Get lost

- **Topic:** random walks, particle density, diffusion, brownian motion
- **Task A:**
  1. Implement a function to make the lattice-bound random walker take unit steps on a grid.
  2. Simulate the motion of a single particle and check that the simulator works.
  3. Simulate the motion of, say, 10000 particles.
  4. Study the distribution of these particles as a function of time. What does this tell us about diffusion?
- **Task B:**
  1. Implement a function to have the freely moving particle from previous lecture take normally distributed random steps.
  2. Simulate this random walk.
- **Templates:**
  - random_walk_lattice.py
  - random_walk_free.py
- **Further reading:**
  - https://en.wikipedia.org/wiki/Random_walk
  - https://en.wikipedia.org/wiki/Diffusion
  - https://en.wikipedia.org/wiki/Brownian_motion

random_walk_lattice.py

class random_walk_lattice.**Walker**(*x*, *y*) [source]

A particle moving on a lattice.

Since the particle can only move on a lattice, its coordinates are always integers.

Parameters:
- **x** (*int*) – initial x coordinate
- **y** (*int*) – initial y coordinate

get_distance_sq() [source]

Calculates the squared distance between the initial and current coordinates of this particle.

Returns: squared distance $r^2 = (x - x_0)^2 + (y - y_0)^2$
Return type: float

move_randomly() [source]

Makes the particle move.

The particle takes a step of length 1 in one of the cardinal directions: up, down, left or right.

There is a 20 % chance for each direction. In addition, there is a 20 % chance the particle stays in place.

> **Note**
> This function is incomplete!

random_walk_lattice.**animate**(*history*) [source]

Animate the simulation.

Parameters:
- **history** (*list*) – list of systems at different times
- **size** (*lattice*) – system size N

random_walk_lattice.**calculate_density_histogram**(*particles*, *range_steps=10*, *max_range=50*) [source]

Calculates a histogram of particle density as a function of displacement.

The, function only takes into account the particles for which the displacement r_i is not greater than a given maximum R. The range of possible values is split in k separate subranges or *bins*, $[0, R/k), [R/k, 2R/k)$ etc. The midpoints of each bin, $\frac{R}{2k}, \frac{3R}{2k}, \frac{5R}{2k}$ etc. are also saved.

Having defined these bins, the function calculates the displacement r_i for each particle using `Walker.get_distance_sq()` and checks which bin the displacement belongs to. The function then counts how many particles are placed in each bin, $n(r)$.

Finally, the function calculates the surface density of particles at each displacement range. That is, the number of particles in a given region is divided by the area of that region,

$$\sigma(r) = \frac{n(r)}{A(r)}.$$

As this is a 2D simulation, the particles whose displacement hit the range $[r, r + \Delta r]$ are inside a circular edge with inner radius r and thickness Δr. The area of this edge is $A(r) = \pi(r + \frac{1}{2}\Delta r)\Delta r$.

Parameters:
- **particles** (*list*) – list of `Walker` objects
- **range_steps** (*int*) – the number of bins, k
- **max_range** (*float*) – maximum displacement, R

Returns: average r, particle count $n(r)$, particle density $\sigma(r)$ in each bin
Return type: array, array, array

random_walk_lattice.**calculate_distance_statistics**(*particles*) [source]

Calculates the average and error of mean of all particles squared displacement.

The squared displacement r_i^2 is calculated with `Walker.get_distance_sq()` for every particle i. The function then calculates the mean

$$\langle r^2 \rangle = \frac{1}{N} \sum_i r_i^2$$

and error of mean

$$\Delta\langle r^2 \rangle = \frac{s_{r^2}}{\sqrt{N}},$$

where s_{r^2} is the sample standard deviation

$$s_{r^2} = \sqrt{\frac{1}{N-1} \sum_i (r_i^2 - \langle r^2 \rangle)^2}.$$

Parameters: **particles** (*list*) – list of `Walker` objects
Returns: average $\langle r^2 \rangle$, error $\Delta\langle r^2 \rangle$
Return type: float, float

random_walk_lattice.**draw**(*frame*, *history*, *scale=5*) [source]

Draws the system for animation.

Parameters:
- **frame** (*int*) – index of the frame to draw
- **history** (*list*) – list of systems at different times
- **scale** (*int*) – values above this will be shown as black

random_walk_lattice.**generate_empty_lattice**(*lattice_size=100*) [source]

Creates an empty lattice as an $N \times N$ array of zeros.

Parameters: **size** (*lattice*) – system size N
Returns: the lattice
Return type: array

random_walk_lattice.**generate_lattice**(*particles*, *lattice_size*, *label=1*, *additive=True*) [source]

Creates a lattice with particles as an $N \times N$ array.

The lattice sites with no particles are given the value of zero. The sites with 1 or more particle are given a non-zero value.

