slice1":
        ylabel = "Intensity (Cy3)"
    elif second_key=="Slice2":
        ylabel = "Intensity (A647)"
    elif second_key=="Velocity (um%s)":
        ylabel = "Velocity (μm/s)"
    elif second_key=="Run Length (um)":
        ylabel = "Run Length (μm)"
    else:
        ylabel = second_key
    plt.xlabel("")
    plt.ylabel(ylabel)
    plt.grid()
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the upper frame border
    plt.tight_layout()

    # Save
    output = "{}¥¥{}".format(dir_out, file_name.replace(".csv", "_" + second_key))
```

```
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    #plt.savefig(dir_out + "¥¥" + file_name.replace(".csv", "_" + second_key + ".png"))
    plt.show()
    plt.close()

if __name__ == '__main__':
    main()
```

py0067_3_csv_concater_by_2nd_file_key_a_220126.py

```
#
# py0067:
# Concat files which has the same key on the second key separated by "_" in their file names
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from IPython.display import display
import numpy as np
import py0052_7_my_library_a_220122 as ml

def main(dir_in="", dir_out_master="", label="", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt"):
    # Initialize and select a directory for analysis
    # importlib.reload(ml)
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)

    # Select files from the designated directory and process each file
    csvs = ml.get_sublayer_items(dir_in, ".csv", visualize=False)

    # Separate the file name by "_" and assign the second (counted from 0) str as a key.
    keys = [csv.split("_")[2] for csv in csvs]
    keys_unique = list(set(keys))
    print("¥nUnique keys: {}".format(keys_unique))
    keys_processed = []
    for key in keys_unique:
        print("¥nKey: {}".format(key))
        keys_processed = sub_script(dir_in, key, csvs, id=id, dir_out=dir_out,
keys_processed=keys_processed)

    # Logging
    keys_unprocessed = keys_unique  # Detect unprocessed keys
    for key_processed in keys_processed:
        keys_unprocessed.remove(key_processed)
    print("¥n* Processed keys: {}".format(keys_processed))
    print("* Unprocessed keys: {}".format(keys_unprocessed))
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out

def sub_script(dir_in, key, files, id, dir_out, keys_processed):
    # Initialization
    dir_name_input = os.path.basename(dir_in)
    file_date = ""

    for csv in files:
```

```
        #print(csv)
        file_ret = csv.split("_")
        key_tmp = file_ret[2]
        if key_tmp == key:
            # Load source data
            print("Matched file: {}".format(csv))
            data_src = pd.read_csv(dir_in + "¥¥" + csv)

            # Drop unrelated columns from the source data
            for column_candidate in [" ", "Unnamed: 0"]:
                try:
                    data = data.drop(column_candidate, axis=1)
                except:
                    pass

            # Concat data (pd.concat)
            if not file_date:    # The first process using the key
                data_dst = data_src  # data_dst is generated as a first data
                file_date = file_ret[0]    # Load data of the zeroth column
                key_1 = file_ret[1].split("-")[0]
                key_2 = file_ret[2]
                keys_processed.append(key_tmp)
            else:              # Second or more processes
                data_dst = pd.concat([data_dst, data_src], ignore_index=True)    # Concat

    # Log and Save
    data_dst.to_csv(dir_out + "¥¥" + file_date + "_" + key_1 + "_" + key_2 + "_Master_" +
dir_name_input + "_Work_" + id + "_concat.csv")
    return keys_processed


if __name__ == "__main__":
    main()
```

```
#
# py0116:
# Make histograms of velocity, fluorescence intensities and run length from data generated by
py0060, py0059 or py0064
# Applicable for both data of 1 and 2 channel(s).
#
# -*- coding: utf8 -*-

# Import modules/scripts
import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
#from pandas.tools.plotting import scatter_matrix
import seaborn as sns
import pprint
import subprocess
from IPython.display import display

# Number (150 mM NaCl) 用
xrange = [(-30, 300), (0, 2.5), (-40, 500), (-20, 300), (0, 20)]
# Sx4 用
#xrange = [(-30, 300), (0, 1), (-40, 700), (-20, 500), (0, 20)]
# Sx2 用
#xrange = [(-30, 300), (0, 1.5), (-40, 500), (-20, 400), (0, 20)]
# free kinesin用
#xrange = [(-30, 300), (0, 2.6), (-40, 500), (-20, 400), (0, 20)]
# monomer_layout_Sx4
#xrange = [(-30, 300), (0, 0.8), (-40, 200), (-20, 100), (0, 20)]
# monomer_layout_Sx2
#xrange = [(-30, 300), (0, 0.4), (-20, 300), (-20, 100), (0, 20)]
# monomer_number
#xrange = [(-30, 300), (0, 0.4), (-20, 300), (-20, 100), (0, 20)]

#def main(dir_in="", dir_out_master="", label="hist", config="test", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", fontsize=20, matrix=False, xrange=xrange):
def main(dir_in="", dir_out_master="", label="hist", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", fontsize=20, matrix=True, xrange=xrange):
    # 初期化および処理データの選択 (my library の initialize 関数を用いる)
    # importlib.reload(ml)
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock

    # Settings
    # columns = ["Intensity", "Velocity (um/s)", "Intensity in Slice1", "Intensity in Slice2",
"Length (um)"]
```

```
    columns = ["Intensity", "Velocity (um/s)", "Slice_1", "Slice_2", "Run Length (um)"]
    xlabels = ["Intensity", "Velocity (μm/s)", "Intensity (Cy3)", "Intensity (A647)", "Run Length
(μm)"]
    width_bins = [10, 0.02, 10, 10, 2]
    # width_bins = [10, 0.1, 20, 10, 2]
    #width_bins = [10, 0.02, 10, 10, 1]

    # Dataframes for summarizing data
    df_int = pd.DataFrame()
    df_vel = pd.DataFrame()
    df_run_len = pd.DataFrame()
    df_slice1 = pd.DataFrame()
    df_slice2 = pd.DataFrame()
    list_df_sum = [df_int, df_vel, df_slice1, df_slice2, df_run_len]

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)
    for x in items:
        # Initialize and load source data
        print("¥nFile:{}".format(x))
        ret = x.split("_")
        x_core = "{}_{}_{}".format(ret[0], ret[1], ret[2])
        data = pd.read_csv(dir_in + "¥¥" + x, engine="python")

        # Drop unrelated columns from the source data
        for column_candidate in [" ", "Unnamed: 0", "Unnamed: 0.1", "File", "Index", "x", "y",
"Width (pixel)", "Height (pixel)",
                                 "SD_Slice_1", "SD_Slice_2", "Mock SD_Slice_1", "Mock SD_Slice_2",
"x_run", "y_run",
                                 "Width_run (pixel)", "Height_run (pixel)", "Run Time (s)", "Full_x",
"Full_y"]:
            #data = data.drop(column_candidate, axis=1)
            try:
                data = data.drop(column_candidate, axis=1)
            except:
                pass

        if matrix:
            # Make a matrix with pairplot
            context = "talk"  # 文字の大きさ paper < notebook < talk < poster
            sns.set_context(context)
            sns.plot_kws = {"s": 10}
            sns.pairplot(data, plot_kws={'alpha': 0.4})
            sns.plot_kws = {"s": 10}
            plt.gca().spines['right'].set_visible(False)    # 右の枠線を消す
            plt.gca().spines['top'].set_visible(False)    # 上の枠線を消す
            plt.tight_layout()

            # Save
            output = "{}¥¥{}_{}_matrix".format(dir_out, x_core, id)
            plt.savefig(output + ".png")  # png として保存
            plt.savefig(output + ".svg")  # emf として直接出力できないため、一度 svg として保存
            subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)  #
inkscape を用いて svg を emf に変換
```

```
            #plt.savefig(dir_out + "¥¥" + x.replace(".csv", "_matrix.png"))
            plt.show()
            plt.close()

        #
        #  Set xrange of a histogram depending on data
        #
        # Number (150 mM NaCl) 用
        #xrange = [(-30, 300), (0, 2.5), (-40, 500), (-20, 300), (0, 20)]
        # Sx4 用
        #xrange = [(-30, 300), (0, 2.5), (-40, 500), (-20, 300), (0, 20)]
        #
        #xrange = [(-30, 300), (0, 2.2), (-40, 300), (-20, 200), (0, 20)]
        #xrange = [(-30, 300), (0, 1.2), (-40, 300), (-20, 200), (0, 20)]
        #bins = [33, 22, 17, 22, 10]

