1 Appendix

1.1 Code for P3

```python
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt

# %% Declare functions

# Generate uniform distribution function
def gen_uni_dist(a, b):
    """
    Generates a callable uniform distribution for a continuous random
    variable. Pass the 'a' and 'b' values ('b' > 'a').
    """
    if (a > b):
        return gen_uni_dist(b, a)
    cv = (1/(b-a))
    ud_func = lambda x: cv if a <= x and x <= b else 0
    return ud_func

# Generate normal distribution function
def gen_norm_dist(mu, si2):
    """
    Generates a callable normal distribution for a continuous random
    variable. Pass the 'mu' (mean) and 'si2' (variance = square of the
    standard deviation)
    """
    nd_func = lambda x: 1/( np.sqrt(2*np.pi*si2) ) * \
        np.exp( -(x-mu)**2/(2*si2) )
    return nd_func

# %% Plot everything
if __name__ == "__main__":
    # Configurations
    a, b = 2, 4
    mu, si2 = (a+b)/2, ((b-a)**2)/12
    xlim = [0, 5]
    # Actual code to generate data to be plotted
    xvals = np.linspace(xlim[0], xlim[1], 100)
    uni_dist = gen_uni_dist(a, b)
    uni_vals = np.array([uni_dist(xv) for xv in xvals])
    norm_vals = gen_norm_dist(mu, si2)(xvals)
    # Plot everything
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.plot(xvals, uni_vals, 'b--', label='Uniform')
    ax.plot(xvals, norm_vals, 'r--', label='Normal')
    ax.legend()
    ax.set_title(fr"$\mu={mu:.2f}\,\,and\,\,\sigma^2={si2:.3f}$")
    fig.savefig("plot_p3.png", dpi=600)
    plt.show()

# %%
```

Listing 1: Code to generate Figure **??**

1.2 Evaluating integral of e^{-x^2} over $(-\infty, \infty)$

Here, we shall prove the result

$$\int_{-\infty}^{\infty} e^{-x^2} \mathrm{d}x = \sqrt{\pi} \tag{1}$$

To prove 1, consider the integral to be I, that is

$$I = \int_{-\infty}^{\infty} e^{-x^2} \mathrm{d}x \tag{2}$$

Consider evaluating the integral below (we'll introduce evaluation in multiple dimensions)

$$\begin{aligned} J &= \int_{y=-\infty}^{\infty} \int_{x=-\infty}^{\infty} e^{-x^2-y^2} \mathrm{d}x \, \mathrm{d}y = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-x^2} e^{-y^2} \mathrm{d}x \, \mathrm{d}y \\ &= \int_{-\infty}^{\infty} e^{-y^2} \left(\int_{-\infty}^{\infty} e^{-x^2} \mathrm{d}x \right) \mathrm{d}y = \left(\int_{-\infty}^{\infty} e^{-x^2} \mathrm{d}x \right) \left(\int_{-\infty}^{\infty} e^{-y^2} \mathrm{d}y \right) \\ &= I \cdot I = I^2 \end{aligned} \tag{3}$$

We will evaluate a value for J, then, using result of $J = I^2$ (from 3), we will solve for I. It is also useful to recall the conversion from cartesian system to polar coordinate system

$$x = r\cos(\theta);\ y = r\sin(\theta);\ \mathrm{d}x \, \mathrm{d}y = r \mathrm{d}\theta \mathrm{d}r;\ r \to (0, \infty);\ \theta \to (0, 2\pi) \tag{4}$$

Using this, we can compute J as follows

$$\begin{aligned} J &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-x^2} e^{-y^2} \mathrm{d}x \, \mathrm{d}y = \int_{0}^{\infty} \int_{0}^{2\pi} e^{-r^2} r \mathrm{d}\theta \mathrm{d}r \\ &= \int_{0}^{\infty} e^{-r^2} r \left(\int_{0}^{2\pi} \mathrm{d}\theta \right) dr = 2\pi \int_{0}^{\infty} e^{-r^2} r dr = 2\pi \frac{\left[-e^{-r^2} \right]_0^{\infty}}{2} \\ &= 2\pi \frac{[(-0) - (-1)]}{2} = \pi \end{aligned} \tag{5}$$

