Homework #2

Winter 2020, STATS 509
Dino Bektesevic

Problem 1

Find PMF and CDF for X_1 and X_2

a. in the case k=1 so the winnings can only be 0 or 2/3, i.e. $X_1 = \{0, 2/3\}$. The probability of each coin toss is 1/2. We can write the pmf function as:

$$f(x) = \begin{cases} 1/2 & \text{if } x \in X_1 \\ 0 & \text{if } x \notin X_1 \end{cases}$$

essentially saying that the probability of seeing random variable x be either win or lose is 50 percent. the cumulative distribution function, i.e. cdf, is given by $F_X(x) = P(X \leq x) = \sum_i p(x_i)$. In our case this boils down to three cases, corresponding to the two mass points - a win or no win:

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1/2 & \text{if } 0 \leq x \leq 2/3 \\ 1 & \text{if } x > 2/3 \end{cases}$$

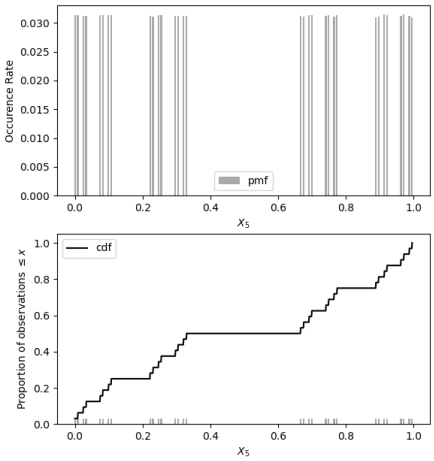
b. in the of k=2 there are 4 different winnings $X_2 = \{0, 2/3, 2/9, 8/9\}$ corresponding to four outcome cases (where ordering matters) $x = \{tt, ht, th, hh\} = \{0{+}0, 2/3{+}0, 0{+}2/9, 2/3{+}2/9\}$. The probability of each outcome is 1/4 because these are two independent events each with probability of 1/2. Written down:

$$f(x) = \begin{cases} 1/4 & \text{if } x \in X_2 \\ 0 & \text{if } x \notin X_2 \end{cases}$$

Similarly there are now 5 cases corresponding to the 4 mass points in the cdf:

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 0.25 & \text{if } 0 \leq x \leq 2/9 \\ 0.5 & \text{if } 2/9 < x \leq 2/3 \\ 0.75 & \text{if } 2/3 < x \leq 8/9 \\ 1 & \text{if } x > 8/9 \end{cases}$$

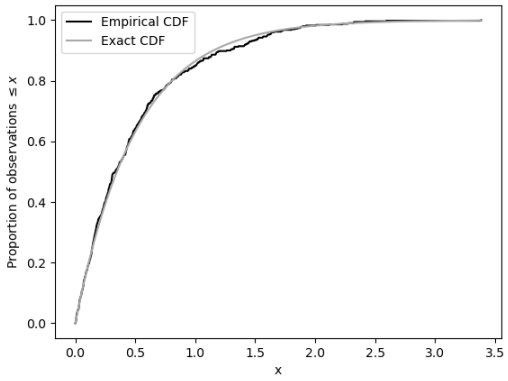
c. by simulating winnings. I run a million simulations counting up all the winnings from each toss in each simulation in Python.




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)


def problem1c(n=1000000):
    """Performs n simulations of 5 coin tosses and returns the total winnings
    earned in each simulation. Win earnings scale as  2/3**k where k is the
    toss count, i.e. integer from 1 to 5, for each simulation.

    Parameters
    ----------
    n : `int`
        Number of simulations to run, a simulation being 5 random draws from
        the set [0,1].

