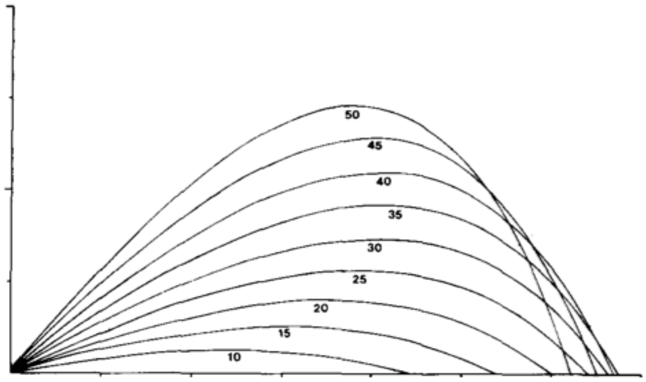
Problem statement

Overview



A gas canister explodes on a flat surface. Several small pieces of debris are sent flying in all directions. Where will they land?

Modelling assumptions

- The pieces can be modelled as free projectiles under uniform gravity $g = 9.81\ ms^{-2}$ with no air resistance.
- Each piece started from the same point, with randomly-varying initial speeds and angles.
- The speed u and angle θ have (arbitrary) probability density functions written as

$$f_U(u) \ \text{ and } \ f_\Theta(\theta), \ \text{ for } \ 0 \leq u < \infty \ \text{ and } \ 0 \leq \theta \leq \frac{\pi}{2}.$$

- The speed and the angle are independent.

Useful formulas

- The range of a projectile is given by

$$x = g(u, \theta) = \frac{u^2 \sin 2\theta}{g}.$$

In [2]:

```
# import all required modules at the start

import warnings

import numpy as np
from matplotlib import pyplot as plt
from matplotlib import rcParams
from scipy.integrate import dblquad, quad
from scipy.stats import uniform, triang
import plotly.graph_objects as go

# settings for notebook

%matplotlib inline
%config InlineBackend.figure_formats = 'svg'
```

```
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Arial']

warnings.filterwarnings('ignore')
```

Finding the distribution for the range x

From the definition of the CDF, we have $F_X(x) = P(X \leq x) = P(g(U, \Theta) \leq x)$. To work out the RHS, we integrate the joint PDF over the region of all u and θ which gives X less than x:

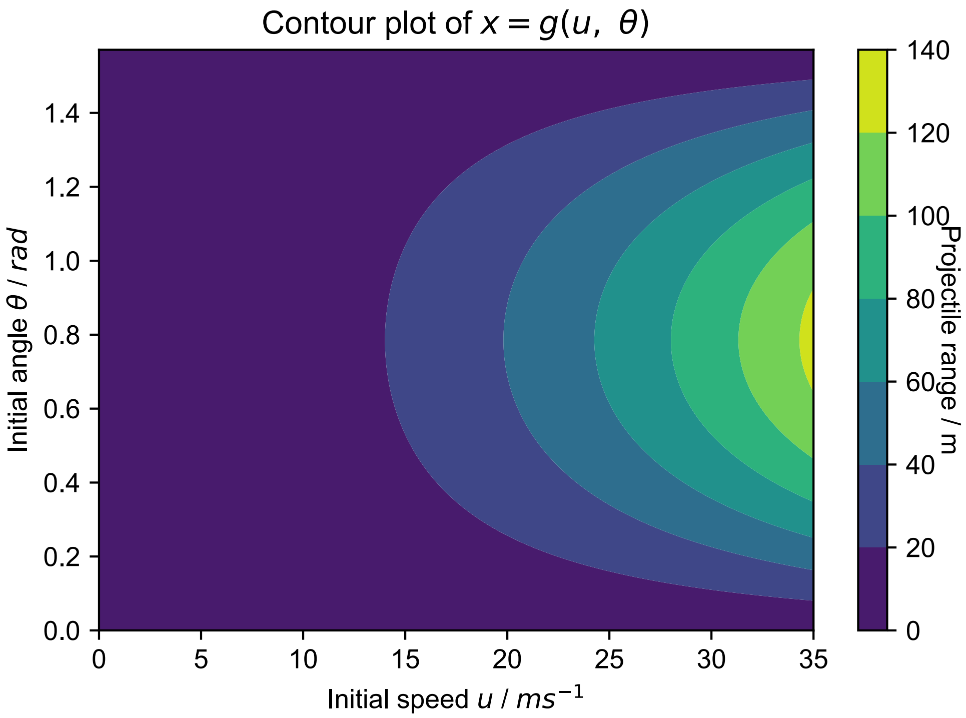
$$F_X(x) = \int_{(u,\ \theta):\ g(u,\ \theta)\ \leq\ x} f_{u,\ \theta}(u,\ \theta)\ \mathrm{d}u\ \mathrm{d}\theta.$$

