Machine Learning and Computational Statistics

Homework 4

1) Correct: 2, 4, 5

2) Correct: 1, 4

3) Correct: 2, 3

4) Correct: 2

5) Correct: 1, 2, 4

6) Correct: 1, 4

7) Correct: 2, 4

8) Correct: 1

9) Correct: 1, 3

10) Correct: 1, 4

11) Correct: 1, 3, 4.

12) Correct: 1

13) a) → 1, b) → 2, c) → ~~3~~ ← 2 still quadratic!

* The green points don't add new info. Model → quadratic

14) Correct: 1, 4

15) Correct: 4

16) Correct: 1, 2, 3

17) Correct: 1

21) a) $\int_0^1 \int_{x^3}^1 p(x,y)\,dx\,dy = 4/3 \int_0^1 [y]_{x^3}^1\,dx = 4/3 \int_0^1 (1-x^3)\,dx$

$= 4/3\,[x - x^4/4]_0^1 = 1.$

b) $p(x) = \int_{x^3}^1 p(x,y)\,dy = 4/3\,[y]_{x^3}^1 = 4/3(1-x^3)$

c) $p(y|x) = p(x,y)/p(x) = \dfrac{1}{1-x^3}$

d) $E[y|x] = \int_{x^3}^1 y\,p(y|x)\,dy = \int_{x^3}^1 y\,\dfrac{1}{1-x^3}\,dy = \dfrac{1}{1-x^3}\left[\dfrac{y^2}{2}\right]_{x^3}^1 =$

$= \dfrac{1-x^6}{2(1-x^3)} = \dfrac{1}{2}(1+x^3).$

20) $E_D[(f(x;D)-E[y|x])^2]$ can be zero when:

- $E_D[f(x;D)] = E[y|x]$, in other words perfect, optimal estimator, zero bias
- Variance $E_D[(f(x;D) - E_D[f(x;D)])^2]$ is zero, so D is very large, so we have the same result ~~[illegible]~~

b) Achieving zero MSE is not possible in real world as data are typically noisy leading to errors. Not infinite sample sizes also give limitations for the data and their variations. So the aim must be to have a low MSE as zero MSE is practically impossible

**Exercise 18 (regularization - python code):**

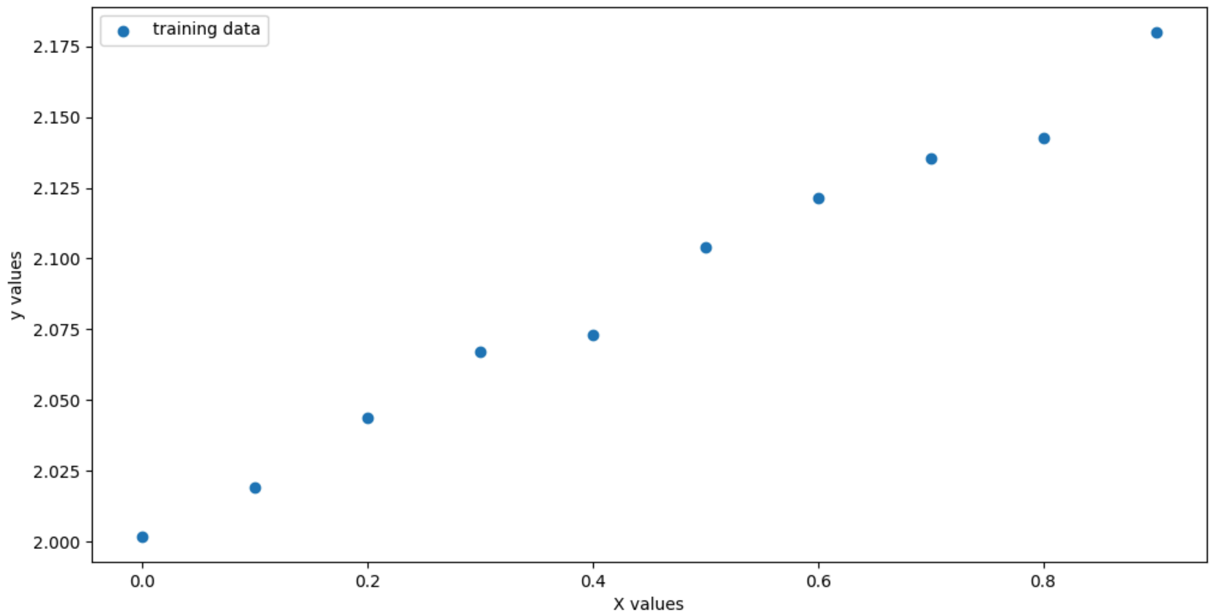
Consider the data set given in the attached file (the code for reading from python is also given). Specifically, it consists of 10 data pairs of the form $(y_i, x_i)$, $i = 1, \ldots, 10$. All $y_i$'s are accumulated in the vector $\boldsymbol{y}$ while all $x_i$'s are accumulated in the vector $\boldsymbol{x}$.

