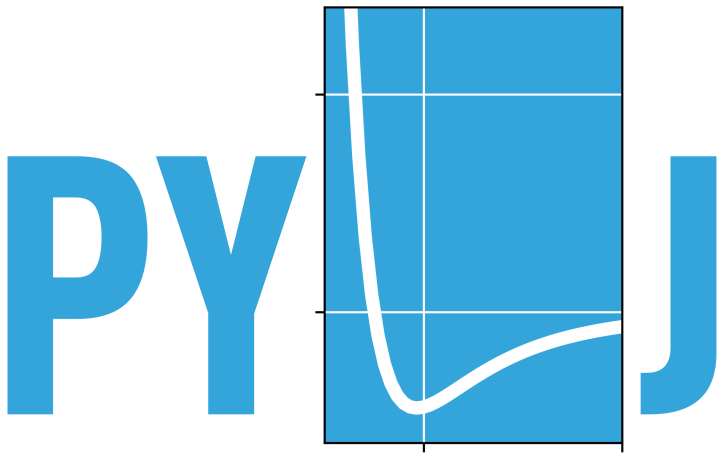
pylj Documentation

Release 1.3.4

Andrew R. McCluskey

Sep 29, 2020

Package Contents:


PYLJ

CHAPTER 1

pylj modules

1.1 pylj.comp

These are the COMPutationally intensive, COMPiled (by cython) functions. Generally these are functions required by pylj which involve a pair-wise comparison of particles. These are written in C++, the source of which can be found in the src/comp.cpp file.

pylj.comp.**calculate_pressure**()

Calculates the instantaneous pressure of the simulation cell, found with the following relationship:

$$p = \langle \rho k_b T \rangle + \left\langle \frac{1}{3V} \sum_i \sum_{j<i} \mathbf{r}_{ij} \mathbf{f}_{ij} \right\rangle$$

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **temperature** (*float*) – Instantaneous temperature of the simulation.
- **cut_off** (*float*) – The distance greater than which the forces between particles is taken as zero.
- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B].

Returns Instantaneous pressure of the simulation.

Return type float

pylj.comp.**compute_energy**()

Calculates the total energy of the simulation. This uses a 12-6 Lennard-Jones potential model for Argon with values:

- A = 1.363e-134 J m ^{12}

- B = 9.273e-78 J m ^{6}

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **cut_off** (*float*) – The distance greater than which the energies between particles is taken as zero.
- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B].

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.
- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.comp.**compute_force**()

Calculates the forces and therefore the accelerations on each of the particles in the simulation. This uses a 12-6 Lennard-Jones potential model for Argon with values:

- A = 1.363e-134 J m ^{12}
- B = 9.273e-78 J m ^{6}

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **cut_off** (*float*) – The distance greater than which the forces between particles is taken as zero.
- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B].
- **mass** (*float (optional)*) – The mass of the particle being simulated (units of atomic mass units).

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.
- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current forces between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.comp.**dist**()

Returns the distance array for the set of particles.

Parameters

- **xpos** (*float, array_like (N)*) – Array of length N, where N is the number of particles, providing the x-dimension positions of the particles.
- **ypos** (*float, array_like (N)*) – Array of length N, where N is the number of particles, providing the y-dimension positions of the particles.
- **box_length** (*float*) – The box length of the simulation cell.

Returns

- *distances float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles.
- *xdistances float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles, in only the x-dimension.
- *ydistances float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles, in only the y-dimension.

pylj.comp.**heat_bath**()

Rescales the velocities of the particles in the system to control the temperature of the simulation. Thereby allowing for an NVT ensemble. The velocities are rescaled according the following relationship,

$$v_{\text{new}} = v_{\text{old}} \times \sqrt{\frac{T_{\text{desired}}}{\bar{T}}}$$

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **temperature_sample** (*float, array_like*) – The temperature at each timestep in the simulation.
- **bath_temp** (*float*) – The desired temperature of the simulation.

Returns Information about the particles with new, rescaled velocities.

Return type util.particle_dt, array_like

1.2 pylj.forcefields

The forcefields packaged with pylj.

pylj.forcefields.**buckingham**

Calculate the energy or force for a pair of particles using the Buckingham forcefield.

$$E = Ae^{(-Bdr)} - \frac{C}{dr^6}$$

$$f = ABe^{(-Bdr)} - \frac{6C}{dr^7}$$

Parameters

- **dr** (*float, array_like*) – The distances between all the pairs of particles.
- **constants** (*float, array_like*) – An array of length three consisting of the A, B and C parameters for the Buckingham function.
- **force** (*bool (optional)*) – If true, the negative first derivative will be found.

