Package 'particles'

February 26, 2018

Type Package

Title A Graph Based Particle Simulator Based on D3-Force

Version 0.2.1

Date 2018-02-22

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description Simulating particle movement in 2D space has many application. The 'particles' package implements a particle simulator based on the ideas behind the 'd3-force' 'JavaScript' library. 'particles' implements all forces defined in 'd3-force' as well as others such as vector fields, traps, and attractors.

License MIT + file LICENSE

SystemRequirements C++11

Encoding UTF-8

LazyData true

Imports tidygraph, rlang, igraph, stats, magrittr, Rcpp, mgcv, digest, dplyr

RoxygenNote 6.0.1.9000

LinkingTo Rcpp

Suggests knitr, rmarkdown, ggraph

VignetteBuilder knitr

URL https://github.com/thomasp85/particles

BugReports https://github.com/thomasp85/particles/issues

NeedsCompilation yes

Author Thomas Lin Pedersen [aut, cre], Andrei Kashcha [ctb]

Repository CRAN

Date/Publication 2018-02-26 12:11:45 UTC

R topics documented:

`particles-package` *particles: A Graph Based Particle Simulator Based on D3-Force*

Description

Simulating particle movement in 2D space has many application. The 'particles' package implements a particle simulator based on the ideas behind the 'd3-force' 'JavaScript' library. 'particles' implements all forces defined in 'd3-force' as well as others such as vector fields, traps, and attractors.

Author(s)

Maintainer: Thomas Lin Pedersen <thomasp85@gmail.com>

Other contributors:

- Andrei Kashcha [contributor]

References

See also the GitHub page for the original JavaScript implementation in D3 by Mike Bostock

See Also

Useful links:

- https://github.com/thomasp85/particles
- Report bugs at https://github.com/thomasp85/particles/issues

---

`center_force` *Center all particles around the origin without affecting velocity*

Description

This force repositions the particles at each generation so they are centered around (0,0). It does not affect the velocity of the particles and are thus mainly a guard against the whole body of particles drifting off.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `x` : The x position to center around (*tidy eval*)
- `y` : The y position to center around (*tidy eval*)

See Also

Other forces: `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

`collision_force` *Models particles as circles with a given radius and pushes overlapping particles apart*

Description

This force pushes overlapping particles apart by assigning a radius to each particle, treating them as circles, and searches for overlaps through an optimised quad tree algorithm.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `strength` : A dampening of the repulsion between overlapping circles. This allows the force to iterate towards the optimal solution through iterative relaxation. Should be a number between 0 and 1. Defaults to 0.7
- `radius` : The radius of each particle. Defaults to 1 (*tidy eval*)
- `n_iter` : The number of iterations to perform in the iterative relaxation. Defaults to 1.

See Also

Other forces: `center_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

`dominator_constraint` *Restrict child position based on parent position*

Description

This constraint requires children to be positioned at a certain side of their parent and with a certain distance. It can be used to enforce a layering of particles for e.g. DAG and tree layouts.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `distance` : The minimum orthogonal distance to the parent. Default to `0`, meaning that children are only required to be positioned to the specific side of their parent. (*tidy eval*)
- `angle` : The direction the children should be enforced to be relative to their parent. Defaults to `-pi/2` which is equivalent to down. (*tidy eval*)

See Also

Other constraints: `infinity_constraint`, `path_constraint`, `polygon_constraint`, `velocity_constraint`, `x_constraint`, `y_constraint`

`evolve` *Move the simulation forward one or more steps*

Description

This is the function that move the simulation forward in time. It is possible to either specify the number of steps that should be simulated or let the simulation terminate as `alpha_min` is reached. Note that some values of `alpha` and `alpha_target` does not allow alpha to converge to `alpha_min` so letting the simulation self-terminate can result in an infinite loop. The default settings will result in `alpha_min` being reached in 300 generations.