The aim is to unravel the relation between $x_i$'s and $y_i$'s.

(a) Plot the data.
(b) Fit a $8^{th}$ degree polynomial on the data using the LS estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial.
(c) Fit a $8^{th}$ degree polynomial on the data using the ridge regression estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial. Experiment with various values of $\lambda$.
(d) Fit a $8^{th}$ degree polynomial on the data using the lasso estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial. Experiment with various values of $\lambda$.
(e) Discuss briefly on the results produced by lasso and compare them with those produced by the LS and ridge regression cases.

In [1]:
```
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
!pip install scikit-learn


Training_Set = sio.loadmat('Training_Set.mat')
X = Training_Set['X']
y = Training_Set['y']
#Plot
plt.figure(figsize=(12, 6))
plt.scatter(X, y, label='training data')
plt.xlabel('X values')  # Optional: Add x-axis label
plt.ylabel('y values')  # Optional: Add y-axis label
plt.legend()
plt.show()
```

```
Requirement already satisfied: scikit-learn in c:\users\steli\anaconda3\lib\site-pac
kages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\steli\anaconda3\lib\site-pa
ckages (from scikit-learn) (2.1.2)
Requirement already satisfied: scipy>=1.6.0 in c:\users\steli\anaconda3\lib\site-pac
kages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\steli\anaconda3\lib\site-pa
ckages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\steli\anaconda3\lib
\site-packages (from scikit-learn) (3.5.0)
```




