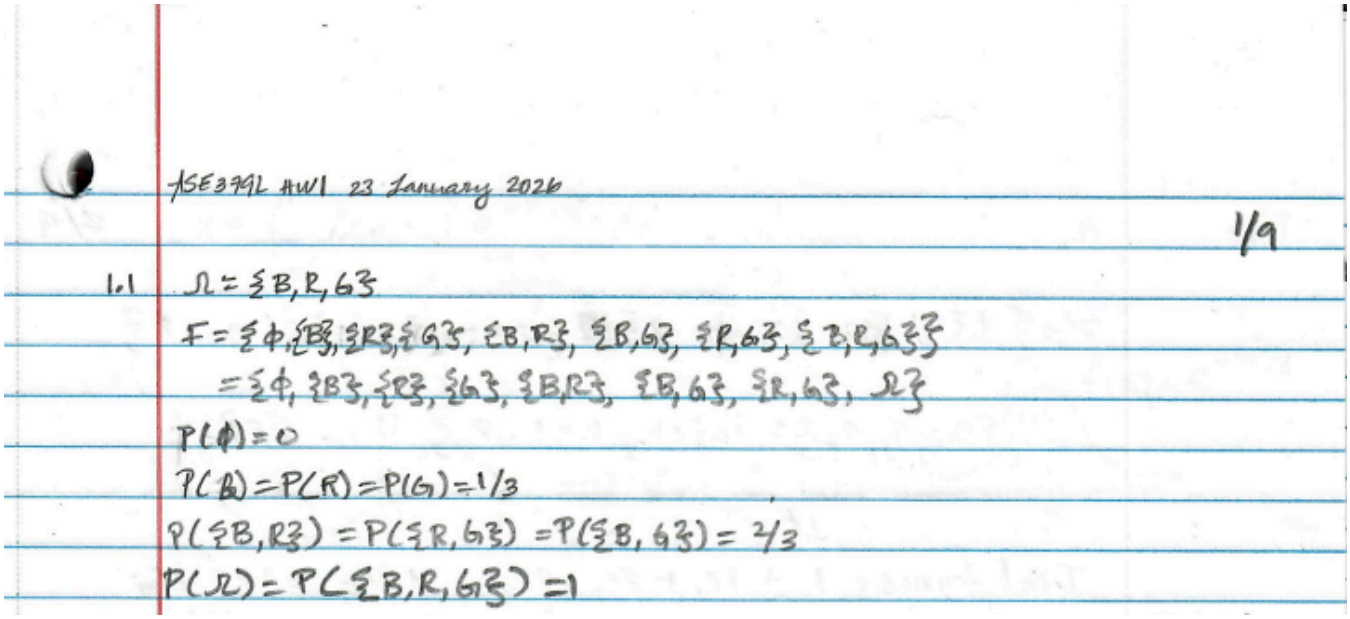
hw01

Exported 1/23/2026, 12:08:18 AM

1.1



ASE379L HW1 23 January 2026

1/9

1.1 $\Omega = \{B, R, G\}$

$F = \{\phi, \{B\}, \{R\}, \{G\}, \{B,R\}, \{B,G\}, \{R,G\}, \{B,R,G\}\}$

$= \{\phi, \{B\}, \{R\}, \{G\}, \{B,R\}, \{B,G\}, \{R,G\}, \Omega\}$

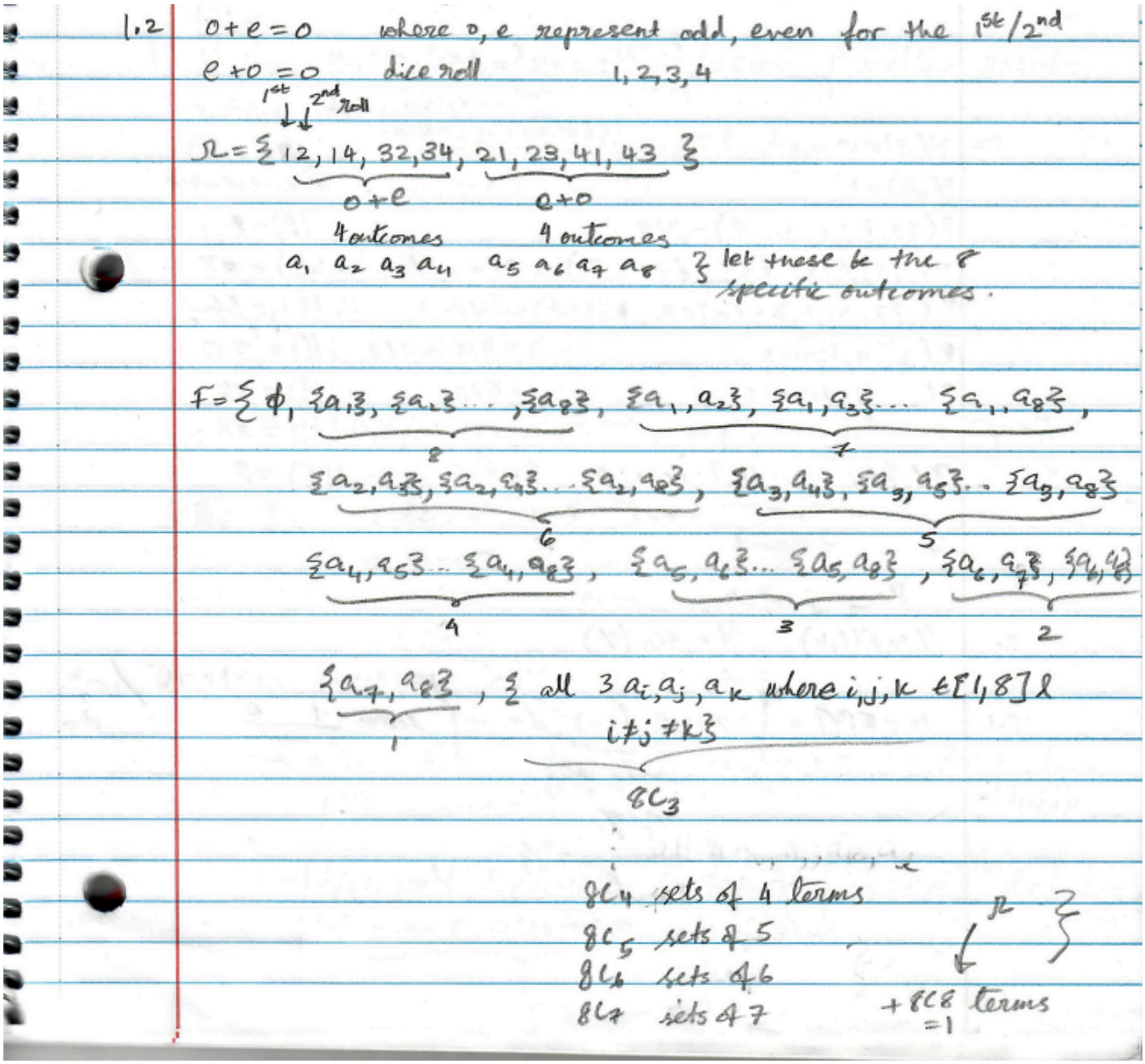
$P(\phi) = 0$

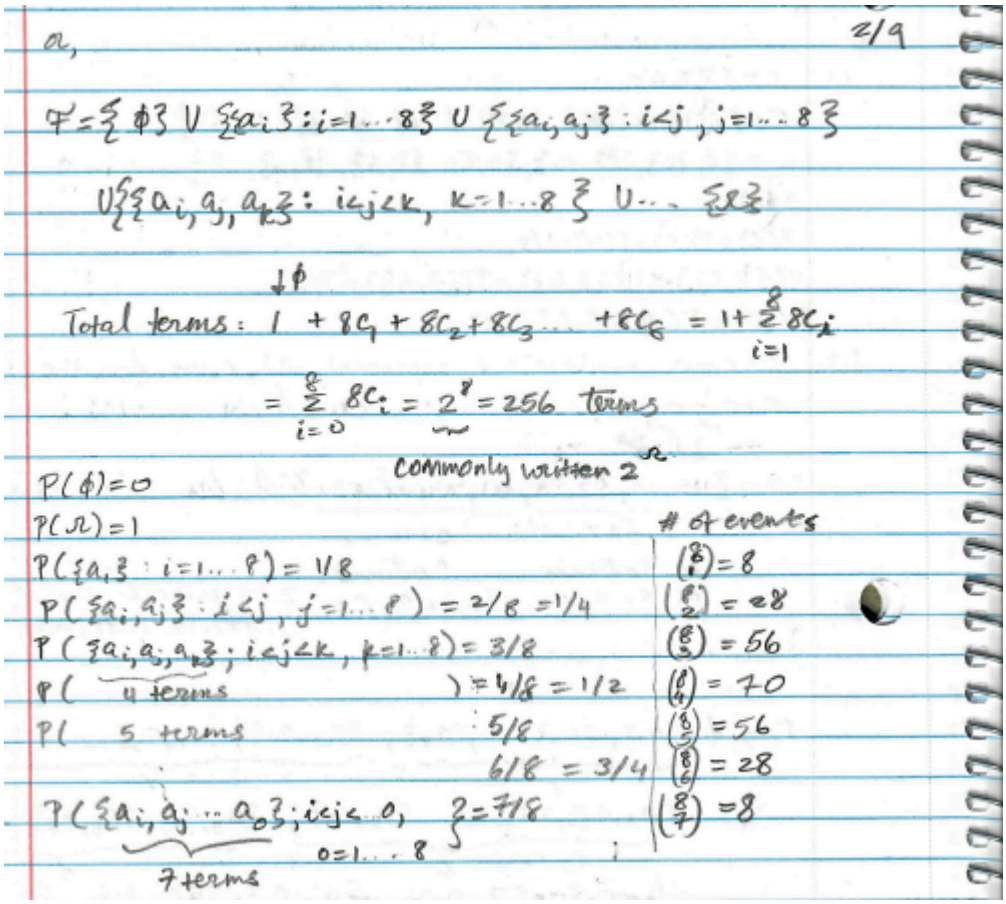
$P(B) = P(R) = P(G) = 1/3$

$P(\{B,R\}) = P(\{R,G\}) = P(\{B,G\}) = 2/3$

$P(\Omega) = P(\{B,R,G\}) = 1$

1.2





2.1

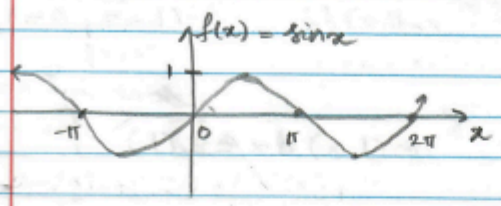
2. $X \sim N(0,4)$ $\quad Y = \sin(X)$ $\quad \mu_x = 0$, $\sigma_x^2 = 4$

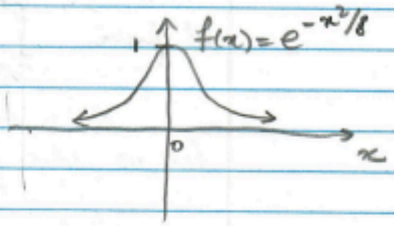
2.1 $\mu = E[Y] = \int_{-\infty}^{\infty} \sin(x) \underbrace{f(x)}_{\text{pdf of } X} dx = \int_{-\infty}^{\infty} \sin x \frac{1}{\sqrt{2\pi}\sigma_x^2} e^{-(x-\mu_x)^2/2\sigma_x^2} dx$

$= \int_{-\infty}^{\infty} (\sin x) \frac{e^{-x^2/8}}{2\sqrt{2\pi}} dx$

$\mu = \int_{-\infty}^{\infty} (\sin x) e^{-x^2/8} dx \cdot \frac{1}{2\sqrt{2\pi}}$ 3/9

odd function, even function





We know the product of an odd & even function results in an odd function.
