

Simulation der Brown'schen Molekularbewegung mit Python 3

Martin Fané

Alfatraining Köln

11. März 2021

Hintergrund

Die Brown'sche Molekularbewegung:

- ▶ Beschreibt die unregelmäßige und ruckartige Wärmebewegung kleiner Teilchen → z.B. Geruchsausbreitung.
- ▶ Mathematisches Modell des „Random Walk“ dient zur Simulation.
- ▶ Random Walk: Verkettung zufälliger Bewegungen.

Motivation

- ▶ Simulationskenntnisse in Python aufbauen.
- ▶ Wichtiges Werkzeug in Wissenschaft, Forschung und Entwicklung.

Hintergrund

Die Brown'sche Molekularbewegung:

- ▶ Beschreibt die unregelmäßige und ruckartige Wärmebewegung kleiner Teilchen → z.B. Geruchsausbreitung.
- ▶ Mathematisches Modell des „Random Walk“ dient zur Simulation.
- ▶ Random Walk: Verkettung zufälliger Bewegungen.

Motivation

- ▶ Simulationskenntnisse in Python aufbauen.
- ▶ Wichtiges Werkzeug in Wissenschaft, Forschung und Entwicklung.

Hintergrund

Die Brown'sche Molekularbewegung:

- ▶ Beschreibt die unregelmäßige und ruckartige Wärmebewegung kleiner Teilchen → z.B. Geruchsausbreitung.
- ▶ Mathematisches Modell des „Random Walk“ dient zur Simulation.
- ▶ Random Walk: Verkettung zufälliger Bewegungen.

Motivation

- ▶ Simulationskenntnisse in Python aufbauen.
- ▶ Wichtiges Werkzeug in Wissenschaft, Forschung und Entwicklung.

Hintergrund

Die Brown'sche Molekularbewegung:

- ▶ Beschreibt die unregelmäßige und ruckartige Wärmebewegung kleiner Teilchen → z.B. Geruchsausbreitung.
- ▶ Mathematisches Modell des „Random Walk“ dient zur Simulation.
- ▶ Random Walk: Verkettung zufälliger Bewegungen.

Motivation

- ▶ Simulationskenntnisse in Python aufbauen.
- ▶ Wichtiges Werkzeug in Wissenschaft, Forschung und Entwicklung.

Hintergrund

Die Brown'sche Molekularbewegung:

- ▶ Beschreibt die unregelmäßige und ruckartige Wärmebewegung kleiner Teilchen → z.B. Geruchsausbreitung.
- ▶ Mathematisches Modell des „Random Walk“ dient zur Simulation.
- ▶ Random Walk: Verkettung zufälliger Bewegungen.

Motivation

- ▶ Simulationskenntnisse in Python aufbauen.
- ▶ Wichtiges Werkzeug in Wissenschaft, Forschung und Entwicklung.

Das Programm

Überblick:


```
1  from matplotlib.figure import Figure
2  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
3  from random import choice
4  import numpy as np
5  import tkinter as tk
6
7
8  class BrownianMotion:
9
10     def open_window(self):...
160
161
162     def random_walker(num_particles, num_steps, step_size):...
211
212
213 ▶ if __name__ == "__main__":
214     bm = BrownianMotion() # Instantiierung eines BrownianMotion()-Objektes.
215     bm.open_window()      # Methodenaufruf zum Öffnen des Fensters.
```

Das Programm

Die Methode `def open_window()` der Klasse `BrownianMotion`:

`def open_window()`

`# Definition/Initialisierung
der Label und Buttons
mittels tkinter sowie ver-
schiedene Funktionsaufrufe`

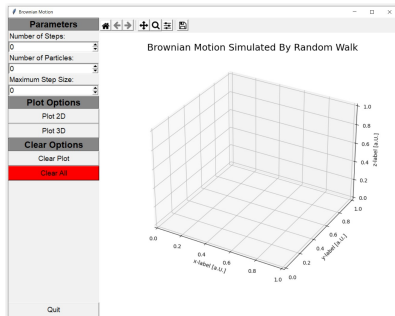


Das Programm

Die Methode `def open_window()` der Klasse `BrownianMotion`:

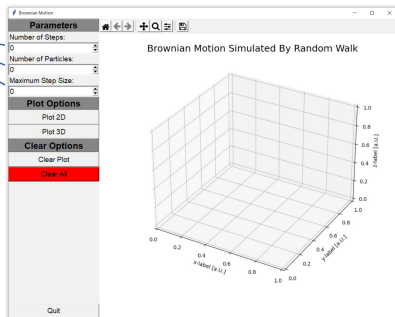
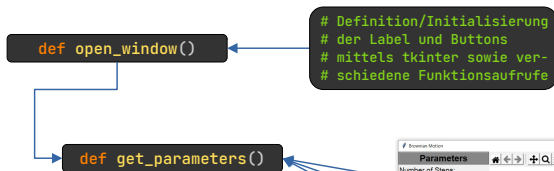
`def open_window()`

Definition/Initialisierung
der Label und Buttons
mittels tkinter sowie ver-
schiedene Funktionsaufrufe



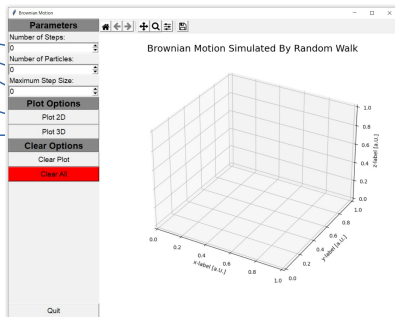
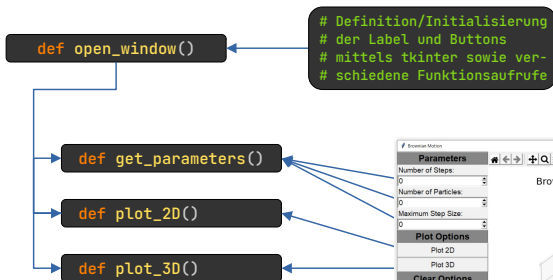
Das Programm

Die Methode `def open_window()` der Klasse `BrownianMotion`:



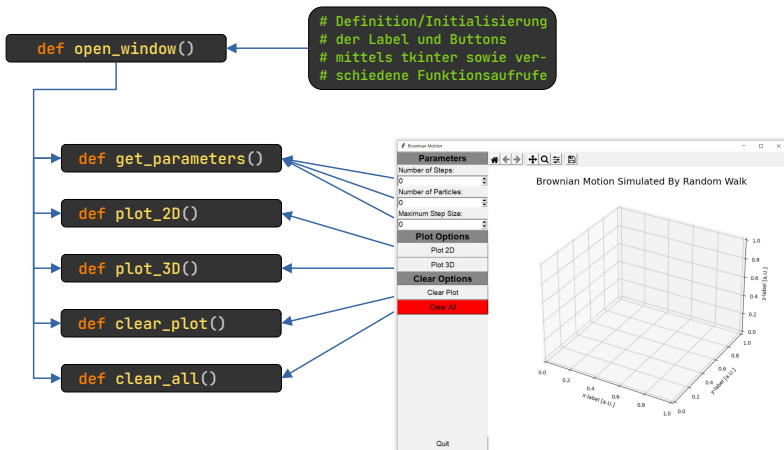
Das Programm

Die Methode `def open_window()` der Klasse `BrownianMotion`:



Das Programm

Die Methode `def open_window()` der Klasse `BrownianMotion`:



Das Programm

Die Funktion `def random_walker()`:

```
162 def random_walker(num_particles, num_steps, step_size):
163     '''In dieser Funktion wird der Random Walk der der Teilchen simuliert.'''
164
165     data_x = []      # Enthalten die Listen der Koordinaten.
166     data_y = []      # Wichtig für den Plot, da so die Koordinaten
167     data_z = []      # pro Teilchen in einer eigenen Liste gespeichert werden.
168                     # Z.B. enthält data_x[[Teilchen1], [Teilchen2], ...] die
169                     # x-Koordinaten aller Teilchen.
170
171     for particle in range(num_particles):...
209
210     return data_x, data_y, data_z
```

- ▶ Bekommt die Parameter für *Random Walk* übergeben.
- ▶ Führt die Berechnung des *Random Walk* durch.
- ▶ Returniert die x-, y- und z-Koordinaten an Funktionen der Methode `open_walk()`.