        # Make histograms
        bins = [int(round((xrange[i][1] - xrange[i][0])/width_bins[i], 0)) for i in
range(len(xrange))]  # xrange と bin 幅から、bin 数を計算
        print("¥nbins: {}".format(bins))
        #print("¥nbin: {}".format((xrange[1][1] - xrange[1][0])/width_bins[1]))
        #print("¥nbins: {}".format(bins))
        rnds = [1, 2, 0, 1, 2]    # Number of decimals for rounding off each parameter value. This
number is used to insert median etc in a graph.
        col_processed = []
        col_unprocessed = []
        for i in range(len(columns)):
            #make_histgram(data, columns[i], dir_out, x, xlabels[i], id=id, xrange=xrange[i],
bin=bins[i], rnd=rnds[i],
            #             fontsize=fontsize)
            try:
                sr_hist_tmp = make_histgram(data, columns[i], dir_out, x, xlabels[i], id=id,
xrange=xrange[i], bin=bins[i], rnd=rnds[i], fontsize=fontsize)
                sr_hist_tmp2 = sr_hist_tmp.rename(columns={"n":ret[2]})
                list_df_sum[i] = pd.concat([list_df_sum[i], sr_hist_tmp2], axis=1)
            except TypeError:
                col_unprocessed.append(columns[i])
            except KeyError:
                col_unprocessed.append(columns[i])
        print("¥n* Processed columns:")
        pprint.pprint(col_processed)
        print("* Unprocessed columns:")
        pprint.pprint(col_unprocessed)

    #for i in range(len(list_df_sum)):
    #    if i==0:
    #        continue
    #    master_histogram(list_df_sum[i], columns[i], dir_out, xlabels[i], id=id)

    print("¥n** {} Script Finished **".format(script_num))
    return dir_out


def make_histgram(data, column, output, file, xlabel, id, xrange=(), bin=40, rnd=0,
```

```
fontsize=20):
    '''
    A method to make a histogram and save it with designated setting.
    :param data: pd.DataFrame of source containing data for a histogram
    :param column: a colomn in param data for the histogram
    :param output: absolute path (str) of a directory to save results
    :param file: str to name result files
    :param xlabel: xlabel for the histogram
    :param id: str to name result files
    :param xrange: range of x-axis of the histogram
    :param bin: number of bins of the histogram
    :param rnd: list of numbers of decimals for rounding off [median, mean, std]. This number is
used to insert them in the histogram.
    :param fontsize: size of font printed on the histgram
    :return: pd.Series containing x and y values of the histogram
    '''
    # Initialization
    sr = data[column]
    plt.close()
    if column == "Slice_2":
        plt.figure(figsize=(6, 4))
        #plt.figure(figsize=(8, 4))
    elif column == "Intensity":
        plt.figure(figsize=(6, 4))
    else:
        plt.figure(figsize=(5, 4))
    plt.tick_params(labelsize=fontsize)
    column_edt, xlabel_edt = column.replace("/", "_").replace(" ", "_"), xlabel.replace("/", "_")
    dir_out_sub = output + "¥¥" + column_edt    # Data are saved in different directories
depending on columns
    try:
        os.makedirs(dir_out_sub)
    except:
        pass
    color = reference(file.split("_")[2])
    file_ret = file.split("_")
    file_core = "{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2])
    if rnd == 0:
        fnc = int
    else:
        fnc = str

    # Main processing (make a histogram)
    #sr = data[column].dropna()
    statistics = sr.describe()
    text = "¥nn={}¥nMedian={}¥nMean={}¥nSD={}". ¥
        format(int(statistics["count"]), fnc(round(statistics["50%"], rnd)),
               fnc(round(statistics["mean"], rnd)), fnc(round(statistics["std"], rnd)))
    print("¥nStatistic parameters in {}:{}".format(column, text))
    statistics.to_csv("{}¥¥{}_{}_hist_statistics.csv".format(dir_out_sub, file_core, id))
    if xrange:
        n, bins, patches = plt.hist(sr, bins=bin, range=xrange, color=color)
        xmax = np.amax(xrange)
    else:
```

```
        n, bins, patches = plt.hist(sr, bins=bin, color=color)
    bins_del = np.delete(bins, -1, 0)
    sr_hist = pd.Series(n, index=bins_del)
    sr_hist.index.name = "bins"
    sr_hist.name = "n"
    display(sr_hist)
    sr_hist.to_csv("{}¥¥{}_{}_hist_dataset.csv".format(dir_out_sub, file_core, id), header=True)

    # Arrange appearance of the graph
    ymax = np.amax(n)
    plt.xlabel(xlabel, fontsize=fontsize)
    plt.ylabel("Number", fontsize=fontsize)
    print("ymax: {}".format(ymax))
    print("res: {}".format((ymax //10) *5))
    #xtick = (0, 0.2, 0.4)
    #ytick = range(0, int(ymax + 2), int(((ymax+1) // 10) *5))
    #print("Tick: {}".format(ytick))
    #if column == "Intensity":
    #    plt.yticks(ytick)
    #elif column == "Velocity (um/s)":
    #    plt.xticks(xtick)
    #ax.yaxis.set_major_locator(MultipleLocator(1))
    #plt.ylim([0, ymax *1.5])
    point = {'start': [statistics["50%"], ymax*1.3], 'end': [statistics["50%"], ymax*1.1]}
    #if column == "Velocity (um/s)" or column == "Slice_1" or column =="Intensity":
    #    plt.annotate("", xy=point["end"], xytext=point["start"],
    #            arrowprops=dict(shrink=0, width=1, headwidth=8,
    #                headlength=10, connectionstyle='arc3', facecolor='r', edgecolor='r'))
    plt.text(xrange[1]/2, ymax/2, text, size=fontsize/5*4)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    plt.tight_layout()

    # Save
    output = "{}¥¥{}_{}".format(dir_out_sub, file_core, id)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)  #
inkscape を用いて svg を emf に変換
    #subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)  #
inkscape を用いて svg を emf に変換
    #plt.savefig(dir_out_sub + "¥¥" + file.replace(".csv", "_" + column_edt + "_" + xlabel_edt +
".png"))
    plt.show()
    plt.close()
    return sr_hist

def master_histogram(df, column, output, xlabel, id, rnd=0, fontsize=20):
    # Initialization
    plt.close()
    if column == "Slice_2":
        plt.figure(figsize=(6, 4))
        #plt.figure(figsize=(8, 4))
    elif column == "Intensity":
```

```
        plt.figure(figsize=(6, 4))
    else:
        plt.figure(figsize=(5, 4))
    plt.tick_params(labelsize=fontsize)
    column_edt, xlabel_edt = column.replace("/", "_").replace(" ", "_"), xlabel.replace("/", "_")
    dir_out_sub = output + "¥¥" + column_edt    # カラム毎に別の directory に保存する
    color = reference(column)
    if rnd == 0:
        fnc = int
    else:
        fnc = str

    # Main processing
    #sr = data[column].dropna()
    display(df)
    statistics = df.describe()
    statistics.to_csv("{}¥¥{}_{}_hist_master_statistics.csv".format(dir_out_sub, column_edt, id))
    for cl in df.columns:
        plt.plot(df.index, df[cl], color=color, label=cl)

    # Arrange appearance of the graph
    #ymax = np.amax(n)
    plt.xlabel(xlabel, fontsize=fontsize)
    plt.ylabel("Number", fontsize=fontsize)
    #print("ymax: {}".format(ymax))
    #print("res: {}".format((ymax //10) *5))
    xtick = (0, 0.2, 0.4)
    #ytick = range(0, int(ymax + 2), int(((ymax+1) // 10) *5))
    #print("Tick: {}".format(ytick))
    #if column == "Intensity":
    #    plt.yticks(ytick)
    #elif column == "Velocity (um/s)":
    #    plt.xticks(xtick)
    #ax.yaxis.set_major_locator(MultipleLocator(1))
    #plt.ylim([0, ymax *1.5])
    #point = {'start': [statistics["50%"], ymax*1.3], 'end': [statistics["50%"], ymax*1.1]}
    #if column == "Velocity (um/s)" or column == "Slice_1" or column =="Intensity":
    #    plt.annotate("", xy=point["end"], xytext=point["start"],
    #            arrowprops=dict(shrink=0, width=1, headwidth=8,
    #                headlength=10, connectionstyle='arc3', facecolor='r', edgecolor='r'))
    #plt.text(xrange[1]/2, ymax/2, text, size=fontsize/5*4)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    plt.tight_layout()