Usage

```
evolve(simulation, steps = NULL, on_generation = NULL, ...)
```

Arguments

`simulation`	A simulation object
`steps`	The number of generations to progress or a function getting the simulation object and returns `TRUE` if the simulation should proceed and `FALSE` if it should stop. If `NULL` the simulation will run until `alpha_min` has been reached.
`on_generation`	A function to be called after each generation has been progressed. The function will get the current state of the simulation as the first argument. If the function returns a simulation object it will replace the current simulation from the next generation. In the case of any other return type the return will be discarded and the function will have no effect outside its side-effects.
`...`	Additional arguments to `on_generation`

Details

Each generation in the simulation progress in the following manner:

1. Check whether the specified number of generations has been reached
2. Check whether `alpha_min` has been reached
3. If either 1. or 2. is true, terminate the simulation
4. Apply the forces on the current particle positions and velocities in the order they have been added
5. Reduce the velocity according to the given `velocity_decay`
6. Update the position and velocity based on any provided constraints
7. Calculate the new particle positions based on the new velocity
8. If given, call the `on_generation` function.

Value

A simulation object with updated positions and velocities

Examples

```
graph <- tidygraph::create_notable('folkman')
sim <- graph %>%
  simulate() %>%
  wield(link_force) %>%
  wield(manybody_force)

# Take 5 steps and tell about it
sim %>% evolve(5, function(sim) {
  cat('Generation: ', evolutions(sim), '\n', sep = '')
})

# Run evolution until alpha_min is reached
sim %>% evolve(NULL)
```

`field_force` *Apply a vector field to particles*

Description

This force adjusts the velocity of particles based on a supplied vector field. The vector field can either be specified using x and y velocities, or angle and magnitude. Velocity adjustments are calculated based on a bilinear interpolation.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `x` : A matrix giving the velocity in the x direction at each grid point
- `y` : A matrix giving the velocity in the y direction at each grid point
- `angle` : A matrix giving the direction of the velocity at each grid point. Will only be considered if `x` and `y` are missing.
- `vel` : A single numeric or a matrix of the same dimensions as `angle` giving the magnitude of velocity at each grid point.
- `xlim` : The coordinate span of the vector field in the x direction.
- `ylim` : The coordinate span of the vector field in the y direction.

See Also

Other forces: `center_force`, `collision_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

genesis *Particle initialisation*

Description

These functions are passed to the simulation and defines how the position and velocity of the particles are initiated. The default is to lay out the nodes in a phyllotactic arrangement (think sunflower seeds) and with no velocity, which is also the default in d3-force.

Usage

```
phyllotactic_genesis(radius = 10, angle = pi * (3 - sqrt(5)))

predefined_genesis(x, y, x_vel = 0, y_vel = 0)

bigbang_genesis(vel_min = 0, vel_max = 1)

aquarium_genesis(width = 10, height = 10, vel_min = 0, vel_max = 1)

petridish_genesis(max_radius = 10, vel_min = 0, vel_max = 1)
```

Arguments

`radius`	The radius modifier (will be multiplied by the square root of the index of the particle)
`angle`	The angular difference between two adjacent particles
`x, y`	The columns holding (or value of) the position coordinates
`x_vel, y_vel`	The columns holding (or value of) the velocity verlets
`vel_min, vel_max`	The bounds of the uniformly distributed velocities
`width, height`	The size of the rectangle holding the particles
`max_radius`	The size of the disc.

Value

A function that takes the particle graph and returns a list with a position and velocity element, each holding a matrix with two columns and a row for each particle giving the x and y position and velocity respectively.

