1) Consider Cash Flows of Indicated Table

a) Write Python function that gives NPV of cash flows for a given discount rate

In [12]:
```
import numpy as np

def NPV(cf, t, r):
    # Convert cf and t to numpy arrays to perform element-wise operations
    cf = np.array(cf)
    t = np.array(t)

    # Calculate NPV using element-wise multiplication and summing up the results
    NPV = lambda r: np.sum(cf * np.exp(-r * t))

    return NPV(r)

# Define cash flows, time periods, and discount rate
cf = [-100, 60, 20, 50]
t = [0, 1, 2, 3]
r = 0.05

# Calculate NPV
x = NPV(cf, t, r)
print(x)
```

```
18.205912652014923
```

b) use part a) to create a function that numerically computes the derivative of NPV wrt the discount rate

In [14]:
```
def dNPV_dr(cf, t, r):
    # define small step
    s = 1e-4

    d = ((NPV(cf,t, r+s)) - (NPV(cf,t, r-s)))/(2*s)

    return d

#Test dNPV_dr function
dNPV_dr(cf, t, r)
```

Out[14]:
```
-222.3734609282424
```

c) Use Newton's method algorithm to find small positive root of function in a). Soln is Internal rate

of return (IRR)

In [16]:
```
# Implement newton's method algorithm to find roots

def NewtonMethod(f,df,g,Iter,t):
    if f(g) == 0:
            print('Guess is correct. Root:', g)
    else:
        for i in range(Iter):
            x = g
            x_n = x - (f(x)/df(x))

            # Check Convergence
            if abs(x_n - x) < t:
                print('root of f:', round(x_n,2))
                print('f(x_root):', round(f(x_n),2))
                break
            else:
                g = x_n



    print('Loop Teminated:\nf(x_n):', round(x_n,2), '\nx_n:',round(f(x_n),2),)
    return f(x_n), x_n
```

In [18]:
```
#Initialize variables and function presets
t = [0,1,2,3,4]
cf = [-100, 50, 20, 70, 10]
cf = np.array(cf)
t = np.array(t)
s = 1e-4
tol = 1e-7


#Simplify functions: NPV ans d_NPV as lambda functions
NPV = lambda r: np.sum(cf*np.exp(-r*t))

d_NPV = lambda r: ((NPV(r+s)) - (NPV(r-s)))/(2*s)

# Apply Newton's Method algorithm
vals = NewtonMethod(NPV,d_NPV,0,1000, tol)
```

```
root of f: 0.19
f(x_root): -0.0
Loop Teminated:
f(x_n): 0.19
x_n: -0.0
```

d) Use root_scalar in scipy.optimize to check c)

In [20]:
```
from scipy.optimize import root_scalar
```

```
#function

t = [0,1,2,3,4]
cf = [-100, 50, 20, 70, 10]
cf = np.array(cf)
t = np.array(t)

NPV = lambda r: np.sum(cf*np.exp(-r*t))

root = root_scalar(NPV, bracket=[0,10])

print(f"Root: {round(root.root,2)}")
```

```
Root: 0.19
```

In [22]:
```
# Confirmation should be near zero
confirmation = round(NPV(root.root),2)
confirmation
```