Also, any odd function integrated from $-\infty$ to ∞ (symmetric) is 0.

So, $\mu = 0 \cdot \frac{1}{2\sqrt{2\pi}} = 0$.

$Var[Y] = E[Y^2] - E[Y]^2 = E[Y^2] - \mu^2 = E[Y^2]$

$E[Y^2] = \int_{-\infty}^{\infty} (\sin x)^2 e^{-x^2/8} dx \frac{1}{2\sqrt{2\pi}}$

even, even

$\sin^2 x = \frac{1-\cos 2x}{2}$ so,

$E[Y^2] = \left[\int_{-\infty}^{\infty} e^{-x^2/8} dx - \int_{-\infty}^{\infty} (\cos 2x) e^{-x^2/8} dx\right] \frac{1}{4\sqrt{2\pi}}$

$= \left[\left(\text{integral of gaussian of mean 0, variance 2} \cdot 2\sqrt{2\pi}\right) - \int_{-\infty}^{\infty} (\cos 2x) e^{-x^2/8} dx\right] \frac{1}{4\sqrt{2\pi}}$

1

$$E[Y^2] = \frac{1}{2} - \underbrace{\int_{-\infty}^{\infty} (\cos 2x)\, e^{-x^2/8}\, dx}_{\text{of form } \int_{-\infty}^{\infty} e^{-ax^2}\cos(bx)\,dx \quad (b=2,\ a=1/8)} \cdot \frac{1}{4\sqrt{2\pi}} \qquad 4/9$$

let us solve $\int_{-\infty}^{\infty} e^{-ax^2}\cos(bx)\,dx = I(b)$ ← some integral function

$$I(0) = \int_{-\infty}^{\infty} e^{-ax^2}\,dx = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-(x^2)/2\sigma^2}\,dx \cdot \sqrt{\frac{\pi}{a}}$$

