Random Walk in Python

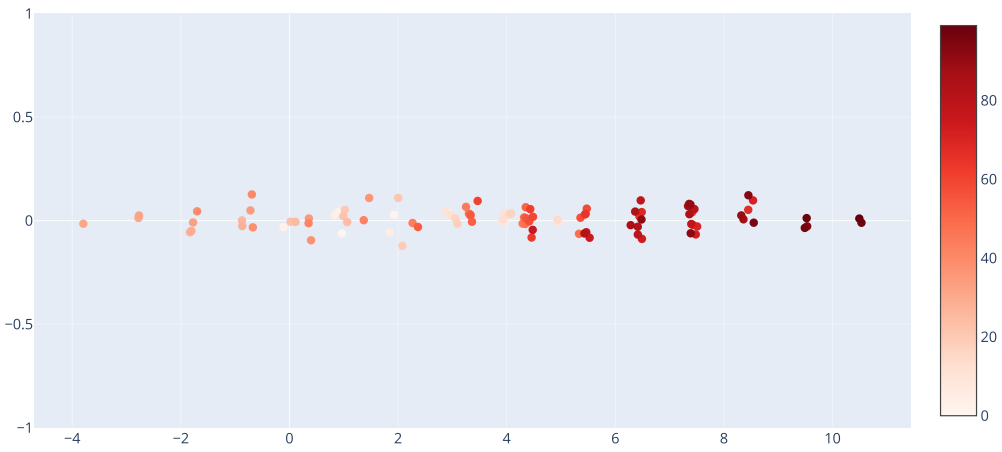
Learn how to use Python to make a Random Walk

A random walk (https://en.wikipedia.org/wiki/Random_walk) can be thought of as a random process in which a token or a marker is randomly moved around some space, that is, a space with a metric used to compute distance. It is more commonly conceptualized in one dimension ($\mathbb{Z}$), two dimensions ($\mathbb{Z}^2$) or three dimensions ($\mathbb{Z}^3$) in Cartesian space, where $\mathbb{Z}$ represents the set of integers. In the visualizations below, we will be using scatter plots (https://plotly.com/python/line-and-scatter/) as well as a colorscale to denote the time sequence of the walk.

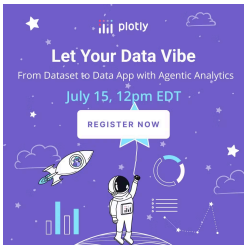
Random Walk in 1D

The jitter in the data points along the x and y axes are meant to illuminate where the points are being drawn and what the tendency of the random walk is.

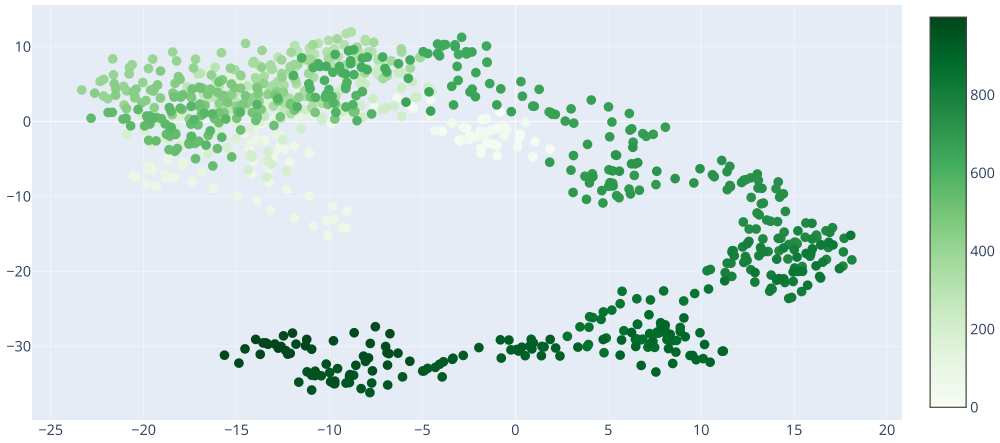
```
import plotly.graph_objects as go
import numpy as np
np.random.seed(1)

l = 100
steps = np.random.choice([-1, 1], size=l) + 0.05 * np.random.randn(l) # l steps
position = np.cumsum(steps) # integrate the position by summing steps values
y = 0.05 * np.random.randn(l)

fig = go.Figure(data=go.Scatter(
    x=position,
    y=y,
    mode='markers',
    name='Random Walk in 1D',
    marker=dict(
        color=np.arange(l),
        size=7,
        colorscale='Reds',
        showscale=True,
    )
))

fig.update_layout(yaxis_range=[-1, 1])
fig.show()
```