where $f_{u,\ \theta}(u,\ \theta)$ is the joint probability density function, which depends on the distributions we choose to model the random variables u and θ as having. We will leave this as an arbitrary function for now.

In order to work out the region of integration, we will look at the contour plot for $g(u,\ \theta)$.

In [4]:

```
G = 9.81
g = lambda u, theta: (u ** 2 * np.sin(2 * theta)) / G

u_range = np.linspace(0, 35, 100)
theta_range = np.linspace(0, np.pi / 2, 100)

U, Theta = np.meshgrid(u_range, theta_range)
X = g(U, Theta)

contour_plot = plt.contourf(U, Theta, X)
color_bar = plt.colorbar(contour_plot)
color_bar.set_label('Projectile range / m', rotation=270)

plt.title(r'Contour plot of $ x = g(u, \ \theta) $')
plt.xlabel(r'Initial speed $ u $ / $ ms^{-1} $')
plt.ylabel(r'Initial angle $ \theta $ / $ rad $')
plt.show()
```



After seeing the contour plot, it will be easier to integrate over the region where x is *larger* than some value, then subtract it from 1. So our formula will use

$$F_X(x) = 1 - \int_{(u,\theta):\ g(u,\theta)\geq x} f_{u,\theta}(u,\theta)\ \mathrm{d}u\ \mathrm{d}\theta$$

We now need to find the limits of integration. On the u-axis, the region will be from the left-most peak of the contour, all the way up to infinity. This peak will always have $\theta = \frac{pi}{4}$, which allows us to easily solve for value of u at the peak:

$$x = \frac{u^2 \sin\left(2 \times \frac{\pi}{4}\right)}{g} \quad \rightarrow \quad u = \sqrt{gx}.$$

Next, for a given value of u, we need the range of θ. This will be a closed interval between the lower and upper intersections with the contour. The lower limit and upper limit will be, respectively,

$$\theta_- = \frac{1}{2}\sin^{-1}\frac{gx}{u^2} \ \text{ and } \ \theta_+ = \frac{1}{2}\cos^{-1}\frac{gx}{u^2}.$$

So our formula is

$$F_X(x) = 1 - \int_{\sqrt{gx}}^{\infty} \int_{\frac{1}{2}\sin^{-1}\frac{gx}{u^2}}^{\frac{\pi}{2}-\frac{1}{2}\sin^{-1}\frac{gx}{u^2}} f_{u,\,\theta}(u,\ \theta)\ \mathrm{d}\theta\ \mathrm{d}u.$$

If we know $f_{u,\,\theta}(u,\ \theta)$, then this is a formula which we can work with...

Choosing our Statistical Model

There is lots of room for experimentation here i.e. how does changing the input distributions affect the distribution of projectile landing points. In this particular model, we will use:

- u has a Rayleigh distribution with $\sigma = 12$
- θ has a triangular distribution with a mode of $\theta = 45^\circ$ and is zero at $\theta = 0$ and $\theta = 90^\circ$.