    Returns
    -------
    outcome : `np.array`
        a list of n pmf values corresponding to each simulation.
    """
    # a matrix in which rows are simulations and columns are tosses
    sims = np.random.randint(2, size=(n, 5))

    # covnert tosses of each simulation into individual earnings by multiplying
    # column-wise with the earnings for that toss, i.e. 2/3**k
    tossEarnings = []
    for k in range(1, 6):
        tossEarnings.append(sims[:,k-1].astype(float) * 2/3**k)

    # add all individual toss earnings into a total earning per simulation.
    simEarnings = np.column_stack(tossEarnings)
    return simEarnings.sum(axis=1)


def plot1c():
    """Displays plots for problem 1c."""
    n = 1000000
    simEarnings = problem1c(n)
    possibleEarnings = set(simEarnings)

    fig, axes = plt.subplots(2,1)

    # PMF
    for earning in possibleEarnings:
        occurenceRate = np.sum(simEarnings == earning) / n
        axes[0].bar(earning, occurenceRate, width=0.005, color='darkgray')
        axes[1].bar(earning, occurenceRate, width=0.005, color='darkgray')
    # replot the last one again to get a legend entry correct
    axes[0].bar(earning, occurenceRate, width=0.005, color='darkgray', label="pmf")

    # CDF evaluated at equaly space points within the domain
    xs = np.linspace(0, max(simEarnings), 1000)
    cumsum = []
    for x in xs:
        cumsum.append(np.sum(simEarnings <= x))
    cumsum = np.array(cumsum)/n
    axes[1].plot(xs, cumsum, 'black', label="cdf")

    for ax in axes:
        ax.set_xlabel("$X_5$")
        ax.legend()
    axes[0].set_ylabel("Occurence Rate")
    plt.ylabel("Proportion of observations $\leq x$")
    plt.show()


if __name__ == "__main__":
    plot1c()
```

Problem 2

Simulate and plot empirical CDF of exponential random variable.




```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
np.random.seed(0)


def problem2(n=500, lambd=2):
    """Instantiates an scaled exponential random variable, draws n samples from
    it and calculates an empirical CDF at a 1000 evenly spaced points in the
    range of the drawn samples.

    Parameters
    ----------
    n : `int`
        number of samples drawn from the distribution
    lambd : `float`
        The rate parameter ``lambda*exp(-lambda*x)`` from which scale will be
        calculated from as ``1/lambda``

    Returns
    -------
    x : `np.array`
        Points at which empirical CDF was evaluated at
    empiricalCDF : `np.array`
        Empirical CDF evaluated at points in ``x``
    exactCDF : `np.array`
        Exact CDF evaluated at points in ``x``
    """
    scale = 1/lambd
    rv = stats.expon(scale=scale)
    expSamples = rv.rvs(size=n)

    x = np.linspace(0, max(expSamples), 1000)
    empiricalCDF = []
    for i in x:
        empiricalCDF.append(np.sum(expSamples <= i))

    empiricalCDF = np.array(empiricalCDF)/n
    exactCDF = rv.cdf(x)

    return x, empiricalCDF, exactCDF


def plot2():
    """Plots problem 2."""
    fig, ax = plt.subplots()

    x, empiricalCDF, exactCDF = problem2()

    ax.plot(x, empiricalCDF, 'black', label="Empirical CDF")
    ax.plot(x, exactCDF, 'darkgray',  label="Exact CDF")

    ax.set_xlabel("x")
    ax.set_ylabel("Proportion of observations $\leq x$")
    ax.legend()
    plt.show()


if __name__ == "__main__":
    plot2()
```

Problem 3

a. As in all the problems above the empirical CDF is just a counting function:

$$\mathbb{F}(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{x_i \leq t\}$$

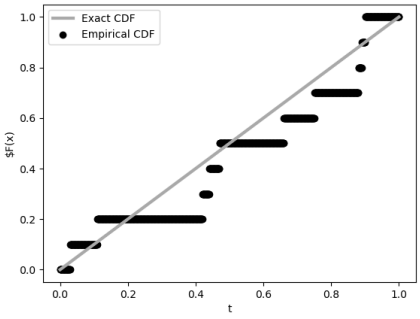
b. I assume Rectangular distribution is the Uniform distribution given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

In this problem we are given that $b = 1$ and $a = 0$ so that $f(t) = 1$ for $0 \leq t \leq 1$. CDF, given by the definition $\mathbb{F}(x) = \int f(x)dx$, in this problem is then:

$$\mathbb{F}(t) = \int_{-\infty}^{\infty} f(t)dt = \begin{cases} 0 & \text{for } t < 0 \\ \int_0^t dt = t & \text{for } 0 \leq t \leq 1 \\ 1 & \text{for } t > 1 \end{cases}$$

c. Code producing this plot can be found at the end of this problem.



d. Code producing this output can be found at the end of this problem.