Returns **float** – The potential energy or force between the particles.

Return type array_like

pylj.forcefields.**lennard_jones**
Calculate the energy or force for a pair of particles using the Lennard-Jones (A/B variant) forcefield.

$$E = \frac{A}{dr^{12}} - \frac{B}{dr^{6}}$$

$$f = \frac{12A}{dr^{13}} - \frac{6B}{dr^{7}}$$

Parameters

- **dr** (*float, array_like*) – The distances between the all pairs of particles.
- **constants** (*float, array_like*) – An array of length two consisting of the A and B parameters for the 12-6 Lennard-Jones function
- **force** (*bool (optional)*) – If true, the negative first derivative will be found.

Returns **float** – The potential energy or force between the particles.

Return type array_like

pylj.forcefields.**lennard_jones_sigma_epsilon**
Calculate the energy or force for a pair of particles using the Lennard-Jones (sigma/epsilon variant) forcefield.

$$E = \frac{4e * a^{12}}{dr^{12}} - \frac{4e * a^{6}}{dr^{6}}$$

$$f = \frac{48e * a^{12}}{dr^{13}} - \frac{24e * a^{6}}{dr^{7}}$$

Parameters

- **dr** (*float, array_like*) – The distances between the all pairs of particles.
- **constants** (*float, array_like*) – An array of length two consisting of the sigma (a) and epsilon (e) parameters for the 12-6 Lennard-Jones function
- **force** (*bool (optional)*) – If true, the negative first derivative will be found.

Returns **float** – The potential energy or force between the particles.

Return type array_like

pylj.forcefields.**square_well**(*dr*, *constants*, *max_val=inf*, *force=False*)
Calculate the energy or force for a pair of particles using a square well model.

$$E = if dr < sigma : E = max_valelif sigma <= dr < lambda * sigma : E = -epsilonelif r >= lambda * sigma : E = 0$$

$$f = if sigma <= dr < lambda * sigma : f = infelse : f = 0$$

Parameters

- **dr** (*float, array_like*) – The distances between all the pairs of particles.
- **constants** (*float, array_like*) – An array of length three consisting of the epsilon, sigma, and lambda parameters for the square well model.
- **max_val** (*int (optional)*) – Upper bound for values in square well - replaces usual infinite values
- **force** (*bool (optional)*) – If true, the negative first derivative will be found.

Returns **float** – The potential energy between the particles.

Return type array_like

1.3 pylj.mc

Functions related to the Monte-Carlo components of pylj.

pylj.mc.**accept**(*new_energy*)
Accept the move.

Parameters **new_energy** (*float*) – A new total energy for the system.

Returns A new total energy for the system.

Return type float

pylj.mc.**get_new_particle**(*particles*, *random_particle*, *box_length*)
Generates a new position for the particle.

Parameters

- **particles** (*util.particle.dt, array_like*) – Information about the particles.
- **random_particle** (*int*) – Index of the random particle that is selected.
- **box_length** (*float*) – Length of a single dimension of the simulation square.

Returns Information about the particles, updated to account for the change of selected particle position.

Return type util.particle.dt, array_like

pylj.mc.**initialise**(*number_of_particles, temperature, box_length, init_conf, mass=39.948, constants=[1.363e-134, 9.273e-78], forcefield=CPUDispatcher(<function lennard_jones>)*)
Initialise the particle positions (this can be either as a square or random arrangement), velocities (based on the temperature defined, and # calculate the initial forces/accelerations.

Parameters

- **number_of_particles** (*int*) – Number of particles to simulate.
- **temperature** (*float*) – Initial temperature of the particles, in Kelvin.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **init_conf** (*string, optional*) – The way that the particles are initially positioned. Should be one of: - 'square' - 'random'
- **mass** (*float (optional)*) – The mass of the particles being simulated.
- **constants** (*float, array_like (optional)*) – The values of the constants for the forcefield used.
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

Returns System information.

Return type *System*

pylj.mc.**initialize**(*number_particles*, *temperature*, *box_length*, *init_conf*)
Maps to the mc.initialise function to account for US english spelling.

pylj.mc.**metropolis**(*temperature*, *old_energy*, *new_energy*, *n=0.8244327950328006*)
Determines if the move is accepted or rejected based on the metropolis condition.

