Package ‘puff’

April 10, 2025

Title Simulate and Visualize the Gaussian Puff Forward Atmospheric Model

Version 0.1.0

Description Simulate and run the Gaussian puff forward atmospheric model in sensor (specific sensor coordinates) or grid (across the grid of a full oil and gas operations site) modes, following Jia, M., Fish, R., Daniels, W., Sprinkle, B. and Hammerling, D. (2024) <doi:10.26434/chemrxiv-2023-hc95q-v3>. Numerous visualization options, including static and animated, 2D and 3D, and a site map generator based on sensor and source coordinates.

License MIT + file LICENSE

Encoding UTF-8

URL `https://github.com/Hammerling-Research-Group/puff`

BugReports `https://github.com/Hammerling-Research-Group/puff/issues`

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports ggplot2, dplyr, tidyr, magrittr, htmlwidgets, patchwork, plotly, scales, tidyselect

Suggests knitr, rmarkdown, devtools, akima, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Teagan Ward [aut],
Philip Waggoner [aut, cre],
Will Daniels [aut],
Meng Jia [aut],
Dorit Hammerling [aut, ths]

Maintainer Philip Waggoner <`philip.waggoner@mines.edu`>

Repository CRAN

Date/Publication 2025-04-10 09:40:02 UTC

Contents

compute_sigma_vals . . . 2
create_site_map . . . 3
faceted_time_series_plot . . . 3
faceted_time_series_plot2 . . . 5
get_stab_class . . . 6
gpuff . . . 7
interpolate_wind_data . . . 8
is_day . . . 8
plot_2d_animated . . . 9
plot_3d_animated . . . 11
simulate_grid_mode . . . 12
simulate_sensor_mode . . . 15
single_emission_rate_plot . . . 17
time_series_plot . . . 18
wind_vector_convert . . . 19

Index **20**

`compute_sigma_vals` *Compute Sigma Values Based on Stability Class and Distance*

Description

Compute Sigma Values Based on Stability Class and Distance

Usage

```
compute_sigma_vals(stab_class, total_dist)
```

Arguments

`stab_class`	Character vector of stability classes ("A" to "F")
`total_dist`	Numeric vector of distances in km (must match length of stab_class or be scalar)

Value

2-row matrix with sigma_y (row 1) and sigma_z (row 2) values

Examples

```
out <- compute_sigma_vals("A", 0.7)
```

`create_site_map` *Create a Site Map of Sensors and Sources*

Description

This function generates a site map with an adjacent compass rose.

Usage

```
create_site_map(sensors, sources, text_size = 12)
```

Arguments

`sensors`	Coordinates for sensor locations.
`sources`	Coordinates for source locations.
`text_size`	Numeric. Font size for labels. Default is 12.

Value

A patchwork-combined ggplot object: site map + compass rose.

Examples

```
source_coords <- c(0, 0, 2.5)

n_sensors <- 8
radius <- 20
z_height <- 2.0

angles <- seq(0, 2 * pi, length.out = n_sensors + 1)[- (n_sensors + 1)]

sensor_coords <- matrix(
  cbind(radius * cos(angles), radius * sin(angles), rep(z_height, n_sensors)),
  ncol = 3
)

create_site_map(sensor_coords, source_coords)
```

`faceted_time_series_plot` *Faceted Time Series Plot of Methane Concentrations and Wind Data*

Description

This function creates a faceted bubble plot of methane concentrations and a shared wind rose plot.

Usage

```
faceted_time_series_plot(
  sensor_concentrations,
  sensor_coords,
  wind_data,
  start_time,
  end_time,
  output_dt,
  text_size = 12
)
```

Arguments

`sensor_concentrations`	Data frame. Output from simulate_sensor_mode().
`sensor_coords`	A data frame or matrix containing sensor locations.
`wind_data`	A list with wind_u and wind_v components.
`start_time`	POSIXct start of simulation.
`end_time`	POSIXct end of simulation.
`output_dt`	Time step (in seconds) for aligning wind data with concentration data.
`text_size`	Default at 12.

Value

A ggplot object: faceted concentration plot + single wind rose plot.