Then we have the following marginal PDFs:

$$f_U(u) = \begin{cases} \frac{u}{144} \exp\left(-\frac{u^2}{288}\right), & u \geq 0, \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad f_\Theta(\theta) = \begin{cases} \frac{4\pi - |4\pi - 16\theta|}{\pi^2}, & 0 \leq \theta \leq \pi/2, \\ 0, & \text{otherwise} \end{cases}$$

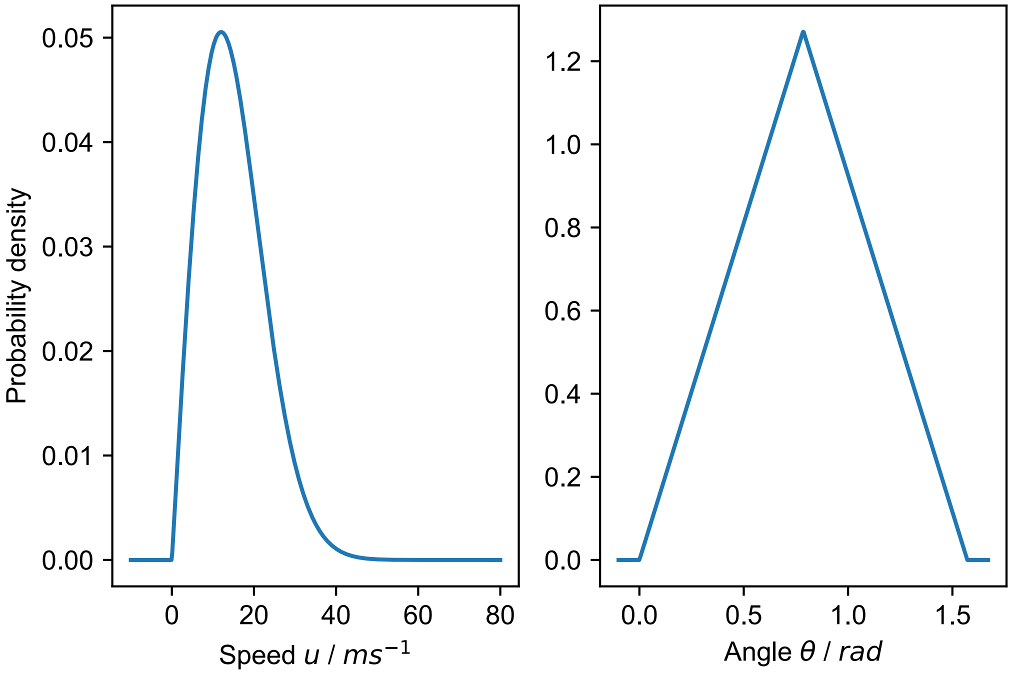
Since u and θ are independent, the joint PDF is simply their product:

$$f_{U,\,\Theta} = f_U(u) \times f_\Theta(\theta) = \begin{cases} \frac{u}{144} \exp\left(-\frac{u^2}{288}\right) \frac{4\pi - |4\pi - 16\theta|}{\pi^2}, & u \geq 0 \ \text{ and } \ 0 \leq \theta \leq \pi/2, \\ 0, & \text{otherwise} \end{cases}.$$

These functions are implemented and plotted here.

In [5]:

```
# marginal PDFs

def f_U(u: float) -> float:
    # Rayleigh PDF
    if u >= 0:
        return (u / 144) * np.exp(-u ** 2 / 288)
    else:
        return 0

def f_Theta(theta: float) -> float:
    # triangular PDF
    if 0 <= theta <= np.pi / 2:
        return (4 * np.pi - np.abs(4 * np.pi - 16 * theta)) / (np.pi ** 2)
    else:
        return 0

# joint PDF
def f_U_Theta(u: float, theta: float) -> float:
    if u >= 0 and 0 <= theta <= np.pi / 2:
        return (u / 144) * np.exp(-u ** 2 / 288) * (4 * np.pi - np.abs(4 * np.pi - 1
    else:
        return 0

# plots
u_range = np.linspace(-10, 80, 500)
theta_range = np.linspace(-0.1, np.pi / 2 + 0.1, 500)

fig, (ax0, ax1) = plt.subplots(1, 2)
ax0.plot(u_range, [f_U(u) for u in u_range])
ax0.set_xlabel(r'Speed $ u $ / $ ms^{-1} $')
ax0.set_ylabel(r'Probability density')
ax1.plot(theta_range, [f_Theta(theta) for theta in theta_range])
ax1.set_xlabel(r'Angle $ \theta $ / $ rad $')
plt.show()
```