Parameters

- **temperature** (*float*) – Simulation temperature.
- **old_energy** (*float*) – The total energy of the simulation in the previous configuration.
- **new_energy** (*float*) – The total energy of the simulation in the current configuration.
- **n** (*float, optional*) – The random number against which the Metropolis condition is tested. The default is from a numpy uniform distribution.

Returns True if the move should be accepted.

Return type bool

pylj.mc.**reject**(*position_store*, *particles*, *random_particle*)
Reject the move and return the particle to the original place.

Parameters

- **position_store** (*float, array_like*) – The x and y positions previously held by the particle that has moved.
- **particles** (*util.particle.dt, array_like*) – Information about the particles.
- **random_particle** (*int*) – Index of the random particle that is selected.

Returns Information about the particles, with the particle returned to the original position

Return type util.particle.dt, array_like

pylj.mc.**sample**(*total_energy*, *system*)
Sample parameters of interest in the simulation.

Parameters

- **total_energy** (*float*) – The total system energy.
- **system** (System) – Details about the whole system

Returns Details about the whole system, with the new temperature, pressure, msd, and force appended to the appropriate arrays.

Return type *System*

pylj.mc.**select_random_particle**(*particles*)
Selects a random particle from the system and return its index and current position.

Parameters **particles** (*util.particle.dt, array_like*) – Information about the particles.

Returns

- *int* – Index of the random particle that is selected.
- *float, array_like* – The current position of the chosen particle.

1.4 pylj.md

Functions related to the molecular dynamics components of pylj.

pylj.md.**calculate_msd**(*particles*, *initial_particles*, *box_length*)
Determines the mean squared displacement of the particles in the system. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param initial_particles: Information about the initial state of the particles. :type initial_particles: util.particle_dt, array_like :param box_length: Size of the cell vector. :type box_length: float

Returns Mean squared deviation for the particles at the given timestep.

Return type float

pylj.md.**calculate_temperature**(*particles*, *mass*)
Determine the instantaneous temperature of the system. :param particles: Information about the particles. :type particles: util.particle_dt, array_like

Returns Calculated instantaneous simulation temperature.

Return type float

pylj.md.**compute_energy**(*particles*, *box_length*, *cut_off*, *constants*, *forcefield*)
Calculates the total energy of the simulation. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param cut_off: The distance greater than which the energies between particles is

taken as zero.

Parameters

- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B]
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.
- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.md.**compute_force**(*particles*, *box_length*, *cut_off*, *constants*, *forcefield*, *mass*)
Calculates the forces and therefore the accelerations on each of the particles in the simulation. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param cut_off: The distance greater than which the forces between particles is taken

as zero.

Parameters

- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B]
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.
- **mass** (*float (optional)*) – The mass of the particle being simulated (units of atomic mass units).

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.

- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current forces between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.md.**heat_bath**(*particles*, *temperature_sample*, *bath_temperature*)

Rescales the velocities of the particles in the system to control the temperature of the simulation. Thereby allowing for an NVT ensemble. The velocities are rescaled according the following relationship, .. math:

```
v_{\text{new}} = v_{\text{old}} \times
\sqrt{\frac{T_{\text{desired}}}{\bar{T}}}
```

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **temperature_sample** (*float, array_like*) – The temperature at each timestep in the simulation.
- **bath_temp** (*float*) – The desired temperature of the simulation.

Returns Information about the particles with new, rescaled velocities.

Return type util.particle_dt, array_like

pylj.md.**initialise**(*number_of_particles, temperature, box_length, init_conf, timestep_length=1e-14, mass=39.948, constants=[1.363e-134, 9.273e-78], forcefield=CPUDispatcher(<function lennard_jones>)*)

Initialise the particle positions (this can be either as a square or random arrangement), velocities (based on the temperature defined, and calculate the initial forces/accelerations.

Parameters

- **number_of_particles** (*int*) – Number of particles to simulate.
- **temperature** (*float*) – Initial temperature of the particles, in Kelvin.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **init_conf** (*string, optional*) – The way that the particles are initially positioned. Should be one of: - 'square' - 'random'
- **timestep_length** (*float (optional)*) – Length for each Velocity-Verlet integration step, in seconds.
- **mass** (*float (optional)*) – The mass of the particles being simulated.
- **constants** (*float, array_like (optional)*) – The values of the constants for the forcefield used.
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

Returns System information.