Examples

```
set.seed(123)
sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)
source_coords <- c(0, 0, 2.5)
sensor_coords <- matrix(c(-6.525403221327715e-15, -35.52264, 2.01775), ncol = 3, byrow = TRUE)

out <- simulate_sensor_mode(
  start_time, end_time, source_coords,
 emission_rate, wind_data, sensor_coords, sim_dt, puff_dt, output_dt, puff_duration = 1200
)

faceted_time_series_plot(out, sensor_coords,
  wind_data, as.POSIXct(start_time), as.POSIXct(end_time),
  output_dt
  )
```

`faceted_time_series_plot2`

Alternate version with wind rose at each time step + scatter plot of methane concentration time series

---

Description

Alternate version with wind rose at each time step + scatter plot of methane concentration time series

Usage

```
faceted_time_series_plot2(
  sensor_concentrations,
  sensor_coords,
  wind_data,
  start_time,
  end_time,
  output_dt,
  text_size = 12
)
```

Arguments

`sensor_concentrations`	Data frame. Output from a sensor simulation function, which must include a column named "Group.1" which contains the timestamps (e.g., "YYYY-MM-DD HH:MM:SS") and a column "Sensor_1" for the sensor concentration values.
`sensor_coords`	A data frame or matrix containing sensor locations.
`wind_data`	A list containing wind data with components u and v
`start_time`	POSIXct. Start time of the simulation.
`end_time`	POSIXct. End time of the simulation.
`output_dt`	Integer. Desired time resolution (in seconds) for the final output of concentrations.
`text_size`	Default at 12.

Value

A ggplot object with faceted time series plots of methane concentrations and wind rose data.

Examples

```
set.seed(123)
sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)
source_coords <- c(0, 0, 2.5)
sensor_coords <- matrix(c(-6.525403221327715e-15, -35.52264, 2.01775), ncol = 3, byrow = TRUE)

out <- simulate_sensor_mode(
  start_time, end_time, source_coords,
 emission_rate, wind_data, sensor_coords, sim_dt, puff_dt, output_dt, puff_duration = 1200
)

faceted_time_series_plot2(out, sensor_coords,
  wind_data, as.POSIXct(start_time), as.POSIXct(end_time),
  output_dt
  )
```

---

get_stab_class *Determine Stability Class Based on Wind Speed and Time of Day*

Description

This function calculates the stability class based on wind speed (U) and the time of day. It categorizes the atmosphere's stability as one of several classes (A-F) depending on the inputs.

Usage

```
get_stab_class(U, time)
```

Arguments

`U`	A numeric value representing the wind speed in meters per second.
`time`	A time value that is used to determine whether it's day or night.

Value

A character vector representing the stability class(es) ("A" to "F").

Examples

```
out <- get_stab_class(3, 12)
```

gpuff *Gaussian Puff Concentration Calculation*

Description

Calculates the concentration of an emission event at a specified location and time due to a Gaussian puff.

This function uses wind speed and direction components, advection adjustments, and stability class calculations to accurately measure the dispersion of a puff in the atmosphere.

Usage

```
gpuff(Q, stab_class, x_p, y_p, x_r_vec, y_r_vec, z_r_vec, total_dist, H, U)
```

Arguments

`Q`	Numeric. Mass per puff. E.g., for 100 puffs/hour of a 100 kg/hr emission, put 1 kg of mass into each puff.
`stab_class`	Character vector. Stability class ("A" to "F").
`x_p`	Numeric. Puff position in the X direction.
`y_p`	Numeric. Puff position in the Y direction.
`x_r_vec`	Numeric vector. The x-coordinate (east-west) where the concentration is calculated.
`y_r_vec`	Numeric vector. The y-coordinate (north-south) where the concentration is calculated.
`z_r_vec`	Numeric vector. The z-coordinate (height) where the concentration is calculated.
`total_dist`	Numeric. The total distance the puff has traveled from the source in m.
`H`	Numeric. Source height.
`U`	Numeric. Wind speed in m/s.

Value

Numeric. Pollutant concentration at the specified (x, y, z) locations and time ‘t‘.

Examples

```
out <- gpuff(Q = 1, stab_class = "D", x_p = 0, y_p = 0,
  x_r_vec = 100, y_r_vec = 0, z_r_vec = 2,
  total_dist = 100, H = 2, U = 5
)
```

interpolate_wind_data *Resample wind_speeds and wind_directions to the simulation resolution by interpolation*

Description

Resample wind_speeds and wind_directions to the simulation resolution by interpolation

Usage

```
interpolate_wind_data(wind_speeds, wind_directions, sim_start, sim_end, puff_dt)
```