# (b) Fit a 8th degree polynomial on the data using the LS estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial.

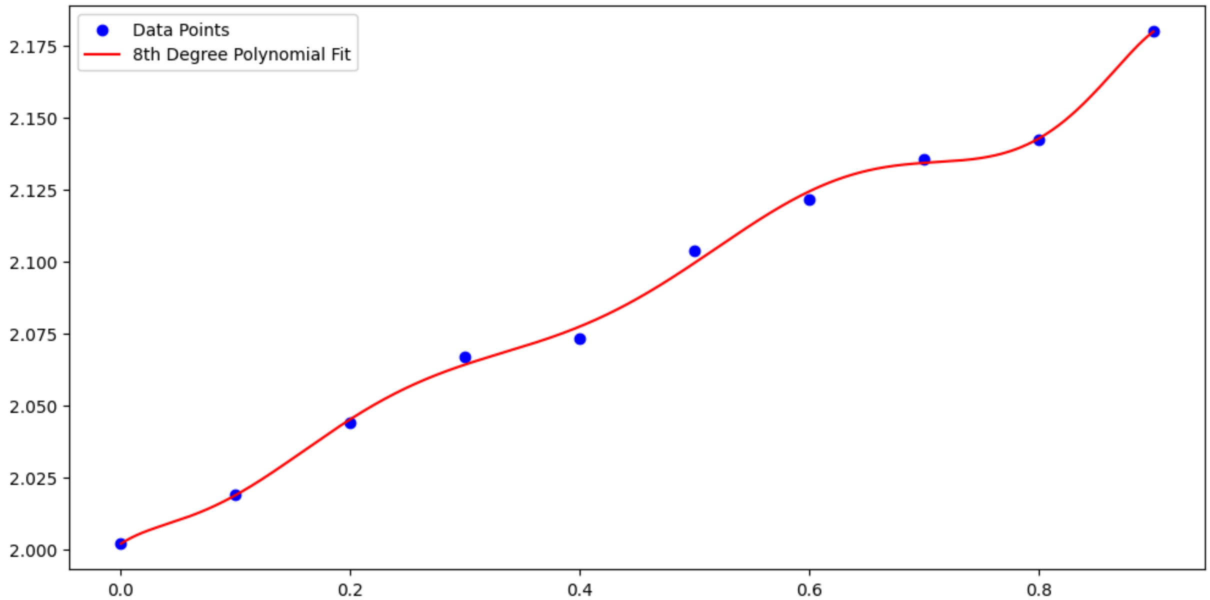
In [2]:

```
# Load data
Training_Set = sio.loadmat('Training_Set.mat')
X = Training_Set['X'].flatten()  # Ensure X is a 1D array
y = Training_Set['y'].flatten()  # Ensure y is a 1D array

#  matrix manually for an 8th-degree polynomial
degree = 8
n = len(X)
X_design = np.zeros((n, degree + 1))

for i in range(degree + 1):
    X_design[:, i] = X ** i  # Fill each column with X raised to the ith power

# polynomial coefficients using the normal equation
coefficients = np.linalg.inv(X_design.T @ X_design) @ X_design.T @ y

# polynomial fit line
x_fit = np.linspace(min(X), max(X), 1000)
y_fit = sum(coefficients[i] * x_fit ** i for i in range(degree + 1))
```

In [3]:

```
# Plot data points and fitted polynomial curve
plt.figure(figsize=(12, 6))
plt.scatter(X, y, label='Data Points', color='blue', marker='o')
plt.plot(x_fit, y_fit, label=f'{degree}th Degree Polynomial Fit', color='red')
plt.legend()
plt.show()

# Output polynomial parameters
```

```
print("Polynomial coefficients (from constant term to highest degree):")
print(coefficients)
```



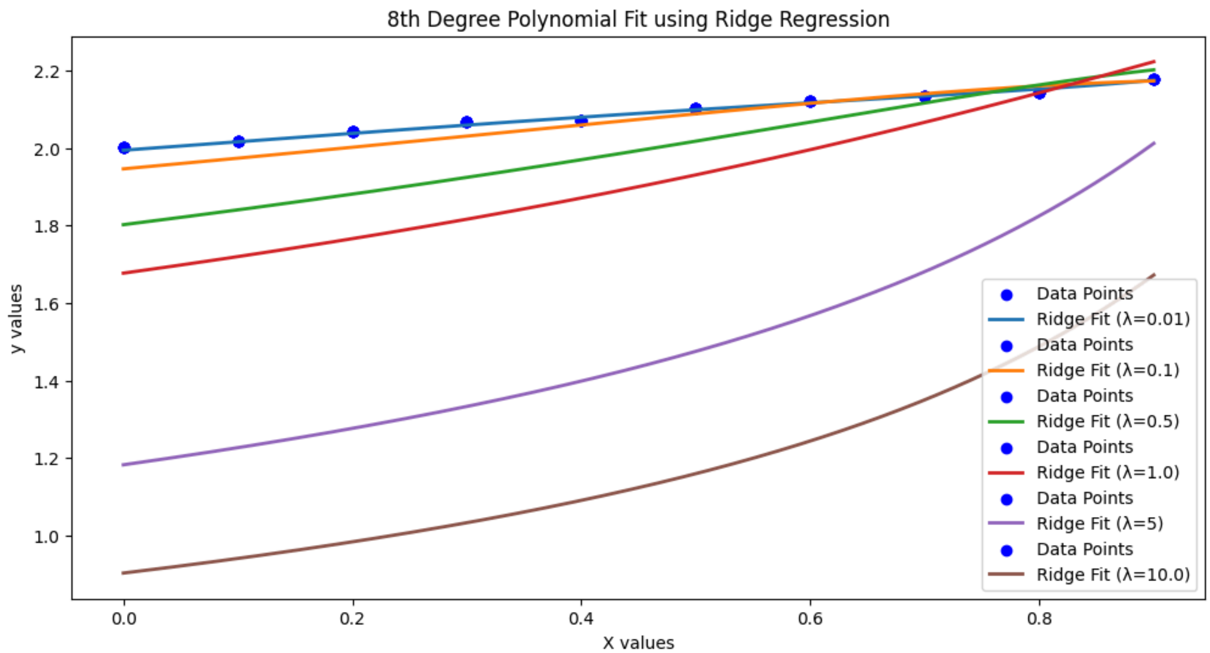

```
Polynomial coefficients (from constant term to highest degree):
[ 2.00198649e+00  2.59415131e-01 -3.83250479e+00  4.94210227e+01
 -2.62209216e+02  6.98309881e+02 -9.86598010e+02  7.06148417e+02
 -2.01452196e+02]
```