Therefore, from 5, we have $J = \pi$. Since $J = I^2$ (from 3), we have $I = \sqrt{\pi}$, that is

$$I = \int_{-\infty}^{\infty} e^{-x^2} \mathrm{d}x = \sqrt{\pi} \tag{6}$$

1.3 Evaluating integral of $x^2e^{-x^2}$ over $(-\infty, \infty)$

Here, we shall prove the result

$$\int_{-\infty}^{\infty} x^2 e^{-x^2}\,\mathrm{d}x = \frac{\sqrt{\pi}}{2} \tag{7}$$

To prove this, let

$$I = \int_{-\infty}^{\infty} x^2 e^{-x^2}\,\mathrm{d}x \tag{8}$$

To evaluate the integral, we use the identity

$$\int u\,\mathrm{d}v = uv - \int v\,\mathrm{d}u \tag{9}$$

Let us use the following substitution

$$u = x;\ \mathrm{d}v = xe^{-x^2}\mathrm{d}x \Rightarrow \mathrm{d}u = \mathrm{d}x;\ v = \frac{-e^{-x^2}}{2} \tag{10}$$

Using 9 to solve for I, we can get

$$I = \int_{-\infty}^{\infty} (x)\left(xe^{-x^2}\right)\,\mathrm{d}x = \left[\frac{-xe^{-x^2}}{2}\right]_{-\infty}^{\infty} + \frac{1}{2}\int_{-\infty}^{\infty} e^{-x^2}\mathrm{d}x = \frac{\sqrt{\pi}}{2} \tag{11}$$

Note that $-xe^{-x^2} = 0$ for $x = \pm\infty$. This proves the result

$$\int_{-\infty}^{\infty} x^2 e^{-x^2}\,\mathrm{d}x = \frac{\sqrt{\pi}}{2} \tag{12}$$

1.4 Code for P6

1.4.1 Normal Distribution with $\mu = 0$ and $\sigma = 3.0$

```
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt
from scipy import special as sps

# %% Define functions

# CDF Inverse of Normal Distribution
def normal_cdfinv(yvals, mu, sig_sq):
    """
    Returns the point-wise inverse CDF for 'yvals', all values in the
    range (0, 1). The mean is 'mu' and variance is 'sig_sq'
    """
    x = mu + np.sqrt(2 * sig_sq) * sps.erfinv(2 * yvals - 1)
    return x

# Normal distribution function
def pdf_normal(xvals, mu, sig_sq):
    """
    Evaluates the value of the Normal Probability Density Function at
    the given 'xvals'. The value 'mu' is mean and variance is 'sig_sq'
    """
    cv = 1/np.sqrt(2*np.pi*sig_sq)
    return cv * np.exp(-(xvals-mu)**2 / (2*sig_sq))

# %% Main code
if __name__ == "__main__":
    # Random number generator
    rng = np.random.default_rng(10)
    mu, sigma = 0, 3    # Mean and standard deviation
    N = 10000
    # Generate 10000 samples in U[0, 1]
    yvals = rng.uniform(0, 1, N)
    xvals = normal_cdfinv(yvals, mu, sigma**2)
    # Histogram
    hvals, hedges = np.histogram(xvals, 50, density=True)
    lhedges = hedges[:-1]   # Left edges
    bar_width = 0.85*(lhedges[1] - lhedges[0])
    # Normal distribution (to cross-verify)
    x_n = np.linspace(hedges[0], hedges[-1], 100)
    norm_vals = pdf_normal(x_n, mu, sigma**2)
    # Plot everything
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.bar(lhedges, hvals, align='edge', width=bar_width, \
        label="inv CDF")
    ax.plot(x_n, norm_vals, 'r--', label="Normal Dist")
    ax.set_title("Normal Distribution using inverse CDF")
    ax.legend()
    ax.set_xlabel("x")
    ax.set_ylabel("p(x)")
    fig.savefig("plot_p6_normal.png", dpi=600)
    plt.show()