Functions

- `phyllotactic_genesis`: Initiates particles in a phyllotactic arrangement with zero velocity
- `predefined_genesis`: Uses information from the node data to set position and velocity.
- `bigbang_genesis`: Initiates particles at center position and a random velocity

- aquarium_genesis: Places particles randomly in a rectangle and gives them a random velocity
- petridish_genesis: Places particles randomly on a disc and gives them a random velocity

Examples

```
# A contrieved example
graph <- tidygraph::create_notable('bull')
genesis <- phyllotactic_genesis()
genesis(graph)

# Usually used as an argument to simulate
graph %>%
  simulate(setup = phyllotactic_genesis())
```

---

impose *Assign a force or constraint to a simulation*

Description

This function adds a new force/constraint to the simulation and trains the it on the current particle graph. The parameters passed on to the training are using tidy evaluation from the rlang package. Depending on the force/constraint the data getting referenced is either the node or the edge data of the particle graph. Both forces and constraints manipulate position and velocity of the particles but they differ in when the are applied during a generation. First forces are applied sequentially and the resulting velocity is added to the resulting position after velocity_decay has been applied. After this operation any constraint is imposed on the results. In general, forces tends to calculate velocity adjustments, while constraints modify position and velocity directly, but this difference is not in any way enforced.

Usage

```
impose(simulation, constraint, ..., name, include = TRUE)

reimpose(simulation, name, ...)

unimpose(simulation, name)

wield(simulation, force, ..., name, include = TRUE)

rewield(simulation, name, ...)

unwield(simulation, name)
```

Arguments

simulation	A simulation object
constraint	A constraint object
...	Parameters passed on to the training of the force or constraint
name	The name of the force. For use when accessing the force at a later stage. If no name is given the force is accessible by its index in the stack.
include	The particles to be affected by this force. Defaults to every particle in the simulation (*tidy eval*)
force	A force object

Details

`wield()` and `impose()` adds forces and constraints to the simulation respectively. `unwield()` and `unimpose()` removes forces and constraints based on name or index. `rewield()` and `reimpose()` modifies existing forces and constraints based on name or index and retrains them.

Value

A simulation with the force or constraint added

Examples

```
graph <- tidygraph::create_notable('folkman')
graph %>%
  simulate() %>%
  wield(link_force)
```

---

infinity_constraint *Reposition particles outside a canvas so they wrap around*

Description

This constraint keeps particles inside of a defined area by positioning exiting particles on the other side of the area. In effect this makes particles that moves outside the upper bound reenter at the lower bound and vice versa.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `xlim` : The left and right bound of the area
- `ylim` : The upper and lower bound of the area

See Also

Other constraints: dominator_constraint, path_constraint, polygon_constraint, velocity_constraint, x_constraint, y_constraint

---

`link_force` *Attract or repel linked particles*

Description

This force works between linked particles and either attracts or repel them from each other depending on the value of the strength and distance parameters. The force is stronger the longer the linked particles are from each other, mimicking the mechanics of a rubber band.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `strength` : The attractive force between the linked particles. The default weighs edges from low-degree particles higher (`strength = 1 / (min(degree(from), degree(to)))`). (*tidy eval*)
- `distance` : The desired distance between linked particles. Defaults to `30` (*tidy eval*)
- `n_iter` : The number of iteration towards the optimal solution per generation. Higher values leads to faster convergence (measured in number of generations) at the expense of longer computation time per generation. Defaults to `1`.

See Also

Other forces: center_force, collision_force, field_force, manybody_force, map_force, mean_force, random_force, reset_force, trap_force, x_force, y_force

---

`manybody_force` *Model attraction or repulsion between all particles in the system*

Description

This force implements a n-body simulation using the Barnes-Hut approximation for improved performance. An n-body simulation calculates attraction or repulsion between all particles in a system based on their relative distances and each particles capacity and can thus mimick gravity or electrostatic repulsion.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `strength` : The attractive or repulsive force of the particles. If positive the particle attracts, if negative the particle repulses. The default is -30. (*tidy eval*)
- `theta` : The Barnes-Hut criterion governing the precision of the approximation. If 0, no approximation is made. Defaults to 0.9.
- `min_dist` : A lower distance threshold below which the forces will be damped, in order to avoid explosive forces when two particles gets very near each other.
- `max_dist` : A distance threshold above which the forces between particles are ignored. Using this will result in more local changes.