## (c) Fit a 8th degree polynomial on the data using the ridge regression estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial. Experiment with various values of $\lambda$.

In [4]:

```
#  matrix for an 8th-degree polynomial
degree = 8
n = len(X)
X_design_ridge = np.zeros((n, degree + 1))

for i in range(degree + 1):
    X_design_ridge[:, i] = X ** i  # Filling each column with X raised to the ith p
# Define the Ridge regression function
def ridge_regression(X, y, lambda_):
    n = X.shape[1]
    I = np.eye(n)  # Identity matrix
    coefficients = np.linalg.inv(X.T @ X + lambda_ * I) @ X.T @ y
    return coefficients
plt.figure(figsize=(12, 6))

# Defining different values of λ
lambdas = [0.01,0.1,0.5, 1.0, 5, 10.0]
for lambda_ in lambdas:
    # Compute polynomial coefficients using Ridge regression
    coefficients = ridge_regression(X_design_ridge, y, lambda_)
```

```
    # Generate polynomial fit line
    x_fit = np.linspace(min(X), max(X), 1000)
    y_fit = sum(coefficients[i] * x_fit ** i for i in range(degree + 1))

    # Plot data points and fitted polynomial curve
    plt.scatter(X, y, label='Data Points', color='blue', marker='o')
    plt.plot(x_fit, y_fit, label=f'Ridge Fit (λ={lambda_})', linewidth=2)

# Customize plot
plt.legend()
plt.xlabel('X values')
plt.ylabel('y values')
plt.title('8th Degree Polynomial Fit using Ridge Regression')
plt.show()

# Output polynomial parameters for each lambda
for lambda_ in lambdas:
    coefficients = ridge_regression(X_design_ridge, y, lambda_)
    print(f"Polynomial coefficients for λ={lambda_}:")
    print(coefficients)
```



```
Polynomial coefficients for λ=0.01:
[ 1.99471461  0.21909439  0.00406036 -0.03389617 -0.02809453 -0.01125288
  0.00688224  0.02298818  0.03599   ]
Polynomial coefficients for λ=0.1:
[ 1.94657363  0.27166295  0.04077954 -0.01282617 -0.02248477 -0.02009638
 -0.0147972  -0.00943891 -0.00487436]
Polynomial coefficients for λ=0.5:
[ 1.8024133   0.3714051   0.11204427  0.02721759 -0.00480358 -0.01729449
 -0.02164996 -0.02240455 -0.02154775]
Polynomial coefficients for λ=1.0:
[ 1.6771831   0.41167511  0.15549847  0.06257238  0.02254727  0.00368133
 -0.00552871 -0.00995018 -0.0118701 ]
Polynomial coefficients for λ=5:
[1.18222158 0.41850661 0.22616662 0.14174991 0.09694072 0.07020591
 0.05295344 0.04117141 0.03277152]
Polynomial coefficients for λ=10.0:
[0.9026125  0.35527217 0.20706347 0.13826342 0.09978578 0.07569215
 0.05943095 0.04785359 0.03927451]
```