Arguments

`wind_speeds`	A list of float values of wind speeds in m/s at each time stamp
`wind_directions`	A list of float values of wind directions in degrees at each time stamp following the conventional definition: 0 -> wind blowing from North, 90 -> E, 180 -> S, 270 -> W
`sim_start`	Date & time stamps of simulation start time
`sim_end`	Date & time stamps of simulation end time
`puff_dt`	A scalar time interval between two puffs

Value

Quantities corresponding to the conversion direction

Examples

```
out <- interpolate_wind_data(c(2, 3), c(90, 180),
  "2024-01-01 00:00:00", "2024-01-01 01:00:00", 300
)
```

is_day *Determine Whether a Time is During the Day*

Description

This function checks the time and classifies it as day or not

Usage

```
is_day(time)
```

Arguments

`time`	A time value that is used to determine whether it's day or night.

Value

A character T or F representing whether or not it is daytime.

Examples

```
out <- is_day(8)
```

`plot_2d_animated` *Plot a 2D Animated Heatmap for Concentration Over Time*

Description

This function generates a 2D animated heatmap using ‘plotly‘ to visualize the movement of a plume over time. The animation is based on grid concentration data from ‘simulate_grid_mode()‘ output.

Usage

```
plot_2d_animated(
  data,
  grid_coords,
  start,
  end,
  output_dt,
  frames = 100,
  transition = 99,
  save = FALSE,
  interpolate_grid = FALSE,
  granularity = 100
)
```

Arguments

`data`	A matrix or array of grid concentration results from ‘simulate_grid_mode()‘.
`grid_coords`	A list containing the same grid coordinates passed to ‘simulate_grid_mode()‘.
`start`	A character string specifying the start time of the simulation (e.g., "YYYY-MM-DD HH:MM:SS").
`end`	A character string specifying the end time of the simulation (e.g., "YYYY-MM-DD HH:MM:SS").
`output_dt`	A character string or numeric value specifying the time interval between outputs.
`frames`	Numeric. Duration between frames in the animation (milliseconds). Default is 100.

`transition`	Numeric. Duration for transitioning between frames (milliseconds). Default is 99.
`save`	Logical. If ‘TRUE‘, saves the plot as an HTML file named ‘2D_heatmap.html‘ and specifies saved location. Default set to ‘FALSE‘.
`interpolate_grid`	Logical. If ‘TRUE‘, applies interpolation to refine grid resolution and make the heatmap smoother. Default ‘FALSE‘.
`granularity`	Numeric. Sets the number of points in the finer grid resolution when ‘interpolate_grid = TRUE‘. Default ‘100‘.

Value

A ‘plotly‘ object representing the animated heatmap.

Examples

```
set.seed(123)

sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
source_coords <- c(0, 0, 2.5)
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)

grid_coords <- list(
  x = seq(-2, 2, by = 1),
  y = seq(-2, 2, by = 1),
  z = seq(0, 5, by = 1)
)

out <- simulate_grid_mode(
  start_time = start_time,
  end_time = end_time,
  source_coords = source_coords,
  emission_rate = emission_rate,
  wind_data = wind_data,
  grid_coords = grid_coords,
  sim_dt = sim_dt,
  puff_dt = puff_dt,
  output_dt = output_dt,
  puff_duration = 1200
)

plot_2d_animated(data = out,
  grid_coords = grid_coords,
  start = start_time,
  end = end_time,
  output_dt = output_dt)
```

`plot_3d_animated` *Plot a 3D Animated Plot for Concentration Over Time*

Description

This function generates a 3D animated plot (scatter or contour) using ‘plotly‘ to visualize the movement of a plume over time. The animation is based on grid concentration data from ‘simulate_grid_mode()‘ output.

Usage

```
plot_3d_animated(
  data,
  grid_coords,
  start,
  end,
  output_dt,
  frames = 100,
  transition = 99,
  plot_type = "contour",
  save = FALSE
)
```

Arguments

`data`	A matrix or array of grid concentration results from ‘simulate_grid_mode()‘.
`grid_coords`	A list containing the same grid coordinates passed to ‘simulate_grid_mode()‘.
`start`	A character string specifying the start time of the simulation (e.g., "YYYY-MM-DD HH:MM:SS").
`end`	A character string specifying the end time of the simulation (e.g., "YYYY-MM-DD HH:MM:SS").
`output_dt`	A character string or numeric value specifying the time interval between outputs.
`frames`	Numeric. Duration between frames in the animation (milliseconds). Default is 100.
`transition`	Numeric. Duration for transitioning between frames (milliseconds). Default is 99.
`plot_type`	Character. "contour" (default) or "scatter" to specify the type of plot.
`save`	Logical. If ‘TRUE‘, saves the plot as an HTML file named ‘2D_heatmap.html‘ and specifies saved location. Default set to ‘FALSE‘.

