Computer Exercise 3: Sampling from Continuous 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
from scipy.stats import t, chi2, kstest
```

Part 1 - Generating simulated values

In this exercise, we will generate random samples from a continuous distribution using the inverse transform method.

First we define the relevant functions needed to transform uniform samples to the needed samples.

In [ ]:
```
def exp_pdf(x, y=1):
    return y*np.exp(-y*x)

def pareto_pdf(x, k=20, beta=1):
    return k*beta**k / x**(k+1)

def gaussian_pdf(x, mu=0, sigma=1):
    return 1/np.sqrt(2*np.pi)*np.exp(-((x-mu)**2)/(2*sigma**2))

def uniform_2_exponential(U, y=1):
    X = -np.log(U)/y
    return X

def uniform_2_pareto(U, k=20, beta=1):
    X = beta*(U**(-1/k))
    return X

def uniform_2_normal(U):
    n = len(U)
    U1, U2 = U[:int(n/2)], U[int(n/2):]
    theta = 2*np.pi*U2
    r = np.sqrt(-2*np.log(U1))
    Z1, Z2 = r*np.array([np.cos(theta), np.sin(theta)])
    return np.concatenate((Z1, Z2))
```

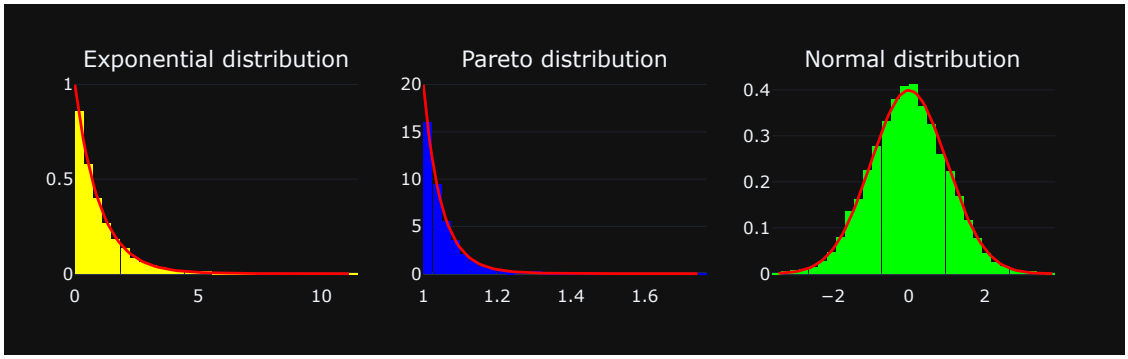
Generate random numbers from exponential, pareto and normal distributions

Using the defined functions, we can generate random samples from the exponential, pareto and normal distributions.

In [ ]:
```
U = np.random.rand(10000)

X_exp = uniform_2_exponential(U)
X_pareto = uniform_2_pareto(U)
X_norm = uniform_2_normal(U)
```

Plot histograms of the generated random numbers with superimposed pdf

In [ ]:
```
# Plot
num_bins = 30
fig = sp.make_subplots(rows=1, cols=3, subplot_titles=("Exponential distribution", "Pareto distribution", "Normal distribution"))

x_exp = np.linspace(min(X_exp), max(X_exp), 100, endpoint=True)
fig.add_trace(go.Scatter(x=x_exp, y=exp_pdf(x_exp), mode='lines', line=dict(color='red')), row=1, col=1)
fig.add_trace(go.Histogram(x=X_exp, xbins=dict(start=0, size=(x_exp[-1]-x_exp[0])/num_bins), histnorm='probability density', name='

x_pareto = np.linspace(min(X_pareto), max(X_pareto), 100, endpoint=True)
fig.add_trace(go.Scatter(x=x_pareto, y=pareto_pdf(x_pareto), mode='lines', line=dict(color='red'), showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x=X_pareto, xbins=dict(start=1, size=(x_pareto[-1]-x_pareto[0])/num_bins), histnorm='probability density

x_norm = np.linspace(min(X_norm), max(X_norm), 100, endpoint=True)
fig.add_trace(go.Scatter(x=x_norm, y=gaussian_pdf(x_norm), mode='lines', line=dict(color='red'), showlegend=False), row=1, col=3)
fig.add_trace(go.Histogram(x=X_norm, xbins=dict(size=(x_norm[-1]-x_norm[0])/num_bins), histnorm='probability density', name='Sample