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

`map_force` *Apply a map to particles*

Description

In mathematics, maps are a functions that translates its input into new values. In the context of particles a map is a translation function that translates the current particle positions to a new one

Details

Normally a map has no notion of velocity — it simply translates positions. In particles it is possible to decide whether positions should be modified directly or whether the translation magnitude should be added to the velocity verlet using the `fixed` parameter.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `map` : A function that accepts the particle position matrix and returns the new positions in the same format.
- `fixed` : Logical. Should position be modified directly (`TRUE`) or should the translation be added to the velocity verlet (`FALSE`)

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

mean_force *Apply the mean velocity of all the neighbors to a particle*

Description

This force takes the mean of all the neighbors (in the graph sense) of a particle (and optionally itself) and applies it to itself.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `include_self` : Should the velocity of itself be included in the mean calculation
- `mode` : How should neighbors be found? `'all'` uses all edges. `'out'` only uses outbound edges, and `'in'` only uses inbound edges. Ignored for undirected particle graphs

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `random_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

path_constraint *Limit particle position to be along a path or outline*

Description

This constraint repositions particles to their closest point along a given path and sets their velocity to zero.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `path` : A two column matrix giving the path, or a list of matrices to use multiple disconnected paths.
- `closed` : Should the path close on itself. Defaults to `FALSE`

See Also

Other constraints: `dominator_constraint`, `infinity_constraint`, `polygon_constraint`, `velocity_constraint`, `x_constraint`, `y_constraint`

polygon_constraint *Fixes particles to be inside a polygon*

Description

This constraint prevents particles from moving outside of one or more polygons. If a particle ventures outside it will be moved back to its closest point inside the specified polygon(s) and have its velocity set to zero.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `polygon` : A two column matrix giving the polygon, or a list of matrices to use multiple polygons. Overlapping polygons will be subtracted from each other so it is possible to define polygons with holes.

See Also

Other constraints: `dominator_constraint`, `infinity_constraint`, `path_constraint`, `velocity_constraint`, `x_constraint`, `y_constraint`

random_force *Modify the velocity randomly at each step*

Description

This force applies a random velocity modification to all particles. The modification is uniformly distributed and bound be the parameters provided during initialisation.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `xmin, xmax` : The bounds of the modification in the horizontal direction
- `ymin, ymax` : The bounds of the modification in the vertical direction

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `reset_force`, `trap_force`, `x_force`, `y_force`

reset_force *Reset the velocity verlet of particles to a fixed value*

Description

This force resets the velocity of particles at each generation. It can be used if each generation should start from the same foundation rather than accumulate as the simulation evolve. Particles where the parameters evaluates to NA will ignore this force.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `xvel` : The x-velocity to reset to at each generation (*tidy eval*)
- `yvel` : The y-velocity to reset to at each generation (*tidy eval*)

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `trap_force`, `x_force`, `y_force`

simulate *Start a simulation based on a graph*

Description

This function initiates a simulation based on the provided graph and parameters. Any graph structure with a `tidygraph::as_tbl_graph()` method is supported as input. This function does not start the simulation but merely sets it up.

Usage

```
simulate(graph, alpha = 1, alpha_min = 0.001, alpha_decay = 1 -
  alpha_min^(1/300), alpha_target = 0, velocity_decay = 0.4,
  setup = phyllotactic_genesis(), ...)

is.simulation(x)

record(simulation, ...)

clear_history(simulation)

get_history(simulation, age = -1)

history_length(simulation)
```

```
reheat(simulation, alpha)

particles(simulation)

position(simulation)

velocity(simulation)

evolutions(simulation)
```

Arguments

graph	A graph in a format supported by tidygraph
alpha	The starting alpha value. See Details.
alpha_min	The minimum alpha value after which the simulation is terminated. See Details.
alpha_decay	The speed at which the alpha value decreases. See Details.
alpha_target	The alpha value that alpha drifts towards. See Details.
velocity_decay	The dampening factor of the system. See Details.
setup	A function that takes the particle graph and returns a start position and velocity to each particle. particles provides a range of genesis functions to choose from.
...	Additional parameters for the simulation (currently ignored)
x, simulation	A simulation object
age	The version to retrieve. Positive numbers count from the beginning, while negative numbers counts backwards from current version. Defaults to -1.