```
KS statistic K=0.2194194194194194 at point t=0.4194194194194194
```

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
np.random.seed(0)


def problem2(n=500, lambd=2):
    """Instantiates an scaled exponential random variable, draws n samples from
    it and calculates an empirical CDF at a 1000 evenly spaced points in the
    range of the drawn samples.

    Parameters
    ----------
    n : `int`
        number of samples drawn from the distribution
    lambd : `float`
        The rate parameter ``lambda*exp(-lambda*x)`` from which scale will be
        calculated from as ``1/lambda``

    Returns
    -------
    x : `np.array`
        Points at which empirical CDF was evaluated at
    empiricalCDF : `np.array`
        Empirical CDF evaluated at points in ``x``
    exactCDF : `np.array`
        Exact CDF evaluated at points in ``x``
    """
    scale = 1/lambd
    rv = stats.expon(scale=scale)
    expSamples = rv.rvs(size=n)

    x = np.linspace(0, max(expSamples), 1000)
    empiricalCDF = []
    for i in x:
        empiricalCDF.append(np.sum(expSamples <= i))

    empiricalCDF = np.array(empiricalCDF)/n
    exactCDF = rv.cdf(x)

    return x, empiricalCDF, exactCDF


def plot2():
    """Plots problem 2."""
    fig, ax = plt.subplots()

    x, empiricalCDF, exactCDF = problem2()

    ax.plot(x, empiricalCDF, 'black', label="Empirical CDF")
    ax.plot(x, exactCDF, 'darkgray',  label="Exact CDF")

    ax.set_xlabel("x")
    ax.set_ylabel("Proportion of observations $\leq x$")
    ax.legend()
    plt.show()


if __name__ == "__main__":
    plot2()
```

Problem 4

For each of the following, use the CDF approach to obtain the PDF of Y:

a. $Y = 2X$ if X is a distributed exponential distribution. The exponential distribution is defined as:

$$F(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

In a similar manner we can define $G(y) = P(Y \leq y)$ as the CDF of Y. Since CDF is just an integrated PDF we know we can recover the integrand, i.e. the PDF, by deriving G. First, since we are given the CDF of X, but not of Y we need to write out the explicit form of G:

$$\begin{aligned} G(y) = P(Y \leq y) = P(2X \leq y) = P(X \leq y/2) = F(y/2) \\ = \begin{cases} 1 - e^{-\lambda y/2} & y \geq 0, \\ 0 & y < 0. \end{cases} \end{aligned}$$

Deriving case $y \geq 0$ we get:

$$\begin{aligned} g(y) = \frac{\partial}{\partial y} G(y) = \frac{\partial}{\partial y}\left(1 - e^{-\frac{\lambda y}{2}}\right) \\ = -e^{-\frac{\lambda y}{2}} \frac{\partial}{\partial y}\left(-\lambda \frac{y}{2}\right) \\ = \frac{\lambda}{2} e^{-\frac{\lambda}{2} y} \end{aligned}$$

Finally we note that deriving the case $y, 0$ retrieves just 0.

b. $Y = -\ln X$ if X is a Rectangular on [0.1] I, again, assume Rectangular distribution is the Uniform distribution given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

whose CDF we determined in a previous problem as:

$$\mathbb{F}(t) = \int_{-\infty}^{\infty} f(t)dt = \begin{cases} 0 & \text{for } t < 0 \\ t & \text{for } 0 \leq t \leq 1 \\ 1 & \text{for } t > 1 \end{cases}$$

Again, write the CDF of Y in terms of F as:

$$G(y) = P(Y \leq y) = P(-\ln X \leq y) = P(\ln X \geq -y) = P(X \geq e^{-} y)$$

Here we have to notice the complementarity $P(X \geq a) + P(X < a) = 1$ from which it follows that $G(y) = 1 - P(X \leq e^{-y}) = 1 - F(e^{-y})$. Finally, we can write

$$G(y) = \begin{cases} 0 & \text{for } y \leq 0, \\ 1 - e^{-y} & \text{for } y > 0 \end{cases}$$