fig.update_layout(height=250, width=800, showlegend=False, bargap=0.1, margin=dict(l=50, r=50, t=50, b=50))
```



Above are histrograms of the generated samples from the three distributions along with the probability density functions. It is seen that the samples generated using the inverse transform method are consistent with the theoretical pdfs.

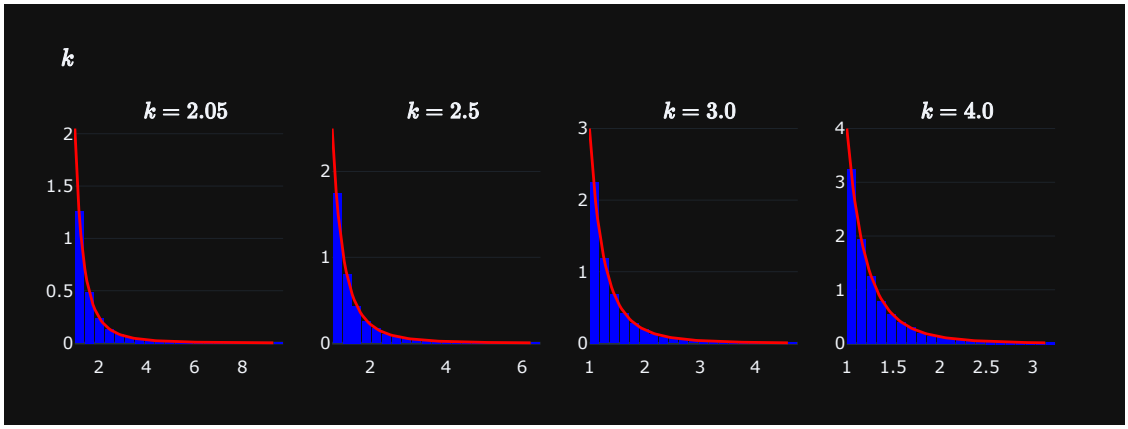
In [ ]:
```
# Perform Kolmogorov-Smirnov test for distribution type
ks_exp, exp_p = kstest(X_exp, 'expon')
ks_pareto, pareto_p = kstest(X_pareto, 'pareto', args=(20,))
ks_norm, norm_p = kstest(X_norm, 'norm')


print(f"The p-value for the exponential distribution: {exp_p:.4f}")
print(f"The p-value for the pareto distribution: {pareto_p:.4f}")
print(f"The p-value for the normal distribution: {norm_p:.4f}")
```

```
The p-value for the exponential distribution: 0.1354
The p-value for the pareto distribution: 0.1354
The p-value for the normal distribution: 0.1135
```

Using the kolmogorov-smirnov test, we can test the null hypothesis that the samples are drawn from the theoretical distribution. The p-values are all greater than 0.05, so we cannot reject the null hypothesis.

We further investigate sampling from the pareto distribution with different values of k.

In [ ]:
```
ks = np.array([2.05, 2.5, 3, 4])
ps = np.zeros(len(ks))

# plot using plotly for each of the ks
fig = sp.make_subplots(rows=1, cols=len(ks), subplot_titles=[r"$k = {}$".format(k) for k in ks])
for i, k in enumerate(ks):
    X = uniform_2_pareto(U, k)
    X_99 = X[X < np.quantile(X, 0.99)] # Remove the highest 1% of the values for plotting
    x = np.linspace(min(X_99), max(X_99), 100, endpoint=True)
    T, p = kstest(X, 'pareto', args=(k,))
    ps[i] = p
    fig.add_trace(go.Scatter(x=x, y=pareto_pdf(x, k), mode='lines', name='Pareto PDF', line=dict(color='red')), row=1, col=i+1)
    fig.add_trace(go.Histogram(x=X_99, xbins=dict(start=1, size=(max(X_99)-min(X_99))/20), histnorm='probability density', name='Pa