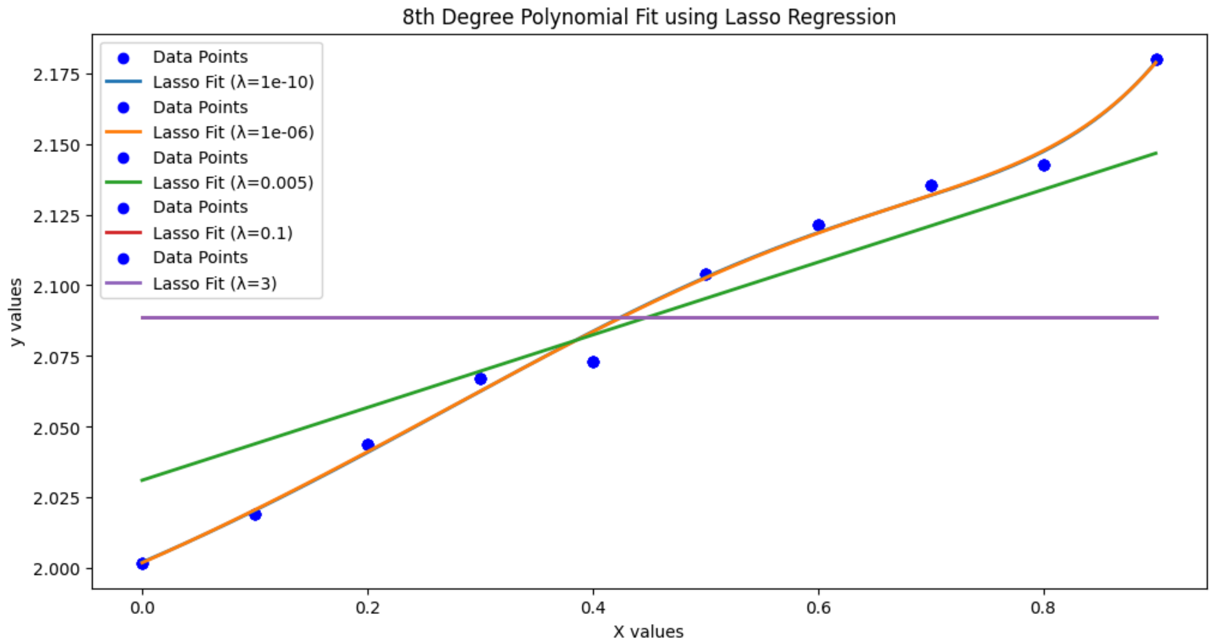
For lower lambda we have better fitting

## (d) Fit a 8th degree polynomial on the data using the lasso estimator and plot the results (data points and the curve resulting from the fit). Output also the estimates of the parameters of the polynomial. Experiment with various values of $\lambda$.

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
from sklearn.linear_model import Lasso

# Load data
Training_Set = sio.loadmat('Training_Set.mat')
X = Training_Set['X'].flatten()  # Ensure X is a 1D array
y = Training_Set['y'].flatten()  # Ensure y is a 1D array

degree = 8
n = len(X)
X_design_lasso = np.zeros((n, degree + 1))

for i in range(degree + 1):
    X_design_lasso[:, i] = X ** i

# Experiment with different values of λ
lambdas = [1e-10,1e-06,5e-03,0.1,3]

plt.figure(figsize=(12, 6))

for lambda_ in lambdas:
    # Fit the Lasso regression model
    model = Lasso(alpha=lambda_)
    model.fit(X_design_lasso, y)
```

```python
    # Generate polynomial fit line
    x_fit = np.linspace(min(X), max(X), 1000)
    X_fit_design = np.zeros((len(x_fit), degree + 1))

    for i in range(degree + 1):
        X_fit_design[:, i] = x_fit ** i  # Create the design matrix for x_fit

    y_fit = model.predict(X_fit_design)  # Predict using the Lasso model

    # Plot data points and fitted polynomial curve
    plt.scatter(X, y, label='Data Points', color='blue', marker='o')
    plt.plot(x_fit, y_fit, label=f'Lasso Fit (λ={lambda_})', linewidth=2)