Random Walk in 2D



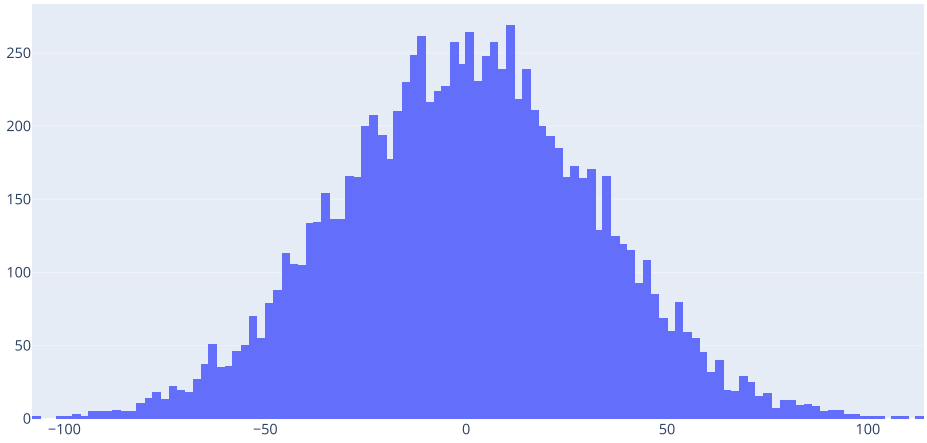
```
import plotly.graph_objects as go
import numpy as np

l = 1000
x_steps = np.random.choice([-1, 1], size=l) + 0.2 * np.random.randn(l) # l steps
y_steps = np.random.choice([-1, 1], size=l) + 0.2 * np.random.randn(l) # l steps
x_position = np.cumsum(x_steps) # integrate the position by summing steps values
y_position = np.cumsum(y_steps) # integrate the position by summing steps values

fig = go.Figure(data=go.Scatter(
    x=x_position,
    y=y_position,
    mode='markers',
    name='Random Walk',
    marker=dict(
        color=np.arange(l),
        size=8,
        colorscale='Greens',
        showscale=True
    )
))

fig.show()
```