Das Programm

Die Funktion `def random_walker()`:

```
162 def random_walker(num_particles, num_steps, step_size):
163     '''In dieser Funktion wird der Random Walk der der Teilchen simuliert.'''
164
165     data_x = []      # Enthalten die Listen der Koordinaten.
166     data_y = []      # Wichtig für den Plot, da so die Koordinaten
167     data_z = []      # pro Teilchen in einer eigenen Liste gespeichert werden.
168                     # Z.B. enthält data_x[[Teilchen1], [Teilchen2], ...] die
169                     # x-Koordinaten aller Teilchen.
170
171     for particle in range(num_particles):...
209
210     return data_x, data_y, data_z
```

- ▶ Bekommt die Parameter für *Random Walk* übergeben.
- ▶ Führt die Berechnung des *Random Walk* durch.
- ▶ Returniert die x-, y- und z-Koordinaten an Funktionen der Methode `open_walk()`.

Das Programm

Die Funktion `def random_walker()`:

```
162 def random_walker(num_particles, num_steps, step_size):
163     '''In dieser Funktion wird der Random Walk der der Teilchen simuliert.'''
164
165     data_x = []      # Enthalten die Listen der Koordinaten.
166     data_y = []      # Wichtig für den Plot, da so die Koordinaten
167     data_z = []      # pro Teilchen in einer eigenen Liste gespeichert werden.
168                     # Z.B. enthält data_x[[Teilchen1], [Teilchen2], ...] die
169                     # x-Koordinaten aller Teilchen.
170
171     for particle in range(num_particles):...
209
210     return data_x, data_y, data_z
```

- ▶ Bekommt die Parameter für *Random Walk* übergeben.
- ▶ Führt die Berechnung des *Random Walk* durch.
- ▶ Returniert die x-, y- und z-Koordinaten an Funktionen der Methode `open_walk()`.

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):  
2     data_x, data_y, data_z = [], [], []
```


Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):  
2     data_x, data_y, data_z = [], [], []  
  
3     for particle in num_particles:
```

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):  
2     data_x, data_y, data_z = [], [], []  
3     for particle in num_particles:  
4         x_coord, y_coord, z_coord = [0], [0], [0]
```

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):  
2     data_x, data_y, data_z = [], [], []  
3     for particle in num_particles:  
4         x_coord, y_coord, z_coord = [0], [0], [0]  
5         while len(x_coord) < num_steps:
```

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):  
2     data_x, data_y, data_z = [], [], []  
3     for particle in num_particles:  
4         x_coord, y_coord, z_coord = [0], [0], [0]  
5         while len(x_coord) < num_steps:  
6             x_direction = choice([-1, 1])
```

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):
2     data_x, data_y, data_z = [], [], []
3
4     for particle in num_particles:
5
6         x_coord, y_coord, z_coord = [0], [0], [0]
7
8         while len(x_coord) < num_steps:
9
10            x_direction = choice([-1, 1])
11
12            x_distance = choice(list(np.arange(0, step_size, .1)))
```

Das Programm

Berechnung des *Random Walk*:

```
1  def random_walker(num_particles, num_steps, step_size):  
2      data_x, data_y, data_z = [], [], []  
  
3      for particle in num_particles:  
  
4          x_coord, y_coord, z_coord = [0], [0], [0]  
  
5          while len(x_coord) < num_steps:  
  
6              x_direction = choice([-1, 1])  
  
7              x_distance = choice(list(np.arange(0, step_size, .1)))  
  
8              x_step = x_direction * x_distance
```

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):
2     data_x, data_y, data_z = [], [], []
3
4     for particle in num_particles:
5
6         x_coord, y_coord, z_coord = [0], [0], [0]
7         while len(x_coord) < num_steps:
8             x_direction = choice([-1, 1])
9             x_distance = choice(list(np.arange(0, step_size, .1)))
10            x_step = x_direction * x_distance
11            x = x_coord[-1] + x_step
```

Analog und
simultan für
y- und
z-Koordinate

Das Programm

Berechnung des *Random Walk*:

```
1 def random_walker(num_particles, num_steps, step_size):
2     data_x, data_y, data_z = [], [], []
3
4     for particle in num_particles:
5
6         x_coord, y_coord, z_coord = [0], [0], [0]
7         while len(x_coord) < num_steps:
8             x_direction = choice([-1, 1])
9             x_distance = choice(list(np.arange(0, step_size, .1)))
10            x_step = x_direction * x_distance
11            x = x_coord[-1] + x_step
12            x_coord.append(x), y_coord.append(y), z_coord.append(z)
```

Analog und
simultan für
y- und
z-Koordinate

Das Programm

Berechnung des *Random Walk*:

```
1  def random_walker(num_particles, num_steps, step_size):
2      data_x, data_y, data_z = [], [], []
3
4      for particle in num_particles:
5
6          x_coord, y_coord, z_coord = [0], [0], [0]
7
8          while len(x_coord) < num_steps:
9
10             x_direction = choice([-1, 1])
11             x_distance = choice(list(np.arange(0, step_size, .1)))
12             x_step = x_direction * x_distance
13             x = x_coord[-1] + x_step
14
15             x_coord.append(x), y_coord.append(y), z_coord.append(z)
16
17             data_x.append(x_coord)
18             data_y.append(y_coord)
19             data_z.append(z_coord)
```

Analog und
simultan für
y- und
z-Koordinate

Das Programm

Berechnung des *Random Walk*:

```
1  def random_walker(num_particles, num_steps, step_size):
2      data_x, data_y, data_z = [], [], []
3
4      for particle in num_particles:
5
6          x_coord, y_coord, z_coord = [0], [0], [0]
7
8          while len(x_coord) < num_steps:
9
10             x_direction = choice([-1, 1])
11             x_distance = choice(list(np.arange(0, step_size, .1)))
12             x_step = x_direction * x_distance
13             x = x_coord[-1] + x_step
14             x_coord.append(x), y_coord.append(y), z_coord.append(z)
15
16             data_x.append(x_coord)
17             data_y.append(y_coord)
18             data_z.append(z_coord)
19
20     return data_x, data_y, data_z
```

Analog und
simultan für
y- und
z-Koordinate

Das Programm

Berechnung des *Random Walk*:

```
1  def random_walker(num_particles, num_steps, step_size):
2      data_x, data_y, data_z = [], [], []
3
4      for particle in num_particles:
5
6          x_coord, y_coord, z_coord = [0], [0], [0]
7
8          while len(x_coord) < num_steps:
9
10             x_direction = choice([-1, 1])
11             x_distance = choice(list(np.arange(0, step_size, .1)))
12             x_step = x_direction * x_distance
13             x = x_coord[-1] + x_step
14             x_coord.append(x), y_coord.append(y), z_coord.append(z)
15
16             data_x.append(x_coord)
17             data_y.append(y_coord)
18             data_z.append(z_coord)
19
20     return data_x, data_y, data_z
```

Analog und
simultan für
y- und
z-Koordinate


```
# data_x = [[x_Teilchen1], [x_Teilchen2],...]
# data_y = [[y_Teilchen1], [y_Teilchen2],...]
# data_z = [[z_Teilchen1], [z_Teilchen2],...]
```

Das Programm

Ablauf des Programms:

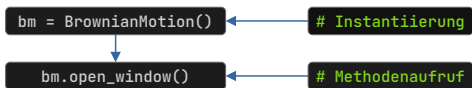
```
bm = BrownianMotion()
```

```
# Instantiierung
```



Das Programm

Ablauf des Programms:

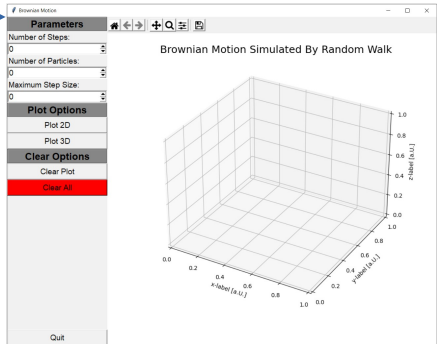


Das Programm

Ablauf des Programms:

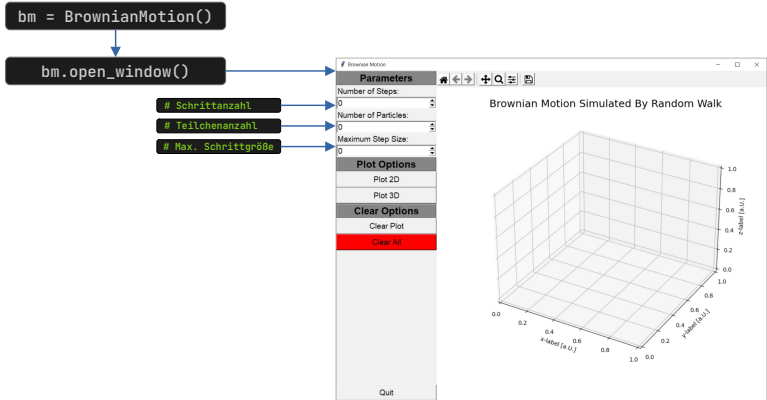
```
bm = BrownianMotion()
```