Details

A simulation in the context of the particles package is a series of equidistant steps where the velocity and position of each particle is updated. A few global rules applies to this cycle irrespectively of the forces added to the simulation. Once a simulation is initiated an alpha value is defined (defaults to 1). At each step this alpha value is decreased according to its distance to the alpha_target (defaults to 0) and alpha_decay (defaults to ~0.023). Once the alpha value gets below alpha_min (defaults to 0.001) the simulation seizes to take additional steps. The default values is adapted from the d3-force implementation and corresponds to 300 steps. Conceptually the alpha progression can be seen as a cooling off of the system as the value decreases quickly in the beginning and then slowly reach the target value. If it is not intended to have a system that cools off, simply set the alpha_target value to the same as alpha. At each step, after the new particle velocities has been calculated but before they have been applied to the positions, a dampening factor (velocity_decay) is applied in order to simulate the gradual loss of momentum. If this is not intended for the simulation, simply set the value to 0.

Value

A simulation object

Functions

- `record`: Save the current state in the simulation's history
- `clear_history`: Clear the current history from the simulation
- `get_history`: Retrieve a simulation from the history
- `history_length`: Get the number of versions stored in the history of the simulation
- `reheat`: set the cooling of the simulation to a new value
- `particles`: Extract the particle graph from a simulation
- `position`: Extract the position coordinates from a simulation
- `velocity`: Extract the velocity verlets from a simulation
- `evolutions`: Get the number of generations the simulation has undergone

Examples

```
graph <- tidygraph::create_notable('folkman')
graph %>%
  simulate()
```

---

simulation_modification

Modify the particles in a simulation

Description

The particles that are modelled in a simulation are encoded as a `tbl_graph`, giving support for the particles as well as their interactions (nodes and edges in graph parlor). A simulation supports a subset of the tidygraph/dplyr verbs in order to allow modification of the particles after they have been included in the simulation. In general it is possible to add and remove particles and interactions as well as modify the metadata associated with them. The API follows the tidygraph API where `activate()` is used to select either particles or interactions and subsequent operations are thus related to the last activated datatype. The simulation is automatically retrained after modifying the state of the particles and their interactions.

Usage

```
add_particles(.data, ..., interactions = NULL, setup = NULL)

add_interaction(.data, ...)
```

Arguments

`.data`	A simulation object
`...`	Parameters passed on to the main verbs in tidygraph/dplyr
`interactions`	A data.frame of interactions/edges to add along with the particles
`setup`	A function to calculate the starting conditions for the particles. It receives all particles with the current position and velocity encoded in the `x`, `y`, `x_vel`, and `y_vel` columns. New particle will have NA. The function must return a position and velocity for all particles even though the values for the current particles will be discarded. If NULL it will use the genesis function used when creating the simulation.