# Customize plot
plt.legend()
plt.xlabel('X values')
plt.ylabel('y values')
plt.title('8th Degree Polynomial Fit using Lasso Regression')
plt.show()

# Output polynomial parameters for each lambda
for lambda_ in lambdas:
    model = Lasso(alpha=lambda_)
    model.fit(X_design_lasso, y)
    print(f"Polynomial coefficients for λ={lambda_} (from constant term to highest
    print(model.coef_)
```

```
C:\Users\steli\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:697: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 9.678e-05, tolerance: 2.981e-06
  model = cd_fast.enet_coordinate_descent(
C:\Users\steli\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:697: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 4.340e-05, tolerance: 2.981e-06
  model = cd_fast.enet_coordinate_descent(
```




```
Polynomial coefficients for λ=1e-10 (from constant term to highest degree):
[ 0.          0.16995658  0.14098216 -0.07624334 -0.12936744 -0.09749301
 -0.01729305  0.08160893  0.18043564]
Polynomial coefficients for λ=1e-06 (from constant term to highest degree):
[ 0.          0.17547491  0.11999218 -0.06109269 -0.12454477 -0.08486763
 -0.01077142  0.03825068  0.19828017]
Polynomial coefficients for λ=0.005 (from constant term to highest degree):
[0.        0.1283484 0.        0.        0.        0.        0.
 0.        0.       ]
Polynomial coefficients for λ=0.1 (from constant term to highest degree):
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
Polynomial coefficients for λ=3 (from constant term to highest degree):
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
C:\Users\steli\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:697: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 9.678e-05, tolerance: 2.981e-06
  model = cd_fast.enet_coordinate_descent(
C:\Users\steli\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:697: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 4.340e-05, tolerance: 2.981e-06
  model = cd_fast.enet_coordinate_descent(
```

# (e) Discuss briefly on the results produced by lasso and compare them with those produced by the LS and ridge regression cases.

In Lasso, we can notice many coefficients being set to zero, indicating feature selection. In contrast, Ridge will show all coefficients with reduced magnitudes, while LS retains all original coefficients. Lasso and Ridge can generalize to new data , while LS be overfitted.

