Computer Exercise 2: Sampling from Discrete Distributions

In [ ]:
```
# Plotting
import plotly.graph_objects as go
import plotly.express as px
import plotly.subplots as sp
import plotly.io as pio
pio.renderers.default = "notebook+pdf"
pio.templates.default = "plotly_dark"

# Utilities
import numpy as np
import pandas as pd
from scipy.stats import geom
from utils import chi_sq_test, kolmogorov_smirnov_test
```

Part 1 - Simulation using p

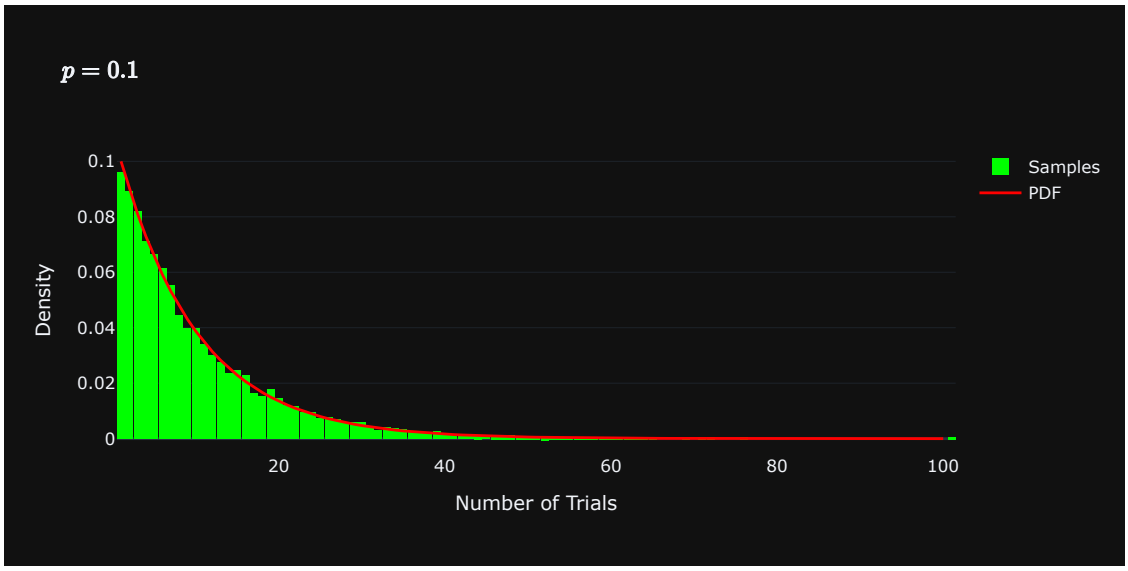
First we will simulate numbers from the geometric distribution with parameter p. We sample numbers from the uniform distribution and use the formula from the slides to calculate our samples. We choose $p = 0.1$.

In [ ]:
```
# Set p to a fixed value on the open interval between 0 and 1
p = 0.1
N = 10000

# Generate uniform random numbers
U = np.random.uniform(0, 1, N)

# Use formula from lecture 3 slide 9 to get geometric distributed discrete numbers
X = np.ceil(np.log(U)/np.log(1-p))

# Histogram
fig = go.Figure(go.Histogram(x=X, histnorm='probability density', marker_color='Lime', name='Samples'))
fig.add_trace(go.Scatter(x=np.arange(min(X), max(X)), y=geom.pmf(np.arange(min(X), max(X)), p), mode='lines', name='PDF', line=dict
fig.update_layout(title=f"Geometric Distribution with $p={p}$", xaxis_title="Number of Trials", yaxis_title="Density", width=800, h
fig.show()
```



We see that the samples generated clearly follow a geometric distribution with parameter $p = 0.1$.

Part 2 - Simulating 6-point distribution

For the next part we are given probabilities for a 6-point distribution. We will then sample numbers from this distribution using different methods.

First we implement functions for the methods before running the sampling and comparing.

In [ ]:
```
ps = np.array([7/48, 5/48, 1/8, 1/16, 1/4, 5/16])
```

(a) Direct method

For the direct method, we approximate the CDF from the given probabilities. We then take samples from the uniform distribution and do a linear search from the approximated CDF and return the corresponding value.

In [ ]:
```
def direct(n, ps):
    # Generate uniform random numbers
    U = np.random.rand(N)

    # Convert to discrete random numbers using the given probabilities
    X = np.searchsorted(np.cumsum(ps), U)

    return X
```

(b) Rejection method

With the rejection method we sample two numbers from the uniform distribution. We use the first number to calculate the sample value and the second number to decide if we accept the sample or not.

In [ ]:
```
def rejection(n, ps):
    c = max(ps)
    k = len(ps)

    X = np.zeros(n, dtype=int)
    for i in range(len(X)):
        while True: # Could theoretically run forever...
            U1, U2 = np.random.rand(2)
            I = np.floor(k * U1).astype(int)
            if U2 <= ps[I]/c:
                X[i] = I + 1
                break

    return X
```

(c) Alias method

To use the Alias method we first need to generate two tables. These tables will be used to keep track of when we are going to use the alias method and what the corresponding value is. We then sample from the uniform distribution and use the tables to return the corresponding value.

