Other properties of variance include, for real constant $c$,

$$\begin{aligned}\mathrm{Var}\,[c] &= 0 \\ \mathrm{Var}\,[X + c] &= \mathrm{Var}\,[X] \\ \mathrm{Var}\,[cX] &= c^2\,\mathrm{Var}\,[X]\,.\end{aligned}$$

The **standard deviation** is defined as

$$\sigma_X = \sqrt{\sigma_X^2}.$$

Although the variance is mathematically more convenient, the standard deviation has the same physical units as $X$, so it is often the more physically meaningful quantity. Due to its meaning as the width or spread of the probability distribution, and its sharing of physical units, it is a convenient choice for error bars on plots of a random variable.

The **skewness** Skew $[X]$ is a normalized third central moment:

$$\mathrm{Skew}\,[X] = \frac{\mathrm{E}\left[(X - \mu_X)^3\right]}{\sigma_X^3}.$$

Skewness is a measure of **asymmetry** of a random variable's PDF or PMF. For a symmetric PMF or PDF, such as the Gaussian PDF, Skew $[X] = 0$.

The **kurtosis** Kurt $[X]$ is a normalized fourth central moment:

$$\mathrm{Kurt}\,[X] = \frac{\mathrm{E}\left[(X - \mu_X)^4\right]}{\sigma_X^4}.$$

Kurtosis is a measure of the **tailedness** of a random variable's PDF or PMF. "Heavier" tails yield higher kurtosis.

A Gaussian random variable has PDF with kurtosis 3. Given that for Gaussians both skewness and kurtosis have nice values (0 and 3), we can think of skewness and and kurtosis as measures of similarity to the Gaussian PDF.

## 3.9 Transforming Random Variables

SR

TODO: describe the theory and formulae

For random variables $X$ and $Y$ with PDFs $f_X$ and $f_Y$, and with invertible transformation $Y = g(X)$, we have the linear approximation

$$f_Y(y) = \frac{1}{|dy/dx|} f_X(x)\bigg|_{x \mapsto g^{-1}(y)}. \qquad (3.10)$$

**Example 3.11**

Suppose we are to probabilistically quantify a parachutist's chances of landing within a certain horizontal distance of a landing target, accounting for random wind displacements. Develop a PDF for the random variable $R$, the landing distance from the target.

Without much intuition, no data, or a very good physical model of the situation, we are left to bootstrap a solution. A toehold can perhaps be found by narrowing the problem to a drop of a fixed, relatively short distance, such as that shown in figure 3.11.

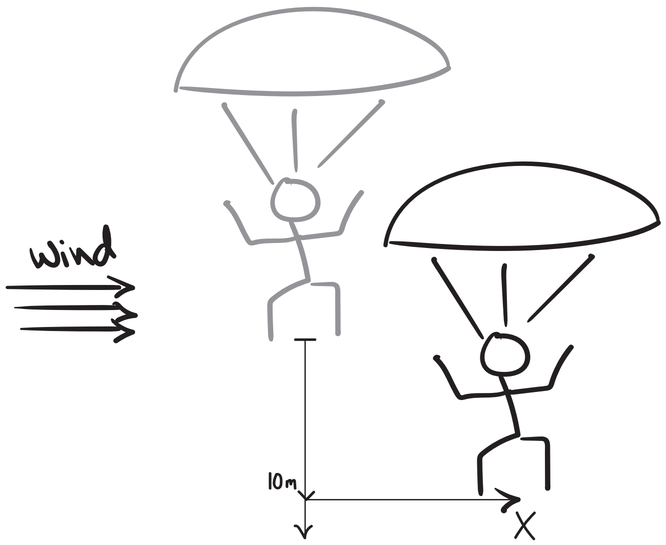

Figure 3.11. A parachutist falling 10 m and being displaced by wind an amount modeled by random variable $X$.