Noting that differentiating first case will just produce 0, we focus on the case $y > 0$:

$$\begin{aligned} g(y) = \frac{\partial}{\partial y} G(y) = \frac{\partial}{\partial y}\left(1 - e^{-y}\right) \\ = e^{-y} \end{aligned}$$

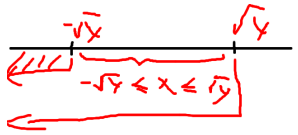
c. $Y = X^2$ if X follows normal distribution. Again the trick is to find a clever way to write the CDF $G(y)$. The CDF of normal distribution can not be expressed using elementary functions and is instead given as an integral:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$$

We can write G as:

$$\begin{aligned} G(y) &= P(Y \leq y) = P(X^2 \leq y) = \\ &= P(-\sqrt{y} \leq X \leq \sqrt{y}) \\ &= P(X \leq \sqrt{y}) - P(X \leq -\sqrt{y}) = F(\sqrt{y}) - F(-\sqrt{y}) \end{aligned}$$

Simply because it's just the way it is:



In addition to the ways things just are what they are it's also important to remember that the derivative of a CDF is just the PDF: $\partial_x F(x) = f(x)$, i.e. the normal distribution. Finally differentiating the CDF we have:

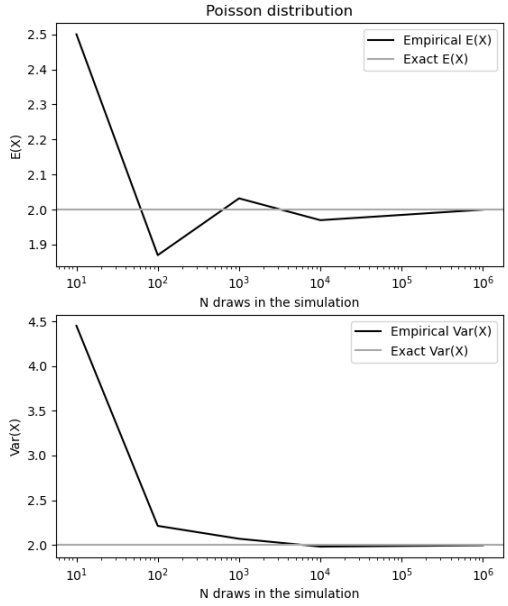
$$\begin{aligned} g(y) &= \frac{\partial}{\partial y} \left[F(\sqrt{y}) - F(-\sqrt{y})\right] \\ &= \frac{\partial \sqrt{y}}{\partial y} \frac{\partial F(\sqrt{y})}{\partial y} - \frac{\partial -\sqrt{y}}{\partial y} \frac{\partial F(-\sqrt{y})}{\partial y} \\ &= \frac{y^{-1/2}}{2} f(\sqrt{y}) - \frac{y^{-1/2}}{2} f(-\sqrt{y}) \end{aligned}$$

Where we just applied the chain rule $\partial_x f(g(x)) = g'(x)f'(g(x))$. Since the normal distribution is symmetric around zero we can say that $f(\sqrt{y}) = f(-\sqrt{y})$, pull the common factors out and simplify:

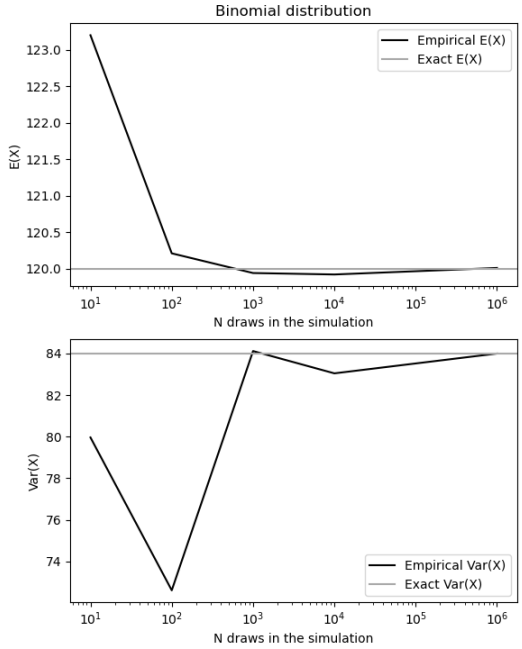
$$g(y) = y^{-\frac{1}{2}} f(\sqrt{y}) = \frac{1}{\sqrt{2\pi y}} e^{-\frac{y}{2}}$$