Out[22]:
```
-0.0
```

2) Use partition pairs in the form P and T to compute the defined function

See scanned attachments

3) Inner Product Question

a) See attached pages

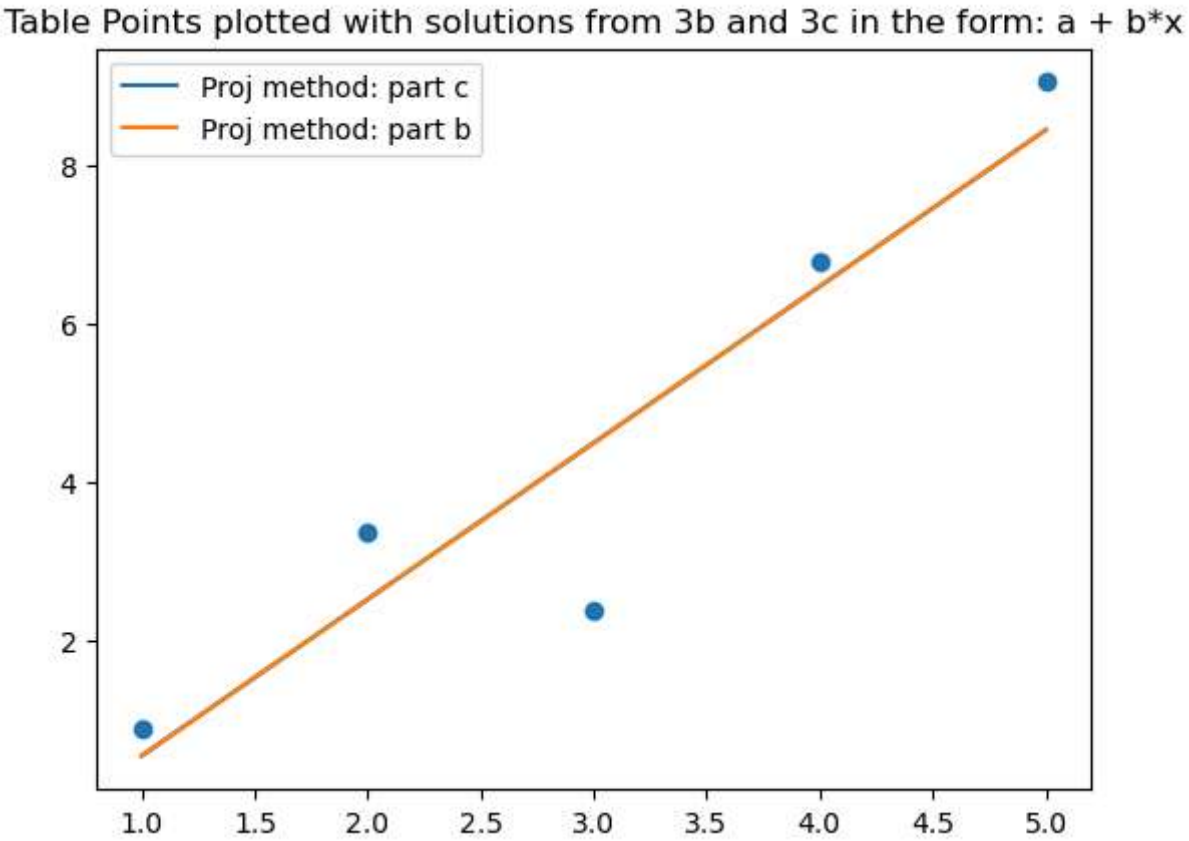
b) See attached pages

c) See attached pages

d) Use scatter to plot table points, Use plot to graph line corresponding to coeff of part b

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt

# table as x and y arrays
x = np.array([1,2,3,4,5])
y = np.array([0.87, 3.37, 2.37, 6.79, 9.05])

# Line with coefficients from part c
sol = lambda x: -361/250 + (989/500)*x
```

```
# Line with coefficients from part b
sol2 = lambda x: -1.444 + 1.978*x

x_sol = np.linspace(1,5,100)

#Plot to verify and show
plt.title('Table Points plotted with solutions from 3b and 3c in the form: a + b*x'
plt.plot(x_sol, sol(x_sol), label = 'Proj method: part c')
plt.plot(x_sol, sol2(x_sol), label = 'Proj method: part b')

plt.scatter(x, y)


plt.legend()


plt.show()
```



Our Solutions from methods 3b (inner product projection) and 3c (minimizing sum of squared error) are identical as shown by the plot above

4) Central Limit Theorem

a) Generate 4 arrays. Take mean of each column. Standardize the array

In [3]:
```
import numpy as np
from scipy.stats import poisson