    # Save
    output = "{}¥¥{}_summary_{}".format(dir_out_sub, column_edt, id)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    plt.show()
    plt.close()

def reference(key):
```

```
    '''
    Return str of a color referring the passed key.
    :param key: t073, f004, f005, etc
    :return: str showing color
    '''
    # Initialization
    color = "k"

    # Return str of a color referring the passed key.
    if key=="t073" or key=="free" or key=="1xbb":
        color = "k"
    elif key=="f004" or key=="bgx1":
        color = "r"
    elif key=="f005" or key=="bgx2":
        color = "b"
    elif key=="x1-f005":
        color = "c"
    elif key=="x2-f005" or key=="plane":
        color = "g"
    elif key == "bleach":
        color = "y"
    return color

# For debug
def set_trace():
    from IPython.core.debugger import Pdb
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)

def debug(f, *arg, **kwargs):
    from IPython.core.debugger import Pdb
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *arg, **kwrgs)

if __name__ == "__main__":
    main()
```

```
#
# py0081:
# Called by py0075.
# Concat columns of the same file_core from a table of n.
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
from IPython.display import display
import numpy as np
import py0052_7_my_library_a_220122 as ml

def main(dir_in="", dir_in_n="",dir_out_master="", label="", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt"):
    # Initialize and select a directory for analysis
    # importlib.reload(ml)
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)

    # Select files from the designated directory
    csvs = ml.get_sublayer_items(dir_in, ".csv", visualize=False)
    for file in csvs:
        print("¥n** File: {}".format(file))
        sub_script(dir_in, file, id, dir_out)

    print("¥n** {} Script Finished **".format(script_num))
    return dir_out

def sub_script(dir_in, file, id, dir_out):
    # Initialization
    dir_name_input = os.path.basename(dir_in)

    # Load source data
    data_src = pd.read_csv(dir_in + "¥¥" + file, index_col=0)
    #print("¥ndata_src")
    #display(data_src)

    # Drop unrelated columns from the source data
    #for column_candidate in [" ", "Unnamed: 0"]:
    #    try:
    #        data_src = data_src.drop(column_candidate, axis=1)
    #    except:
    #        pass

    # Assign the second (counted from "0") str separated by "_" in their column name, as a key
    columns = data_src.columns
```

```
#print("[Log] column name: {}".format(columns))
keys = [column.split("_")[2] for column in columns]
keys_unique = list(set(keys))
#print("¥nUnique keys: {}".format(keys_unique))
keys_processed = []

first = True  # 初期化
for key in keys_unique:
    # Initialization
    file_date = ""
    #print("¥nKey: {}".format(key))

    # Main process
    for column in columns:
        # Load data of the designated column
        data_tmp = data_src[column]
        file_ret = column.split("_")
        key_tmp = file_ret[2]

        # Process data when the key matches to a reference.
        if key_tmp == key:
            #print("Matched column: {}".format(column))
            # Concat data (pd.concat)
            if not file_date:  # The first process of the key
                sr_key = data_tmp  # sr_key is generated as the first data
                #print("¥n1:")
                #display(sr_key)
                file_date = file_ret[0]  # Load the zeroth str from the file name
                key_1 = file_ret[1].split("-")[0]
                key_2 = file_ret[2]
                keys_processed.append(key_tmp)
            else:  # Second or more processes
                sr_key += data_tmp    # Add n of the same key
                #print("¥n>2:")
                #display(sr_key)
                sr_key.name = "{}_{}_{}".format(file_date, key_1, key_2)

    # Concat unique columns
    if first:
        table_dst = sr_key
        #print("¥nFirst:")
        #display(table_dst)
        first = False
    else:
        table_dst = pd.concat([table_dst, sr_key], axis=1)
        #print("2>:")
        #display(table_dst)

# Log and save
keys_unprocessed = keys_unique  # Detect unprocessed keys
for key_processed in keys_processed:
    keys_unprocessed.remove(key_processed)
print("¥n* Processed keys: {}".format(keys_processed))
print("* Unprocessed keys: {}".format(keys_unprocessed))
```

```
    file_out = file.replace(".csv", "_{}_concat.csv".format(id))
    table_dst = table_dst.sort_index(axis=1)
    display(table_dst)
    table_dst.to_csv("{}¥¥{}".format(dir_out, file_out))
    #return keys_processed, table_dst


if __name__ == "__main__":
    main()
```

```
#
# py0043_run_length_analyzer_e
# Make graphs of run lengths with fitting curve from csv file generated by py0042 etc.#
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from scipy.optimize import curve_fit
import pprint
import subprocess

def main(dir_in="", dir_out_master="", label="non-o-fit", config="GUI", pre_set_id="",
        path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", markersize=10):
    # Initialize and select a directory for analysis
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, "csv", visualize=False)
    dir_processed = []
    dir_unprocessed = []
    for x in items:
        try:
            sub_script(dir_in, x, id, dir_log, dir_out, markersize)
            dir_processed.append(x)
        except RuntimeError:
            dir_unprocessed.append(x)
        except FileNotFoundError:
            dir_unprocessed.append(x)
        except TypeError:
            dir_unprocessed.append(x)
    print("¥n* Processed directories:")
    pprint.pprint(dir_processed)
    print("¥n* Unprocessed directories:")
    pprint.pprint(dir_unprocessed)
    #dir_out = input + id + "¥¥"  # Directory for result files
    #try:
    #    os.makedirs(dir_out)  # Make a directory for result files
    #except:
    #    pass
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out
```

```
def sub_script(input, file, id, dir_log, dir_out, markersize):
    data_row = pd.read_csv(input + "¥¥" + file)
    # print(data_row)
    # data_row["Velocity' (um/s)"] = data_row["Length (um)"]/data_row["Time (s)"]
    color= reference(file.split("_")[2])
    data_tmp1 = data_row.sort_values(by="Run Length (um)")
    x = data_tmp1["Run Length (um)"].values
    n = x.shape[0]
    y = np.arange(1, n+1, 1)/n
    index = ["Run Length (μm)", "Cumuative Prob."]
    data_dst = pd.DataFrame([x, y], index=index).T
    print("¥n¥ndata_dst is,¥n", data_dst)
    x_c, amp = fitting_exp(x, y)  # f(x) = 1 - b * exp(-x/x_c)
    sr_fit = pd.Series([x_c, amp], index=["x_c", "amp"])  # Arrange data to pd.Series
    sr_fit_name = file.replace(".csv", "_" + id + "_fitting_parameters.csv")
    sr_fit.to_csv(dir_out + "¥¥" + sr_fit_name)  # save the data
    x_fit = np.array(range(int(30 * 100))) / 100

    # Plot
    theoretical_values_row = 1 -1 * amp * np.exp(-1 * x_fit / x_c)
    data_fit_row = pd.DataFrame([x_fit, theoretical_values_row], index=index).T
    make_plot(data_dst, data_fit_row, columns=index, dir_out=dir_out, id=id,
            file=file, label="integral", range=(0, 30), x_c=x_c, markersize=markersize)


def make_plot(data, data_fitting, columns, dir_out, file, label, id, range=(), x_c=0,
markersize=10):
    '''
    A function to plot data with fitting curve, with designated settings of xrange, markersize,
etc.
    :param data: pd.DataFrame containing data for plot
    :param data_fitting: pd.DataFrame containing data for fitting
    :param columns: [x, y] used for plot and fitting
    :param dir_out: absolute path of a directory to save result files
    :param file: str of file name as a reference of result file names
    :param label: str which will be added to result file names as a label
    :param xrange: [min, max] of x range
    :param x_c: str of mean run length which will be printed in the graph
    :param markersize: int of markersize in the graph
    :param fontsize: int of fontsize in the graph
    :return: --
    '''
    # Initialization
    plt.close()
    plt.tick_params(labelsize=25)
    print("¥n¥n", columns)
    print("¥nn =", data[columns].count())
    color= reference(file.split("_")[2])
    file_ret = file.split("_")
    file_core = "{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2])

    # Plot
    statistics = data[columns[1]].describe()
```

```
    statistics.to_csv(dir_out +  "¥¥" + file.replace(".csv", label + "_description.csv"))
    plt.scatter(data[columns[0]], data[columns[1]], color=color, s=markersize)
    if range:
        plt.xlim(range[0], range[1])
    else:
        pass
    plt.plot(data_fitting[columns[0]].values, data_fitting[columns[1]].values, color="g")