Return type *System*

pylj.md.**initialize**(*number_particles*, *temperature*, *box_length*, *init_conf*, *timestep_length=1e-14*)

Maps to the md.initialise function to account for US english spelling.

pylj.md.**sample**(*particles*, *box_length*, *initial_particles*, *system*)

Sample parameters of interest in the simulation. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param initial_particles: Information about the initial particle conformation. :type initial_particles: util.particle_dt, array-like :param system: Details about the whole system :type system: System

Returns Details about the whole system, with the new temperature, pressure, msd, and force appended to the appropriate arrays.

Return type *System*

pylj.md.**update_positions**(*positions*, *old_positions*, *velocities*, *accelerations*, *timestep_length*, *box_length*)

Update the particle positions using the Velocity-Verlet integrator. :param positions: Where N is the number of particles, and the first row are the x

positions and the second row the y positions.

Parameters

- **old_positions** (*(2, N) array_like*) – Where N is the number of particles, and the first row are the previous x positions and the second row are the y positions.
- **velocities** (*(2, N) array_like*) – Where N is the number of particles, and the first row are the x velocities and the second row the y velocities.
- **accelerations** (*(2, N) array_like*) – Where N is the number of particles, and the first row are the x accelerations and the second row the y accelerations.
- **timestep_length** (*float*) – Length for each Velocity-Verlet integration step, in seconds.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.

Returns Updated positions.

Return type (2, N) array_like

pylj.md.**update_velocities**(*velocities*, *accelerations_old*, *accelerations_new*, *timestep_length*)

Update the particle velocities using the Velocity-Verlet algoritm. :param velocities: Where N is the number of particles, and the first row are the x

velocities and the second row the y velocities.

Parameters

- **accelerations** (*(2, N) array_like*) – Where N is the number of particles, and the first row are the x accelerations and the second row the y accelerations.
- **timestep_length** (*float*) – Length for each Velocity-Verlet integration step, in seconds.

Returns Updated velocities.

Return type (2, N) array_like

pylj.md.**velocity_verlet**

Uses the Velocity-Verlet integrator to move forward in time. The Updates the particles positions and velocities in terms of the Velocity Verlet algorithm. Also calculates the instanteous temperature, pressure, and force and appends these to the appropriate system array. :param particles: Information about the particles. :type particles:

util.particle_dt, array_like :param timestep_length: Length for each Velocity-Verlet integration step, in seconds. :type timestep_length: float :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float

Returns Information about the particles, with new positions and velocities.

Return type util.particle_dt, array_like

1.5 pylj.pairwise

Generally these are functions required by pylj which involve a pair-wise comparison of particles. This is the python implimentation of the *pylj.comp* module.

pylj.pairwise.**calculate_pressure**(*particles*, *box_length*, *temperature*, *cut_off*, *constants*, *forcefield*)

Calculates the instantaneous pressure of the simulation cell, found with the following relationship: .. math:

```
p = \langle \rho k_b T \rangle + \bigg\langle \frac{1}{3V}\sum_{i}
\sum_{j<i} \mathbf{r}_{ij}\mathbf{f}_{ij} \bigg\rangle
```

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **box_length** (*float*) – Length of a single dimension of the simulation square, in Angstrom.
- **temperature** (*float*) – Instantaneous temperature of the simulation.
- **cut_off** (*float*) – The distance greater than which the forces between particles is taken as zero.
- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B]
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

Returns Instantaneous pressure of the simulation.

Return type float

pylj.pairwise.**compute_energy**(*particles*, *box_length*, *cut_off*, *constants*, *forcefield*)

Calculates the total energy of the simulation. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param cut_off: The distance greater than which the energies between particles is

taken as zero.

Parameters

- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B]
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.
- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.pairwise.**compute_force**

Calculates the forces and therefore the accelerations on each of the particles in the simulation. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param cut_off: The distance greater than which the forces between particles is taken

as zero.

Parameters

- **constants** (*float, array_like (optional)*) – The constants associated with the particular forcefield used, e.g. for the function forcefields.lennard_jones, theses are [A, B]
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.
- **mass** (*float (optional)*) – The mass of the particle being simulated (units of atomic mass units).

Returns

- *util.particle_dt, array_like* – Information about particles, with updated accelerations and forces.
- *float, array_like* – Current distances between pairs of particles in the simulation.
- *float, array_like* – Current forces between pairs of particles in the simulation.
- *float, array_like* – Current energies between pairs of particles in the simulation.

pylj.pairwise.**dist**

Returns the distance array for the set of particles. :param xpos: Array of length N, where N is the number of particles, providing the

x-dimension positions of the particles.

Parameters

- **ypos** (*float, array_like (N)*) – Array of length N, where N is the number of particles, providing the y-dimension positions of the particles.
- **box_length** (*float*) – The box length of the simulation cell.

