# `fillstreamline2` Function Documentation

## Function Overview

The `fillstreamline2` function simulates the distribution of non-overlapping objects (referred to as beads) along a given 2D streamline within a flow field. The function is designed to handle complex streamlines and fluid dynamics scenarios, efficiently populating the streamline with beads while ensuring they do not overlap based on the specified radius.

## Usage

```
traj = fillstreamline2(s, Xgrid, Ygrid, Vxgrid, Vygrid [,rbead, l0, verbose])
[traj, errout] = fillstreamline2(...)
```

## Parameters

- **s**: `n x 2` array representing a single 2D streamline, or `1 x m` cell array (output of `streamline` function).
- **Xgrid**: 2D array representing the X coordinates (output of `meshgrid` ).
- **Ygrid**: 2D array representing the Y coordinates (output of `meshgrid` ).
- **Vxgrid**: 2D array representing the x-component of velocity.
- **Vygrid**: 2D array representing the y-component of velocity.
- **rbead**: Optional. Bead radius (default value = 1).
- **l0**: Optional. Initial position of the first bead along the curvilinear coordinate (default value = 0).
- **verbose**: Optional. Flag to control verbosity (default = true).

## Outputs

- **traj**: Structure containing fields:
  - `streamline` : Original streamline coordinates.
  - `curvilinear` : Curvilinear coordinate along the streamline.
  - `pseudotime` : Pseudotime to travel along the streamline.
  - `curv_velocity` : Velocity along the curvilinear coordinate.
  - `cart_velocity` : Cartesian velocity components.
  - `curv_distribution` : Distribution of beads along the curvilinear coordinate.
  - `cart_distribution` : Corresponding Cartesian coordinates of beads.
  - `t_distribution` : Corresponding times of bead positions.
  - `curv_v_distribution` : Corresponding velocities along the streamline for each bead.

    - `cart_v_distribution` : Corresponding Cartesian velocities for each bead.
- **errout**: Array of error messages generated during the function execution.

## Description

The function initializes by verifying input dimensions and setting default values for optional parameters. It handles multiple streamlines by recursively applying the distribution process to each streamline. Beads are distributed based on their specified radius to ensure no overlapping, and their positions are interpolated from the streamline based on fluid dynamics properties such as velocity and time.

## Example

```
matlabCopy code% Define streamline and grid data
s = [1.0, 1.5; 0.5, 2.0; 2.0, 0.5];
Xgrid = linspace(0, 3, 10);
Ygrid = linspace(0, 3, 10);
[Vxgrid, Vygrid] = meshgrid(-0.1:-0.1:-1.0, 0.1:0.1:1.0);
rbead = 0.1;
l0 = 0;
verbose = true;

% Calculate trajectory of beads
[traj, errout] = fillstreamline2(s, Xgrid, Ygrid, Vxgrid, Vygrid, rbead, l0,
verbose);
disp('Bead Trajectory:');
disp(traj.cart_distribution);
```

## Algorithm Details

The `fillstreamline2` function processes a streamline to distribute beads along it based on specified fluid dynamics and geometric constraints. The algorithm can be broken down into several key steps:

### Initialization and Parameter Verification #

- The function starts by checking the dimensions of the input arrays ( `Xgrid` , `Ygrid` , `Vxgrid` , `Vygrid` ) to ensure they match, setting default values for optional parameters like bead radius ( `rbead` ), initial position ( `l0` ), and verbosity ( `verbose` ).
- Handles multiple streamlines if the input `s` is a cell array by recursively applying the algorithm to each streamline segment.

### Streamline Processing #

1. **Streamline Simplification**: Removes any invalid points (NaN values) from the streamline and ensures it is appropriately formatted for processing.
2. **Curvilinear Coordinate Calculation**:
    - Calculates the differential lengths between consecutive points on the streamline.

    - Cumulatively sums these differential lengths to produce a curvilinear coordinate, representing the distance along the streamline.
3. **Velocity Interpolation**:
    - Interpolates the velocity components ( `Vx` , `Vy` ) from the velocity grids at the positions specified by the streamline.
    - Calculates the tangent vector at each point of the streamline, which is used to determine the streamline's local direction.
    - Projects the interpolated velocity onto the tangent vector to get the velocity along the streamline ( `curv_velocity` ).
4. **Streamline Direction Check**:
    - Ensures the streamline is oriented in the direction of fluid flow by checking the sign of the averaged velocity along the streamline. If it's negative, the streamline is flipped.
5. **Bead Distribution**:
    - Determines the number of beads that can be placed along the streamline based on the curvilinear coordinate and the specified bead radius, ensuring no two beads overlap.
    - Calculates a pseudo-travel time for each bead based on the velocity along the streamline.
    - Distributes the beads in such a way that they are spaced to maintain a constant density along the streamline.
6. **Output Construction**:
    - Constructs a structured output containing both curvilinear and Cartesian coordinates for each bead, their respective velocities, and the times at which they reach these positions.

## Error Handling and Verbosity #

- Provides detailed error messages for various failure cases, such as invalid input dimensions or an empty streamline.
- If verbose mode is enabled, outputs additional information during processing, such as computation progress and any warnings about the data or process.

## Recursive Handling for Multiple Streamlines #

- If the input consists of multiple streamlines, the function recursively applies the distribution algorithm to each, collecting outputs and errors into arrays that reflect the processing of each individual streamline.

This approach allows `fillstreamline2` to handle complex fluid dynamics scenarios while ensuring an efficient and accurate distribution of objects along a given streamline.