    # Arrange appearance of the graph
    text = "¥n{} μm".format(round(x_c, 1))
    ymax = np.amax(data[columns[1]])
    plt.text(range[1] / 2, ymax / 2, text, size=25 / 5 * 4)
    #plt.text(range[1] / 2, ymax / 2, text, size=fontsize / 5 * 4)
    plt.xlabel(columns[0], fontsize=25)
    plt.ylabel(columns[1], fontsize=25)
    plt.xlim(0, 30)
    plt.ylim(0, 1.1)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    plt.tight_layout()

    # Save
    output = "{}¥¥{}".format(dir_out, file_core)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    #plt.savefig(dir_out +  "¥¥" + file.replace(".csv", label + ".png"))
    plt.show()
    plt.close()

def reference(key):
    '''
    Return str showing a color depending on a designated key
    :param key: t073, f004, f005, etc
    :return: str (color),
    '''
    color = "k"
    if key == "t073":
        color = "k"
    elif key == "f004":
        color = "r"
    elif key == "f005":
        color = "b"
    elif key == "x2-f005":
        color = "k"
    return color


def fitting_exp(x, y):
    '''
    A function to fit data with a function.
    :param x: x for the function
    :param y: y for the function
    :return: (a, b) calculated by fitting
```

```
    '''
    betaID, cov = curve_fit(exp_curve, x, y)
    a = betaID[0]*betaID[0]    # betaID[0] is the positive root of the solved value in order to
limit the value within positive one.
    b = betaID[1]
    print("¥n¥nParameters are solved,¥n a=", a, "¥nb =", b)    # Logging
    return a, b


def exp_curve(x, x_c_root, amp):
    '''
    A function passed as a first parameter of scipy.curve_fit. The function formula is:
    y = 1 -b * exp(-x/x_c).
    Here, A is an amplitude and x_c is mean x (positive).
    :param x: values of run length of particles
    :param x_c_root: sqrt of mean run length which will be calculated
    :param amp: an amplitude of the function, that will be calculated
    :return: f = 1 -b * exp(-x/x_c)
    '''
    f = 1 -1 * amp * np.exp(-1 *x /x_c_root /x_c_root)    # (1/x_c_root)**2 is used to limit the
x_c value within positive values
    return f

if __name__ == "__main__":
    main()
```

```
#
# py0086
# Analyze data of run lengths using bootstrapping and determine the SD.
#
# -*- coding: utf8 -*-


import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from scipy.optimize import curve_fit
from sklearn.utils import resample
import pprint
from IPython.display import display
import subprocess

def main(dir_in="", dir_out_master="", label="bootstrap_non-o-fit", config="GUI", pre_set_id="",
        path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", markersize=10):
    # Initialization
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, "csv", visualize=False)
    dir_processed = []
    dir_unprocessed = []
    list_row_fit = []
    list_bs_statistics = []
    for x in items:
        #try:
            sr_fit_tmp, statistics_tmp = sub_script(dir_in, x, id, dir_log, dir_out, markersize)
            list_row_fit.append(sr_fit_tmp)
            list_bs_statistics.append(statistics_tmp)
            dir_processed.append(x)
        #except RuntimeError:
        #    dir_unprocessed.append(x)
        #except FileNotFoundError:
        #    dir_unprocessed.append(x)
        #except TypeError:
        #    dir_unprocessed.append(x)
    print("¥n* Processed directories:")
    pprint.pprint(dir_processed)
    print("¥n* Unprocessed directories:")
    pprint.pprint(dir_unprocessed)
```

```
    # Concat the statistics
    first = True
    for i in range(len(list_row_fit)):
        if first:
            table_row_dst = list_row_fit[i]
            table_bs_dst = list_bs_statistics[i]
            # print("¥nFirst:")
            # display(table_dst)
            first = False
        else:
            table_row_dst = pd.concat([table_row_dst, list_row_fit[i]], axis=1)
            table_bs_dst = pd.concat([table_bs_dst, list_bs_statistics[i]], axis=1)
    table_row_dst.T.to_csv("{}¥¥{}_{}_row_fitting_parametes.csv".format(dir_out, label, id))
    table_bs_dst.T.to_csv("{}¥¥{}_{}_bootstrap_statistics.csv".format(dir_out, label, id))
    print("¥n* ------------ Summary ------------- *")
    display(table_row_dst.T)
    display(table_bs_dst.T)
    #dir_out = input + id + "¥¥"  # Directory for output
    #try:
    #    os.makedirs(dir_out)  # Make a directory for output
    #except:
    #    pass
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out

def sub_script(input, file, id, dir_log, dir_out_master, markersize):
    # Initialization
    print("¥n* {}".format(file))
    file_ret = file.split("_")
    file_core = "{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2])
    dir_out = "{}¥¥{}".format(dir_out_master, file_core)
    #dir_out = dir_out_master
    os.makedirs(dir_out)

    # Load source data
    data_row = pd.read_csv(input + "¥¥" + file)
    data_tmp1 = data_row.sort_values(by="Run Length (um)")
    row_x = data_tmp1["Run Length (um)"].values
    n = row_x.shape[0]
    y = np.arange(1, n + 1, 1) / n
    index = ["Run Length (μm)", "Cum. Prob."]
    #index = ["Run Length (μm)", "Cumuative Prob."]
    # print("¥n¥nValues are,¥n", x, "¥n¥nn =", x.shape[0])
    # print("¥n¥ny=,¥n", y, "¥n¥nn =", y.shape[0])

    #
    # Process row data and calculate parameters by fitting
    #
    data_dst = pd.DataFrame([row_x, y], index=index).T
    print("¥n¥ndata_dst:")
    display(data_dst)
    data_dst.to_csv("{}¥¥{}_{}_row_dataset.csv".format(dir_out, file_core, id))
    x_c, amp = fitting_exp(row_x, y)  # f(x) = 1 - b * exp(-x/x_c)
    sr_fit_row = pd.Series([x_c, amp], index=["x_c", "amp"])  # Arrange data to pd.Series
```

```
    sr_fit_row.name = file_core
    sr_fit_row.to_csv("{}¥¥{}_{}_row_fitting_parameters.csv".format(dir_out, file_core, id))
    x_fit = np.array(range(int(30 * 100))) / 100

    # Standard
    theoretical_values_row = 1 - 1 * amp * np.exp(-1 * x_fit / x_c)
    data_fit_row = pd.DataFrame([x_fit, theoretical_values_row], index=index).T
    data_fit_row.to_csv("{}¥¥{}_{}_row_fitting_curve_value.csv".format(dir_out, file_core, id))
    make_plot(data_dst, data_fit_row, columns=index, dir_out=dir_out,
            file=file_core, label="{}_row".format(id), xrange=(0, 30), x_c=x_c,
markersize=markersize)

    #
    # Bootstrap sampling
    #
    #K = 3  # for test
    K = 200    # Number of re-sampling
    SIZE = row_x.shape[0]    # Size of re-sampled data
    list_fit = []
    for i in range(K):
        resampled_x = resample(row_x, n_samples=SIZE)
        sorted_x = np.sort(resampled_x)
        print("¥n------%d resample ------" % i)
        #print(resampled_x)
        #print(sorted_x)
        data_dst = pd.DataFrame([sorted_x, y], index=index).T
        #print("¥n¥ndata_dst is,¥n", data_dst)
        x_c, amp = fitting_exp(sorted_x, y)  # f(x) = 1 - b * exp(-x/x_c)
        sr_fit = pd.Series([x_c, amp], index=["x_c", "amp"])  # Arrange data to pd.Series
        sr_fit.name = i
        list_fit.append(sr_fit)
        x_fit = np.array(range(int(30 * 100))) / 100

        # Standard
        theoretical_values_row = 1 - 1 * amp * np.exp(-1 * x_fit / x_c)
        data_fit_row = pd.DataFrame([x_fit, theoretical_values_row], index=index).T
        #make_plot(data_dst, data_fit_row, columns=index, output=dir_out,
        #         file=file_core, label="integral_{}".format(i), xrange=(0, 30), x_c=x_c,
markersize=markersize)

    # Concat the statistics
    first = True
    for i in range(len(list_fit)):
        if first:
            table_dst = list_fit[i]
            # print("¥nFirst:")
            # display(table_dst)
            first = False
        else:
            table_dst = pd.concat([table_dst, list_fit[i]], axis=1)
    table_dst = table_dst.T
    table_dst.to_csv("{}¥¥{}_{}_bootstrap_fitting_parameters.csv".format(dir_out, file_core, id))
    print("¥n ----------- Fitting parametes ------------- ")
    display(table_dst)
```