Value

A ‘plotly‘ object representing the animated plot.

Examples

```
set.seed(123)

sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
source_coords <- c(0, 0, 2.5)
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)

grid_coords <- list(
  x = seq(-2, 2, by = 1),
  y = seq(-2, 2, by = 1),
  z = seq(0, 5, by = 1)
)

out <- simulate_grid_mode(
  start_time = start_time,
  end_time = end_time,
  source_coords = source_coords,
  emission_rate = emission_rate,
  wind_data = wind_data,
  grid_coords = grid_coords,
  sim_dt = sim_dt,
  puff_dt = puff_dt,
  output_dt = output_dt,
  puff_duration = 1200
)

plot_3d_animated(out,
   grid_coords,
   start_time, end_time,
   output_dt)
```

`simulate_grid_mode` *Simulate Atmospheric Concentration on a Grid*

Description

Simulates methane concentrations at each grid point over time using the Gaussian puff forward model. Supports one or more emission sources. Each puff retains constant wind speed and direction throughout its lifetime, and corresponding dispersion parameters are determined at the time of emission.

Usage

```
simulate_grid_mode(start_time, end_time, source_coords, emission_rate, wind_data,
  grid_coords, sim_dt, puff_dt, output_dt, puff_duration, ws, wd)
```

Arguments

`start_time`	POSIXct. Start time of the simulation.
`end_time`	POSIXct. End time of the simulation.
`source_coords`	Numeric vector or matrix. Coordinates of the emission source(s) in meters (x, y, z). If simulating multiple sources, provide a matrix with one row per source. E.g., `matrix(c(0, 0, 2.5, 10, 10, 2.5), ncol = 3, byrow = TRUE)`.
`emission_rate`	Numeric. Emission rate in kg/hr per source. If multiple sources are provided, this value will be assumed the same for each. (Note: source-specific rates are not yet supported.)
`wind_data`	Data frame. Must contain either columns ‘wind_u‘ and ‘wind_v‘ (wind vector components in x/y directions) or columns representing wind speed and direction, declared as ‘ws‘ and ‘wd‘.
`grid_coords`	List. A list with three numeric vectors specifying the grid points for `x`, `y`, and `z` coordinates, e.g., `list(x = seq(-50, 50, by = 5), y = seq(-50, 50, by = 5), z = c(2.5))`.
`sim_dt`	Integer. Simulation time step in seconds (default = 1). Determines how frequently puff positions are updated.
`puff_dt`	Integer. Puff emission interval in seconds (default = 1). New puffs are emitted from each source at this frequency.
`output_dt`	Integer. Desired time resolution (in seconds) for final output concentrations.
`puff_duration`	Numeric. Maximum puff lifetime in seconds (default = 1200). Puffs beyond this age are discarded.
`ws`	Optional. String. Name of the column in ‘wind_data‘ containing wind speeds (m/s). Required if ‘wind_data‘ contains polar wind components instead of Cartesian (‘wind_u‘, ‘wind_v‘).
`wd`	Optional. String. Name of the column in ‘wind_data‘ containing wind directions (degrees from).

Details

- Each source (from one to many) emits puffs at intervals of `puff_dt`. - Each puff maintains a fixed wind vector and dispersion parameters. - Puffs are advected over time based on their individual wind vectors. - Concentration contributions from all active puffs are computed at each grid point and summed. - Concentrations are aggregated and returned at a coarser time resolution defined by `output_dt`.

Value

A matrix of concentrations (ppm) with rows representing output time steps and columns representing grid points. Columns correspond to the flattened grid defined by `expand.grid(grid_coords)`.

Note

All time parameters should be positive, with ‘puff_dt > sim_dt‘ and ‘out_dt > sim_dt‘. Also, ‘puff_dt‘ should be a positive integer multiple of ‘sim_dt‘, i.e. ‘puff_dt = n*sim_dt‘ for some positive integer ‘n‘. This prevents the code having to interpolate the concentration values in time, although it is likely that this constraint could be avoided.