# mu and n values
mu = 1
n_values = [10, 50, 100, 1000]

array1 = poisson.rvs(mu=mu, size=(n_values[0], 100000))
array1_columnmeans = array1.mean(axis=0)
array1_columnmeans_stand = np.sqrt(n_values[0]) * (array1_columnmeans - 1)

array2 = poisson.rvs(mu=mu, size=(n_values[1], 100000))
array2_columnmeans = array2.mean(axis=0)
array2_columnmeans_stand = np.sqrt(n_values[1]) * (array2_columnmeans - 1)

array3 = poisson.rvs(mu=mu, size=(n_values[2], 100000))
array3_columnmeans = array3.mean(axis=0)
array3_columnmeans_stand = np.sqrt(n_values[2]) * (array3_columnmeans - 1)

array4 = poisson.rvs(mu=mu, size=(n_values[3], 100000))
array4_columnmeans = array4.mean(axis=0)
array4_columnmeans_stand = np.sqrt(n_values[3]) * (array4_columnmeans - 1)
```

b) for each n estimate the probability that the corresponding standardized sample means from (a) are greater than 1.5. Also find P(Z>1.5) where Z ~ N(0,1^2)

In [27]:
```
# n = 10
mean1 = np.mean(array1_columnmeans_stand)
std_dev1 = np.std(array1_columnmeans_stand)
print(f"Mean1: {mean1}")
print(f"Standard Deviation: {std_dev1}")
print(f"""Considering the mean of n=10 dataset is mean1: {round(mean1, 3)} and std
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.""
```

```
Mean1: -0.005252543193539667
Standard Deviation: 0.9954664287609101
Considering the mean of n=10 dataset is mean1: -0.005 and std dev1: 0.995.
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.
```

In [29]:
```
# n = 50
mean2 = np.mean(array2_columnmeans_stand)
std_dev2 = np.std(array2_columnmeans_stand)
print(f"Mean1: {mean2}")
print(f"Standard Deviation: {std_dev2}")
print(f"""Considering the mean of n=50 dataset is mean2: {round(mean2, 3)} and std
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.""
```

```
Mean1: -0.003795749201409347
Standard Deviation: 0.9974647824800633
Considering the mean of n=10 dataset is mean1: -0.004 and std dev1: 0.997.
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.
```

In [31]:
```
# n = 100
mean3 = np.mean(array3_columnmeans_stand)
std_dev3 = np.std(array3_columnmeans_stand)
print(f"Mean3: {mean3}")
print(f"Standard Deviation: {std_dev3}")
print(f"""Considering the mean of n=100 dataset is mean3: {round(mean3, 3)} and std
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.""
```

```
Mean3: 0.0015940000000000867
Standard Deviation: 1.0004942074614924
Considering the mean of n=100 dataset is mean3: 0.002 and std dev3: 1.0.
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.
```

In [33]:
```
# n = 1000
mean4 = np.mean(array4_columnmeans_stand)
std_dev4 = np.std(array4_columnmeans_stand)
print(f"Mean4: {mean4}")
print(f"Standard Deviation: {std_dev4}")
print(f"""Considering the mean of n=1000 dataset is mean4: {round(mean4, 3)} and st
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.""
```