```
    # Plot the results of bootstrapping
    bs_x_c = table_dst["x_c"].values
    statistics_tmp = make_histgram(bs_x_c, dir_out=dir_out, file=file_core,
label="{}_bootstrap".format(id), xrange=(0, 10), bins=40)
    return sr_fit_row, statistics_tmp


def make_histgram(data, dir_out, file, label, xrange=(), bins=20, fontsize=20):
    '''
    A function to make a histogram with designated settings of xrange, bins, fontsize etc.
    :param data: pd.DataFrame for histogram
    :param dir_out: absolute path of a directory to save result files
    :param file: str of file name as a reference of result file names
    :param label: str which will be added to result file names as a label
    :param xrange: [min, max] of x range
    :param bins: int of bin width
    :param fontsize: int of fontsize in the graph
    :return: pd.Series containing the analyzed statistics
    '''
    # Initialization
    plt.close()
    plt.figure(figsize=(5, 4))
    plt.tick_params(labelsize=fontsize)

    # Main processing
    #statistics = data[column1].describe()
    #text = "¥n{} μm".format(round(1 / a, 1))
    statistics = pd.Series(data.ravel()).describe()
    statistics.name = file
    text = "¥nMean {}¥nSD {}".format(round(np.mean(data), 2), round(np.std(data), 2))
    print("¥n-------------- Bootstrap summary ---------------")
    print(statistics)
    #print("Statistic parameters in {}".format(text))
    statistics.to_csv("{}¥¥{}_{}_discription.csv".format(dir_out, file, label))
    if xrange:
        ret=plt.hist(data, bins=bins, range=xrange, color="k")
    else:
        ret=plt.hist(data, bins=bins, color="k")
    #plt.plot(data_fitting[columns2[0]].values, data_fitting[columns2[1]].values, color="g")

    # Arrange appearance of the graph
    ymax = np.amax(ret[0])
    #plt.text(range[1] / 1.5, ymax / 1.5, text, size=25 / 5 * 4)
    point = {'start': [np.mean(data), ymax * 1.3], 'end': [np.mean(data), ymax * 1.1]}
    plt.annotate("", xy=point["end"], xytext=point["start"],
                arrowprops=dict(shrink=0, width=1, headwidth=8,
                                headlength=10, connectionstyle='arc3', facecolor='r', edgecolor='r'))
    plt.xlabel("l_c (μm)", fontsize=fontsize)
    plt.ylabel("Counts", fontsize=fontsize)
    plt.ylim(0, ymax*1.5)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    plt.tight_layout()
```

```
    # Save
    output = "{}¥¥{}_{}_bootstrap_hist".format(dir_out, file, label)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    plt.show()
    plt.close()
    return statistics

def make_plot(data, data_fitting, columns, dir_out, file, label, xrange=(), x_c=0,
markersize=10, fontsize=20):
    '''
    A function to plot data with fitting curve, with designated settings of xrange, markersize,
etc.
    :param data: pd.DataFrame containing data for plot
    :param data_fitting: pd.DataFrame containing data for fitting
    :param columns: [x, y] used for plot and fitting
    :param dir_out: absolute path of a directory to save result files
    :param file: str of file name as a reference of result file names
    :param label: str which will be added to result file names as a label
    :param xrange: [min, max] of x range
    :param x_c: str of mean run length which will be printed in the graph
    :param markersize: int of markersize in the graph
    :param fontsize: int of fontsize in the graph
    :return: --
    '''
    # Initialization
    plt.close()
    plt.figure(figsize=(5, 4))
    plt.tick_params(labelsize=fontsize)
    color= reference(file.split("_")[2])

    # Plot
    statistics = data[columns[1]].describe()
    statistics.to_csv("{}¥¥{}_{}_description.csv".format(dir_out, file, label))
    plt.scatter(data[columns[0]], data[columns[1]], color=color, s=markersize)
    if xrange:
        plt.xlim(xrange[0], xrange[1])
    else:
        pass
    plt.plot(data_fitting[columns[0]].values, data_fitting[columns[1]].values, color="g")

    # Arrange appearance of the graph
    text = "¥n{} μm".format(round(x_c, 1))
    ymax = np.amax(data[columns[1]])
    plt.text(xrange[1] / 2, ymax / 2, text, size=25 / 5 * 4)
    #plt.text(range[1] / 2, ymax / 2, text, size=fontsize / 5 * 4)
    plt.xlabel(columns[0], fontsize=fontsize)
    plt.ylabel(columns[1], fontsize=fontsize)
    plt.xlim(0, 21)
    plt.xticks([0, 10, 20])
    plt.ylim(0, 1.1)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
```

```
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    plt.tight_layout()

    # Save
    output = "{}¥¥{}_{}_fitting".format(dir_out, file, label)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    plt.show()
    plt.close()


def reference(key):
    '''
    Return str showing a color depending on a designated key
    :param key: t073, f004, f005, etc
    :return: str (color),
    '''
    color = "k"
    if key == "t073":
        color = "k"
    elif key == "f004":
        color = "r"
    elif key == "f005":
        color = "b"
    elif key == "x2-f005":
        color = "k"
    return color


def fitting_exp(x, y):
    '''
    A function to fit data with a function.
    :param x: x for the function
    :param y: y for the function
    :return: (a, b) calculated by fitting
    '''
    betaID, cov = curve_fit(exp_curve, x, y)
    a = betaID[0]*betaID[0]    # betaID[0] is the positive root of the solved value in order to
limit the value within positive one.
    b = betaID[1]
    print("¥n¥nParameters are solved,¥n a=", a, "¥nb =", b)    # Logging
    return a, b


def exp_curve(x, x_c_root, amp):
    '''
    A function passed as a first parameter of scipy.curve_fit. The function formula is:
    y = 1 -b * exp(-x/x_c).
    Here, A is an amplitude and x_c is mean x (positive).
    :param x: values of run length of particles
    :param x_c_root: sqrt of mean run length which will be calculated
    :param amp: an amplitude of the function, that will be calculated
    :return: f = 1 -b * exp(-x/x_c)
```

```
    '''
    f = 1 -1 * amp * np.exp(-1 *x /x_c_root /x_c_root)    # (1/x_c_root)**2 is used to limit the
x_c value within positive values
    return f

if __name__ == "__main__":
    main()
```

```
#
# py0087:
# Make a scatter plot with error bars. Bootstrap ver.
# This script is passed by py0075.
#
# -*- coding: utf8 -*-

# Import modules
import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from scipy.optimize import curve_fit
import matplotlib.ticker as ticker
import itertools
import math
import subprocess
from IPython.display import display

def main(dir_in="", dir_out_master="", label="scatter_graph", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", mode="r", ylim=(1.3, 2)):
#def main(dir_in="", dir_out_master="", label="scatter_graph", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", mode="v", ylim=(0.8, 10)):
    # Initialization
    #config = "test"
    #config = "GUI"
    #mode = "v"
    #ylim = (0.8, 10)
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock
        print("¥n****** Directly Run *******")
    #l_fontsize = [15, 20, 22, 24, 26, 28, 30]
    #l_legend = [True, False]

    subscript(dir_in, id, dir_out, fontsize=30, mode=mode, ylim=ylim)

    print("¥n** {} Script Finished **".format(script_num))
    return dir_out

def subscript(dir_in, id, dir_out, fontsize=30, mode="r", ylim=(1, 10)):
    # Initialization
    if mode=="r":
        ref = {"config":"r", "ylabel":"Run length (μm)"}
    if mode == "v":
        ref = {"config": "v", "ylabel": "Velocity (μm/s)"}
```

```
    #width = 0.3
    #fig = plt.figure()
    #ax = fig.add_subplot(1, 1, 1)
    plt.tick_params(labelsize=fontsize)
    #plt.hlines([1, 2], 0, 4, color="k", alpha=1, linestyles='dashed', linewidth=1)