fig.update_layout(height=300, width=800, showlegend=False, bargap=0.1, margin=dict(l=50, r=50, t=80, b=50), title="Pareto distribut
fig.show()
```



Above we have plotted samples from the pareto distribution with different values of k along with the analytical pdf. The analytical expression for the expectation of the pareto distribution is:

$$E[X] = \frac{k}{k-1}\beta$$

So when we increase k we expect that the mean should increase towards β. This is also what see happening in the histograms above. The analytical expression for the variance of the pareto distribution is:

$$Var[X] = \frac{\beta^2 k}{(k-1)^2(k-2)}$$

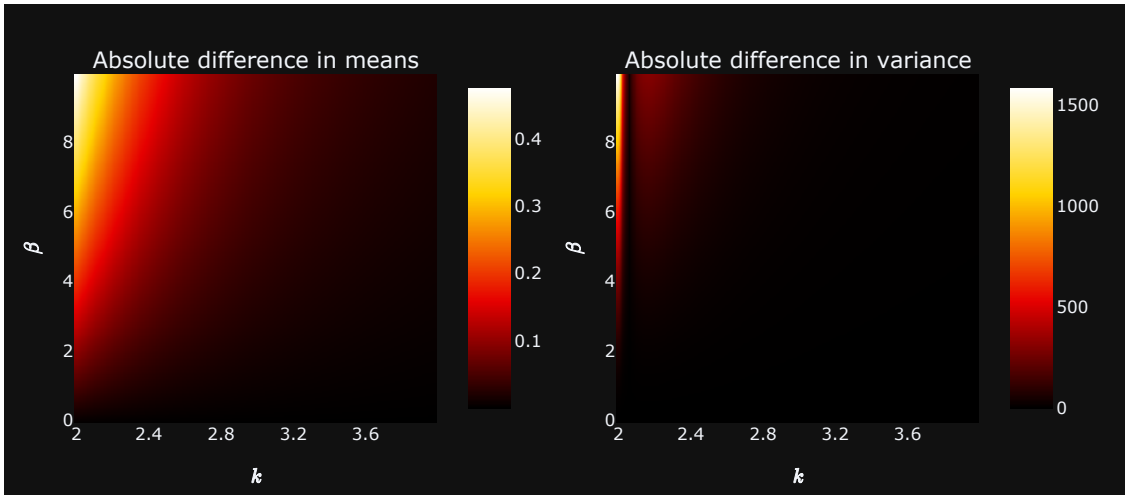
So when we increase k we expect that the variance should decrease. This is also what we see happening in the histograms above as the x-axis' range is getting smaller and smaller as k increases.

In [ ]:
```
for i, k in enumerate(ks):
    print(f"The p-value for the pareto distribution with k = {k}: {ps[i]:.4f}")
```

```
The p-value for the pareto distribution with k = 2.05: 0.1354
The p-value for the pareto distribution with k = 2.5: 0.1354
The p-value for the pareto distribution with k = 3.0: 0.1354
The p-value for the pareto distribution with k = 4.0: 0.1354
```

Part 2 - Comparison of simulation and analytic results for Pareto distribution

In [ ]:
```
nk = 100
nbeta = 100

ks = np.linspace(2.05, 4, nk)
betas = np.linspace(0.05, 10, nbeta)

mean_diff = np.zeros((nk, nbeta))
var_diff = np.zeros((nk, nbeta))
E = np.zeros((nk, nbeta))
V = np.zeros((nk, nbeta))

U = np.random.rand(10000)

for i, k in enumerate(ks):
    for j, beta in enumerate(betas):
        X = uniform_2_pareto(U, k, beta)
        EX = beta*k/(k-1)
        VarX = beta**2 * k / ((k-1)**2 * (k-2))
        mean_diff[j, i] = abs(np.mean(X) - EX)
        var_diff[j, i] = abs(np.var(X) - VarX)