where $\sigma = \frac{1}{\sqrt{2a}}$

$$= \sqrt{\pi/a}$$

$$I'(b) = \int_{-\infty}^{\infty} -x e^{-ax^2}\sin(bx)\,dx$$

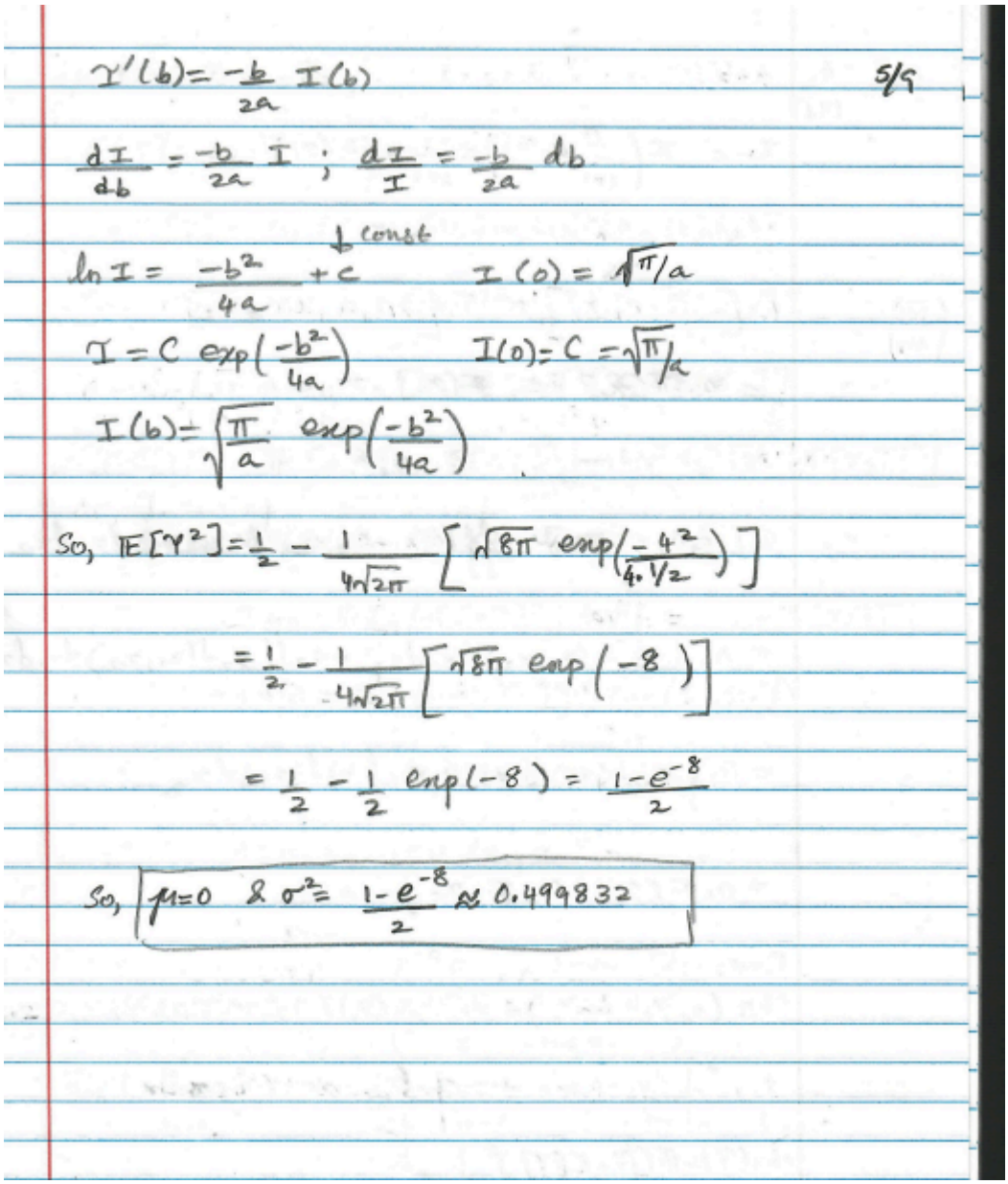
$$= \int_{-\infty}^{\infty} \frac{1}{2a}\frac{\partial}{\partial x}\left(e^{-ax^2}\right)\sin bx\,dx$$

let $u = \sin bx$, $du = b\cos bx\,dx$

$dv = \frac{1}{2a}\frac{\partial}{\partial x}\left(e^{-ax^2}\right)dx$, $v = \frac{1}{2a}e^{-ax^2}$

$$I'(b) = \underbrace{\frac{1}{2a}\sin bx\, e^{-ax^2}\Big|_{-\infty}^{\infty}}_{\text{odd}} - \int_{-\infty}^{\infty} \frac{1}{2a} e^{-ax^2} b\cos bx\,dx$$

$$= 0 - \frac{b}{2a} I(b)$$



$$\gamma'(b) = \frac{-b}{2a} I(b) \qquad 5/5$$

$$\frac{dI}{db} = \frac{-b}{2a} I \;;\; \frac{dI}{I} = \frac{-b}{2a} db$$

$$\ln I = \frac{-b^2}{4a} + c \qquad \text{(+ const)} \qquad I(0) = \sqrt{\pi/a}$$

$$I = C \exp\left(\frac{-b^2}{4a}\right) \qquad I(0) = C = \sqrt{\pi/a}$$

$$I(b) = \sqrt{\frac{\pi}{a}} \exp\left(\frac{-b^2}{4a}\right)$$

$$\text{So, } \mathbb{E}[\gamma^2] = \frac{1}{2} - \frac{1}{4\sqrt{2\pi}} \left[\sqrt{8\pi} \exp\left(\frac{-4^2}{4 \cdot 1/2}\right)\right]$$

$$= \frac{1}{2} - \frac{1}{4\sqrt{2\pi}} \left[\sqrt{8\pi} \exp(-8)\right]$$

$$= \frac{1}{2} - \frac{1}{2} \exp(-8) = \frac{1 - e^{-8}}{2}$$

$$\text{So, } \boxed{\mu = 0 \;\&\; \sigma^2 = \frac{1 - e^{-8}}{2} \approx 0.499832}$$

2.2

In [1]:

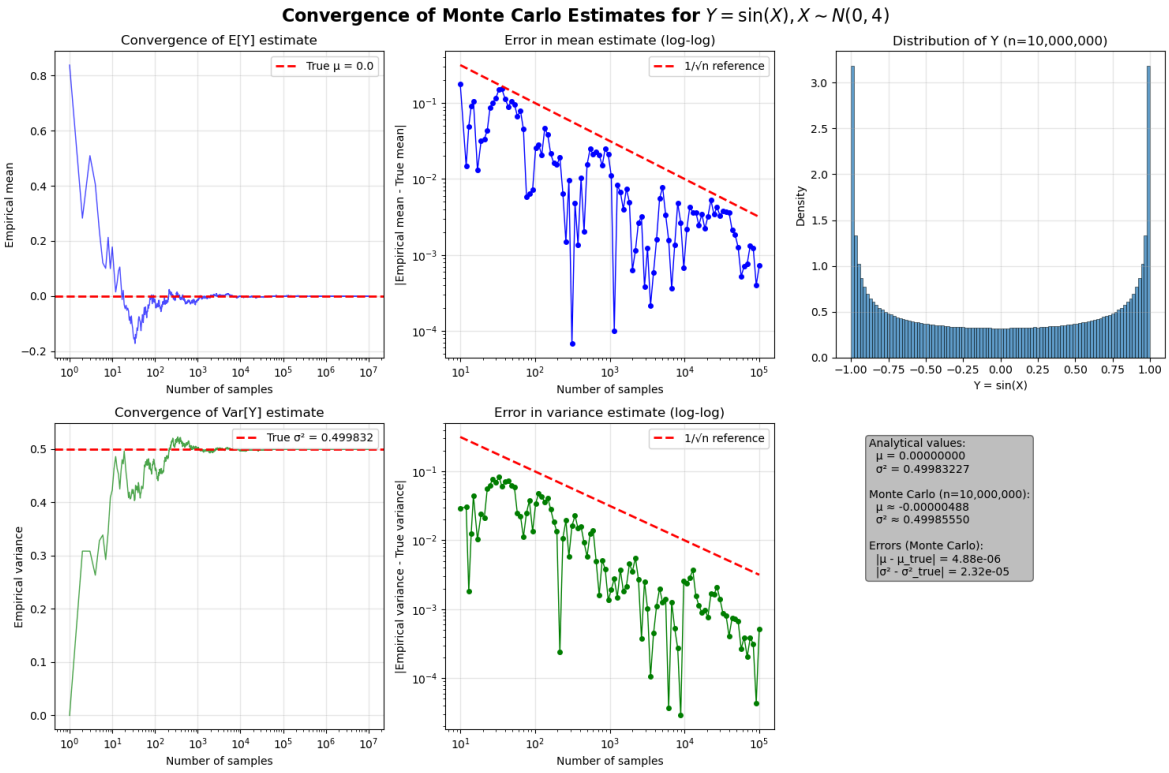
```
import numpy as np
import matplotlib.pyplot as plt

# ============================================
# 2.1: Solve using computer integration/simulation
# ============================================

# True analytical values (from theory)
mu_true = 0.0
sigma2_true = (1 - np.exp(-8)) / 2
print(f"Analytical values:")
print(f"  μ = E[Y] = {mu_true}")
print(f"  σ² = Var[Y] = {sigma2_true:.8f}")

# ============================================
# Method 1: Direct Monte Carlo simulation
# ============================================

np.random.seed(42)

# Use a large number of samples for accurate estimation
n_mc = int(1e7)  # 10 million samples
print(f"\nMonte Carlo simulation with n = {n_mc:,} samples...")

# Generate X ~ N(0, 4) and Y = sin(X)
X = np.random.normal(loc=0, scale=2, size=n_mc)  # scale=2 for
variance 4
Y = np.sin(X)

# Compute empirical mean and variance
mu_mc = np.mean(Y)
sigma2_mc = np.var(Y, ddof=1)

print(f"Monte Carlo results:")
print(f"  E[Y] ≈ {mu_mc:.8f} (error = {abs(mu_mc - mu_true):.8f})")
print(f"  Var[Y] ≈ {sigma2_mc:.8f} (error = {abs(sigma2_mc -
sigma2_true):.8f})")



# ============================================
print(f"\nVisualizing convergence of Monte Carlo estimates...")

# Compute cumulative means and variances
cumulative_means = np.cumsum(Y) / np.arange(1, n_mc + 1)
cumulative_sumsq = np.cumsum(Y**2)
cumulative_vars = (cumulative_sumsq / np.arange(1, n_mc + 1) -
cumulative_means**2)
```

```
# Create figure with convergence plots
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Convergence of Monte Carlo Estimates for $Y = \sin(X), X
\sim N(0,4)$',
             fontsize=16, fontweight='bold')

# Plot 1: Mean convergence (full range)
ax = axes[0, 0]
ax.plot(np.arange(1, n_mc + 1), cumulative_means, 'b-', alpha=0.7,
linewidth=1)
ax.axhline(y=mu_true, color='r', linestyle='--', linewidth=2,
label=f'True μ = {mu_true}')
ax.set_xscale('log')
ax.set_xlabel('Number of samples')
ax.set_ylabel('Empirical mean')
ax.set_title('Convergence of E[Y] estimate')
ax.legend()
ax.grid(True, alpha=0.3)

# Plot 2: Variance convergence (full range)
ax = axes[1, 0]
ax.plot(np.arange(1, n_mc+ 1), cumulative_vars, 'g-', alpha=0.7,
linewidth=1)
ax.axhline(y=sigma2_true, color='r', linestyle='--', linewidth=2,
           label=f'True σ² = {sigma2_true:.6f}')
ax.set_xscale('log')
ax.set_xlabel('Number of samples')
ax.set_ylabel('Empirical variance')
ax.set_title('Convergence of Var[Y] estimate')
ax.legend()
ax.grid(True, alpha=0.3)

# Plot 3: Error in mean (log-log)
ax = axes[0, 1]
sample_sizes = np.logspace(1, 5, 100).astype(int)
sample_sizes = np.unique(sample_sizes)  # Remove duplicates
errors_mean = []

for n in sample_sizes:
    sample = Y[:n]
    errors_mean.append(abs(np.mean(sample) - mu_true))

ax.loglog(sample_sizes, errors_mean, 'bo-', markersize=4, linewidth=1)
ax.loglog(sample_sizes, 1/np.sqrt(sample_sizes), 'r--', linewidth=2,
label='1/√n reference')
ax.set_xlabel('Number of samples')
ax.set_ylabel('|Empirical mean - True mean|')
```

```
ax.set_title('Error in mean estimate (log-log)')
ax.legend()
ax.grid(True, alpha=0.3)

# Plot 4: Error in variance (log-log)
ax = axes[1, 1]
errors_var = []

for n in sample_sizes:
    sample = Y[:n]
    errors_var.append(abs(np.var(sample, ddof=1) - sigma2_true))

ax.loglog(sample_sizes, errors_var, 'go-', markersize=4, linewidth=1)
ax.loglog(sample_sizes, 1/np.sqrt(sample_sizes), 'r--', linewidth=2,
label='1/√n reference')
ax.set_xlabel('Number of samples')
ax.set_ylabel('|Empirical variance - True variance|')
ax.set_title('Error in variance estimate (log-log)')
ax.legend()
ax.grid(True, alpha=0.3)

# Plot 5: Distribution of Y
ax = axes[0, 2]
ax.hist(Y, bins=100, density=True, alpha=0.7, edgecolor='black',
linewidth=0.5)
ax.set_xlabel('Y = sin(X)')
ax.set_ylabel('Density')
ax.set_title(f'Distribution of Y (n={n_mc:,})')
ax.grid(True, alpha=0.3)

# Plot 6: Summary statistics
ax = axes[1, 2]
ax.axis('off')
summary_text = (
    f'Analytical values:\n'
    f'  μ = {mu_true:.8f}\n'
    f'  σ² = {sigma2_true:.8f}\n\n'
    f'Monte Carlo (n={n_mc:,}):\n'
    f'  μ ≈ {mu_mc:.8f}\n'
    f'  σ² ≈ {sigma2_mc:.8f}\n\n'
    f'Errors (Monte Carlo):\n'
    f'  |μ - μ_true| = {abs(mu_mc - mu_true):.2e}\n'
    f'  |σ² - σ²_true| = {abs(sigma2_mc - sigma2_true):.2e}'
)
ax.text(0.1, 0.95, summary_text, transform=ax.transAxes, fontsize=10,
        verticalalignment='top', bbox=dict(boxstyle='round',
facecolor='gray', alpha=0.5))
```

```
plt.tight_layout()
plt.show()
```

```
Analytical values:
  μ = E[Y] = 0.0
  σ² = Var[Y] = 0.49983227

Monte Carlo simulation with n = 10,000,000 samples...
Monte Carlo results:
  E[Y] ≈ -0.00000488 (error = 0.00000488)
  Var[Y] ≈ 0.49985550 (error = 0.00002323)

Visualizing convergence of Monte Carlo estimates...
```



The absolute errors of both the sample mean and the sample variance converge at an $O(n^{-1/2})$ rate, as evidenced by the approximately $-\frac{1}{2}$ slope on the log--log plot. By the central limit theorem, the standard deviation of the average of n independent and identically distributed (IID) samples scales as $1/\sqrt{n}$. Consequently, for the sample mean $\bar{X}_n$, $|\bar{X}_n - \mu| = O(n^{-1/2})$.

A similar convergence rate holds for the sample variance s_n^2. Writing $s_n^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2$, we see that it is itself an average of transformed random variables. Although the terms $(X_i - \bar{X}_n)^2$ are not independent due to the shared dependence on $\bar{X}_n$, this dependence becomes negligible asymptotically. By applying the central limit theorem to the sequence $Z_i = (X_i - \mu)^2$, which has finite variance under standard assumptions, we obtain $s_n^2 - \sigma^2 = O(n^{-1/2})$.

Both the sample mean and the sample variance can therefore be viewed as sample averages of functions of the data: $\bar{X}_n$ averages X_i, while s_n^2 averages squared deviations. Any such sample mean of IID data with finite variance inherits the characteristic $1/\sqrt{n}$ scaling of its standard error. As a result, on a log--log plot, the absolute errors of both statistics display an approximate slope of $-\frac{1}{2}$, reflecting the fundamental $n^{-1/2}$ convergence rate of Monte Carlo estimators.