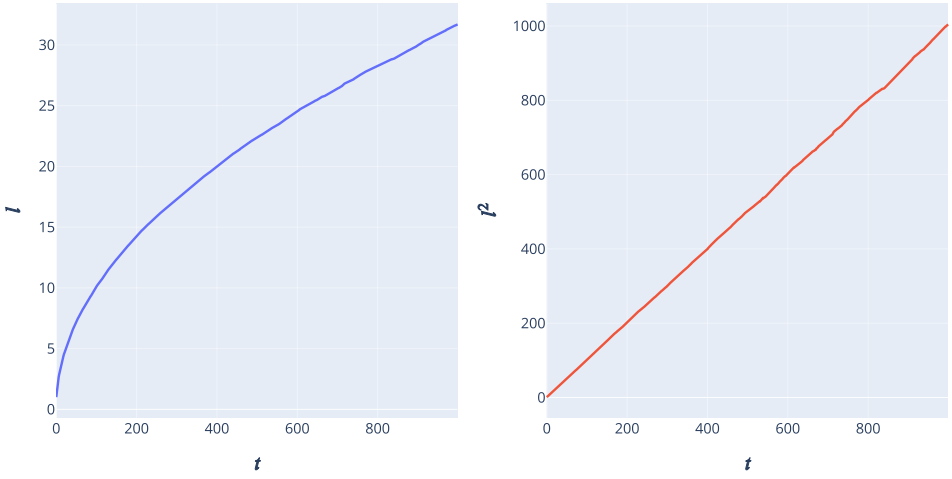
Random walk and diffusion

In the two following charts we show the link between random walks and diffusion. We compute a large number N of random walks representing for examples molecules in a small drop of chemical. While all trajectories start at 0, after some time the spatial distribution of points is a Gaussian distribution. Also, the average distance to the origin grows as $\sqrt(t)$.

```
import plotly.graph_objects as go
import numpy as np

l = 1000
N = 10000
steps = np.random.choice([-1, 1], size=(N, l)) + 0.05 * np.random.standard_normal((N, l)) # l steps
position = np.cumsum(steps, axis=1) # integrate all positions by summing steps values along time axis

fig = go.Figure(data=go.Histogram(x=position[:, -1])) # positions at final time step
fig.show()
```



```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

l = 1000
N = 10000
t = np.arange(l)
steps = np.random.choice([-1, 1], size=(N, l)) + 0.05 * np.random.standard_normal((N, l)) # l steps
position = np.cumsum(steps, axis=1) # integrate the position by summing steps values
average_distance = np.std(position, axis=0) # average distance

fig = make_subplots(1, 2)
fig.add_trace(go.Scatter(x=t, y=average_distance, name='mean distance'), 1, 1)
fig.add_trace(go.Scatter(x=t, y=average_distance**2, name='mean squared distance'), 1, 2)
fig.update_xaxes(title_text='$t$')
fig.update_yaxes(title_text='$l$', col=1)
fig.update_yaxes(title_text='$l^2$', col=2)
fig.update_layout(showlegend=False)
fig.show()
```



Advanced Tip

We can formally think of a 1D random walk as a point jumping along the integer number line. Let Z_i be a random variable that takes on the values +1 and -1. Let this random variable represent the steps we take in the random walk in 1D (where +1 means right and -1 means left). Also, as with the above visualizations, let us assume that the probability of moving left and right is just $\frac{1}{2}$. Then, consider the sum

$$S_n = \sum_{i=0}^{n} Z_i$$

where S_n represents the point that the random walk ends up on after n steps have been taken.

To find the expected value of S_n, we can compute it directly. Since each Z_i is independent, we have

$$E(S_n) = \sum_{i=0}^{n} E(Z_i)$$

but since Z_i takes on the values +1 and -1 then

$$E(Z_i) = 1 \cdot P(Z_i = 1) + -1 \cdot P(Z_i = -1) = \frac{1}{2} - \frac{1}{2} = 0$$

r random walk to hover around 0 regardless of how many steps we take in our walk.

What About Dash?

Dash (https://dash.plot.ly/) is an open-source framework for building analytical applications, with no Javascript required, and it is tightly integrated with the Plotly graphing library.

Learn about how to install Dash at https://dash.plot.ly/installation (https://dash.plot.ly/installation).

Everywhere in this page that you see fig.show(), you can display the same figure in a Dash application by passing it to the figure argument of the Graph component (https://dash.plot.ly/dash-core-components/graph) from the built-in dash_core_components package like this:

```
import plotly.graph_objects as go # or plotly.express as px
fig = go.Figure() # or any Plotly Express function e.g. px.bar(...)
# fig.add_trace( ... )
# fig.update_layout( ... )

from dash import Dash, dcc, html

app = Dash()
app.layout = html.Div([
    dcc.Graph(figure=fig)
])

app.run(debug=True, use_reloader=False)  # Turn off reloader if inside Jupyter
```