```

```
# %%
```

Listing 2: Code to generate Figure **??**

1.4.2 Rayleigh Distribution with $\sigma = 1.0$

```
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt

# %% Define functions

# CDF Inverse of Rayleigh Distribution
def rayleigh_cdfinv(yvals, sig_sq):
    """
    Returns the point-wise inverse CDF for 'yvals' (whose all values
    must be in the range (0, 1)). CDF is for Rayleigh distribution.
    The 'sig_sq' is squared of sigma (a parameter for the
    distribution).
    """
    return np.sqrt( 2*sig_sq*np.log(1/(1-yvals)) )

# Rayleigh distribution function
def pdf_rayleigh(xvals, sig_sq):
    """
    Evaluates the value of Rayleigh Probability Density Function at
    given 'xvals'. The value for 'sig_sq' is the squared of sigma (a
    parameter for the distribution).
    """
    return (xvals/sig_sq) * np.exp( -(xvals**2)/(2*sig_sq) )

# %% Main code
if __name__ == "__main__":
    # Random number generator
    rng = np.random.default_rng(10)
    sigma = 1.0
    N = 10000
    # Generate 10,000 samples in U[0, 1]
    yvals = rng.uniform(0, 1, N)
    xvals = rayleigh_cdfinv(yvals, sigma**2)
    # Histogram
    hvals, hedges = np.histogram(xvals, 50, density=True)
    lhedges = hedges[:-1]   # Left edges
    bar_width = 0.85*(lhedges[1] - lhedges[0])
    # Rayleigh distribution (to cross-verify)
    x_n = np.linspace(hedges[0], hedges[-1], 100)
    rayleigh_vals = pdf_rayleigh(x_n, sigma**2)
    # Plot everything
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.bar(lhedges, hvals, align='edge', width=bar_width, \
        label="inv CDF")
    ax.plot(x_n, rayleigh_vals, 'r--', label="Rayleigh Dist")
    ax.set_title("Rayleigh Distribution using inverse CDF")
    ax.legend()
    ax.set_xlabel("x")
    ax.set_ylabel("p(x)")
    fig.savefig("plot_p6_rayleigh.png", dpi=600)
```

```
    plt.show()

# %%
```

Listing 3: Code to generate Figure **??**

1.4.3 Exponential Distribution with $\lambda = 1.5$

```
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt

# %% Define functions

# CDF Inverse of Exponential Distribution
def exp_cdfinv(yvals, ld):
    """
    Returns the point-wise inverse CDF for 'yvals' (whose all values
    must be in the range (0, 1)). CDF is for Exponential distribution.
    The 'ld' is the lambda value for the exponential distribution.
    """
    return (1/ld)*np.log( 1/(1-yvals) )

# Exponential distribution function
def pdf_exp(xvals, ld):
    """
    Evaluates the value of Exponential Probability Density Function at
    given 'xvals'. The 'ld' is the lambda value for the exponential
    distribution.
    """
    return ld * np.exp(-ld*xvals)

# %% Main code
if __name__ == "__main__":
    # Random number generator
    rng = np.random.default_rng(10)
    lbda = 1.5  # Lambda value for the distribution
    N = 10000
    # Generate 10,000 samples in U[0, 1]
    yvals = rng.uniform(0, 1, N)
    xvals = exp_cdfinv(yvals, lbda)
    # Histogram
    hvals, hedges = np.histogram(xvals, 50, density=True)
    lhedges = hedges[:-1]   # Left edges
    bar_width = 0.85*(lhedges[1] - lhedges[0])
    # Exponential distribution (to cross-verify)
    x_n = np.linspace(hedges[0], hedges[-1], 100)
    exponential_vals = pdf_exp(x_n, lbda)
    # Plot everything
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.bar(lhedges, hvals, align='edge', width=bar_width, \
        label="inv CDF")
    ax.plot(x_n, exponential_vals, 'r--', label="Exp Dist")
    ax.set_title("Exponential Distribution using inverse CDF")
    ax.legend()
    ax.set_xlabel("x")
    ax.set_ylabel("p(x)")
```

```
    fig.savefig("plot_p6_exp.png", dpi=600)
    plt.show()

# %%
```