Returns

- *drr float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles.
- *dxr float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles, in only the x-dimension.
- *dyr float, array_like ((N - 1) * N / 2))* – The pairs of distances between the particles, in only the y-dimension.

pylj.pairwise.**heat_bath**(*particles*, *temperature_sample*, *bath_temp*)

Rescales the velocities of the particles in the system to control the temperature of the simulation. Thereby allowing for an NVT ensemble. The velocities are rescaled according the following relationship, .. math:

```
v_{\text{new}} = v_{\text{old}} \times
\sqrt{\frac{T_{\text{desired}}}{\bar{T}}}
```

Parameters

- **particles** (*util.particle_dt, array_like*) – Information about the particles.
- **temperature_sample** (*float, array_like*) – The temperature at each timestep in the simulation.
- **bath_temp** (*float*) – The desired temperature of the simulation.

Returns Information about the particles with new, rescaled velocities.

Return type util.particle_dt, array_like

pylj.pairwise.**lennard_jones_energy**(*A*, *B*, *dr*)

pairwise.lennard_jones_energy has been deprecated, please use forcefields.lennard_jones instead

Calculate the energy of a pair of particles at a given distance.

Parameters

- **A** (*float*) – The value of the A parameter for the Lennard-Jones potential.
- **B** (*float*) – The value of the B parameter for the Lennard-Jones potential.
- **dr** (*float*) – The distance between the two particles.

Returns The potential energy between the two particles.

Return type float

pylj.pairwise.**lennard_jones_force**(*A*, *B*, *dr*)

pairwise.lennard_jones_energy has been deprecated, please use forcefields.lennard_jones with force=True instead

Calculate the force between a pair of particles at a given distance.

Parameters

- **A** (*float*) – The value of the A parameter for the Lennard-Jones potential.
- **B** (*float*) – The value of the B parameter for the Lennard-Jones potential.
- **dr** (*float*) – The distance between the two particles.

Returns The force between the two particles.

Return type float

pylj.pairwise.**pbc_correction**

Correct for the periodic boundary condition. :param position: Particle position. :type position: float :param cell: Cell vector. :type cell: float

Returns Corrected particle position.

Return type float

pylj.pairwise.**second_law**(*f*, *m*, *d1*, *d2*)
Newton's second law of motion to get the acceleration of the particle in a given dimension. :param f: The force on the pair of particles. :type f: float :param m: Mass of the particle. :type m: float :param d1: Distance between the particles in a single dimension. :type d1: float :param d2: Distance between the particles across all dimensions. :type d2: float

Returns Acceleration of the particle in a given dimension.

Return type float

pylj.pairwise.**separation**(*dx*, *dy*)
Calculate the distance in 2D space. :param dx: Vector in the x dimension :type dx: float :param dy: Vector in the y dimension :type dy: float :param Returns: :param float: Magnitude of the 2D vector.

pylj.pairwise.**update_accelerations**(*particles*, *f*, *m*, *dx*, *dy*, *dr*)
Update the acceleration arrays of particles. :param particles: Information about the particles. :type particles: util.particle_dt, array_like :param f: The force on the pair of particles. :type f: float :param m: Mass of the particles. :type m: float :param dx: Distance between the particles in the x dimension. :type dx: float :param dy: Distance between the particles in the y dimension. :type dy: float :param dr: Distance between the particles. :type dr: float

Returns Information about the particles with updated accelerations.

Return type util.particle_dt, array_like

1.6 pylj.sample

This module is dedicated visualisation of the pylj output. More details of how to implement custom visualisation see *visualisation*.

class pylj.sample.**CellPlus**(*system*, *xlabel*, *ylabel*, *size='medium'*)
Bases: object

The CellPlus class will plot the particles positions in addition to one user defined one-dimensional dataset. This is designed to allow user interaction with the plotting data.

Parameters

- **system** (System) – The whole system information.
- **xlabel** (*string*) – The label for the x-axis of the custom plot.
- **ylabel** (*string*) – The label for the y-axis of the custom plot.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

update(*system*, *xdata*, *ydata*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so use this wisely.

Parameters

- **system** (System) – The whole system information.
- **xdata** (*float, array-like*) – x-data that should be plotted in the custom plot.