We can now implement the CDF of x:

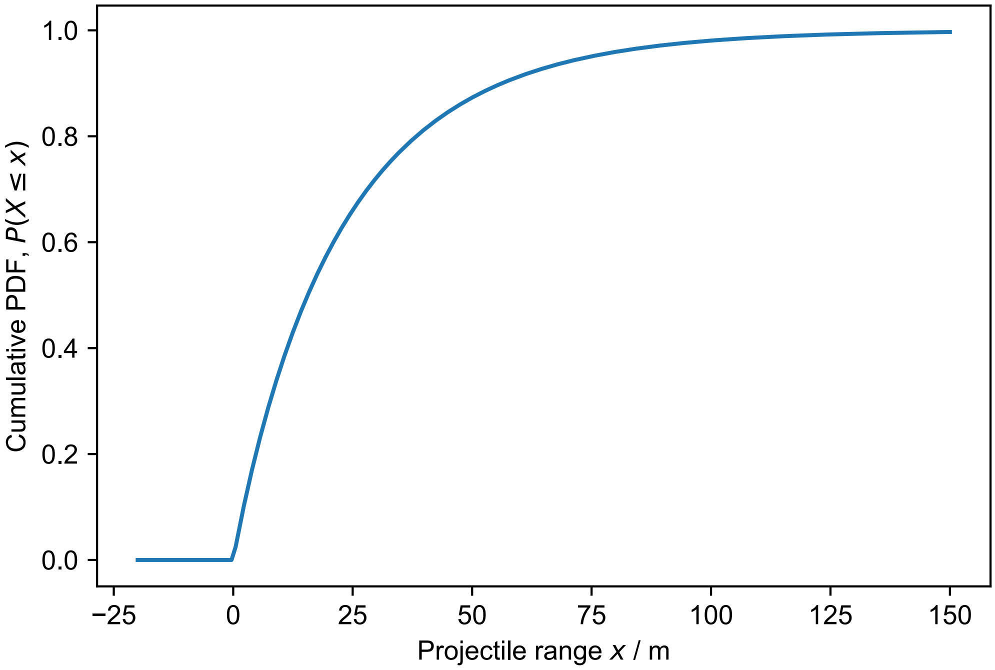
In [6]:

```
def F_X(x: float) -> float:
    '''
    Represents the CDF of x.

    #### Arguments

    `x` (float): the projectile range, in m.

    #### Returns

    float: the value of F_X(x).
    '''

    if isinstance(x, np.ndarray):
        return np.array(list(map(F_X, x)))
    else:
        if x > 0:

            f_theta_u = lambda theta, u: f_U_Theta(u, theta)  # scipy's x and y are
            theta_min = lambda u: 1/2 * np.arcsin(G * x / u ** 2)
            theta_max = lambda u: 1/2 * (np.pi - np.arcsin(G * x / u ** 2))

            fx, abserr = dblquad(f_theta_u, np.sqrt(G * x), np.Infinity, theta_min,
            return 1 - fx

        else:
            return 0
```

Plot the CDF of x:

In [7]:

```
x_range = np.linspace(-20, 150, 200)

plt.plot(x_range, F_X(x_range))
plt.xlabel(r'Projectile range $ x $ / m')
plt.ylabel(r'Cumulative PDF, $ P(X \leq x) $')
plt.show()
```



Next, the PDF. Instead of differentiating this numerically, which would be slow and inaccurate due to the complexity of the function, we will manipulate algebraically first. We have

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x) = -\frac{\mathrm{d}}{\mathrm{d}x} \int_{\sqrt{gx}}^{\infty} \int_{\frac{1}{2}\sin^{-1}\frac{gx}{u^2}}^{\frac{\pi}{2}-\frac{1}{2}\sin^{-1}\frac{gx}{u^2}} f_{u,\,\theta}(u,\ \theta)\ \mathrm{d}\theta\ \mathrm{d}u.$$