Listing 4: Code to generate Figure **??**

1.4.4 Plotting Normal, Rayleigh and Exponential PDFs

```
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt
# PDFs in earlier codes
from p6_normal import pdf_normal
from p6_rayleigh import pdf_rayleigh
from p6_exp import pdf_exp

# %% Main code
if __name__ == "__main__":
    xlim = [-5, 6]
    n_m, n_sigsq = 0, 1.25 # Mean and Variance of Normal Distribution
    r_sigsq = 1.25 # Sigma squared of Rayleigh Distribution
    e_lambda = 1.5  # Lambda value of Exponential Distribution
    # X values
    xvals = np.linspace(xlim[0], xlim[1], 100)
    # All distribution values
    normal_vals = pdf_normal(xvals, n_m, n_sigsq)
    xvpos = xvals[xvals >= 0]   # Only positive side of X
    rayleigh_vals = pdf_rayleigh(xvpos, r_sigsq)
    exp_vals = pdf_exp(xvpos, e_lambda)
    # Plot names
    normal_name = fr"$N(x;\mu={n_m}, \sigma^2={n_sigsq})$"
    rayleigh_name = fr"$R(x; \sigma^2={r_sigsq})$"
    exponential_name = fr"$E(x; \lambda = {e_lambda})$"
    # Plot everything
    fig = plt.figure()
    ax = fig.add_subplot()
    ax.axvline(0, c='k', ls='-.')
    ax.plot(xvals, normal_vals, '--', label=normal_name)
    ax.plot(xvpos, rayleigh_vals, '--', label=rayleigh_name)
    ax.plot(xvpos, exp_vals, '--', label=exponential_name)
    ax.set_xlim(xlim)
    ax.set_ylim([0, 0.8])
    ax.set_title("Different Probability Density Functions")
    ax.legend()
    ax.grid()
    fig.savefig("plot_p6_all.png", dpi=600)
    plt.show()

# %%
```

Listing 5: Code to generate Figure **??**

1.5 Code for P7

```python
# %% Import everything
import numpy as np
from matplotlib import pyplot as plt

# %% Define functions

# Run the experiment once
def experiment_trial(rng: np.random.Generator, N = 500):
    """
    Runs the experiment: Generates 'N' random numbers from Uniform
    Probability Density Function (low = 0, high = 1), adds them all up
    and returns the resultant sum. The function requires the 'rng'
    which is the random number generator object (numpy.random)
    """
    # Generate N samples
    samples = rng.uniform(0, 1, N)
    return np.sum(samples)  # Return their sum

# Normal distribution function
def pdf_normal(xvals, mu, sig_sq):
    """
    Evaluates the value of the Normal Probability Density Function at
    the given 'xvals'. The value 'mu' is mean and variance is 'sig_sq'
    """
    cv = 1/np.sqrt(2*np.pi*sig_sq)
    return cv * np.exp(-(xvals-mu)**2 / (2*sig_sq))

# %% Main code
if __name__ == "__main__":
    N = 500 # n value for each experiment (number of samples of U)
    N_iter = 50000   # Number of times the experiment must be run
    # Rnadom Number Generator
    rng = np.random.default_rng(10)
    exp_results = []
    for _ in range(N_iter):
        exp_results.append(experiment_trial(rng, N))
    # Convert results to numpy array
    exp_results = np.array(exp_results, dtype=float)
    # Generate normalized histogram from experiment results
    hvals, hedges = np.histogram(exp_results, 50)
    hvals_n = hvals/np.sum(np.diff(hedges)*hvals)   # Normalize hist.
    ledges = hedges[0:-1]
    # Approximating a normal distribution
    x_n = np.linspace(hedges[0], hedges[-1], 100)
    app_norm_vals = pdf_normal(x_n, N/2, N/12)
    # Plot the resultant histogram
    fig = plt.figure()
    ax = fig.add_subplot()
    bin_width = 0.85 * (ledges[1]-ledges[0])
    ax.bar(ledges, hvals_n, width=bin_width, align="edge", \
        label="Irwin-Hall")
    ax.plot(x_n, app_norm_vals, 'r--', label="Normal")
    ax.set_title("Irwin-Hall distribution " + \
        r"$X \rightarrow \sum^{" + str(N) + r"}_{k=1} U(0,1) $")
    ax.set_xlabel(r"$X$")
    ax.set_ylabel(r"Normalized Histogram of $X$ sampled " \
```

```
        + str(N_iter) + r" times")
    ax.grid()
    ax.legend()
    fig.savefig("plot_p7.png", dpi=600)
    plt.show()

# %%
```

Listing 6: Code to generate Figure **??**