```
bm.open_window()
```



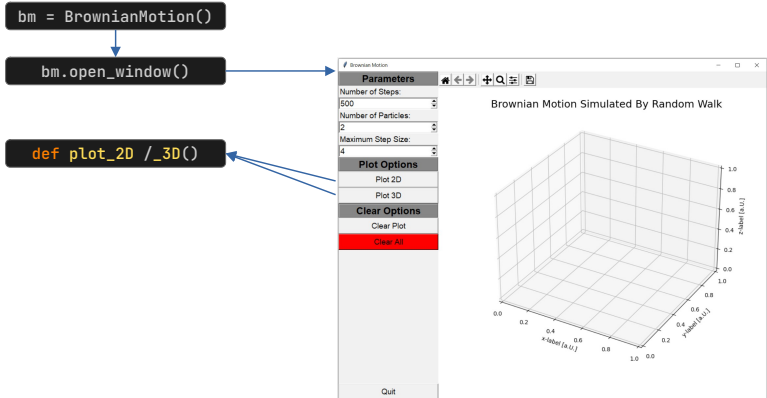
Das Programm

Ablauf des Programms:



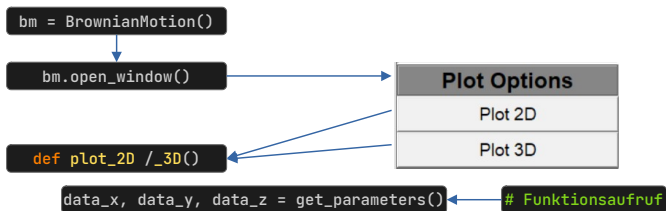
Das Programm

Ablauf des Programms:



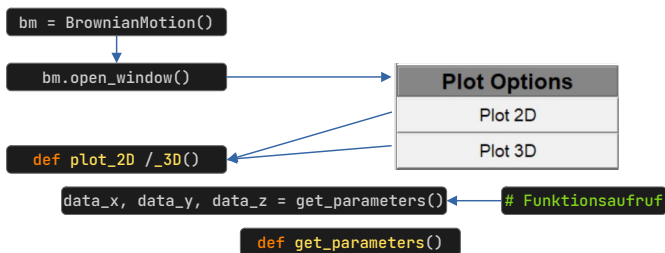
Das Programm

Ablauf des Programms:



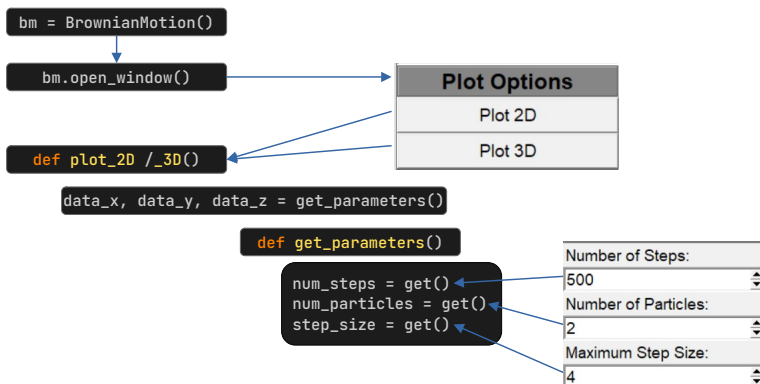
Das Programm

Ablauf des Programms:



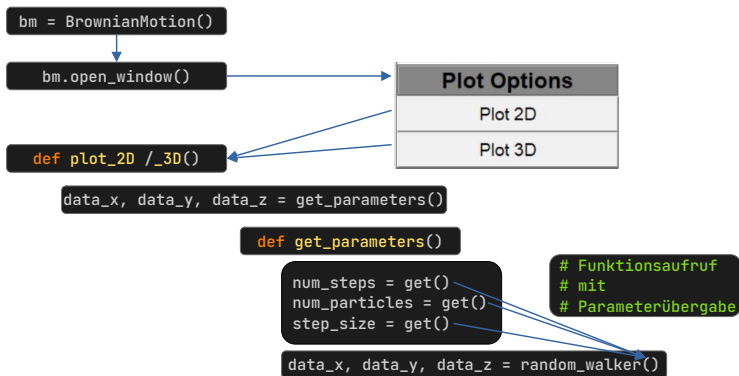
Das Programm

Ablauf des Programms:



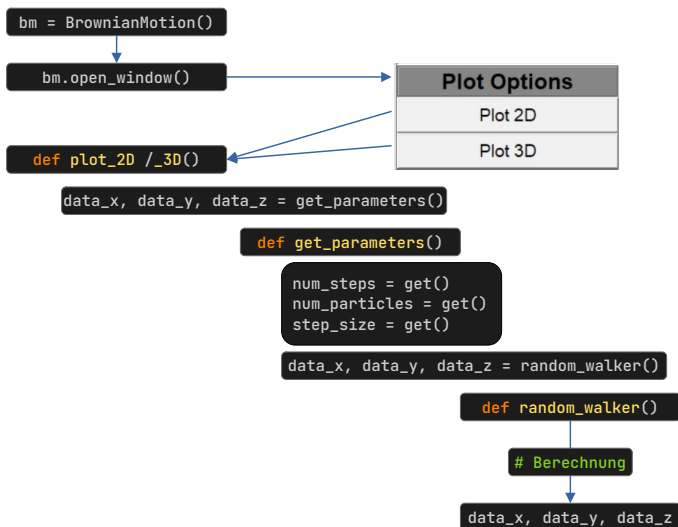
Das Programm

Ablauf des Programms:



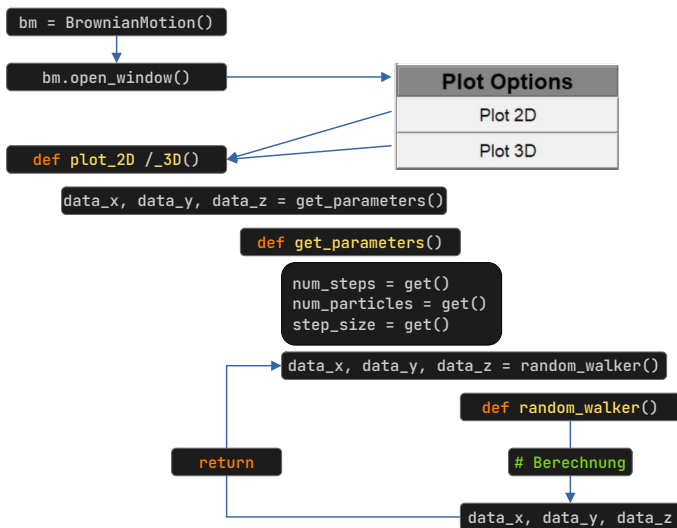