- **ydata** (*float, array-like*) – y-data that should be plotted in the custom plot.

class pylj.sample.**Energy**(*system*, *size='medium'*)
Bases: object

The energy class will plot the particle positions and potential energy of the system.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**Interactions**(*system*, *size='medium'*)
Bases: object

The Interactions class will plot the particle positions, total force, simulation pressure and temperature. This class is perfect for showing the interactions between the particles and therefore the behaviour of ideal gases and deviation when the conditions of an ideal gas are not met.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**JustCell**(*system*, *size='medium'*, *scale=1*)
Bases: object

The JustCell class will plot just the particles positions. This is a simplistic sampling class for quick visualisation.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'

- 'medium' (default)
- 'large'

- **scale** (*float (optional)*) – The amount by which to scale down the size of the particles

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so use this wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**MaxBolt**(*system*, *size='medium'*)
Bases: object

The MaxBolt class will plot the particle positions and a histogram of the particle velocities.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**Phase**(*system*, *size='medium'*)
Bases: object

The Phase class will plot the particle positions, radial distribution function, mean squared deviation and total energy of the simulation. This sampling class is ideal for observing the phase transitions between solid, liquid, gas.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

average()

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**RDF**(*system*, *size='medium'*)
Bases: object

The RDF class will plot the particle positions and radial distribution function. This sampling class is can be used to show the relative RDFs for solid, liquid, gas.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

average()

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

class pylj.sample.**Scattering**(*system*, *size='medium'*)
Bases: object

The Scattering class will plot the particle positions, radial distribution function, mean squared deviation and scattering profile (as a fft of the rdf). This sampling class is ideal for observing the phase transitions between solid, liquid, gas.

Parameters

- **system** (System) – The whole system information.
- **size** (*string*) –

  The size of the visualisation:

  - 'small'
  - 'medium' (default)
  - 'large'

average()

update(*system*)
This updates the visualisation environment. Often this can be slower than the cythonised force calculation so used is wisely.

Parameters **system** (System) – The whole system information.

pylj.sample.**environment**(*panes*, *size='medium'*)
The visualisation environment consists of a series of panes (1, 2, or 4 are allowed). This function allows the number of panes in the visualisation to be defined.

Parameters

- **panes** (*int*) – Number of visualisation panes.
- **size** (*string*) –

  The size of the visualisation:

- 'small'
- 'medium' (default)
- 'large'

Returns

- *Matplotlib.figure.Figure object* – The relevant Matplotlib figure.
- *Axes object or array of axes objects* – The axes related to each of the panes. For panes=1 this is a single object, for panes=2 it is a 1-D array and for panes=4 it is a 2-D array.

pylj.sample.**setup_cellview**(*ax*, *system*, *scale=1*)
Builds the particle position visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.
- **scale** (*float (optional)*) – The amount by which the particle size should be scaled down.

pylj.sample.**setup_diffview**(*ax*)
Builds the scattering profile visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_energyview**(*ax*)
Builds the total force visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_forceview**(*ax*)
Builds the total force visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_maxbolthist**(*ax*)
Builds the simulation velocity histogram visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_msdview**(*ax*)
Builds the simulation mean squared deviation visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_pressureview**(*ax*)
Builds the simulation instantaneous pressure visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**setup_rdfview**(*ax*, *system*)
Builds the radial distribution function visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**setup_tempview**(*ax*)
Builds the simulation instantaneous temperature visualisation pane.

Parameters **ax** (*Axes object*) – The axes position that the pane should be placed in.

pylj.sample.**update_cellview**(*ax*, *system*)
Updates the particle positions visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_diffview**(*ax*, *system*, *average_diff*, *q*)
Updates the scattering profile visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.
- **average_diff** (*array_like*) – The scattering profile's i(q) for each timestep, to later be averaged.
- **q** (*array_like*) – The scattering profile's q for each timestep, to later be averaged.

pylj.sample.**update_energyview**(*ax*, *system*)
Updates the total force visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_forceview**(*ax*, *system*)
Updates the total force visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_maxbolthist**(*ax*, *system*, *velocities*)
Updates the simulation velocity histogram visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_msdview**(*ax*, *system*)
Updates the simulation mean squared deviation visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_pressureview**(*ax*, *system*)
Updates the simulation instantaneous pressure visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

pylj.sample.**update_rdfview**(*ax*, *system*, *average_rdf*, *r*)
Updates the radial distribution function visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.
- **average_rdf** (*array_like*) – The radial distribution functions g(r) for each timestep, to later be averaged.
- **r** (*array_like*) – The radial distribution functions r for each timestep, to later be averaged.

pylj.sample.**update_tempview**(*ax*, *system*)
Updates the simulation instantaneous temperature visualisation pane.