We will need to use the Leibniz rule twice to expand this. First, treat the inner integral as an arbitrary function $F(x, u)$:

$$\frac{\mathrm{d}}{\mathrm{d}x} \int_{\sqrt{gx}}^{\infty} F(x, u)\ \mathrm{d}u = -F(x, \sqrt{gx}) \times \frac{g}{2\sqrt{gx}} + \int_{\sqrt{gx}}^{\infty} \frac{\partial F}{\partial x}\ \mathrm{d}u$$

Now, work out the partial derivative of F:

$$\frac{\partial F}{\partial x} = \frac{\partial}{\partial x} \int_{\frac{1}{2}\sin^{-1}\frac{gx}{u^2}}^{\frac{\pi}{2}-\frac{1}{2}\sin^{-1}\frac{gx}{u^2}} f_{u,\,\theta}(u,\ \theta)\ \mathrm{d}\theta = -\frac{g}{2\sqrt{u^4 - g^2x^2}} \Bigg(f_{u,\,\theta}(u,\ \frac{\pi}{2} - \frac{1}{2}\sin^{-1}\frac{gx}{u^2}) + f_{u,\,\theta}(u,\ \cdot$$

Since $F(x, \sqrt{gx})$ = 0, we get our PDF as:

$$f_X(x) = \int_{\sqrt{gx}}^{\infty} \frac{g}{2\sqrt{u^4 - g^2x^2}} \Bigg(f_{u,\,\theta}(u,\ \frac{\pi}{2} - \frac{1}{2}\sin^{-1}\frac{gx}{u^2}) + f_{u,\,\theta}(u,\ \frac{1}{2}\sin^{-1}\frac{gx}{u^2}) \Bigg)\ \mathrm{d}u.$$

This function is implemented now.

In [8]:

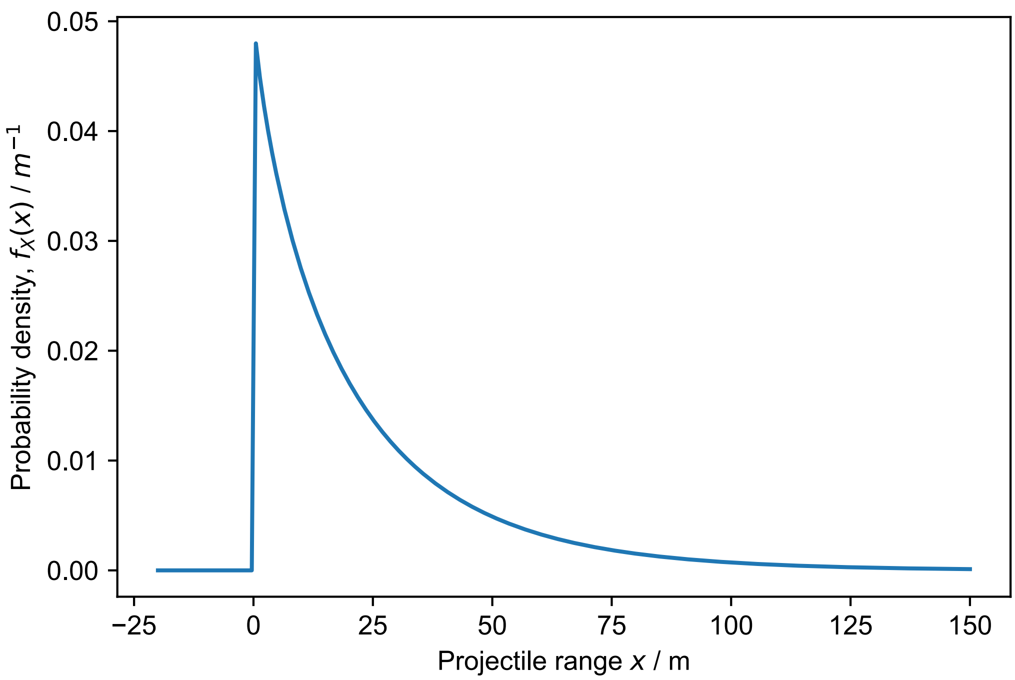
```
def f_X(x: float) -> float:
    '''
    Represents the PDF of x.
```

```
    #### Arguments

    `x` (float): the projectile range, in m.

    #### Returns

    float: the value of f_X(x).
    '''

    if isinstance(x, np.ndarray):
        return np.array(list(map(f_X, x)))
    else:
        if x > 0:

            p1 = lambda u: f_U_Theta(u, 1/2 * (np.pi - np.arcsin(G * x / u ** 2)))
            p2 = lambda u: f_U_Theta(u, 1/2 * np.arcsin(G * x / u ** 2))
            p3 = lambda u: np.sqrt(u ** 4 - (G * x) ** 2)
            f_x_integrand = lambda u: (p1(u) + p2(u)) / p3(u)

            y, abserr = quad(f_x_integrand, np.sqrt(G * x), np.Infinity)
            return (G / 2) * y
        else:
            return 0
```