3: Heat Equation

In [ ]:

```
from heat_rod import SteadyStateRod
import numpy as np

# --- Parameters from Example 3.5 ---
a = 0.95  # cm
b = 0.95  # cm
h = 0.002  # W/cm^2·C
k = 2.37  # W/cm·C
Phi = -18  # W/cm^2 (use negative for heat flux)
Tamb = 21  # C
L = 70
# --- Create rod instance ---
rod = SteadyStateRod(a, b, h, k, Phi, Tamb, L)

# --- Measurement locations ---
x0 = 10
dx = 4
x_vals = x0 + np.arange(15)*dx  # 15 locations: 10, 14, 18, ..., 66

# --- Evaluate Ts and derivatives ---
Ts_vals = rod.Ts(x_vals)
dTs_dh_vals = rod.dTs_dh(x_vals)
dTs_dk_vals = rod.dTs_dk(x_vals)
dTs_dPhi_vals = rod.dTs_dPhi(x_vals)

# --- Print results like Table 3.2 ---
print(f"{'x (cm)':>6} | {'Ts (C)':>7} | {'dTs/dh (C^2 cm^2/W)':>10} |
{'dTs/dk (C^2 cm/W)':>10} | {'dTs/dPhi () b b':>10} ")
print("-"*55)
for i, x in enumerate(x_vals):
    print(f"{x:6.1f} | {Ts_vals[i]:7.4f} | {dTs_dh_vals[i]:10.4f} |
{dTs_dk_vals[i]:10.4f} | {dTs_dPhi_vals[i]:10.4f}")
```

```
x (cm) |  Ts (C) |     dTs/dh |     dTs/dk |   dTs/dPhi
-----------------------------------------------------
  10.0 | 91.2698 | -28169.0259 |    -5.8784 |    -3.9039
  14.0 | 76.3885 | -25543.2402 |    -1.8152 |    -3.0771
  18.0 | 64.6710 | -22787.0997 |     0.8030 |    -2.4262
  22.0 | 55.4481 | -20079.4108 |     2.4096 |    -1.9138
  26.0 | 48.1929 | -17530.1660 |     3.3196 |    -1.5107
  30.0 | 42.4909 | -15201.9433 |     3.7607 |    -1.1939
  34.0 | 38.0165 | -13125.3738 |     3.8963 |    -0.9454
  38.0 | 34.5142 | -11310.2370 |     3.8423 |    -0.7508
  42.0 | 31.7837 | -9753.3536 |     3.6806 |    -0.5991
  46.0 | 29.6693 | -8444.1446 |     3.4679 |    -0.4816
  50.0 | 28.0500 | -7368.5026 |     3.2435 |    -0.3917
  54.0 | 26.8334 | -6511.4519 |     3.0335 |    -0.3241
  58.0 | 25.9501 | -5858.9472 |     2.8556 |    -0.2750
  62.0 | 25.3495 | -5399.0651 |     2.7210 |    -0.2416
  66.0 | 24.9973 | -5122.7715 |     2.6364 |    -0.2221
```

3.1 & 3.2

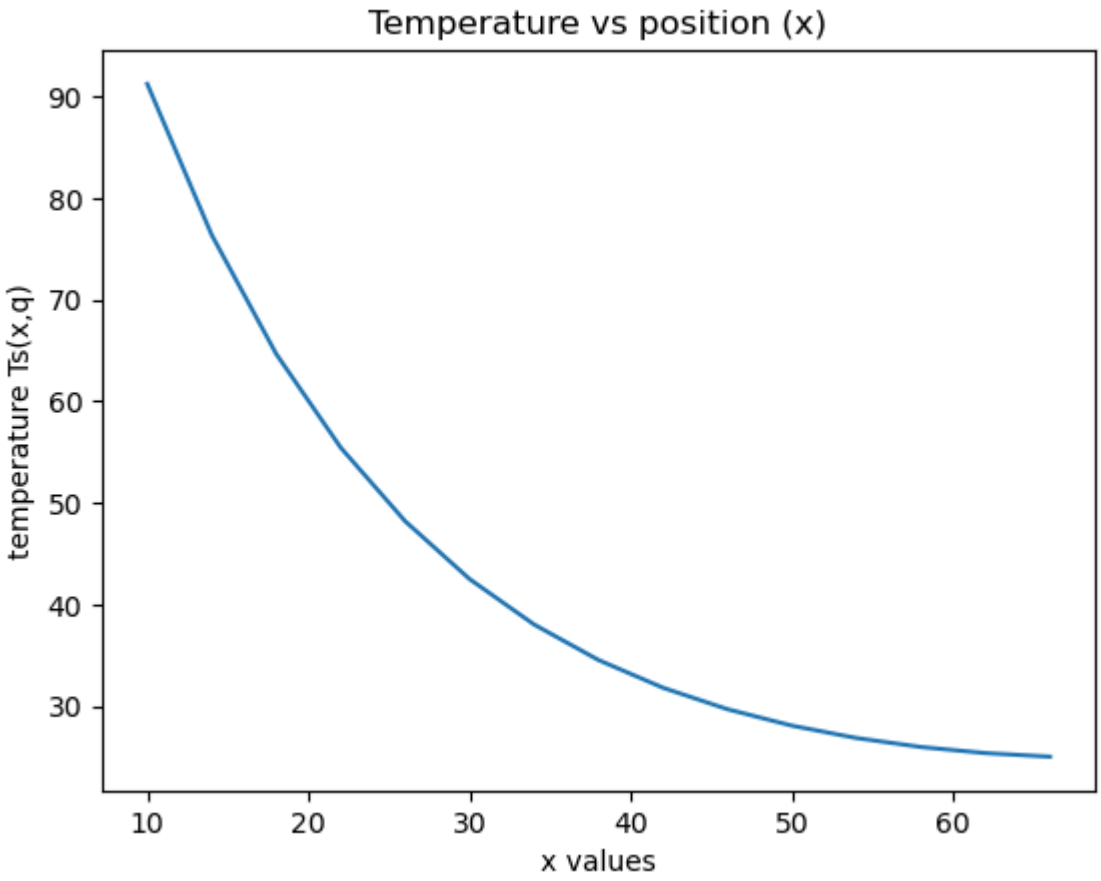
The values of $T_s(x, q)$ at x_j and the respective partials at the same x_j are presented in the below table.

x (cm)	Ts (C)	dTs/dh	dTs/dk	dTs/dPhi
10.0	91.2698	-28169.0259	-5.8784	-3.9039
14.0	76.3885	-25543.2402	-1.8152	-3.0771
18.0	64.6710	-22787.0997	0.8030	-2.4262
22.0	55.4481	-20079.4108	2.4096	-1.9138
26.0	48.1929	-17530.1660	3.3196	-1.5107
30.0	42.4909	-15201.9433	3.7607	-1.1939
34.0	38.0165	-13125.3738	3.8963	-0.9454
38.0	34.5142	-11310.2370	3.8423	-0.7508
42.0	31.7837	-9753.3536	3.6806	-0.5991
46.0	29.6693	-8444.1446	3.4679	-0.4816
50.0	28.0500	-7368.5026	3.2435	-0.3917
54.0	26.8334	-6511.4519	3.0335	-0.3241
58.0	25.9501	-5858.9472	2.8556	-0.2750
62.0	25.3495	-5399.0651	2.7210	-0.2416
66.0	24.9973	-5122.7715	2.6364	-0.2221