**Exercise 19 (python code + text):**

Consider the set-up of exercise 15 from Homework 3. Consider also the ridge regression estimators resulting from eq. (A) in the exercise 1 above, for $\lambda = 0, 0.1, 0.2, \ldots, 10000$. For each one of these values of $\lambda$, apply the steps (a), (b), (c1) of exercise 15 above, in order to compute the MSE. Plot MSE versus $\lambda$ and

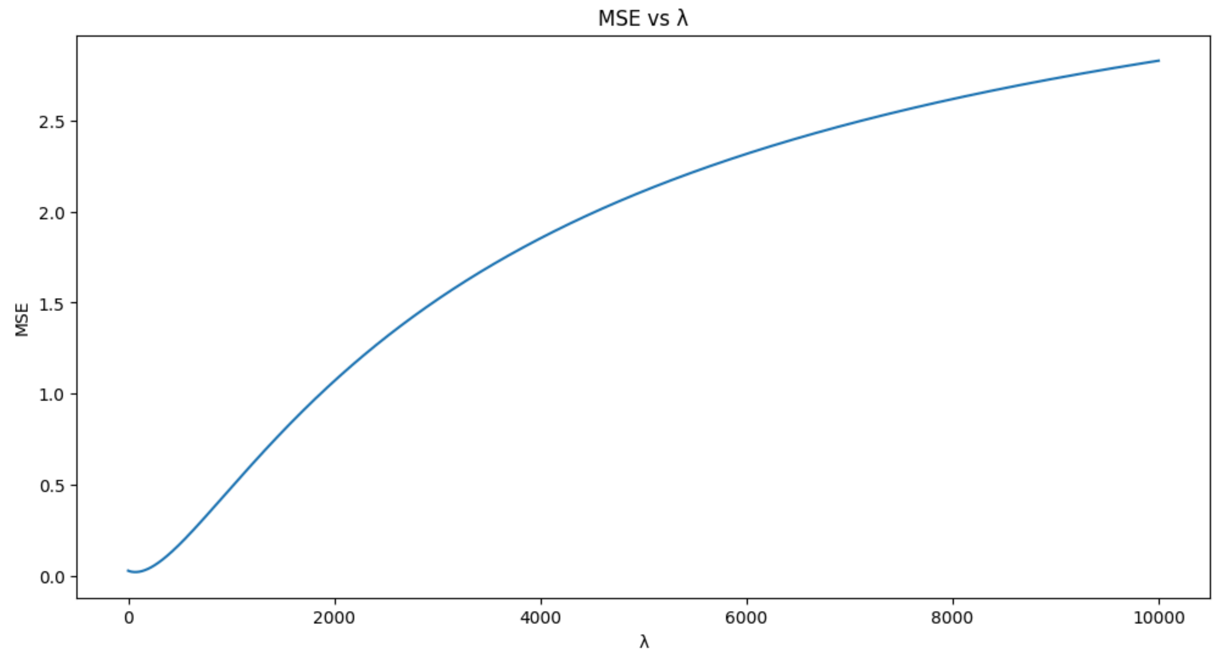
**(i)** determine the range of values of $\lambda$ where the MSE is smaller than that of the unbiased LS estimator,

**(ii)** Comment on the results.

In [6]:

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

# Parameters
mean = 0
sigma = 8
N = 30
theta = 2
num_datasets = 50

# Generate a single set of datasets
X, Y = [], []
for _ in range(num_datasets):
    x = np.random.normal(mean, sigma, N)
    y = theta * x + np.random.normal(0, np.sqrt(64), N)
    X.append(x)
    Y.append(y)

X = np.array(X).T
Y = np.array(Y).T

# Estimation of theta for different lambda values
Theta_est0_lambda = []
lambda_values = np.arange(0.0, 10000.0, 0.1)

for lamda in lambda_values:
    Theta_est = []
    for i in range(num_datasets):
        arr_ones = np.ones(len(X))
        X_new = np.column_stack((arr_ones, X[:, i]))
        XTXinv = np.linalg.inv(np.dot(X_new.T, X_new) + lamda * np.identity(2))
        theta_est = np.dot(XTXinv, X_new.T).dot(Y[:, i])
        Theta_est.append(theta_est[1])
    Theta_est0_lambda.append(Theta_est)

# Calculate MSE
MSE_lambda = []
for theta_est in Theta_est0_lambda:
    MSE = np.mean((np.array(theta_est) - theta) ** 2)
    MSE_lambda.append(MSE)
```

```
# Plotting MSE
plt.figure(figsize=(12, 6))
plt.plot(lambda_values, MSE_lambda)
plt.title('MSE vs λ')
plt.xlabel('λ')
plt.ylabel('MSE')
plt.show()
```




**<u>Exercise 22</u>** <u>(python code + text):</u>

Consider the **regression problem** (1-dep., 1-indep. variables)

$$y = g(x) + \eta$$

where y and x are jointly distributed according to the normal distribution $p(y,x) = N(\boldsymbol{\mu}, \Sigma)$

with $\boldsymbol{\mu} = \begin{bmatrix} \mu_y \\ \mu_x \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} {\sigma_y}^2 & \sigma_{yx} \\ \sigma_{yx} & {\sigma_x}^2 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 3 & 5 \end{bmatrix}$

(a) Determine $E[y|\boldsymbol{x}]$ and plot the corresponding curve (recall the relevant theory concerning the normal distribution case).

(b) Generate 100 data sets $D_i$, $i = 1, \dots 100$, each one consisting of $N = 50$ randomly selected pairs $(y_n, x_n)$, $n$=1,…,$N$, from $p(y, x)$.