References

Jia, M., Fish, R., Daniels, W., Sprinkle, B. and Hammerling, D. (2024) <doi:10.26434/chemrxiv-2023-hc95q-v3>

Examples

```
set.seed(123)

sim_dt <- 7
puff_dt <- 7
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
source_coords <- c(0, 0, 2.5)
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)

grid_coords <- list(
  x = seq(-2, 2, by = 1),
  y = seq(-2, 2, by = 1),
  z = c(2.5)
)
  out <- simulate_grid_mode(
    start_time = start_time,
    end_time = end_time,
    source_coords = source_coords,
    emission_rate = emission_rate,
    wind_data = wind_data,
    grid_coords = grid_coords,
    sim_dt = sim_dt,
    puff_dt = puff_dt,
    output_dt = output_dt,
    puff_duration = 1200
  )
```

`simulate_sensor_mode` *Simulate Atmospheric Concentration at Sensor Locations*

Description

This function simulates atmospheric methane concentrations at one or more sensor locations using a Gaussian puff forward model. It supports one or multiple emission sources and assumes each puff maintains a constant wind speed and direction throughout its lifetime. The function accounts for puff dispersion based on wind conditions and atmospheric stability class.

Usage

```
simulate_sensor_mode(start_time, end_time, source_coords, emission_rate, wind_data,
  sensor_coords, sim_dt, puff_dt, output_dt, puff_duration, ws, wd)
```

Arguments

`start_time`	POSIXct. Start time of the simulation.
`end_time`	POSIXct. End time of the simulation.
`source_coords`	Numeric vector or matrix. Source coordinates in meters (x, y, z). If a single source, pass as a vector. For multiple sources, use a matrix where each row is a source.
`emission_rate`	Numeric. Emission rate from each source in kg/hr. Applied uniformly to all sources.
`wind_data`	Data frame. Must contain either columns ‘wind_u‘ and ‘wind_v‘ (wind vector components in x/y directions) or columns representing wind speed and direction, declared as ‘ws‘ and ‘wd‘.
`sensor_coords`	Numeric matrix. Sensor coordinates in meters (x, y, z); one row per sensor.
`sim_dt`	Integer. Simulation time step in seconds (default: 1). Controls how often the simulation updates concentrations.
`puff_dt`	Integer. Puff emission interval in seconds (default: 1). Controls how often a new puff is emitted.
`output_dt`	Integer. Desired resolution in seconds for output concentrations.
`puff_duration`	Numeric. Lifetime of each puff in seconds (default: 1200). Puffs are removed after this time.
`ws`	Optional character. If your ‘wind_data‘ uses wind speed and direction instead of ‘wind_u‘/‘wind_v‘, supply the name of the wind speed column here (e.g., ‘"ws"‘ or ‘"wind_speed"‘).
`wd`	Optional character. If your ‘wind_data‘ uses wind direction in degrees, supply the name of the wind direction column here (e.g., ‘"wd"‘ or ‘"wind_direction"‘).

Details

- Each source emits puffs at regular intervals (‘puff_dt‘) with a fixed mass based on ‘emission_rate‘. - Wind speed and direction at the time of puff emission are used to advect the puff and determine dispersion. - Puff position is analytically computed at each timestep based on wind, without tracking in-between steps. - Puff dispersion is computed using stability-class-based sigma values from a fast lookup. - Total sensor concentration is the sum of all active puff contributions at each timestep. - Concentrations are aggregated into intervals matching ‘output_dt‘ before being returned.

Value

A data frame with aggregated sensor concentrations across time. Rows represent time intervals (‘output_dt‘), columns represent sensors (‘Sensor_1‘, ‘Sensor_2‘, etc.).

Note

All time parameters should be positive, with ‘puff_dt > sim_dt‘ and ‘out_dt > sim_dt‘. Also, ‘puff_dt‘ should be a positive integer multiple of ‘sim_dt‘, i.e. ‘puff_dt = n*sim_dt‘ for some positive integer ‘n‘. This prevents the code having to interpolate the concentration values in time, although it is likely that this constraint could be avoided.