    # For run lengths
    if mode=="r":
        # Load and concat data from files
        items_row = ml.get_sublayer_items(dir_in, "row_fitting_parametes.csv", visualize=False)
        items_std = ml.get_sublayer_items(dir_in, "statistics.csv", visualize=False)
        for i in range(len(items_row)):
            # Initialization
            file = items_row[i]
            file_ret = file.split("_")
            file_core = "{}_{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2], file_ret[4])
            print("¥nfile_core: {}".format(file_core))
            # Mean
            file_row_fit = items_row[i]
            data_row_fit = pd.read_csv("{}¥¥{}".format(dir_in, file_row_fit), index_col="Unnamed:
0")
            sr_mean = data_row_fit["x_c"]
            sr_mean.name = "mean"
            # Std
            file_bs_stat = items_std[i]
            data_bs_stat = pd.read_csv("{}¥¥{}".format(dir_in, file_bs_stat), index_col="Unnamed:
0")
            sr_std = data_bs_stat["std"]
            # Concat
            table_dst = pd.concat([sr_mean, sr_std], axis=1)

    # For velocities
    elif mode=="v":
        # Load data of velocities from different files
        list_sr_vel = []
        items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)
        for i in range(len(items)):
            # Initialization
            file = items[i]
            file_ret = file.split("_")
            file_core = "{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2])

            # Load source data and pick velocities
            data_row = pd.read_csv("{}¥¥{}".format(dir_in, items[i]))
            sr_tmp = data_row["Velocity (um/s)"]
            sr_tmp.name = file_core
            list_sr_vel.append(sr_tmp)

        # Concat data of velocities
        first = True
        for i in range(len(list_sr_vel)):
            if first:
                df_vel = list_sr_vel[i]
                first = False
```

```
            else:
                df_vel = pd.concat([df_vel, list_sr_vel[i]], axis=1)

        # Get mean ±SD
        df_tmp = df_vel.describe()
        table_dst = df_tmp[1:3].T

    #table_dst = table_dst.T
    print("¥nMean±SD:")
    #table_dst = table_dst.reindex(["190628_1_0k", "190628_2_5k", "190628_0_10k"])
    display(table_dst)
    table_dst.to_csv("{}¥¥{}_bootstrap_fitting_parameters.csv".format(dir_out, id))

    # Plot mean with error bars
    xlabels = []
    #xlabels = [table_dst.index]    #
    for i in range(len(table_dst.index)):
        file_core_tmp = table_dst.index[i]
        file_ret_tmp = file_core_tmp.split("_")
        key_tmp = file_ret_tmp[2]
        xlabels.append(key_tmp)
    y = table_dst["mean"].values
    x = np.array([i+1 for i in range(len(y))])
    #xlabels = table_dst.index
    se = table_dst["std"].values
    file_label = "{}_{}_{}_mean".format(file_ret[0], ref["config"], id)
    plot_scatter(x, y, se=se, xlabels=xlabels, ylabel=ref["ylabel"], dir_out=dir_out,
file_label=file_label, mode=mode, ylim=ylim)



def plot_scatter(x, y, dir_out, file_label="test", se=[], ylabel="", xlabels=["1,2,3,4",
"1,2,7,8", "1,3,5,7"], fontsize=20, mode="r", ylim=(1, 10)):
    '''
    A function to make a scatter plot with error bars.
    :param x: np.array of x
    :param y: np.array of y
    :param dir_out: absolute path of a directory for result files
    :param file_label: str which will be included to result files
    :param se: np.array of errors as error bars
    :param ylabel: str of ylabel of the graph
    :param xlabels: list of factors (xticks) on the x-axis
    :param fontsize: int of fontsize in the graph
    :param mode: "v" (velocity) or "r" (run length)
    :param ylim: [min, max] determining the range of y-axis
    :return: ---
    '''
    # Initialization
    plt.close()
    #plt.tick_params(labelsize=25)
    fig, ax = plt.subplots(figsize=(4.5, 4))
    plt.tick_params(labelsize=fontsize)
    # ax2 = ax1.twinx()
    str_legend = "x"
```

```
    # Plot
    plt.xlim([0, 4])
    if len(x)==4:
        y = [y[i] for i in (3, 0, 1, 2)]
        xlabels = [xlabels[i] for i in (3, 0, 1, 2)]
        se = [se[i] for i in (3, 0, 1, 2)]
        plt.xlim([0, 5])
    elif len(x)==2:
        plt.xlim([0, 3])
    ax.scatter(x, y, marker="s", color="k", s=fontsize)
    if len(se) != 0:
        ax.errorbar(x, y, yerr=se, fmt='none', ecolor="k", capsize=5)

    # Arrange appearance of the graph
    ax.set_ylabel(ylabel, fontsize=fontsize)
    plt.xticks(np.unique(x), xlabels)
    if mode=="r":
        plt.ylim([0, ylim[1]])
        ax.set_ylabel("Run length (μm)", fontsize=fontsize)
    elif mode=="v":
        plt.ylim([0, ylim[0]])
        ax.set_ylabel("Velocity (μm/s)", fontsize=fontsize)
    plt.gca().spines['right'].set_visible(False)
    plt.gca().spines['top'].set_visible(False)
    plt.tight_layout()

    # Save
    output = "{}¥¥{}_graph".format(dir_out, file_label)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    plt.show()
    plt.close()

# For debug
def set_trace():
    from IPython.core.debugger import Pdb
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)

def debug(f, *arg, **kwargs):
    from IPython.core.debugger import Pdb
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *arg, **kwrgs)

if __name__ == "__main__":
    main()
```

py0135_0_histgram_with_Gaussian_fitting_1_220208.py

```
#
# py0135:
# Make histograms of velocityfrom data generated by py0060, py0059 or py0064 and fit it with
Gaussian distribution
#
# -*- coding: utf8 -*-

# Import modules/scripts
import os, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import py0052_7_my_library_a_220122 as ml
import seaborn as sns
import pprint
import subprocess
from IPython.display import display

# Number (150 mM NaCl) 用
#xrange = [(-30, 300), (0, 2.5), (-40, 500), (-20, 300), (0, 20)]
# Sx4 用
#xrange = [(-30, 300), (0, 1), (-40, 700), (-20, 500), (0, 20)]
# Sx2 用
#xrange = [(-30, 300), (0, 1.5), (-40, 500), (-20, 400), (0, 20)]
# free kinesin用
#xrange = [(-30, 300), (0, 2.6), (-40, 500), (-20, 400), (0, 20)]
# monomer_layout_Sx4
#xrange = [(-30, 300), (0, 0.8), (-40, 200), (-20, 100), (0, 20)]
# monomer_layout_Sx2
#xrange = [(-30, 300), (0, 0.4), (-20, 300), (-20, 100), (0, 20)]
# monomer_number
xrange = [(-30, 300), (0, 0.4), (-20, 300), (-20, 100), (0, 20)]

#def main(dir_in="", dir_out_master="", label="hist_Gaussian", config="test", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", fontsize=20, matrix=False, xrange=xrange):
def main(dir_in="", dir_out_master="", label="hist_Gaussian", config="GUI", pre_set_id="",
path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", fontsize=20, matrix=True, xrange=xrange):
    # 初期化および処理データの選択 (my library の initialize 関数を用いる)
    # importlib.reload(ml)
    matrix = False
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    id, dir_log, mock, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id)
    if not dir_in:
        dir_in = mock

    # Settings
    # columns = ["Intensity", "Velocity (um/s)", "Intensity in Slice1", "Intensity in Slice2",
"Length (um)"]
    columns = ["Intensity", "Velocity (um/s)", "Slice_1", "Slice_2", "Run Length (um)"]
```

```
    xlabels = ["Intensity", "Velocity (μm/s)", "Intensity (Cy3)", "Intensity (A647)", "Run Length
(μm)"]
    width_bins = [10, 0.02, 10, 10, 2]    # monomer
    #width_bins = [10, 0.1, 20, 10, 2]    # free, number, dimer

    # Dataframes for summarizing data
    df_int = pd.DataFrame()
    df_vel = pd.DataFrame()
    df_run_len = pd.DataFrame()
    df_slice1 = pd.DataFrame()
    df_slice2 = pd.DataFrame()
    list_df_sum = [df_int, df_vel, df_slice1, df_slice2, df_run_len]
    list_param = []
    list_name = []

    # Select files from the designated directory and process each file
    items = ml.get_sublayer_items(dir_in, ".csv", visualize=False)
    for x in items:
        if True:    # Mock
            # Initialize and load source data
            print("¥nFile:{}".format(x))
            ret = x.split("_")
            x_core = "{}_{}_{}".format(ret[0], ret[1], ret[2])
            data = pd.read_csv(dir_in + "¥¥" + x, engine="python")