For each vertical drop of 10 m, we might expect a horizontal displacement of a few meters. Without any information about average prevailing winds, we cannot expect any particular direction to be most likely. It seems more likely that wind gusts would displace the parachutist a small amount than a large amount, and even less likely to displace a very large amount. These facts suggest a reasonable model to start with is a Gaussian distribution with PDF

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(x-\mu)^2}{2\sigma^2},$$

where $\mu = 0$ m and $\sigma = 5$ m. This model could clearly be improved with some data or a detailed analysis of the physics involved, but this seems to be a reasonable place to begin.

From here, we can extrapolate. For one 10-m drop, the displacement random variable is $X$. For two 10-m drops, the displacement random variable is $2X$, and so on. We conclude that for $N$ drops of 10 m, the landing displacement random variable $R$ is

$$R = NX.$$

Here we have assumed the parachutist lands after $N$ drops of 10 m. Another way of writing this is

$$R = h(X) = NX.$$

The function $h$ transforms random variable $X$ (with value $x$) to random variable $R$ with value $(r)$.

We can apply equation (3.10) directly to find the PDF of $R$ as follows:

$$f_Y(y) = \frac{1}{|dr/dx|} f_X(x)\bigg|_{x \mapsto h^{-1}(r)} \tag{3.11}$$

$$= \frac{1}{N} \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(r/N - \mu)^2}{2\sigma^2}. \tag{3.12}$$

Letting $\mu' = N\mu$ and $\sigma' = N\sigma$, we obtain

$$f_R(r) = \frac{1}{\sqrt{2\pi}\sigma'} \exp \frac{-(x - \mu')^2}{2\sigma'^2}.$$

That is, $R$ also has a Gaussian PDF. We see that the linear transformation has simply transformed the mean $\mu$ and standard deviation $\sigma$ accordingly.

We observe that for greater $N$ (higher jumps), the standard deviation is also greater. This is an intuitive result. We now turn to Python for graphical and simulation purposes.

Load the necessary packages:

```python
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

Define fixed parameters:

```python
mu = 0.0 # Mean of the Gaussian distribution for the 10 m drop
sigma = 5.0 # Standard deviation of the Gaussian distribution for the 10 m drop
```

Define the 10 m drop Gaussian distribution $f_X(x)$ symbolically

```
x, r, N = sp.symbols('x, r, N', real=True)
f_X = 1/(sigma * sp.sqrt(2 * sp.pi)) * sp.exp(-(x - mu)**2 / (2 * sigma**2))
print(f_X)
```

```
0.1*sqrt(2)*exp(-0.02*x**2)/sqrt(pi)
```

Define the functional relationship between $X$ and $R$, the horizontal distance from the initial drop point

```
h_eq = sp.Eq(r, N * x)
h_sol = sp.solve(h_eq, r, dict=True)[0]
h_inv = sp.solve(h_eq, x, dict=True)[0]
dr_dx = sp.diff(h_sol[r], x)
print(dr_dx)
```

```
N
```

Define symbolically $f_R(r)$, the probability density function for the horizontal distance from the initial drop point:

```
f_R = 1/sp.Abs(dr_dx) * f_X.subs(h_inv)
print(f_R)
```

```
0.1*sqrt(2)*exp(-0.02*r**2/N**2)/(sqrt(pi)*Abs(N))
```

Lambdify the PDF for numerical evaluation

```
f_R_fun = sp.lambdify((r, N), f_R, 'numpy')
```

Plot the PDF $f_R(r)$ for several values of $N$:

```
N_vals = np.array([600, 800, 1000])/10  # Drop steps of 10 m
r_vals = np.linspace(-1000, 1000, 1001)
fig, ax = plt.subplots()
for N_val in N_vals:
    p_vals = f_R_fun(r_vals, N_val)
    ax.plot(r_vals, p_vals, label=f'N = {N_val}')
ax.set_xlabel('$r_f$ (m)')
ax.set_ylabel('$p(r_f)$')
ax.legend()
plt.draw()
```

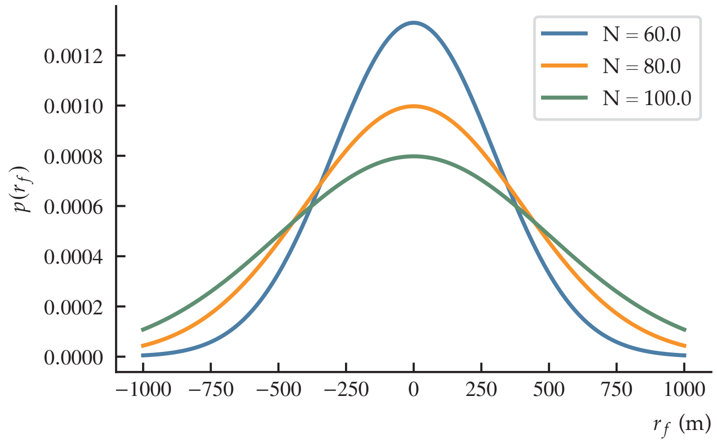

Figure 3.12. Probability density function $f_R(r)$ for several values of $N$

Compute the probability of landing within ±500 m of the initial drop point:

```
r_min, r_max = -500, 500
p_landing = sp.integrate(f_R, (r, r_min, r_max))
print(p_landing)
```

```
Piecewise((0.707106781186548*sqrt(2)*Abs(N)*erf(70.7106781186548/Abs(N))/N,
↪   N >= 0), (-
↪   0.707106781186548*sqrt(2)*Abs(N)*erf(70.7106781186548/Abs(N))/N,
↪   True))
```

Plot the probability of landing within ±500 m of the initial drop point as a function of $N$:

```
N_vals = np.linspace(1, 1200, 1001)
p_landing_fun = sp.lambdify(N, p_landing, 'numpy')
p_landing_vals = np.zeros(N_vals.shape[0])  # Preallocate
for i, N_val in enumerate(N_vals):
    p_landing_vals[i] = p_landing_fun(N_val)  # Evaluate
fig, ax = plt.subplots()
ax.plot(N_vals * 10, p_landing_vals)
ax.set_xlabel('Drop height (m)')
ax.set_ylabel('$p(\pm 500 m)$')
plt.draw()
```

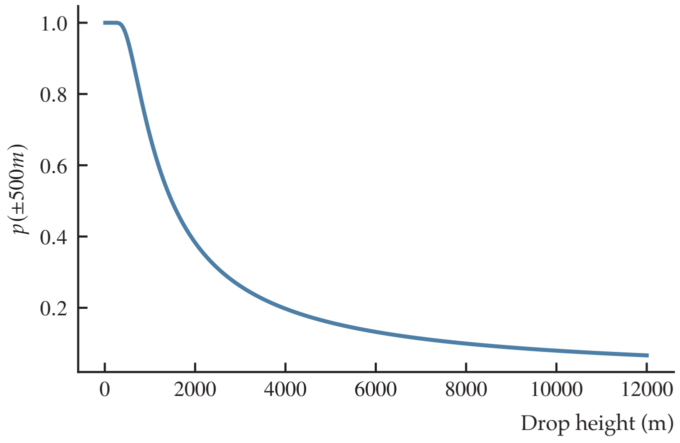

Figure 3.13. Probability of landing within ±500 m of the initial drop point as a function of $N$

Define a function to take one 10 m drop:

```python
def take_drop(x_previous):
    x_new = x_previous + np.random.normal(mu, sigma)
    return x_new
```

Define a function to simulate a random walk:

```python
def simulate_random_walk(N_sim):
    y_sim = np.flip(np.arange(0, N_sim + 1)) * 10  # Heights
    x_sim = np.zeros(N_sim + 1)  # Preallocate
    x_sim[0] = 0  # Initial drop point
    for i in range(1, N_sim + 1):
        x_sim[i] = take_drop(x_sim[i - 1])
    return x_sim, y_sim
```

Simulate several random walks (drops) for various values of $N$:

```python
N_vals = [60, 80, 100]
n_sim = 50  # Number of simulations
x_sims = [np.zeros((n_sim, N_val+1)) for N_val in N_vals]  # Preallocate
y_sims = [np.zeros((n_sim, N_val+1)) for N_val in N_vals]  # Preallocate
for i, N_val in enumerate(N_vals):
    for j in range(n_sim):
        x_sim, y_sim = simulate_random_walk(N_val)
        x_sims[i][j] = x_sim
        y_sims[i][j] = y_sim
```

Plot the random walks (drops) for several values of $N$:

```
fig, ax = plt.subplots()
for i, N_val in enumerate(N_vals):
    for j in range(n_sim):
        ax.plot(
            x_sims[i][j], y_sims[i][j],
            color=f'C{i}', alpha=[0.7, 0.5, 0.3][i]
        )
ax.set_xlabel('Horizontal distance (m)')
ax.set_ylabel('Height (m)')
plt.show()
```

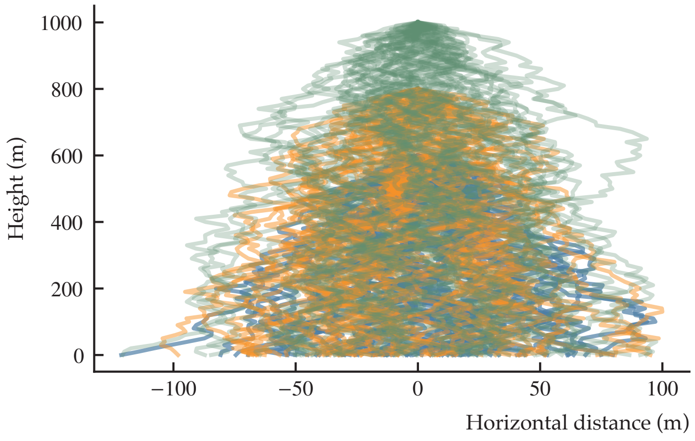

Figure 3.14. Random walks (drops) for several values of $N$

## 3.10 Multivariate Probability and Correlation

Thus far, we have considered probability density and mass functions (PDFs and PMFs) of only one random variable. But, of course, often we measure multiple random variables $X_1, X_2, \ldots, X_n$ during a single event, meaning a measurement consists of determining values $x_1, x_2, \ldots, x_n$ of these random variables.

We can consider an $n$-tuple of continuous random variables to form a sample space $\Omega = \mathbb{R}^n$ on which a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$, called the **joint PDF** assigns a probability density to each outcome $\boldsymbol{x} \in \mathbb{R}^n$. The joint PDF must be greater than or equal to zero for all $\boldsymbol{x} \in \mathbb{R}^n$, the multiple integral over $\Omega$ must be unity, and the multiple integral over a subset of the sample space $A \subset \Omega$ is the probability of the event $A$.

We can consider an $n$-tuple of discrete random variables to form a sample space $\mathbb{N}_0^n$ on which a multivariate function $f : \mathbb{N}_0^n \to \mathbb{R}$, called the **joint PMF** assigns a probability to each outcome $\boldsymbol{x} \in \mathbb{N}_0^n$. The joint PMF must be greater than or equal to zero for all $\boldsymbol{x} \in \mathbb{N}_0^n$, the multiple sum over $\Omega$ must be unity, and the multiple sum over a subset of the sample space $A \subset \Omega$ is the probability of the event $A$.

### Example 3.12

Let's visualize multivariate PDFs by plotting a bivariate gaussian using the `scipy.stats` function `multivariate_normal`

We proceed in Python. First, load packages:

```
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Define the mean and covariance matrix for a bivariate Gaussian distribution.

```
mu = [10, 20]  # Mean
Sigma = [[1, 0], [0, 0.2]]  # Covariance matrix
```

Generate grid points as input for the PDF.

```
x1_a = np.linspace(mu[0] - 5 * np.sqrt(Sigma[0][0]), mu[0] + 5 * np.sqrt(Sigma[0][0])
x2_a = np.linspace(mu[1] - 5 * np.sqrt(Sigma[1][1]), mu[1] + 5 * np.sqrt(Sigma[1][1])
```

Create a meshgrid.

```
X1, X2 = np.meshgrid(x1_a, x2_a)
```

Calculate the PDF.

```
pos = np.dstack((X1, X2))
rv = multivariate_normal(mu, Sigma)
f = rv.pdf(pos)
```