Das Programm

Ablauf des Programms:



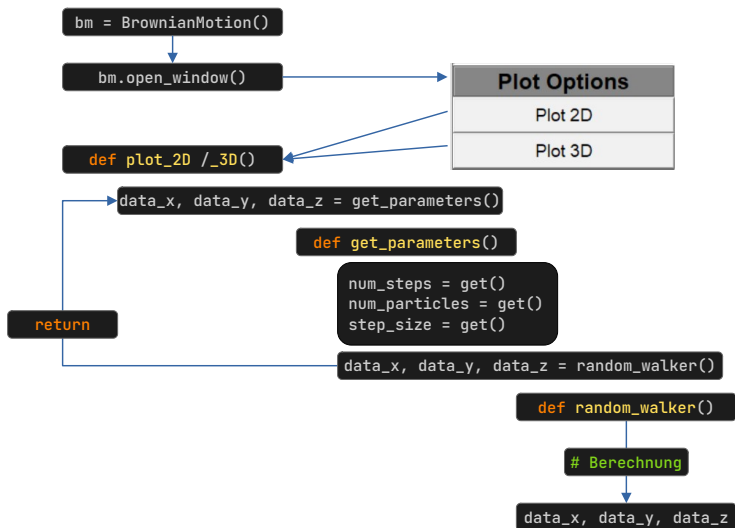
Das Programm

Ablauf des Programms:



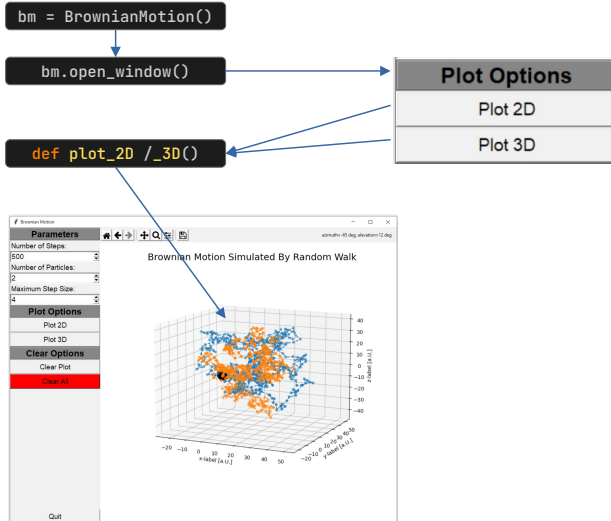
Das Programm

Ablauf des Programms:

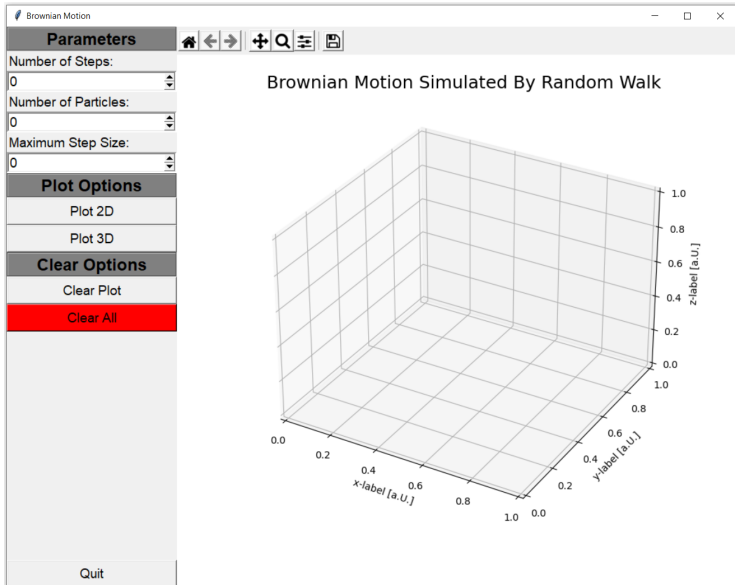


Das Programm

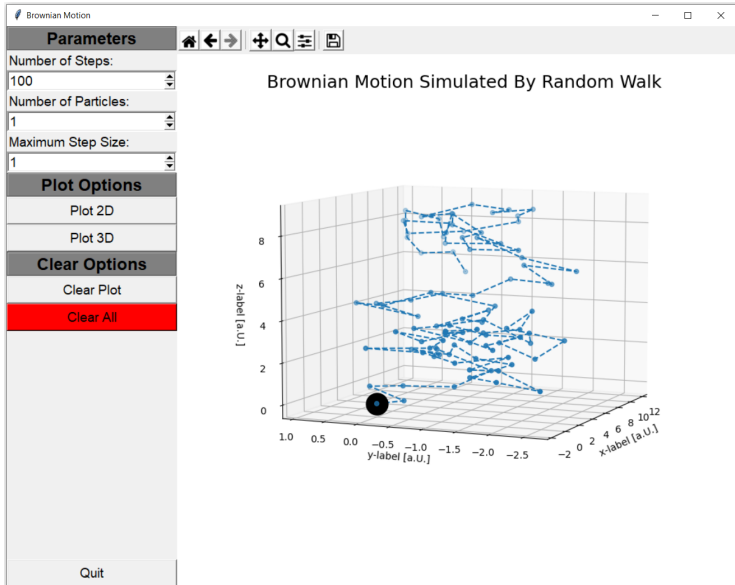