Parameters:
- **particles** (*list*) – list of `Walker` objects
- **size** (*lattice*) – system size N
- **label** (*int*) – value given to sites with particles
- **additive** (*bool*) – If True, the value given to sites is proportional to the number of particles on that site. If False, all sites with however many particles are given the same value.

Returns: the lattice
Return type: array

random_walk_lattice.**linear**(*x*, *a*) [source]

Calculates the linear function $y = ax$ and returns the result.

Parameters:
- **x** (*float*) – the variable
- **a** (*float*) – the slope

Returns: the result
Return type: float

random_walk_lattice.**linear_fit**(*xdata*, *ydata*, *name*) [source]

Fit a linear function $y = ax$ to the given xy data. Also return the optimal fit values obtained for slope a.

To identify this information, you may also name the operation.

Parameters:
- **xdata** (*array*) – x values
- **ydata** (*array*) – y values
- **name** (*str*) – identifier for printing the result

Returns: slope
Return type: float

random_walk_lattice.**main**(*n_particles*, *n_steps*, *n_plots*) [source]

The main program.

Creates particles at the center of the system and then allows them to diffuse by performing a random walk on a lattice.

Animates the motion and if the number of particles is large enough, also calculates statistics for the particle distribution.

Parameters:
- **n_particles** (*int*) – number of particles
- **n_steps** (*int*) – number of simulation steps
- **n_plots** (*int*) – number of times data is recorded

random_walk_lattice.**move**(*particles*) [source]

Makes all particles move.

All particles take a random step using `Walker.move_randomly()`.

Parameters: **particles** (*list*) – list of `Walker` objects

random_walk_lattice.**print_progress**(*step*, *total*) [source]

Prints a progress bar.

Parameters:
- **step** (*int*) – progress counter
- **total** (*int*) – counter at completion

random_walk_lattice.**show_system**(*particles*, *lattice_size*, *scale=5*) [source]

Draws the system as a $N \times N$ pixel image.

Parameters:
- **particles** (*list*) – list of `Walker` objects
- **size** (*lattice*) – system size N
- **scale** (*int*) – values above this will be shown as black

random_walk_free.py

random_walk_free.**animate**() [source]

Animates the trajectory.

A trajectory of the shifting coordinates are saved in the global variable trajectory as a list of coordinates. This function animates the system up to the specified frame.

random_walk_free.**draw**(*frame*) [source]

Draws one frame for animation.

A trajectory of the shifting coordinates are saved in the global variable trajectory as a list of coordinates. This function draws the system as defined in trajectory[frame].

Parameters: **frame** – index of the frame to be drawn.

random_walk_free.**main**(*x_start*, *y_start*, *angle_start*, *simulation_length=500*) [source]

Moves a point step by step and draws the trajectory.

The point is moved using `move()`. The path is visualized using `plot_trajectory()` and `animate()`.

Parameters:
- **x_start** (*float*) – starting point x coordinate
- **y_start** (*float*) – starting point y coordinate
- **angle_start** (*float*) – angle defining the starting direction
- **simulation_length** (*int*) – total number of steps to take

random_walk_free.**move**(*x*, *y*, *step*, *angle*) [source]

Calculates new coordinates by taking a step from a given starting point.

The function starts from position (x, y) and moves the distance L in the direction defined by the angle θ, where $\theta = 0$ means moving in positive x direction and $\theta = \pi/2$ means moving in positive y direction.

The function returns the coordinates of the final position.

Parameters:
- **x** (*float*) – initial x coordinate
- **y** (*float*) – initial y coordinate
- **step** (*float*) – step length L
- **angle** (*float*) – step direction θ in radians

Returns: final coordinates (x, y)
Return type: float, float

random_walk_free.**move_randomly**(*x*, *y*, *std*) [source]

Calculates new coordinates by taking a step from a given starting point.

The function starts from position (x, y) and takes a random step so that both coordinates are changed by a normally distributed shift.

The function returns the coordinates of the final position.

> **Note**
> This function is incomplete!

Parameters:
- **x** (*float*) – initial x coordinate
- **y** (*float*) – initial y coordinate
- **std** (*float*) – standard deviation for the coordinate shifts.

Returns: final coordinates (x, y)
Return type: float, float

random_walk_free.**plot_trajectory**(*trajectory*) [source]

Plots a trajectory.

The trajectory is drawn as a line and the current position is drawn as a point.

Parameters: **trajectory** (*array*) – list of coordinate pairs

random_walk_free.**read_file**(*filename*) [source]

Reads a trajectory from file.

The file must obey the format specified in `write_file()`.

Parameters: **filename** (*str*) – name of the file to read.
Returns: trajectory as an array of coordinate pairs
Return type: array

random_walk_free.**write_file**(*trajectory*, *filename*) [source]

Writes the trajectory to file.

Each line in the file will contain a pair x and y coordinates separated by whitespace:

```
x0  y0
x1  y1
x2  y2
...
```

Parameters:
- **trajectory** (*array*) – list of coordinate pairs
- **filename** (*str*) – name of file to write