            # Drop unrelated columns from the source data
            for column_candidate in [" ", "Unnamed: 0", "Unnamed: 0.1", "File", "Index", "x", "y",
"Width (pixel)",
                                     "Height (pixel)",
                                     "SD_Slice_1", "SD_Slice_2", "Mock SD_Slice_1", "Mock SD_Slice_2",
"x_run", "y_run",
                                     "Width_run (pixel)", "Height_run (pixel)", "Run Time (s)",
"Full_x", "Full_y"]:
                # data = data.drop(column_candidate, axis=1)
                try:
                    data = data.drop(column_candidate, axis=1)
                except:
                    pass


            # Make histograms
            bins = [int(round((xrange[i][1] - xrange[i][0]) / width_bins[i], 0)) for i in
                    range(len(xrange))]  # xrange と bin 幅から、bin 数を計算
            print("¥nbins: {}".format(bins))
            # print("¥nbin: {}".format((xrange[1][1] - xrange[1][0])/width_bins[1]))
            # print("¥nbins: {}".format(bins))
            rnds = [1, 2, 0, 1,
                    2]  # Number of decimals for rounding off each parameter value. This number is
used to insert median etc in a graph.
            col_processed = []
            col_unprocessed = []
            for i in range(len(columns)):
                # make_histgram(data, columns[i], dir_out, x, xlabels[i], id=id, xrange=xrange[i],
bin=bins[i], rnd=rnds[i],
```

```
            #            fontsize=fontsize)
            #try:
            if i == 1:
                sr_hist_tmp, param_tmp = make_histgram(data, columns[i], dir_out, x, xlabels[i],
id=id, xrange=xrange[i],
                                       bin=bins[i], rnd=rnds[i], fontsize=fontsize)
                #sr_hist_tmp2 = sr_hist_tmp.rename(columns={"n": ret[2]})
                sr_hist_tmp2 = sr_hist_tmp.rename({"n": ret[2]})
                list_df_sum[i] = pd.concat([list_df_sum[i], sr_hist_tmp2], axis=1)
                list_param.append(param_tmp)
                list_name.append(ret[2])
            #except TypeError:
            #    col_unprocessed.append(columns[i])
            #except KeyError:
            #    col_unprocessed.append(columns[i])
        print("¥n* Processed columns:")
        pprint.pprint(col_processed)
        print("* Unprocessed columns:")
        pprint.pprint(col_unprocessed)

    # Save
    df_param = pd.DataFrame(list_param, index=list_name, columns=["amp", "mu", "sigma", "R^2"])
    path_param = "{}¥¥velocity_fitting_{}_params.csv".format(dir_out, id)
    df_param.to_csv(path_param, header=True)
    print("¥n** {} Script Finished **".format(script_num))
    return dir_out, path_param


def make_histgram(data, column, output, file, xlabel, id, xrange=(), bin=40, rnd=0,
fontsize=20):
    '''
    A method to make a histogram and save it with designated setting.
    :param data: pd.DataFrame of source containing data for a histogram
    :param column: a colomn in param data for the histogram
    :param output: absolute path (str) of a directory to save results
    :param file: str to name result files
    :param xlabel: xlabel for the histogram
    :param id: str to name result files
    :param xrange: range of x-axis of the histogram
    :param bin: number of bins of the histogram
    :param rnd: list of numbers of decimals for rounding off [median, mean, std]. This number is
used to insert them in the histogram.
    :param fontsize: size of font printed on the histgram
    :return: pd.Series containing x and y values of the histogram
    '''
    # Initialization
    sr = data[column]
    plt.close()
    if column == "Slice_2":
        plt.figure(figsize=(6, 4))
        #plt.figure(figsize=(8, 4))
    elif column == "Intensity":
        plt.figure(figsize=(6, 4))
    else:
```

```
        plt.figure(figsize=(5, 4))
    plt.tick_params(labelsize=fontsize)
    column_edt, xlabel_edt = column.replace("/", "_").replace(" ", "_"), xlabel.replace("/", "_")
    dir_out_sub = output + "¥¥" + column_edt    # Data are saved in different directories
depending on columns
    try:
        os.makedirs(dir_out_sub)
    except:
        pass
    color = reference(file.split("_")[2])
    file_ret = file.split("_")
    file_core = "{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2])
    if rnd == 0:
        fnc = int
    else:
        fnc = str

    # Main processing (make a histogram)
    #sr = data[column].dropna()
    statistics = sr.describe()
    statistics.to_csv("{}¥¥{}_{}_hist_statistics.csv".format(dir_out_sub, file_core, id))
    if xrange:
        n, bins, patches = plt.hist(sr, bins=bin, range=xrange, color=color, align="left")
        xmax = np.amax(xrange)
    else:
        n, bins, patches = plt.hist(sr, bins=bin, color=color, align="left")
    #set_trace()
    bins_del = np.delete(bins, -1, 0)
    sr_hist = pd.Series(n, index=bins_del)
    sr_hist.index.name = "bins"
    sr_hist.name = "n"
    display(sr_hist)
    sr_hist.to_csv("{}¥¥{}_{}_hist_dataset.csv".format(dir_out_sub, file_core, id), header=True)

    # Fitting
    # param_ini = [30, 0.7, 0.02]    # free, number, dimer
    param_ini = [100, 0.15, 0.01]    # monomer
    popt, pcov = curve_fit(func, bins_del, n, p0=param_ini)
    #popt, pcov = curve_fit(func, bins_del, n)
    #set_trace()
    print(popt)
    # [85.43763117 99.90468934  4.60884537]
    new_arr = np.arange(bins[0], bins[-1], 0.01)
    fitting = func(new_arr, popt[0], popt[1], popt[2])
    plt.plot(new_arr, fitting, 'r')

    # Calculate R^2
    residuals = n - func(bins_del, popt[0], popt[1], popt[2])
    rss = np.sum(residuals ** 2)  # residual sum of squares = rss
    tss = np.sum((n - 1 * np.mean(n)) ** 2)  # total sum of squares = tss
    r_squared = 1 - (rss / tss)
    params = np.append(popt, r_squared)
    print("r^2: {}".format(r_squared))
```

```
    # Text annotating in the figure
    #text = "¥nn={}¥nMedian={}¥nMean={}¥nSD={}¥nR^2={}". ¥
    #    format(int(statistics["count"]), fnc(round(statistics["50%"], rnd)),
    #          fnc(round(statistics["mean"], rnd)), fnc(round(statistics["std"], rnd)),
    #          fnc(round(r_squared, 2)))
    text = "¥nn={}¥n$¥mu$={}¥n$¥sigma$={}¥n$R^{{2}}$={}". ¥
        format(int(statistics["count"]), fnc(round(popt[1], rnd)),
              fnc(round(popt[2], rnd)), fnc(round(r_squared, 2)))
    print("¥nStatistic parameters in {}:{}".format(column, text))

    # Arrange appearance of the graph
    ymax = np.amax(n)
    plt.xlabel(xlabel, fontsize=fontsize)
    plt.ylabel("Number", fontsize=fontsize)
    #print("ymax: {}".format(ymax))
    #print("res: {}".format((ymax //10) *5))
    #xtick = (0, 0.2, 0.4)
    #ytick = range(0, int(ymax + 2), int(((ymax+1) // 10) *5))
    #print("Tick: {}".format(ytick))
    #if column == "Intensity":
    #    plt.yticks(ytick)
    #elif column == "Velocity (um/s)":
    #    plt.xticks(xtick)
    #ax.yaxis.set_major_locator(MultipleLocator(1))
    #plt.ylim([0, ymax *1.5])
    point = {'start': [statistics["50%"], ymax*1.3], 'end': [statistics["50%"], ymax*1.1]}
    #if column == "Velocity (um/s)" or column == "Slice_1" or column =="Intensity":
    #    plt.annotate("", xy=point["end"], xytext=point["start"],
    #            arrowprops=dict(shrink=0, width=1, headwidth=8,
    #                headlength=10, connectionstyle='arc3', facecolor='r', edgecolor='r'))
    plt.text(xrange[1]/3 *2, ymax/2, text, size=fontsize/5*4)
    plt.subplots_adjust(left=0.2, right=0.9, bottom=0.1, top=0.9)
    plt.gca().spines['right'].set_visible(False)  # Omit the right frame border
    plt.gca().spines['top'].set_visible(False)  # Omit the top frame border
    #set_trace()
    #plt.tight_layout()