Problem 5

Simulate data from a given distribution and verify Goldberg table 3.1. The code that produced the following plots is located at the end of the problem.



a.



b.

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
np.random.seed(0)


def simulate(nSamples, randVar):
    """Given n number of samples m, performs n simulations drawing m samples
    each time calculating mean and the variance.

    Parameters
    ----------
    nSamples : `array`
        number of samples to draw for each simulation, where the number of sims
        is the length of the array.
    randVar : `scipy.stats.rv_frozen`
        Random variable from which draws will be made.

    Returns
    ------
    expectations : `array`
        calculated empirical expectation values for each simulation
    variances : `array`
        calculated empirical variance values for each simulation
    """
    expectation, variance = [], []

    for nSample in nSamples:
        draws = randVar.rvs(nSample)
        expectation.append(np.mean(draws))
        variance.append(np.var(draws))

    return expectation, variance


def plot(nSamples, expectations, variances, exactE, exactVar, title=""):
    """Plots the given empirical and exact expectation and variance values.

    Parameters
    ----------
    nSamples : `array`
        number of samples to draw each simulation.
    expectations : `array`
        calculated empirical expectation values
    variances : `array`
        calculated empirical variance values
    exactE : `float`
        exact expectation value
    exactVar : `float`
        exact variance value
    title : `str`, optional
        Title of the plot
    """
    fig, axes = plt.subplots(2, 1)

    axes[0].semilogx(nSamples, expectations, color="black", label="Empirical E(X)")
    axes[0].axhline(exactE, color="darkgray", label="Exact E(X)")

    axes[1].semilogx(nSamples, variances, color="black", label="Empirical Var(X)")
    axes[1].axhline(exactVar, color="darkgray", label="Exact Var(X)")

    for ax in axes:
        ax.set_xlabel("N draws in the simulation")
        ax.legend()
    axes[0].set_ylabel("E(X)")
    axes[0].set_title(title)
    axes[1].set_ylabel("Var(X)")
    plt.show()


def plot5a(nSamples):
    """Runs n simulations with m draws each and plots the calculated and exact
    values for Poisson distribution.

    Parameters
    ----------
    nSamples : `array`
        number of samples to draw each simulation.
    """
    rv = stats.poisson(2.0)
    expectations, variances = simulate(nSamples, rv)
    plot(nSamples, expectations, variances, 2.0, 2.0, "Poisson distribution")


def plot5b(nSamples):
    """Runs n simulations with m draws each and plots the calculated and exact
    values for Binomial distribution.

    Parameters
    ----------
    nSamples : `array`
        number of samples to draw each simulation.
    """
    rv = stats.binom(400, 0.3)
    expectations, variances = simulate(nSamples, rv)
    plot(nSamples, expectations, variances, 120, 84, "Binomial distribution")