Value

A simulation object

See Also

`dplyr::mutate()`, `dplyr::mutate_at()`, `dplyr::mutate_all()`, `dplyr::filter()`, `dplyr::slice()`, `tidygraph::activate()`, `tidygraph::bind_nodes()`, `tidygraph::bind_edges()`

`trap_force` *Attract and trap particles within polygons*

Description

This force creates a trap based on any type of polygon that attracts particles as long as they are outside the polygon, while leaving particles inside the polygon unaffected. The trap as such has no walls and particles are allowed to leave it, but they will be pulled back as soon as they exits the polygon.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `polygon` : A two column matrix giving the corners of the polygon, or a list of matrices to use multiple polygons. If multiple polygons are overlapping it is considered a hole.
- `strength` : The attractive force applied to the particle. Particles are attracted towards the closest part of the polygon, rather than the center, and the attraction is stronger for particles moving away from the polygon than for those moving towards it. (*tidy eval*)
- `min_dist` : A lower distance threshold below which the strength is not increased. The attraction of the trap falls of with the square of the distance to the particle, so particles close by can get an enormous attraction unless this threshold is set (so much that the shoot out of the other side of the trap).
- `distance_falloff` : How should the attractive force deteriorate with the distance between the polygon and the particle. Defaults to 2 (quadratic falloff) (*tidy eval*)

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `x_force`, `y_force`

`velocity_constraint` *Limits particles to a specific velocity range*

Description

This constraint puts bounds on the magnitude of velocity a particle can have. Particles where either end of the bound is NA ignores the constraint. If a particle with no velocity is forced to have a velocity the direction will be random.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `v` : The velocity allowed for the particle. (*tidy eval*)
- `vmin` : The lowest permissible velocity. If `NULL` then `v` will be used. (*tidy eval*)
- `vmax` : The highest permissible velocity. If `NULL` then `v` will be used. (*tidy eval*)

See Also

Other constraints: `dominator_constraint`, `infinity_constraint`, `path_constraint`, `polygon_constraint`, `x_constraint`, `y_constraint`

`x_constraint` *Fixes particles to a horizontal position*

Description

This constraint simply prevents particles from moving in the x direction. For particles where the constraint evaluates to `NA` this constraint is ignored. If the constraint is enforced the velocity in the x direction will be set to `0`.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `x` : The position on the x-axis to fix to. (*tidy eval*)
- `xmin` : The lowest permissible x-value. If `NULL` then `x` will be used. (*tidy eval*)
- `xmax` : The highest permissible x-value. If `NULL` then `x` will be used. (*tidy eval*)

See Also

Other constraints: dominator_constraint, infinity_constraint, path_constraint, polygon_constraint, velocity_constraint, y_constraint

---

`x_force` *Attract particles towards a horizontal position*

Description

This force simply pulls particles towards a fixed position on the x-axis.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `strength` : The strength with which the attraction occurs (*tidy eval*)
- `x` : The position on the x-axis to pull towards. (*tidy eval*)

See Also

Other forces: center_force, collision_force, field_force, link_force, manybody_force, map_force, mean_force, random_force, reset_force, trap_force, y_force

---

`y_constraint` *Fixes particles to a vertical position*

Description

This constraint simply prevents particles from moving in the y direction. For particles where the constraint evaluates to `NA` this constraint is ignored. If the constraint is enforced the velocity in the y direction will be set to `0`.

Training parameters

The following parameters defines the training of the constraint and can be passed along a call to `impose()`

- `y` : The position on the y-axis to fix to. (*tidy eval*)
- `ymin` : The lowest permissible y-value. If `NULL` then `y` will be used. (*tidy eval*)
- `ymax` : The highest permissible y-value. If `NULL` then `y` will be used. (*tidy eval*)

See Also

Other constraints: dominator_constraint, infinity_constraint, path_constraint, polygon_constraint, velocity_constraint, x_constraint

`y_force` *Attract particles towards a vertical position*

Description

This force simply pulls particles towards a fixed position on the y-axis.

Training parameters

The following parameters defines the training of the force and can be passed along a call to `wield()`

- `strength` : The strength with which the attraction occurs (*tidy eval*)
- `y` : The position on the y-axis to pull towards. (*tidy eval*)

See Also

Other forces: `center_force`, `collision_force`, `field_force`, `link_force`, `manybody_force`, `map_force`, `mean_force`, `random_force`, `reset_force`, `trap_force`, `x_force`

Index