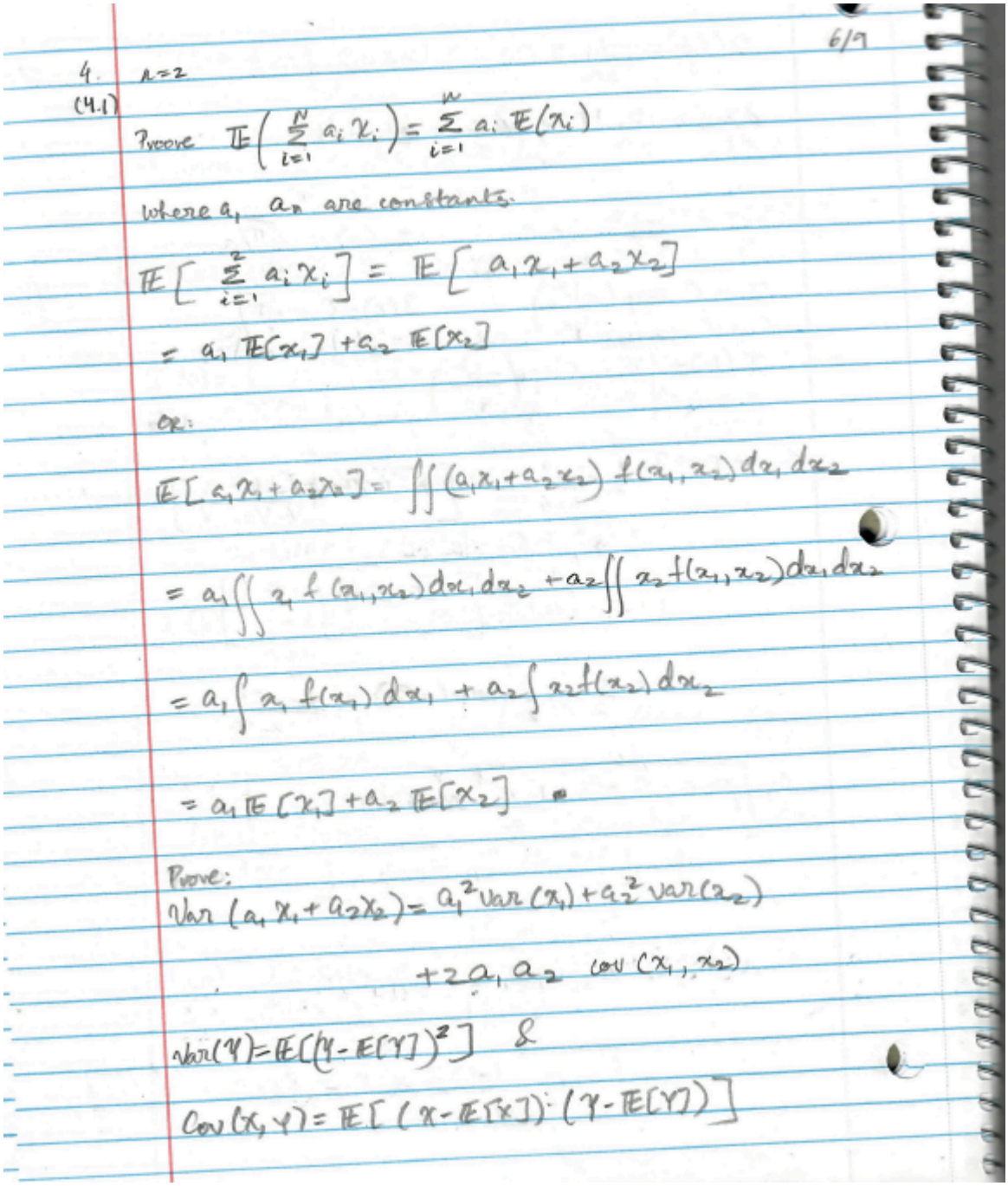
In [5]:

```
import matplotlib.pyplot as plt
plt.plot(x_vals,Ts_vals)
plt.xlabel("x values")
plt.ylabel("temperature Ts(x,q)")
plt.title("Temperature vs position (x)")
```

```
Text(0.5, 1.0, 'Temperature vs position (x)')
```



4 - 4.1



4. n=2

(4.1)

6/9

Prove $\mathbb{E}\left(\sum_{i=1}^{N} a_i X_i\right) = \sum_{i=1}^{N} a_i \mathbb{E}(X_i)$

where a_1 a_n are constants.

$$\mathbb{E}\left[\sum_{i=1}^{2} a_i X_i\right] = \mathbb{E}\left[a_1 X_1 + a_2 X_2\right]$$

$$= a_1 \mathbb{E}[X_1] + a_2 \mathbb{E}[X_2]$$

or:

$$\mathbb{E}[a_1 X_1 + a_2 X_2] = \iint (a_1 x_1 + a_2 x_2) f(x_1, x_2)\, dx_1\, dx_2$$

$$= a_1 \iint x_1 f(x_1, x_2)\, dx_1\, dx_2 + a_2 \iint x_2 f(x_1, x_2)\, dx_1\, dx_2$$

$$= a_1 \int x_1 f(x_1)\, dx_1 + a_2 \int x_2 f(x_2)\, dx_2$$

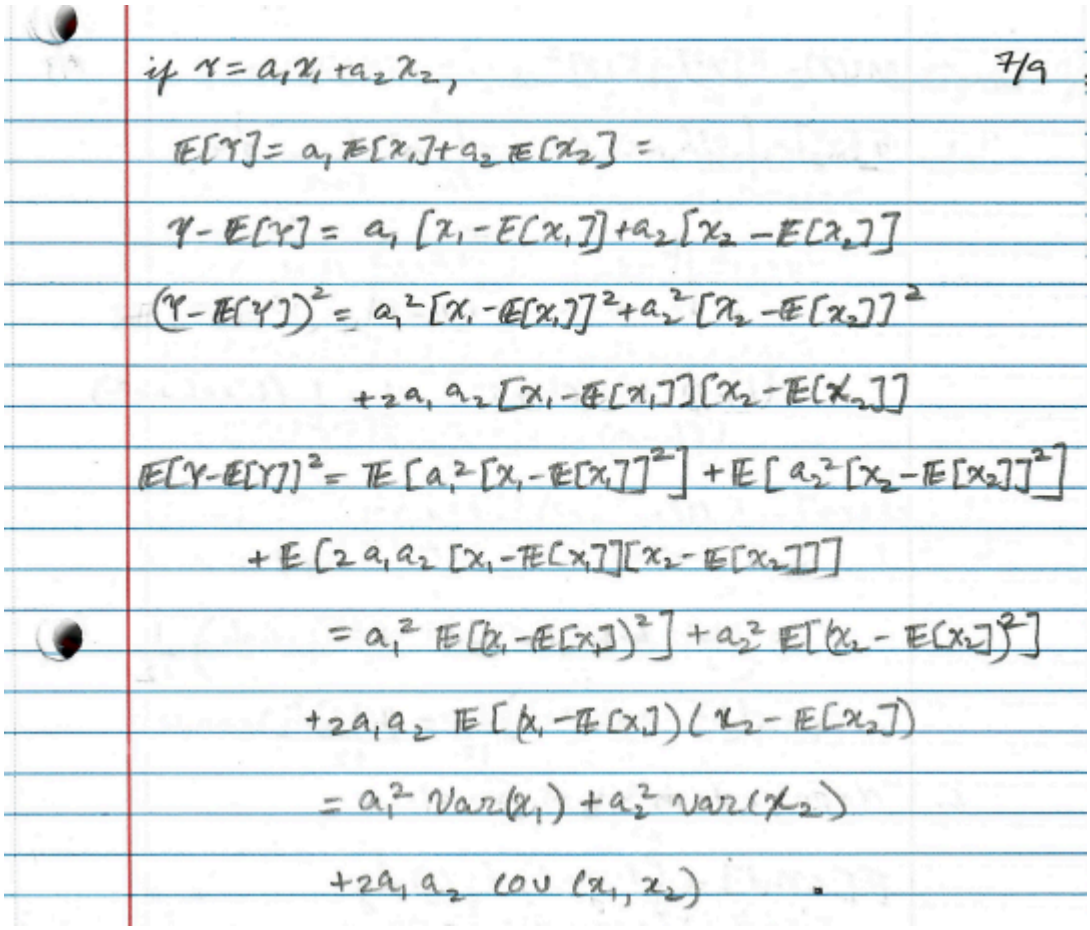
$$= a_1 \mathbb{E}[X_1] + a_2 \mathbb{E}[X_2]$$ ∎