Now we can plot the PDF of x:

In [9]:

```
plt.plot(x_range, f_X(x_range))
plt.xlabel(r'Projectile range $ x $ / m')
plt.ylabel(r'Probability density, $ f_X(x) $ / $ m^{-1} $')
plt.show()
```



At first glance this seems reasonable, but let's evaluate some summary statistics.

Finding some properties of the distribution of x

Let's check more carefully, by looking at the mean range, $E[X]$. By LOTUS, we should get

$$E[X] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(u, \theta) \times f_{u,\,\theta}(u,\ \theta)\ \mathrm{d}u\ \mathrm{d}\theta$$

while from our PDF, we should get

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)\ \mathrm{d}x.$$

We can calculate both of these values explicitly and check they are equal:

In [10]:

```
def find_mean_range_lotus() -> float:
    '''
    Calculates E[X] using only the joint PDF and the function x = g(u, theta).

    #### Returns

    float: calculated mean range E[X].
    '''

    return dblquad(lambda u, theta: g(u, theta) * f_U_Theta(u, theta), np.NINF, np.I


def find_mean_range_pdf() -> float:
    '''
    Calculates E[X] using the PDF of x.

    #### Returns

    float: calculated mean range E[X].
    '''

    return quad(lambda x: x * f_X(x), np.NINF, np.Inf)


mean_lotus, lotus_abserr = find_mean_range_lotus()
mean_pdf, pdf_abserr = find_mean_range_pdf()

print(f'Mean range using LOTUS: {mean_lotus}, abserr: {lotus_abserr}')
print(f'Mean range using PDF: {mean_pdf}, abserr: {pdf_abserr}')
print(f'Difference: {abs(mean_lotus - mean_pdf)}, percent: {abs(mean_lotus - mean_pd
```

```
Mean range using LOTUS: 23.79653487392079, abserr: 4.3583121253261967e-07
Mean range using PDF: 23.796468474823975, abserr: 9.023399982766023e-08
Difference: 6.639909681638301e-05, percent: 2.7902920505465535e-06
```

We can see that they are in fact equal - to within 3 d.p.

We can also get the variance:

In [11]:

```
def find_variance() -> float:
    '''
    Calculates Var[X] using LOTUS.

    #### Returns

    float: calculated variance in range, Var[X].
    '''

    return dblquad(lambda u, theta: (g(u, theta) - mean_lotus) ** 2 * f_U_Theta(u, t
```

```
        np.NINF, np.Inf, np.NINF, np.Inf)


variance, abserr = find_variance()

print(f'Variance of range: {variance}, abserr: {abserr}')
print(f'Standard deviation of range: {np.sqrt(variance)}')
```

```
Variance of range: 644.9121749817065, abserr: 1.0257125448148675e-05
Standard deviation of range: 25.39512108617926
```

By visual inspection of the graph of the PDF again, these values seem reasonable.

Next we can look at the median and quartiles. This will be easiest done by reading from the CDF.