Ablauf des Programms:



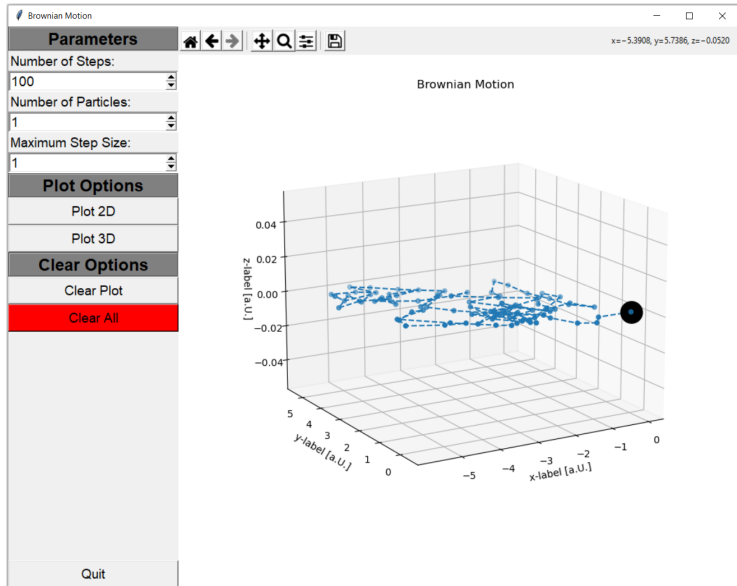
Das Programm



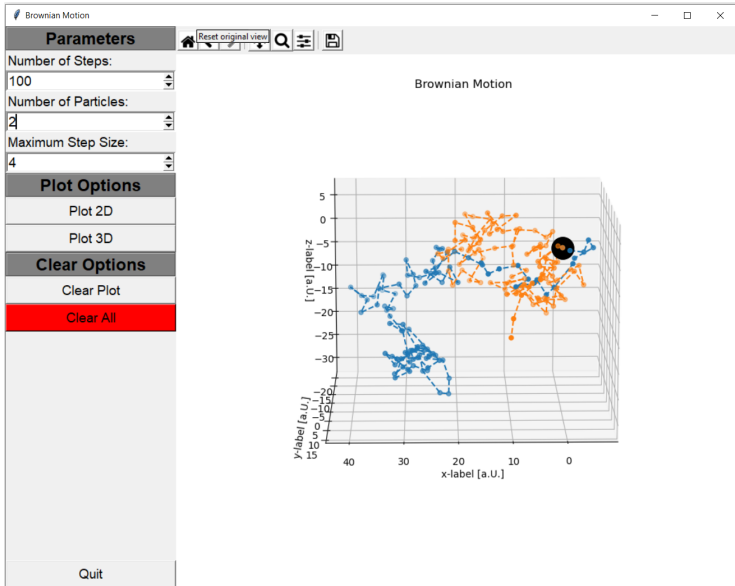
Das Programm



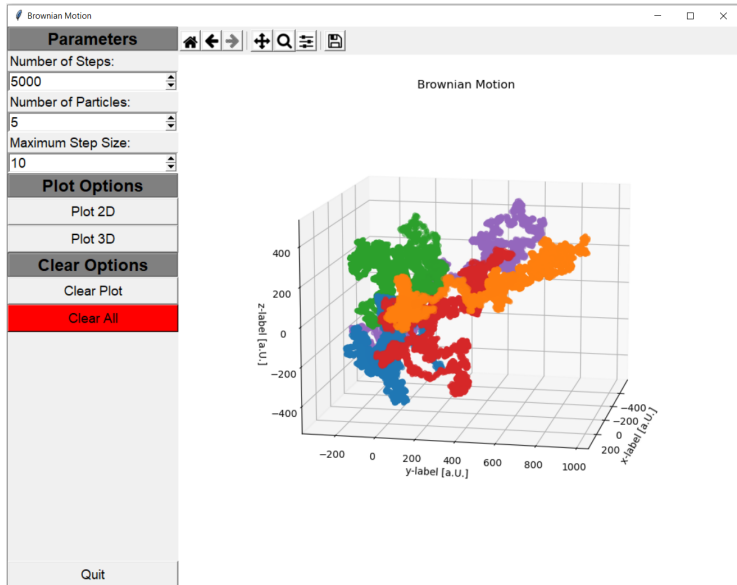
Das Programm



Das Programm



Das Programm



Danke für die Aufmerksamkeit!