if __name__ == "__main__":
    nSamples = [10, 100, 1000, 10000, 1000000]
    plot5a(nSamples)
    plot5b(nSamples)
```

Problem 6

a. From hint we know we have to focus on tossing the unfair coin twice. Then we have four possible outcomes $X = \{hh, ht, th, tt\}$. In this sample space the probability of events $ht = p(1-p) = (1-p)p = th$ could be different from 0.5, i.e. for $p = 0.2$ the probability of $ht = th = 0.2 \cdot 0.8 = 0.16 \neq 0.5$, but the probabilities of ht and th will be equal. However, while individual probabilities $P(ht)$ or $P(th)$ are not 0.5, the probability to get an event from the smaller sample space $P(x|x \in X \cap hh \cap tt) = P(X \in \{ht, th\})$ should be 0.5. I might have mangled the notation a bit, but the point is toss the coin until one gets ht or th outcome, ignoring others. The probability to either get the ht or th in that order are then 50%.

b. The probability that in a single toss we first toss a head and then a tail, or first a tail and then a head would be:

$$P(ht \cup th) = P(ht) + P(th) - P(ht \cap th) = p(1-p) + (1-p)p = 2p(1-p)$$

Geometric distribution lets us model the number of failures before the first success if all of the trials are mutually independent. The distribution is defined as: $P(Y = k) = p(1-p)^k$ where k is the number of failures before the first success, each with probability p. In this case, we do not consider each instance the coin is tossed, but each time we toss it twice. Then probability of each double-toss is given by expression above. So we carefully denote two different p's in these equations by saying $p' = p$ and rewriting: $P(Y = k) = p'(1-p')^k$ The expectation value of the geometric distribution is given as:

$$E = \frac{1}{p'}$$

so in our case we can expect to have to toss pairs of coins:

$$E = \frac{1}{2p(1-p)}$$

Since the question is potentially confusing, inquiring about the number of times requiring to toss the actual coin not pairs, we can double that expectation value - relying on the linearity of expectation value - to get the actual number of times we had to toss the actual coin and not pairs:

$$E = \frac{1}{p(1-p)}$$

c.
```python
import scipy.stats as stats
import numpy as np
np.random.seed(0)


def simulate(nSimulations, p):
    """Runs n simulations, where each simulation is a double coin toss, keeping
    track of how many attempts are requires untill the first case of HT or TH
    is observed.

    Parameters
    ----------
    nSimulations : `int`
        Number of simulations to run
    p : `float`
        Probability that a coin toss is either heads or tails.

    Returns
    -------
    nAttempts : `array`
        Number of coin tosses that were required until HT or TH were tossed.
    """
    nAttempts = []
    for sim in range(nSimulations):
        nTries = 0
        while True:
            flip1 = stats.binom.rvs(1, p)
            flip2 = stats.binom.rvs(1, p)
            nTries += 2
            if flip1 != flip2:
                break
        nAttempts.append(nTries)
    return nAttempts


if __name__ == "__main__":
    nSimulations = 1000
    p = 0.2
    nAttempts = simulate(nSimulations, p)
    print(f"Average number of coin tosses, across all simulations, until success was {np.mean(nAttempts)}")
```

d. Average number of coin tosses, across 1000 simulations, until success was 6.13. When we climb up to 10 000 simulations I get 6.21 and I get really close to theoretically predicted value of $\frac{1}{0.2 \cdot 0.8} = 6.25$ with the average value across a million simulations of 6.2501. Now this is the total number of count tosses total, the total number of the ht and th event tosses are then half of that, due to linearity of expectation values we can just move a factor in or out.

Problem 7

For each of the following distribution calculate expectation value and variance. Expressions taken from Goldberger table 3.1.

a. Discrete uniform with N=9

$$E(X) = \frac{N+1}{2} = 5$$
$$V(X) = \frac{N^2-1}{12} = 6.666...$$

b. Binomial with n=2 and p=0.4

$$E(X) = np = 0.8$$
$$V(X) = np(1-p) = 0.48$$

c. Binomial with n=4 and p=0.6

$$E(X) = np = 2.4$$
$$V(X) = np(1-p) = 0.96$$

d. Poisson with $\lambda = 2$

$$E(X) = \lambda = 1.5$$
$$V(X) = \lambda = 1.5$$

e. Rectangular on $[0, 2]$ (again, assuming uniform)

$$E(X) = \frac{a+b}{2} = 1$$
$$V(X) = \frac{(b-a)^2}{12} = 0.333$$

f. Exponential with $\lambda = 2$

$$E(X) = \frac{1}{\lambda} = 0.5$$
$$V(X) = \frac{1}{\lambda^2} = 0.25$$

g. Powerlaw with $\theta = 2$ on $[0, 1]$

$$E(X) = \theta/(1+\theta) = 0.75$$
$$V(X) = \theta\left[(1+\theta)^2(2+\theta)\right]^{-1} = 0.0555...$$