```
Mean4: 0.0012323396041668297
Standard Deviation: 0.9969199322609114
Considering the mean of n=1000 dataset is mean4: 0.001 and std dev3: 0.997.
The probability a mean value from the dataset is greater than 1.5 is about 9.25%.
```

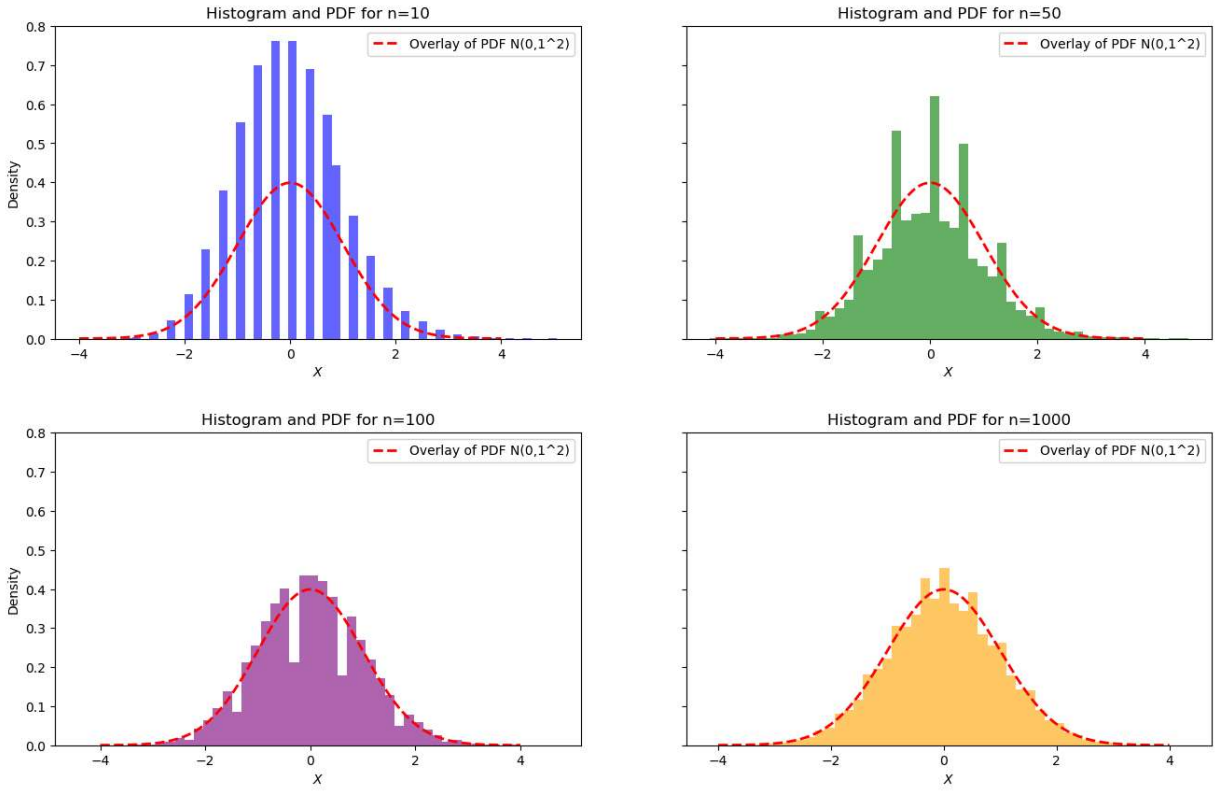
In [5]:
```
# find P(Z>1.5) where Z~N(0,1^2)

# Z = (X - mean)/ Std. Dev

Z = (1.5-0)/1

print('Z is equal to:',Z,)
print('The probability that Z is greater than 1.5 is appoximately 9.25%')
```

```
Z is equal to: 1.5
The probability that Z is greater than 1.5 is appoximately 9.25%
```

c) Display histograms of results from a). Create 2x2 grid of subplots.

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson, norm

# PDF x and y values
x_vals = np.linspace(-4, 4, 1000)
sigma = 1  # Standard deviation for N(0, 1^2)
pdf_vals = norm.pdf(x_vals, loc=0, scale=sigma)

# Subplots Part A
```

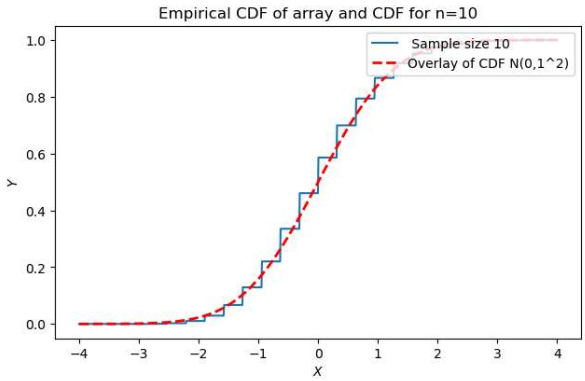
```
fig, ax = plt.subplots(2, 2, sharey=True, figsize=(16, 10))
plt.subplots_adjust(hspace=0.3, wspace=0.2)

# For n=10
ax[0, 0].hist(array1_columnmeans_stand, bins=50, density=True, alpha=0.6, color='b'
ax[0, 0].plot(x_vals, pdf_vals, 'r--', lw=2, label=r'Overlay of PDF N(0,1^2)')
ax[0, 0].set_title(r'Histogram and PDF for n=10')
ax[0, 0].set_xlabel(r'$X$')
ax[0, 0].set_ylabel(r'Density')
ax[0, 0].legend(loc="upper right")

# For n=50
ax[0, 1].hist(array2_columnmeans_stand, bins=50, density=True, alpha=0.6, color='g'
ax[0, 1].plot(x_vals, pdf_vals, 'r--', lw=2, label=r'Overlay of PDF N(0,1^2)')
ax[0, 1].set_title(r'Histogram and PDF for n=50')
ax[0, 1].set_xlabel(r'$X$')
ax[0, 1].legend(loc="upper right")

# For n=100
ax[1, 0].hist(array3_columnmeans_stand, bins=50, density=True, alpha=0.6, color='pu
ax[1, 0].plot(x_vals, pdf_vals, 'r--', lw=2, label=r'Overlay of PDF N(0,1^2)')
ax[1, 0].set_title(r'Histogram and PDF for n=100')
ax[1, 0].set_xlabel(r'$X$')
ax[1, 0].set_ylabel(r'Density')
ax[1, 0].legend(loc="upper right")

# For n=1000
ax[1, 1].hist(array4_columnmeans_stand, bins=50, density=True, alpha=0.6, color='or
ax[1, 1].plot(x_vals, pdf_vals, 'r--', lw=2, label=r'Overlay of PDF N(0,1^2)')
ax[1, 1].set_title(r'Histogram and PDF for n=1000')
ax[1, 1].set_xlabel(r'$X$')
ax[1, 1].legend(loc="upper right")

plt.show()
```