Prove:

$$\mathrm{Var}(a_1 X_1 + a_2 X_2) = a_1^2 \mathrm{Var}(X_1) + a_2^2 \mathrm{Var}(X_2) + 2 a_1 a_2 \mathrm{cov}(X_1, X_2)$$

$\mathrm{Var}(Y) = \mathbb{E}[(Y - \mathbb{E}[Y])^2]$ &

$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X]) \cdot (Y - \mathbb{E}[Y])]$



if $Y = a_1 X_1 + a_2 X_2$, 7/9

$E[Y] = a_1 E[X_1] + a_2 E[X_2] =$

$Y - E[Y] = a_1 [X_1 - E[X_1]] + a_2 [X_2 - E[X_2]]$

$(Y - E[Y])^2 = a_1^2 [X_1 - E[X_1]]^2 + a_2^2 [X_2 - E[X_2]]^2$

$+ 2 a_1 a_2 [X_1 - E[X_1]][X_2 - E[X_2]]$

$E[Y - E[Y]]^2 = E[a_1^2 [X_1 - E[X_1]]^2] + E[a_2^2 [X_2 - E[X_2]]^2]$

$+ E[2 a_1 a_2 [X_1 - E[X_1]][X_2 - E[X_2]]]$

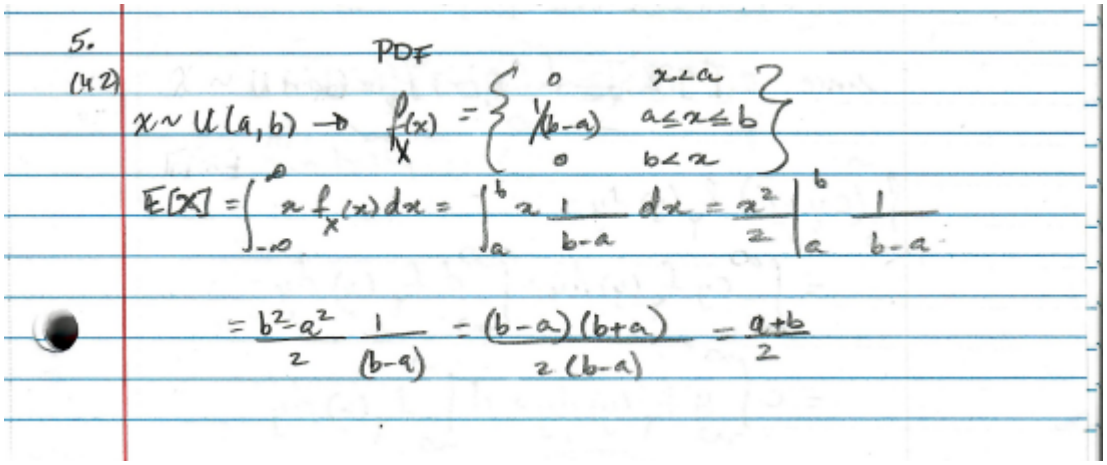
$= a_1^2 E[(X_1 - E[X_1])^2] + a_2^2 E[(X_2 - E[X_2])^2]$

$+ 2 a_1 a_2 E[(X_1 - E[X_1])(X_2 - E[X_2])]$

$= a_1^2 Var(X_1) + a_2^2 Var(X_2)$

$+ 2 a_1 a_2 Cov(X_1, X_2)$.

5 - 4.2



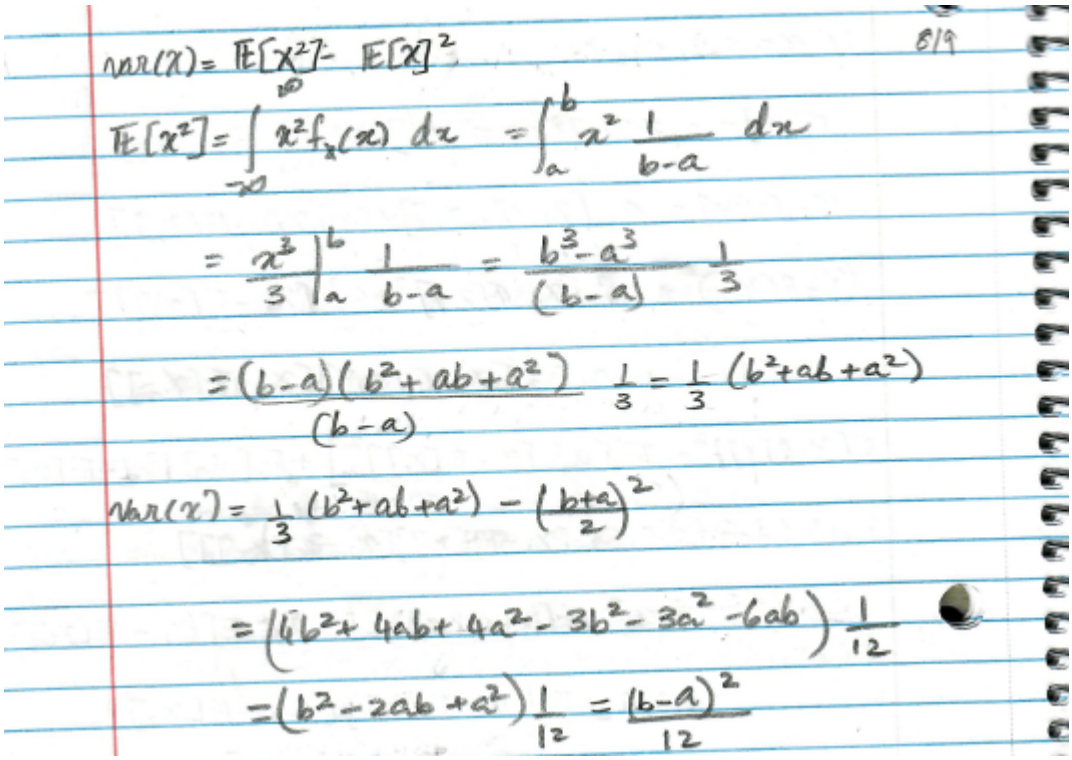
5.
(4.2)

PDF

$X \sim U(a,b) \rightarrow f_X(x) = \begin{cases} 0 & x < a \\ \frac{1}{b-a} & a \le x \le b \\ 0 & b < x \end{cases}$

$E[X] = \int_{-\infty}^{\infty} x f_X(x)\,dx = \int_a^b x \frac{1}{b-a}\,dx = \frac{x^2}{2}\Big|_a^b \frac{1}{b-a}$