In [12]:
```
def find_quartiles() -> tuple[float]:
    '''
    Finds the LQ, Med, UQ of the range x.

    #### Returns

    tuple[float]: (lower quartile, median, upper quartile).
    '''

    x_range = np.linspace(0, 50, 200)
    cdf_vals = F_X(x_range)

    lower_quartile  = x_range[np.argmin(np.abs(cdf_vals - 0.25))]
    median          = x_range[np.argmin(np.abs(cdf_vals - 0.5))]
    upper_quartile  = x_range[np.argmin(np.abs(cdf_vals - 0.75))]

    return (lower_quartile, median, upper_quartile)


lower_quartile, median, upper_quartile = find_quartiles()

print(f'LQ: {lower_quartile}, Med: {median}, UQ: {upper_quartile}')
print(f'IQR: {upper_quartile - lower_quartile}')
```

```
LQ: 6.281407035175879, Med: 15.57788944723618, UQ: 32.66331658291457
IQR: 26.38190954773869
```

Since the median is less than the mean (15.58 < 23.80), the distribution is skewed right (positive skew), which again matches what we see in the PDF graph.

Finally, calculate skewness and kurtosis quantitatively.

In [13]:
```
def find_skew() -> float:
    '''
    Finds the skewness of the range x.

    #### Returns

    float: calculated skewness of x.
    '''

    return dblquad(lambda u, theta: ((g(u, theta) - mean_lotus) / np.sqrt(variance))
        f_U_Theta(u, theta), np.NINF, np.Inf, np.NINF, np.Inf)


def find_kurtosis() -> float:
    '''
```

```
    Finds the kurtosis of the range x.

    #### Returns

    float: calculated kurtosis of x.
    '''

    return dblquad(lambda u, theta: ((g(u, theta) - mean_lotus) / np.sqrt(variance))
        f_U_Theta(u, theta), np.NINF, np.Inf, np.NINF, np.Inf)


skewness, skew_abserr = find_skew()
kurtosis, kurt_abserr = find_kurtosis()
excess_kurtosis = kurtosis - 3

print(f'Skewness: {skewness}, abserr: {skew_abserr}')
print(f'Kurtosis: {kurtosis}, abserr: {kurt_abserr}')
print(f'Excess kurtosis: {excess_kurtosis}')
```

```
Skewness: 2.210139233071541, abserr: 3.252055447708984e-08
Kurtosis: 10.437259602819692, abserr: 1.6724088215913603e-07
Excess kurtosis: 7.437259602819692
```

This confirms the positive skew, and we also see a positive excess kurtosis (leptokurtic), which means the tails are less filled out relative to the Normal distribution.

Simulation with Random Numbers (Monte-Carlo)

We can also simulate the problem directly, by sampling from the chosen distributions and calculating range for a very large number of trials. We will use 100,000 trials here, and also add a degree of freedom to give a radially-symmetric distribution. The top-down angle (written as ϕ) will be uniformly distributed on $[0, 2\pi)$.

In [14]:

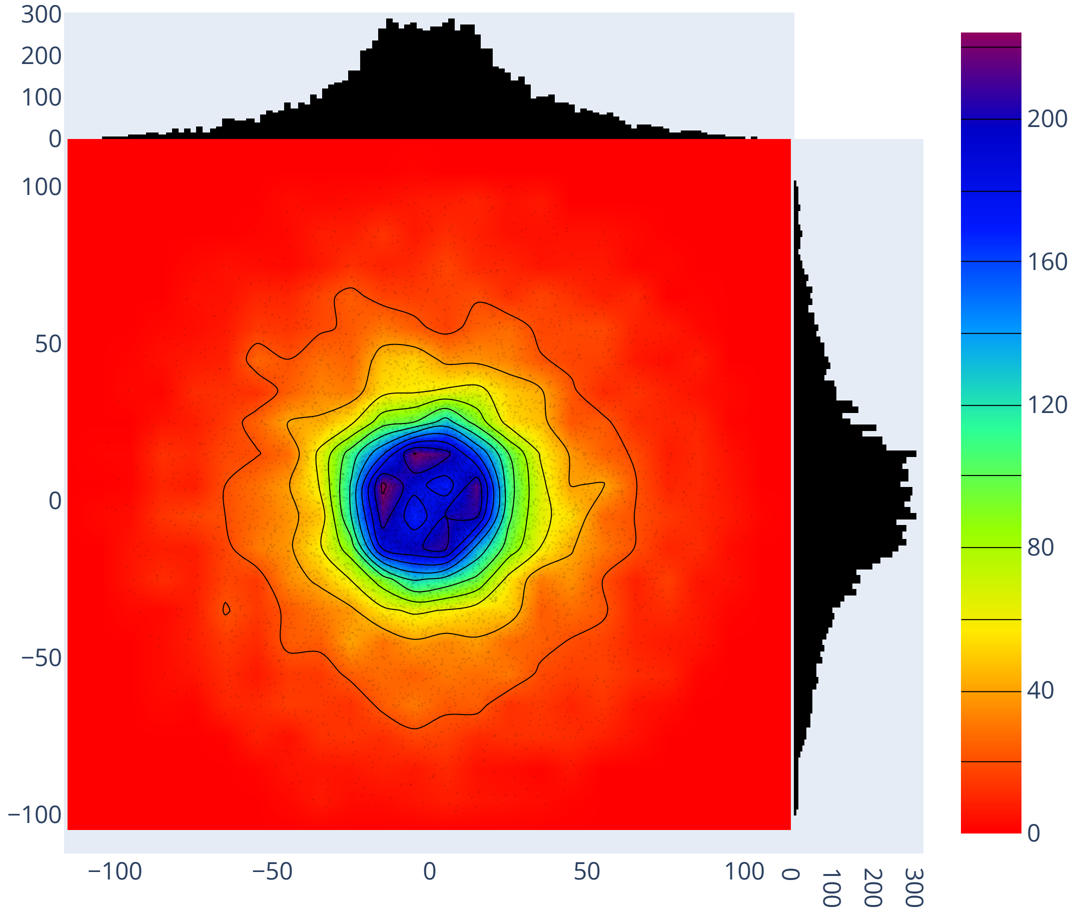
```
NUM_TRIALS = 10_000
PHI_SAMPLES = 10_000

u_samples = uniform.rvs(loc=12, scale=20, size=NUM_TRIALS)
theta_samples = triang.rvs(scale=np.pi / 2, c=np.pi / 4, size=NUM_TRIALS)

# polar angle when viewed top-down: gives a 2D landing position
phi_samples = np.tile(np.linspace(0, 2 * np.pi, PHI_SAMPLES), round(NUM_TRIALS / PHI

r_samples = g(u_samples, theta_samples)  # range is now radial
x_samples, y_samples = r_samples * np.cos(phi_samples), r_samples * np.sin(phi_sampl
```

Finally, let's plot our simulated landing distribution:

In [15]:

```
fig = go.Figure()

fig.add_trace(go.Histogram2dContour(
    x=x_samples, y=y_samples, colorscale='rainbow', reversescale=True, xaxis='x', ya
    contours_coloring='heatmap'))

fig.add_trace(go.Scatter(
    x=x_samples, y=y_samples, xaxis='x', yaxis='y', mode='markers',
    marker=dict(color='rgba(0, 0, 0, 0.1)', size=1)))

fig.add_trace(go.Histogram(y=y_samples, xaxis='x2', marker=dict(color='rgba(0, 0, 0,
```

```
fig.add_trace(go.Histogram(x=x_samples, yaxis='y2', marker=dict(color='rgba(0, 0, 0,

fig.update_layout(
    autosize=False, xaxis=dict(zeroline=False, domain=[0,0.85], showgrid=False),
    yaxis=dict(zeroline=False, domain=[0,0.85], showgrid=False),
    xaxis2=dict(zeroline=False, domain=[0.85,1], showgrid=False),
    yaxis2=dict(zeroline=False, domain=[0.85,1], showgrid=False),
    height=600, width=600, bargap=0, hovermode='closest', showlegend=False)

fig.show()
```



The densely-populated dark blue ring close to the centre shows the region where most of the pieces land. This is consistent with our distribution being more populated at smaller ranges. The marginal distributions, which should match our PDF, also resemble the correct shape.