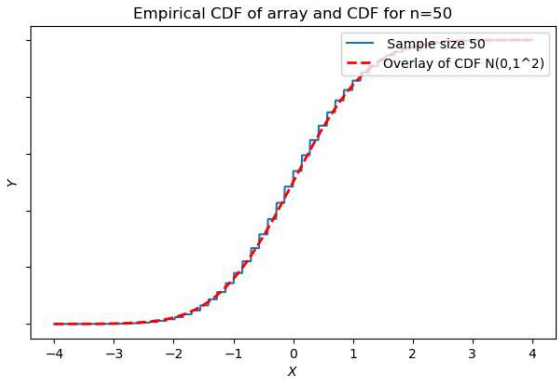


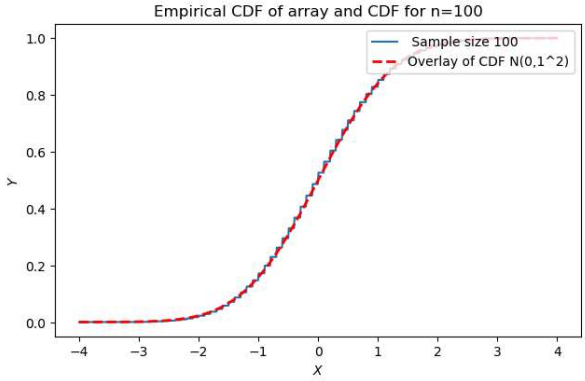
d) Display results from a) using cumulative distribution functions. Then overlay graph of cumulative distribution function

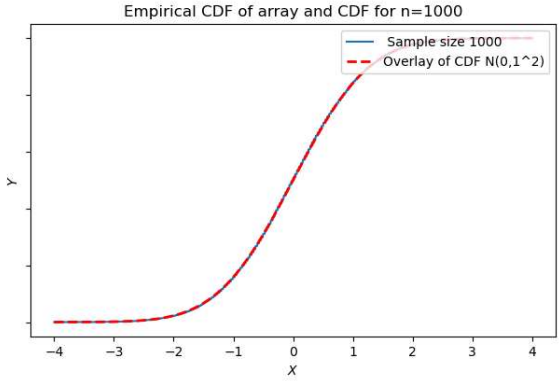
In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson, norm, chi2

#x and y values
x_vals = np.linspace(-4, 4, 1000)
cdf_vals=norm.cdf(x_vals, loc=0, scale=sigma)

#create empirical cdf
ecdf = lambda x, data: np.mean(data <= x)


# Subplots
fig, ax = plt.subplots(2, 2, sharey=True, figsize=(16, 10))
plt.subplots_adjust(hspace=0.3, wspace=0.2)

# For n=10
y_vals1 = [ecdf(x, array1_columnmeans_stand) for x in x_vals]
ax[0, 0].plot(x_vals, y_vals1,label =' Sample size 10')
ax[0, 0].plot(x_vals, cdf_vals, 'r--', lw=2, label=r'Overlay of CDF N(0,1^2)')
ax[0, 0].set_title(r'Empirical CDF of array and CDF for n=10')
ax[0, 0].set_xlabel(r'$X$')
ax[0, 0].set_ylabel(r'$Y$')
ax[0, 0].legend(loc="upper right")