$= \frac{b^2-a^2}{2}\frac{1}{(b-a)} = \frac{(b-a)(b+a)}{2(b-a)} = \frac{a+b}{2}$



$Var(X) = E[X^2] - E[X]^2$

6/9

$E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)\,dx = \int_a^b x^2 \frac{1}{b-a}\,dx$

$= \frac{x^3}{3}\Big|_a^b \frac{1}{b-a} = \frac{b^3-a^3}{(b-a)}\frac{1}{3}$

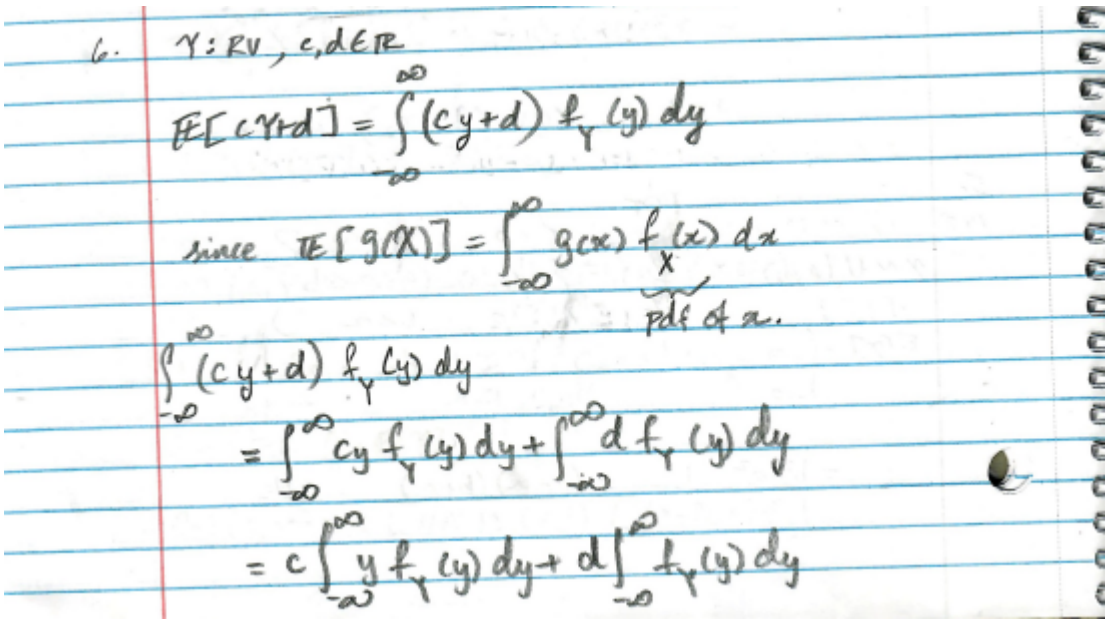
$= \frac{(b-a)(b^2+ab+a^2)}{(b-a)}\frac{1}{3} = \frac{1}{3}(b^2+ab+a^2)$

$Var(X) = \frac{1}{3}(b^2+ab+a^2) - \left(\frac{b+a}{2}\right)^2$

$= \left(4b^2+4ab+4a^2-3b^2-3a^2-6ab\right)\frac{1}{12}$

$= (b^2-2ab+a^2)\frac{1}{12} = \frac{(b-a)^2}{12}$

6 - 4.4



$= c\,\mathbb{E}[Y] + d$, since $\int_{-\infty}^{\infty} f_Y(y)\,dy$ is the integral of a PDF across the whole domain, it equals 1.

$\text{var}(cY+d) = \mathbb{E}[(cY+d)^2] - (\mathbb{E}[cY+d])^2$

$\mathbb{E}[(cY+d)^2] = \mathbb{E}[c^2Y^2 + d^2 + 2cdY]$

$= \mathbb{E}[c^2Y^2] + \mathbb{E}[d^2] + \mathbb{E}[2cdY]$

$= c^2 \int_{-\infty}^{\infty} y^2 f_Y(y)\,dy + d^2 + 2cd \int_{-\infty}^{\infty} y\, f_Y(y)\,dy$

$= c^2\,\mathbb{E}[Y^2] + d^2 + 2cd\,\mathbb{E}[Y]$

so, $\text{var}(cY+d) = c^2\,\mathbb{E}[Y^2] + d^2 + 2cd\,\mathbb{E}[Y]$

$- (c\,\mathbb{E}[Y] + d)^2$

$= c^2\,\mathbb{E}[Y^2] + d^2 + 2cd\,\mathbb{E}[Y]$
$- c^2\,\mathbb{E}[Y]^2 - d^2 - 2cd\,\mathbb{E}[Y]$

$= c^2(\mathbb{E}[Y^2] - \mathbb{E}[Y]^2)$

$= c^2\,\text{var}(Y)$

```python
# filename: heat_rod.py
import sympy as sp
import numpy as np

class SteadyStateRod:
    def __init__(self, a, b, h_val, k_val, Phi_val, Tamb_val, L):
        """
        Initialize the rod parameters.
        a, b: cross-section dimensions (cm)
        h_val: convective heat transfer coefficient (W/cm^2 C)
        k_val: thermal conductivity (W/cm C)
        Phi_val: source heat flux (W/cm^2)
        Tamb_val: ambient temperature (C)
        """
        # Store numeric values
        self.a_val = a
        self.b_val = b
        self.h_val = h_val
        self.k_val = k_val
        self.Phi_val = Phi_val
        self.Tamb_val = Tamb_val
        self.L = L

        # Define symbolic variables
        x, h, k, Phi, Tamb = sp.symbols('x h k Phi Tamb')
        self.x = x
        self.h = h
        self.k = k
        self.Phi = Phi
        self.Tamb = Tamb

        # Precompute gamma
        self.gamma = sp.sqrt(2*(a+b)*h/(a*b*k))

        # Define c1, c2
        c1 = -Phi/(k*self.gamma) * ((sp.exp(self.gamma*self.L)*(h + k*self.gamma)) / (sp.exp(-self.gamma*self.L)*(h - k*self.gamma) \
                                                                                       + sp.exp(self.gamma*self.L)*(h + k*self.gamma)))
        c2 = Phi/(k*self.gamma) + c1
        self.c1 = c1
        self.c2 = c2

        # Ts(x) symbolic
        Ts_expr = c1*sp.exp(-self.gamma*x) + c2*sp.exp(self.gamma*x) + Tamb
        self.Ts_expr = Ts_expr

        # Precompute derivatives symbolically
        self.dTs_dh_expr = sp.diff(Ts_expr, h)
        self.dTs_dk_expr = sp.diff(Ts_expr, k)
        self.dTs_dPhi_expr = sp.diff(Ts_expr, Phi)


        # Lambdify for fast numeric evaluation
        vars = (x, h, k, Phi, Tamb)
        self.Ts_func = sp.lambdify(vars, Ts_expr, 'numpy')
        self.dTs_dh_func = sp.lambdify(vars, self.dTs_dh_expr, 'numpy')
        self.dTs_dk_func = sp.lambdify(vars, self.dTs_dk_expr, 'numpy')
        self.dTs_dPhi_func = sp.lambdify(vars, self.dTs_dPhi_expr, 'numpy')

    def Ts(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        """Evaluate Ts at given x values. Use class defaults if not provided."""
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.Ts_func(x_vals, h, k, Phi, Tamb)

    def dTs_dh(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dh_func(x_vals, h, k, Phi, Tamb)


    def dTs_dk(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dk_func(x_vals, h, k, Phi, Tamb)

    def dTs_dPhi(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dPhi_func(x_vals, h, k, Phi, Tamb)
```