fig = sp.make_subplots(rows=1, cols=2, subplot_titles=["Absolute difference in means", "Absolute difference in variance"], horizonta
fig.add_trace(go.Heatmap(z=mean_diff, colorscale='hot', zsmooth='best', colorbar_x=0.42), row=1, col=1)
fig.add_trace(go.Heatmap(z=var_diff, colorscale='hot', zsmooth='best'), row=1, col=2)
fig.update_xaxes(title_text=r"$k$", tickvals=np.arange(0, nk, 20), ticktext=np.round(ks[::20], 1), row=1, col=1)
fig.update_yaxes(title_text=r"$\beta$", tickvals=np.arange(0, nbeta, 20), ticktext=np.round(betas[::20]), row=1, col=1)
fig.update_xaxes(title_text=r"$k$", tickvals=np.arange(0, nk, 20), ticktext=np.round(ks[::20], 1), row=1, col=2)
fig.update_yaxes(title_text=r"$\beta$", tickvals=np.arange(0, nbeta, 20), ticktext=np.round(betas[::20]), row=1, col=2)
fig.update_layout(height=350, width=800, margin=dict(l=50, r=50, t=50, b=50))
fig.show()
```



As k gets close to 2 it is seen that the difference between the simulated and the analytic variance increases. The analytic variance of the Pareto distribution is undefined for $k \leq 2$, and tends towards ∞ as k approaches 2. The simulated variance is calculated as the sample variance of the generated random numbers, and is therefore always finite. This could explain the increased difference between the simulated and the analytic variance as k approaches 2.

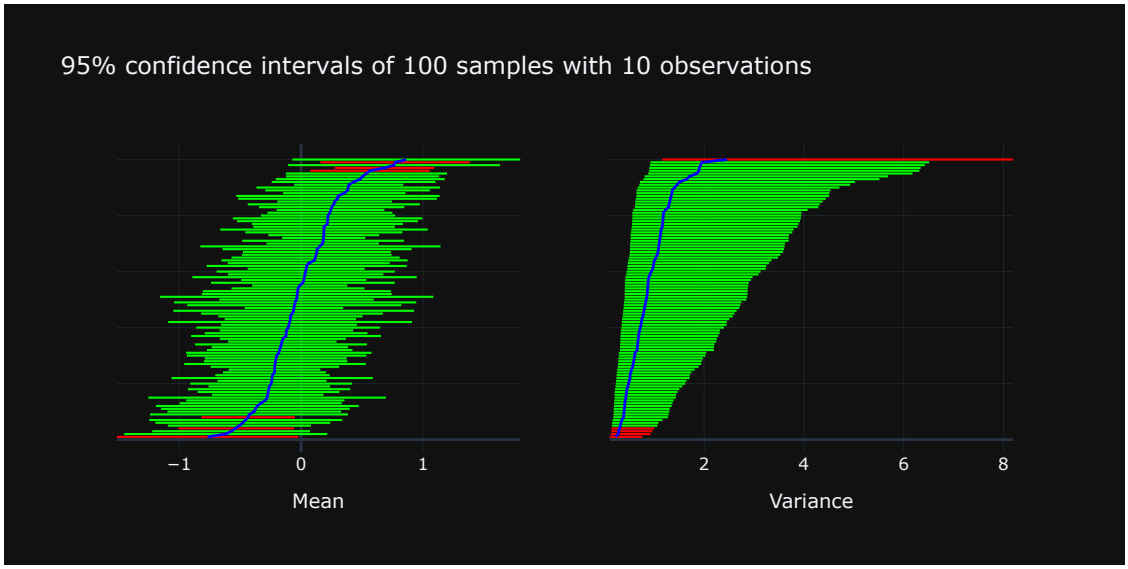
Part 3 - Confidence intervals of normal distribution

For this we generate 10 samples from the normal distribution and calculate a 95% confidence interval for the mean and the variance. We repeat that 100 times and plot the confidence intervals.

In [ ]:
```
alpha = 0.05
n_samples = 100
N = 10

# Generate samples
U = np.random.rand(n_samples, N)
X = uniform_2_normal(U)

# Compute confidence intervals
means = X.mean(axis=1)
stds = X.std(axis=1)
CI_means = means[:, None] + t.ppf([alpha/2, 1-alpha/2], N-1)[None, :] * stds[:, None]/np.sqrt(N)
CI_vars = (N-1)*stds[:, None]**2 / chi2.ppf([1-alpha/2, alpha/2], N-1)[None, :]
```

In [ ]:
```
# Sort samples
sort_idx = np.argsort(means)
means = means[sort_idx]
CI_means = CI_means[sort_idx]

sort_idx = np.argsort(stds)
stds = stds[sort_idx]
CI_vars = CI_vars[sort_idx]

# Count fraction of confidence intervals that contain the true value
mean_fraction = np.mean((CI_means[:, 0] <= 0) & (CI_means[:, 1] >= 0))
var_fraction = np.mean((CI_vars[:, 0] <= 1) & (CI_vars[:, 1] >= 1))
print(f"The fraction of CIs containing the true mean: {mean_fraction:.2f}")
print(f"The fraction of CIs containing the true variance: {var_fraction:.2f}")

# Plot
fig = sp.make_subplots(rows=1, cols=2)

# Mean
for i, CI in enumerate(CI_means):
    color = 'lime' if CI[0] <= 0 and CI[1] >= 0 else 'red'
    fig.add_trace(go.Scatter(x=CI, y=2*[(i+1)/n_samples], mode='lines', line=dict(color=color, width=1.5)), row=1, col=1)
fig.add_trace(go.Scatter(x=means, y=np.linspace(1, n_samples, n_samples, endpoint=True)/n_samples, mode='lines', line=dict(color='b

# Variance
for i, CI in enumerate(CI_vars):
    color = 'lime' if CI[0] <= 1 and CI[1] >= 1 else 'red'
    fig.add_trace(go.Scatter(x=CI, y=2*[(i+1)/n_samples], mode='lines', line=dict(color=color, width=1.5)), row=1, col=2)
fig.add_trace(go.Scatter(x=stds**2, y=np.linspace(1, n_samples, n_samples, endpoint=True)/n_samples, mode='lines', line=dict(color=