# For n=50
```

```
y_vals2 = [ecdf(x, array2_columnmeans_stand) for x in x_vals]
ax[0, 1].plot(x_vals, y_vals2,label =' Sample size 50')
ax[0, 1].plot(x_vals, cdf_vals, 'r--', lw=2, label=r'Overlay of CDF N(0,1^2)')
ax[0, 1].set_title(r'Empirical CDF of array and CDF for n=50')
ax[0, 1].set_xlabel(r'$X$')
ax[0, 1].set_ylabel(r'$Y$')
ax[0, 1].legend(loc="upper right")

# For n=100
y_vals3 = [ecdf(x, array3_columnmeans_stand) for x in x_vals]
ax[1, 0].plot(x_vals, y_vals3,label =' Sample size 100')
ax[1, 0].plot(x_vals, cdf_vals, 'r--', lw=2, label=r'Overlay of CDF N(0,1^2)')
ax[1, 0].set_title(r'Empirical CDF of array and CDF for n=100')
ax[1, 0].set_xlabel(r'$X$')
ax[1, 0].set_ylabel((r'$Y$'))
ax[1, 0].legend(loc="upper right")

# For n=1000
y_vals4 = [ecdf(x, array4_columnmeans_stand) for x in x_vals]
ax[1, 1].plot(x_vals, y_vals4,label =' Sample size 1000')
ax[1, 1].plot(x_vals, cdf_vals, 'r--', lw=2, label=r'Overlay of CDF N(0,1^2)')
ax[1, 1].set_title(r'Empirical CDF of array and CDF for n=1000')
ax[1, 1].set_xlabel(r'$X$')
ax[1, 1].set_ylabel(r'$Y$')
ax[1, 1].legend(loc="upper right")