    # Save
    output = "{}¥¥{}_{}".format(dir_out_sub, file_core, id)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)  #
inkscape を用いて svg を emf に変換
    #subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)  #
inkscape を用いて svg を emf に変換
    #plt.savefig(dir_out_sub + "¥¥" + file.replace(".csv", "_" + column_edt + "_" + xlabel_edt +
".png"))
    plt.show()
    plt.close()
    return sr_hist, params

def reference(key):
    '''
    Return str of a color referring the passed key.
```

```
    :param key: t073, f004, f005, etc
    :return: str showing color
    '''
    # Initialization
    color = "k"

    # Return str of a color referring the passed key.
    if key=="t073" or key=="free" or key=="1xbb":
        color = "k"
    elif key=="f004" or key=="bgx1":
        color = "r"
    elif key=="f005" or key=="bgx2":
        color = "b"
    elif key=="x1-f005":
        color = "c"
    elif key=="x2-f005" or key=="plane":
        color = "g"
    elif key == "bleach":
        color = "y"
    return color

def func(x, a, mu, sigma):
    return a*np.exp(-(x-mu)**2/(2*sigma**2))


# For debug
def set_trace():
    from IPython.core.debugger import Pdb
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)

def debug(f, *arg, **kwargs):
    from IPython.core.debugger import Pdb
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *arg, **kwrgs)

if __name__ == "__main__":
    main()
```

```
#
# py0136:
# Make a scatter plot with error bars.
# mu ± sigma determined by Gaussian fitting (py0135) ver.
#
# -*- coding: utf8 -*-

# Import modules
import datetime, os, tkinter.filedialog, tkinter.messagebox, tkinter.simpledialog, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tkinter as tk
import py0052_7_my_library_a_220122 as ml
from scipy.optimize import curve_fit
import matplotlib.ticker as ticker
import itertools
import math
import subprocess
from IPython.display import display

#def main(dir_in="", dir_out_master="", label="scatter_vel_Gaussian", config="GUI",
pre_set_id="", path_log="C:¥¥fukumoto¥¥script_py¥¥py_log.txt", mode="r", ylim=(1.3, 2)):
def main(path_src="", dir_out_master="", label="scatter_vel_Gaussian", config="GUI",
        pre_set_id="", path_log="C:¥¥fukumoto¥¥script_py¥¥log_files¥¥py_log_0136.txt",
        mode="v_Gaussian", ylim=(0.8, 10), opn="file"):
    '''
    Make a scatter plot with error bars. mu ± sigma determined by Gaussian fitting (py0135) ver.
    :param path_src: path of a csv file containing mu and sigma of velocity
    :param dir_out_master: (used for preset) path of a directory for output
    :param label: str which will be used for output directory and file name
    :param config: "GUI", "test" or "pre_set"
    :param pre_set_id: (used for preset) str which will be written for output directory and file
name
    :param path_log: path of a log file
    :param mode: "v_Gaussian" for this script
    :param ylim: (); tuple of ylim
    :return:
    '''

    # Initialization
    ylim = (0.25, 10)
    #ylim = (0.8, 10)
    #config = "test"
    script_num = __name__.split("_")[0]
    print("¥n¥n¥n**** {} ****".format(__name__))
    if not pre_set_id:
        dir_out_master = os.path.dirname(path_src)
    id, dir_log, input, dir_out = ml.initialize(path_log, label=label, config=config,
                                        dir_in=dir_out_master, pre_set_id=pre_set_id, opn=opn)
    if not pre_set_id:
        path_src = input
```

```
    #   print("¥n****** Directly Run *******")
    #l_fontsize = [15, 20, 22, 24, 26, 28, 30]
    #l_legend = [True, False]

    subscript(path_src, id, dir_out, fontsize=30, mode=mode, ylim=ylim)

    print("¥n** {} Script Finished **".format(script_num))
    return dir_out

def subscript(path_src, id, dir_out, fontsize=30, mode="v_Gaussian", ylim=(1, 10)):
    # Initialization
    ref = {"config": "v_Gaussian", "ylabel": "Velocity (μm/s)"}

    #width = 0.3
    #fig = plt.figure()
    #ax = fig.add_subplot(1, 1, 1)
    plt.tick_params(labelsize=fontsize)
    #plt.hlines([1, 2], 0, 4, color="k", alpha=1, linestyles='dashed', linewidth=1)

    # For velocity determined by Gaussian fitting
    if mode == "v_Gaussian":
        # Initialization
        file = os.path.basename(path_src)
        file_ret = file.split("_")
        #set_trace()
        file_core = "{}_{}_{}_{}".format(file_ret[0], file_ret[1], file_ret[2], file_ret[4])
        print("¥nfile_core: {}".format(file_core))
        # Mean
        data_src = pd.read_csv(path_src, index_col="Unnamed: 0")
        sr_mu = data_src["mu"]
        sr_sigma = data_src["sigma"]
        # Concat
        #table_dst = pd.concat([sr_mean, sr_std], axis=1)

    print("¥ndata_src:")
    #table_dst = table_dst.reindex(["190628_1_0k", "190628_2_5k", "190628_0_10k"])
    display(data_src)

    # Plot mean with error bars
    xlabels = []
    #xlabels = [table_dst.index]    #
    for i in range(len(data_src.index)):
        key_tmp = data_src.index[i]
        xlabels.append(key_tmp)
    y = data_src["mu"].values
    se = data_src["sigma"].values
    x = np.array([i+1 for i in range(len(y))])
    #xlabels = table_dst.index
    file_label = "{}_{}_{}_mean".format(file_core, ref["config"], id)
    plot_scatter(x, y, se=se, xlabels=xlabels, ylabel=ref["ylabel"],
                 dir_out=dir_out, file_label=file_label, mode=mode, ylim=ylim)
```

```
def plot_scatter(x, y, dir_out, file_label="test", se=[], ylabel="", xlabels=["1,2,3,4",
"1,2,7,8", "1,3,5,7"], fontsize=20, mode="r", ylim=(1, 10)):
    '''
    A function to make a scatter plot with error bars.
    :param x: np.array of x
    :param y: np.array of y
    :param dir_out: absolute path of a directory for result files
    :param file_label: str which will be included to result files
    :param se: np.array of errors as error bars
    :param ylabel: str of ylabel of the graph
    :param xlabels: list of factors (xticks) on the x-axis
    :param fontsize: int of fontsize in the graph
    :param mode: "v" (velocity) or "r" (run length)
    :param ylim: [min, max] determining the range of y-axis
    :return: ---
    '''
    # Initialization
    plt.close()
    #plt.tick_params(labelsize=25)
    fig, ax = plt.subplots(figsize=(4.5, 4))
    plt.tick_params(labelsize=fontsize)
    # ax2 = ax1.twinx()
    str_legend = "x"

    # Plot
    plt.xlim([0, 4])
    if len(x)==4:
        y = [y[i] for i in (3, 0, 1, 2)]
        xlabels = [xlabels[i] for i in (3, 0, 1, 2)]
        se = [se[i] for i in (3, 0, 1, 2)]
        plt.xlim([0, 5])
    elif len(x)==2:
        plt.xlim([0, 3])
    # for free
    #else:
    #    y = [y[i] for i in (1, 2, 0)]
    #    xlabels = [xlabels[i] for i in (1, 2, 0)]
    #    se = [se[i] for i in (1, 2, 0)]
    ax.scatter(x, y, marker="s", color="k", s=fontsize)
    if len(se) != 0:
        ax.errorbar(x, y, yerr=se, fmt='none', ecolor="k", capsize=5)

    # Arrange appearance of the graph
    ax.set_ylabel(ylabel, fontsize=fontsize)
    #set_trace()
    plt.xticks(np.unique(x), xlabels)
    if mode=="v_Gaussian":
        plt.ylim([0, ylim[0]])
        ax.set_ylabel("Velocity (μm/s)", fontsize=fontsize)
    plt.gca().spines['right'].set_visible(False)
    plt.gca().spines['top'].set_visible(False)
    plt.subplots_adjust(left=0.2, right=0.9, bottom=0.1, top=0.9)
    #plt.tight_layout()
```

```
    # Save
    output = "{}¥¥{}_graph".format(dir_out, file_label)
    plt.savefig(output + ".png")
    plt.savefig(output + ".svg")
    subprocess.call('inkscape.exe {}.svg -M {}.emf'.format(output, output), shell=True)
    plt.show()
    plt.close()

# For debug
def set_trace():
    from IPython.core.debugger import Pdb
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)

def debug(f, *arg, **kwargs):
    from IPython.core.debugger import Pdb
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *arg, **kwrgs)

if __name__ == "__main__":
    main()
```