# Update the layout
fig.update_layout(height=400, width=800, title_text=f"95% confidence intervals of 100 samples with 10 observations", showlegend=Fals
fig.update_xaxes(title_text="Mean", row=1, col=1); fig.update_xaxes(title_text="Variance", row=1, col=2)
fig.update_yaxes(showticklabels=False, row=1, col=1); fig.update_yaxes(showticklabels=False, row=1, col=2)

fig.show()
```

```
The fraction of CIs containing the true mean: 0.94
The fraction of CIs containing the true variance: 0.95
```

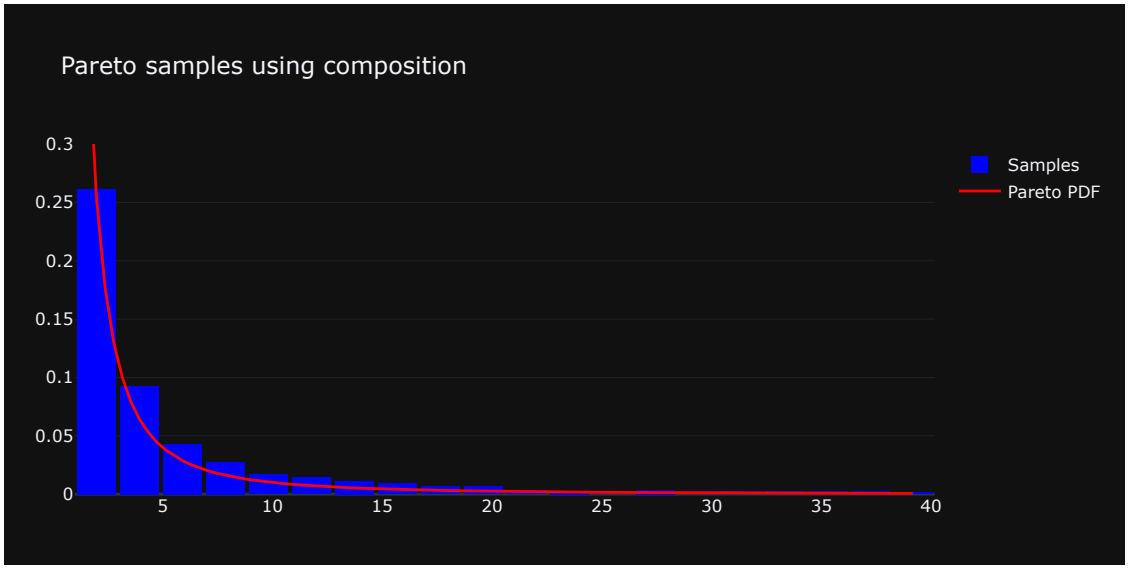


The confidence intervals for the mean and variance of the samples are plotted above. The green lines represent confidence intervals that contains the true mean and variance, while the red lines represent the opposite. It is seen that the fraction of confidence intervals that contain the true mean and variance is close to 0.95, indicating that the confidence intervals are robust.

Part 4 - Pareto distribution using composition

Now we wish to use composition to generate samples from the Pareto distribution. We do this by first generating samples, λ, from an exponential distribution. We then use these samples as the λ coefficient in the exponential distribution to generate samples from the Pareto distribution.

In [ ]:
```
num_samples = 10000

# Generate samples from uniform distribution
U1 = np.random.rand(num_samples)
U2 = np.random.rand(num_samples)

# Transform to Pareto distribution using composition method
beta = 1
y_comp = uniform_2_exponential(U1, beta)
X_comp = uniform_2_exponential(U2, y_comp)

# Plot
X_comp_q975 = X_comp[X_comp < np.quantile(X_comp, 0.975)] # Exclude extreme values for plotting purposes
x_pareto = np.linspace(min(X_comp_q975), max(X_comp_q975), 100, endpoint=True) # PDF points
fig = go.Figure(go.Histogram(x=X_comp_q975, xbins=dict(start=1, size=(max(X_comp_q975)-min(X_comp_q975))/20), histnorm='probability
fig.add_trace(go.Scatter(x=x_pareto, y=pareto_pdf(x_pareto, 1, beta), mode='lines', name='Pareto PDF', line=dict(color='red')))
fig.update_layout(height=400, width=800, title_text="Pareto samples using composition", bargap=0.1, margin=dict(l=50, r=50, t=100,
fig.update_xaxes(range=[1, None]); fig.update_yaxes(range=[0, 0.3])
fig.show()
```



In the plot above we see the samples generated using composition compared with the pdf of the Pareto distribution. We see that the samples generated using composition are consistent with the theoretical pdf.

Note that the shown histogram is truncated at the 97.5 percentile of the Pareto distribution. This is because the Pareto distribution has a long tail, and the histogram would be difficult to interpret if we included the entire range of the distribution.