Parameters

- **ax** (*Axes object*) – The axes position that the pane should be placed in.
- **system** (System) – The whole system information.

1.7 pylj.util

class pylj.util.**System**(*number_of_particles*, *temperature*, *box_length*, *constants*, *forcefield*, *mass*, *init_conf='square'*, *timestep_length=1e-14*, *cut_off=15*)
Bases: object

Simulation system. This class is designed to store all of the information about the job that is being run. This includes the particles object, as will as sampling objects such as the temperature, pressure, etc. arrays. :param number_of_particles: Number of particles to simulate. :type number_of_particles: int :param temperature: Initial temperature of the particles, in Kelvin. :type temperature: float :param box_length: Length of a single dimension of the simulation square, in Angstrom. :type box_length: float :param init_conf: The way that the particles are initially positioned. Should be one of:

- 'square'
- 'random'

Parameters

- **timestep_length** (*float (optional)*) – Length for each Velocity-Verlet integration step, in seconds.
- **cut_off** (*float (optional)*) – The distance apart that the particles must be to consider there interaction to be negliable.
- **constants** (*float, array_like (optional)*) – The values of the constants for the forcefield used.
- **mass** (*float (optional)*) – The mass of the particles being simulated.
- **forcefield** (*function (optional)*) – The particular forcefield to be used to find the energy and forces.

accept()
Maps to the mc.accept function.

compute_energy()
Maps to the compute_energy function in either the comp (if Cython is installed) or the pairwise module and allows for a cleaner interface.

compute_force()
Maps to the compute_force function in either the comp (if Cython is installed) or the pairwise module and allows for a cleaner interface.

heat_bath(*bath_temperature*)
Maps to the heat_bath function in either the comp (if Cython is installed) or the pairwise modules. :param target_temperature: The target temperature for the simulation. :type target_temperature: float

integrate
Maps the chosen integration method. :param method: The integration method to be used, e.g. md.velocity_verlet. :type method: method

mc_sample()
Maps to the mc.sample function. :param energy: Energy to add to the sample :type energy: float

md_sample()
Maps to the md.sample function.

new_random_position()
Maps to the mc.get_new_particle function.

number_of_pairs()
Calculates the number of pairwise interactions in the simulation. :returns: Number of pairwise interactions in the system. :rtype: int

random()
Sets the initial positions of the particles in a random arrangement.

reject()
Maps to the mc.reject function.

select_random_particle()
Maps to the mc.select_random_particle function.

square()
Sets the initial positions of the particles on a square lattice.

pylj.util.**particle_dt**()
Builds the data type for the particles, this consists of: - xposition and yposition - xvelocity and yvelocity - xacceleration and yacceleration - xprevious_position and yprevious_position - xforce and yforce - energy

CHAPTER 2

using pylj

There are a number of ways to get using pylj:

- Fork the code: please feel free to fork the code on GitHub and add functionality that interests you.
- Run it locally: pylj is available through the pip package manager. However, it does require a Jupyter notebooks environment. Therefore we recommend using the anaconda package, which gives Jupyter notebooks as well as packages such as numpy and matplotlib. For best performance a C++ compiler is required, usually the Visual C++ package is best (Windows).
- Use the web app: pylj is available online at the following link.
- Get in touch: Andrew is always keen to chat to potential users or educators so feel free to drop him an email.

CHAPTER 3

bring your own forcefield

Although pylj was originally designed to use only the Lennard-Jones potential model, hence pyLennard-Jones. Following the release of pylj-1.1.0, it is now possible to pass custom forcefields to the simulation.

A quick caveat is that currently the particle size in the `cellview` is defined only by the box size, therefore not by the atom being simulated. The result is that if you try and simulated something significantly different in size to argon, things might look a bit funny. If you have any ideas of how to fix this please check this issue in the GitHub repository.

Writing your own forcefield and passing it to the pylj engine is very simple, firstly the forcefield should have the following form,

```
def forcefield(dr, constants, force=False):
    if force:
        return force
    else:
        return energy
```

An example of this for the Lennard-Jones forcefield (which is the default for pylj) and be found in the *pylj.forcefields* module.

It is necessary to pass this forcefield to the pylj engine. This can be achieved during the initialisation of the `System` class object, by passing the defined function as the variable `forcefield`. This can be seen for the Lennard-Jones forcefield in the `System` class definition in the *pylj.util* module.

CHAPTER 4

visualisation

4.1 existing classes

pylj comes with four different methods for visualising the simulation that is taking place:

- JustCell
- RDF
- Scattering
- Interactions
- Energy

Full information about the existing classes can be found in the *pylj.sample* class documentation.