In [ ]:
```
def alias(N, ps):
    k = len(ps)

    # Generating Alias tables
    L = np.arange(k)
    F = k*ps
    G = np.where(F >= 1)[0]
    S = np.where(F <= 1)[0]

    while len(S) != 0:
        i = G[0]
        j = S[0]
        L[j] = i
        F[i] -= (1 - F[j])
        if F[i] < 1 - np.finfo(float).eps:
            G = np.delete(G, 0)
            S = np.append(S, i)
        S = np.delete(S, 0)

    # Computing values
    X = np.zeros(N, dtype=int)

    # Generate random numbers
    U1 = np.random.rand(N)
    U2 = np.random.rand(N)

    # Perform Alias method
    I = np.array(np.floor(k * U1)).astype(int)
    mask = U2 <= F[I]
    X[mask] = I[mask] + 1
    X[~mask] = L[I[~mask]] + 1

    return X
```

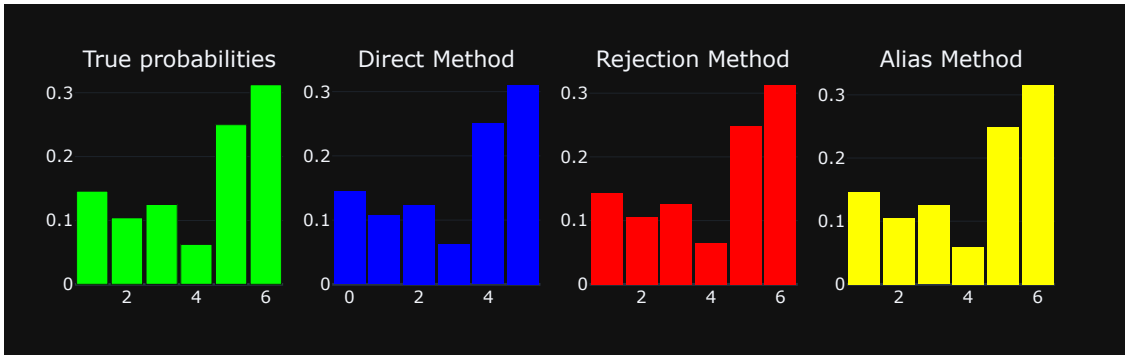
Part 3 - Comparison

We know use the three methods to generate samples from the given 6-point distribution and compare it with the given probabilities.

Histograms

In [ ]:
```
X_d = direct(N, ps)
X_r = rejection(N, ps)
X_a = alias(N, ps)

# Plot histograms
fig = sp.make_subplots(rows=1, cols=4, subplot_titles=("True probabilities", "Direct Method", "Rejection Method", "Alias Method"))
fig.add_trace(go.Bar(x=np.arange(1, 7), y=ps, marker_color='Lime'), row=1, col=1) # True probabilities
fig.add_trace(go.Histogram(x=X_d, histnorm='probability density', marker_color='Blue'), row=1, col=2) # Direct method
fig.add_trace(go.Histogram(x=X_r, histnorm='probability density', marker_color='Red'), row=1, col=3) # Rejection method
fig.add_trace(go.Histogram(x=X_a, histnorm='probability density', marker_color='Yellow'), row=1, col=4) # Alias method
fig.update_layout(height=250, width=800, showlegend=False, bargap=0.1, margin=dict(l=50, r=50, t=50, b=50))
fig.show()
```



All methods yield discrete random numbers that appear to be correctly distributed when inspecting the histogram.

Tests

In [ ]:
```
methods = ["Direct", "Rejection", "Alias"]
print("-"*36)
for i, X in enumerate([X_d, X_r, X_a]):
    tab = np.array([chi_sq_test(X, ps), kolmogorov_smirnov_test(X, ps)])
    df = pd.DataFrame(np.round(tab, 2), index=["Chi squared", "Kol-Smi"], columns=["Test statistic", "p-value"])
    print(">>> " + methods[i] + " method <<<")
    print(df)
    print("-"*36)
```

```
------------------------------------
>>> Direct method <<<
             Test statistic  p-value
Chi squared            1.17     0.95
Kol-Smi                0.00     1.00
------------------------------------
>>> Rejection method <<<
             Test statistic  p-value
Chi squared            1.68     0.89
Kol-Smi                0.00     1.00
------------------------------------
>>> Alias method <<<
             Test statistic  p-value
Chi squared            2.55     0.77
Kol-Smi                0.00     1.00
------------------------------------
```

For each method, we have calculated the p-value using both the chi squared test and the Kolmogorov Smirnov test. All p-values do not give us ground for rejecting the null hypothesis, i.e. that the data is drawn from the same distribution as the given probabilities.

Part 4 - Pros and Cons of the different methods

The rejection method has the con that to achieve the desired number of samples then it potentially needs to run for many more iterations due to the rejections. Many rejections can happen when you have some probabilities that are very low and some probabilities that are quite high. With $c = \max(p_i)$ then p_i/c will also be quite low when p_i is low. This means that the probability of accepting a sample is quite low.

The Alias method does not have any rejection, it does however have some initialization. This initialization can be quite costly if you have a large number of samples to generate. The sampling itself is however quite fast, so in cases where you need to repeatedly sample from the same distribution, the Alias method is a good choice.

The direct method is the simplest of the three methods. It does not require any initialization and is quite fast. The downside is that it requires a linear search for each sample. This can be quite costly if you have a large number of samples to generate.