References

Jia, M., Fish, R., Daniels, W., Sprinkle, B. and Hammerling, D. (2024) <doi:10.26434/chemrxiv-2023-hc95q-v3>

Examples

```
set.seed(123)
sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- as.POSIXct("2024-01-01 12:00:00")
end_time <- as.POSIXct("2024-01-01 13:00:00")

source_coords <- matrix(c(0, 0, 2.5), ncol = 3, byrow = TRUE)

sensor_coords <- matrix(c(
   -20, 0, 2.0,
     0, -20, 2.0,
    20, 0, 2.0,
     0, 20, 2.0,
    10, 10, 2.0
), ncol = 3, byrow = TRUE)

wind_data <- data.frame(
   wind_u = runif(3601, min = -3, max = 0.7),
   wind_v = runif(3601, min = -3, max = 1.5)
)

out <- simulate_sensor_mode(
   start_time = start_time,
   end_time = end_time,
   source_coords = source_coords,
   emission_rate = 3.5,
   wind_data = wind_data,
   sensor_coords = sensor_coords,
   sim_dt = sim_dt,
   puff_dt = puff_dt,
   output_dt = output_dt,
   puff_duration = 1200
)
```

`single_emission_rate_plot` *Plot Multiple Emission Rate Sensor Concentrations*

Description

This function generates a faceted (if multiple sensors) bubble plot of sensor concentrations over time using precomputed sensor data (e.g., from ‘simulate_sensor_mode()‘).

Usage

```
single_emission_rate_plot(sensor_concentrations, sensor_coords, text_size = 12)
```

Arguments

`sensor_concentrations`	Data frame. Output from a sensor simulation function, which must include a column named "Group.1" for timestamps and one or more columns named "Sensor_1", "Sensor_2", etc., for the sensor concentration values.
`sensor_coords`	Numeric vector or matrix. Coordinates (x, y, z) of the sensor(s).
`text_size`	Default at 12.

Value

A ggplot object showing sensor concentrations over time, faceted by sensor.

Examples

```
set.seed(123)
sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)
source_coords <- c(0, 0, 2.5)
sensor_coords <- matrix(c(-6.525403221327715e-15, -35.52264, 2.01775), ncol = 3, byrow = TRUE)

out <- simulate_sensor_mode(
  start_time, end_time, source_coords,
 emission_rate, wind_data, sensor_coords, sim_dt, puff_dt, output_dt, puff_duration = 1200
)

single_emission_rate_plot(out, sensor_coords)
```

`time_series_plot` *Plot Time Series of Sensor Concentrations*

Description

This function plots the time series of sensor concentrations.

Usage

```
time_series_plot(sensor_concentrations, text_size = 12)
```

Arguments

`sensor_concentrations`	A data frame (or matrix) containing the output from the sensor simulation function (e.g., ‘simulate_sensor_mode()‘). It must include: **Group.1** A character vector of timestamps in the format ‘"YYYY-MM-DD HH:MM:SS"‘. **Sensor_1** A numeric vector of sensor concentration values corresponding to each timestamp.
`text_size`	Default at 12.

Value

A ggplot object showing the time series of sensor concentrations.

Examples

```
set.seed(123)
sim_dt <- 10
puff_dt <- 10
output_dt <- 60
start_time <- "2024-01-01 12:00:00"
end_time <- "2024-01-01 13:00:00"
emission_rate <- 3.5
wind_data <- data.frame(
  wind_u = runif(3601, min = -3, max = 0.7),
  wind_v = runif(3601, min = -3, max = 1.5)
)
source_coords <- c(0, 0, 2.5)
sensor_coords <- matrix(c(-6.525403221327715e-15, -35.52264, 2.01775), ncol = 3, byrow = TRUE)

out <- simulate_sensor_mode(
  start_time, end_time, source_coords,
 emission_rate, wind_data, sensor_coords, sim_dt, puff_dt, output_dt, puff_duration = 1200
)

time_series_plot(out)
```

`wind_vector_convert` *Convert between (ws, wd) and (u,v)*

Description

This function converts between coordinate systems by changing from degrees to radians

Usage

```
wind_vector_convert(wind_speeds,wind_directions)
```

Arguments

`wind_speeds`	A list of float values of wind speeds in m/s at each time stamp
`wind_directions`	A list of float values of wind directions in degrees at each time stamp following the conventional definition: 0 -> wind blowing from North, 90 -> E, 180 -> S, 270 -> W

Value

Quantities corresponding to the conversion direction

Examples

```
out <- wind_vector_convert(c(5, 10), c(0, 90))
```

Index

compute_sigma_vals, 2
create_site_map, 3

faceted_time_series_plot, 3
faceted_time_series_plot2, 5

get_stab_class, 6
gpuff, 7

interpolate_wind_data, 8
is_day, 8

plot_2d_animated, 9
plot_3d_animated, 11

simulate_grid_mode, 12
simulate_sensor_mode, 15
single_emission_rate_plot, 17

time_series_plot, 18

wind_vector_convert, 19