4.2 building your own class

Using the inbuild tools it is straightforward to build your own custom sample class, or to have students design their own.

Each sample class consists of at least an `__init__` function and an `update` function. The `__init__` function should select the number of panes in the visualisation environment by calling the `sample.environment(n)` function and setup the various windows using the different setup functions. The `update` function then updates the panes each time it is called, this function should consist of a series of update functions related to each pane. A commented example of the JustCell sample class is shown below.

```
class JustCell(object):
    """The JustCell class will plot just the particle positions.
    This is a simplistic sampling class for quick visualisation.

    Parameters
    ----------
```

(continues on next page)

(continued from previous page)

```
    system: System
        The whole system information.
    """
    def __init__(self):
        # First the fig and ax must be defined. These are the
        # Matplotlib.figure.Figure object for the whole
        # visualisation environment and the axes object (or
        # array of axes objects) related to the individual panes.
        fig, ax = environment(1)

        # This is a setup function (detailed below) for the
        # particle positions
        setup_cellview(ax, system)

        # Generally looks better with the tight_layout function
        # called
        plt.tight_layout()

        # The Matplotlib.figure.Figure and axes object are set to
        # self variables for the class
        self.ax = ax
        self.fig = fig

    def update(self, system):
        """This updates the visualisation environment. Often this
        can be slower than the cythonised force calculation so
        use this wisely.

        Parameters
        ----------
        system: System
            The whole system information.
        """
        # This is the update function (detailed below) for the
        # particle positions
        update_cellview(self.ax, system)

        # The use of the `%matplotlib notebook' magic function in
        # the Jupyter notebook means that the canvas must be
        # redrawn
        self.fig.canvas.draw()
```

The `setup_cellview` and `update_cellview` functions have the following form.

```
def setup_cellview(ax, system):
    """Builds the particle position visualisation pane.

    Parameters
    ----------
    ax: Axes object
        The axes position that the pane should be placed in.
    system: System
        The whole system information.
    """
    # Assign the particles x and y coordinates to new lists
    xpos = system.particles['xposition']
    ypos = system.particles['yposition']
```

(continues on next page)

(continued from previous page)

```
    # This simply defines the size of the particle such that
    # it is proportional to the box size
    mk = (1052.2 / (system.box_length - 0.78921) - 1.2174)

    # Plot the initial positions of the particles
    ax.plot(xpos, ypos, 'o', markersize=mk,
            markeredgecolor='black', color='#34a5daff')

    # Make the box the right size and remove the ticks
    ax.set_xlim([0, system.box_length])
    ax.set_ylim([0, system.box_length])
    ax.set_xticks([])
    ax.set_yticks([])

def update_cellview(ax, system):
    """Updates the particle positions visualisation pane.

    Parameters
    ----------
    ax: Axes object
        The axes position that the pane should be placed in.
    system: System
        The whole system information.
    """
    # Assign the particles x and y coordinates to new lists
    xpos = system.particles['xposition']
    ypos = system.particles['yposition']

    # The plotted data is accessed as an object in the axes
    # object
    line = ax.lines[0]
    line.set_ydata(ypos)
    line.set_xdata(xpos)
```

Hopefully, it is clear how a custom enivornment could be created. Currently there are functions to setup and update the following panes:

- `cellview`: the particle positions
- `rdfview`: the radial distribution function
- `diffview`: the scattering profile
- `msdview`: the mean squared deviation against time
- `pressureview`: the instantaneous pressure against time
- `tempview`: the instantaneous temperature against time
- `forceview`: the total force against time
- `energyview`: the total energy against time

For those plotted against time, the samples are stored as np.arrays in the System object. To design a new sampling pane based on a different variable it may be necessary to impliment this in the System class, and the sampling of it would be added to the sample function in the particular module being used e.g. `md`.

CHAPTER 5

what is this?

This is the documentation for the open-source Python project, pylj. A library designed to facilitate student interaction with classical simulation. For more general information about pylj visit our page on pythoninchemistry.org or visit the GitHub repo for information about how to contribute.

CHAPTER 6

how to cite pylj

If you use this code in a teaching laboratory or a publication we would greatly appreciate if you would cite the JOSE article DOI and the Zenodo DOI for the specific version you use. If you are unsure which version you are using run `pylj.util.__cite__()` in the Jupyter notebook to launch the a webpage to the Zenodo page.

CHAPTER 7

indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

U

V