# Show the plot
plt.show()
```









MFE MATH BOOTCAMP HW #4

2) Use partition pairs:

$P = (1, b^{1/n}, b^{2/n}, \ldots b)$ and $T = (1, b^{1/n}, b^{2/n}, \ldots b^{(n-1)/n})$

to analytically compute: $\int_1^b \frac{dx}{x^p}$ for $b, p > 1$

$$\sum_{k=1}^{n} f(t_k)\Delta x_k = \sum_{k=1}^{n} \frac{1}{(b^{(k-1)/n})^p} \cdot \left(b^{k/n} - b^{(k-1)/n}\right)$$

H $\quad$ W

$\Rightarrow$ Simplify

$$\sum_{k=1}^{n} b^{-p(k-1)/n}\left(b^{k/n} - b^{(k-1)/n}\right)$$

$$\sum_{k=1}^{n} b^{-pk/n + k/n} - b^{-pk+p+k-1/n}$$

$$\sum_{k=1}^{n} b^{k/n}\, b^{(1-p)(k-1)/n} - b^{(1-p)(k-1)/n}$$

* To finish solving, simplify expression above via geometric series summation simplification

* then take limit of expression as n approaches ∞

3) •

x_i	1	2	3	4	5
y_i	0.87	3.37	2.37	6.79	9.05

• Inner prod on vector space of functions from: $\{1,2,3,4,5\}$ to $\mathbb{R}$ as

$$\langle f, g \rangle = \sum_{i=1}^{5} f(x_i)\, g(x_i)$$

a) Orthogonalize 1st 2 elements of basis $(1, x, x^2, x^3, x^4)$

$u_1 = 1$

$u_2 = x - \frac{\langle x, 1 \rangle}{\|1\|^2} \cdot 1$

• $\langle x, 1 \rangle = (1\cdot1) + (1\cdot2) + (1\cdot3) + (1\cdot4) + (1\cdot5) = 15$

• $\langle 1, 1 \rangle = \|1\|^2 = (1) + (1) + (1) + (1) + (1) = 5$

$u_2 = x - \frac{15}{5}$

$u_2 = x - 3$

$u_1 = 1$

$u_2 = x - 3$

* Check $\langle 1, x-3 \rangle \Rightarrow (1\cdot-2) + (1\cdot-1) + (1\cdot0) + (1\cdot1) + (1\cdot2)$

$-2 + -1 + 0 + 1 + 2 = 0$

✓

b) $u_1 = 1$

$u_2 = x - 3$

f

f ⇒

x_i	1	2	3	4	5
y_i	0.87	3.37	2.37	6.79	9.05

project f with xy coordinates in table onto 2 orthogonal basis elements in a). Soln should be in the form $a + bx$

$$\boxed{a + bx}$$

$$a = \frac{\langle y, 1 \rangle}{\langle 1, 1 \rangle} = \frac{0.87 + 3.37 + 2.37 + 6.79 + 9.05}{5} = 4.49$$

$$b = \frac{\langle y, x-3 \rangle}{\langle x-3, x-3 \rangle} = \frac{(0.87 \cdot -2 + 3.37 \cdot -1 + 2.37 \cdot 0 + 6.79 + 9.05 \cdot 2)}{(-2 \cdot -2) + (-1 \cdot -1) + 0 + 1 + (2 \cdot 2)}$$

$$= \frac{16.42}{10} = 1.642 \quad \frac{19.75}{10} =$$

$4.49 + 1.62(x-3)$

$4.49 - 1.62x + 4.86$

$9.35 + 1.62x$

$1.50 + 1.975(x-3)$

$1.50 + 1.975x - 5.934$

$-4.43 + 1.975x$

$4.49 + 1.978(x-3)$

$4.49 + 1.978x - 5.934$

$$\boxed{-1.444 + 1.978x}$$

3) part c

$$\mathcal{L}_2(\alpha, \beta) = \sum_{i=1}^{5} (y_i - \alpha - \beta x_i)^2$$

Minimize $\mathcal{L}_2$ wrt α and β, check if soln is minimum or maximum.

$$\mathcal{L}_2 = (0.87 - \alpha - \beta)^2 + (3.37 - \alpha - 2\beta)^2 + (2.37 - \alpha - 3\beta)^2 + (6.79 - \alpha - 4\beta)^2 + (9.05 - \alpha - 5\beta)^2$$

$$\frac{\partial \mathcal{L}_2}{\partial \alpha} = -2(0.87 - \alpha - \beta) - 2(3.37 - \alpha - 2\beta) - 2(2.37 - \alpha - 3\beta) - 2(6.79 - \alpha - 4\beta) - 2(9.05 - \alpha - 5\beta)$$

$$\Rightarrow -2(0.87 + 3.37 + 2.37 + 6.79 + 9.05) + 10\alpha + (2 + 4 + 6 + 8 + 10)\beta$$

$$\Rightarrow 0 = -44.9 + 10\alpha + 30\beta$$

1) $\Rightarrow 44.9 = 10\alpha + 30\beta$

$$\frac{\partial \mathcal{L}_2}{\partial \beta} = -2(0.87 - \alpha - \beta) - 4(3.37 - \alpha - 2\beta) - 6(2.37 - \alpha - 3\beta) - 8(6.79 - \alpha - 4\beta) - 10(9.05 - \alpha - 5\beta)$$

$$0 = -174.26 + 30\alpha + 110\beta$$

2) $\Rightarrow 174.26 = 30\alpha + 110\beta$

- Find α and β

1) $44.9 = 10\alpha + 30\beta$
2) $174.26 = 30\alpha + 110\beta$

* plugged into online solver

$\alpha = -361/250$

$\beta = 989/500$

$$D = \begin{vmatrix} \frac{\partial^2 f}{\partial \alpha^2} & \frac{\partial^2 f}{\partial \alpha \partial \beta} \\ \frac{\partial^2 f}{\partial \beta \partial \alpha} & \frac{\partial^2 f}{\partial \beta^2} \end{vmatrix}$$

$$= \begin{vmatrix} 10 & 30 \\ 30 & 110 \end{vmatrix}$$

$D = 1100 - 900 = 200$, $D > 0$, test works

$\frac{\partial^2 f}{\partial \alpha^2}\left(-\frac{361}{250}, \frac{989}{500}\right) = 10 \Rightarrow$ local min

$\alpha = \frac{-361}{250}$, $\beta = \frac{989}{500}$ Local min