(c) Adopt a linear estimator $f(x; D)$ and determine its instances $f(x; D_1), \dots, f(x; D_{100})$, utilizing the LS criterion.

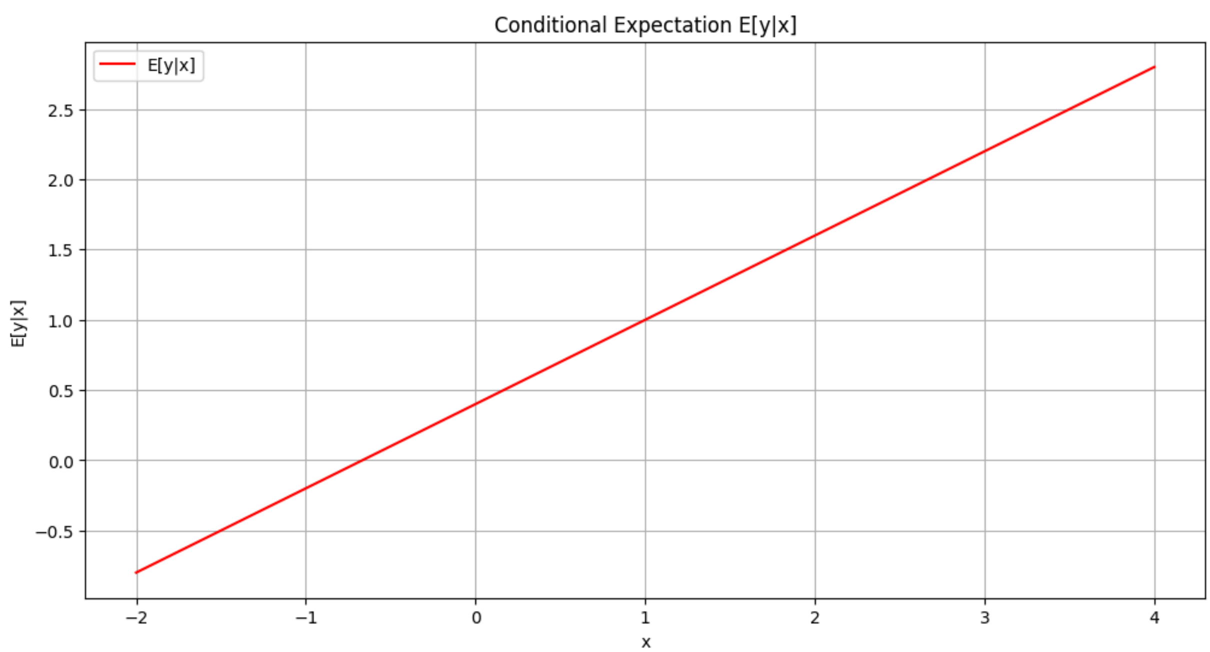
(d) Plot in a single figure **(i)** the lines corresponding to the above 100 estimates (blue color) and **(ii)** the line corresponding to the optimal MSE estimate (green color).

(e) Repeat steps (b)-(d) where now each data set consists of $N = 5000$ points.

(f) Discuss the results (in your discussion, take into account the decomposition of the MSE to a variance and a bias term).

In [2]:

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

# Parameters
mu = np.array([1, 1])
sigma = np.array([[4, 3], [3, 5]])
N_50 = 50
N_5000 = 5000
num_datasets = 100

# (a) E[y|x] and plot the corresponding curve
def conditional_expectation(x):
    mu_y_x = mu[0] + sigma[0, 1] / sigma[1, 1] * (x - mu[1])
    return mu_y_x

x_values = np.linspace(-2, 4, 100)
y_values = conditional_expectation(x_values)

plt.figure(figsize=(12, 6))
plt.plot(x_values, y_values, label='E[y|x]', color='red')
plt.title("Conditional Expectation E[y|x]")
plt.xlabel("x")
plt.ylabel("E[y|x]")
plt.grid()
plt.legend()
plt.show()
```




In [3]:

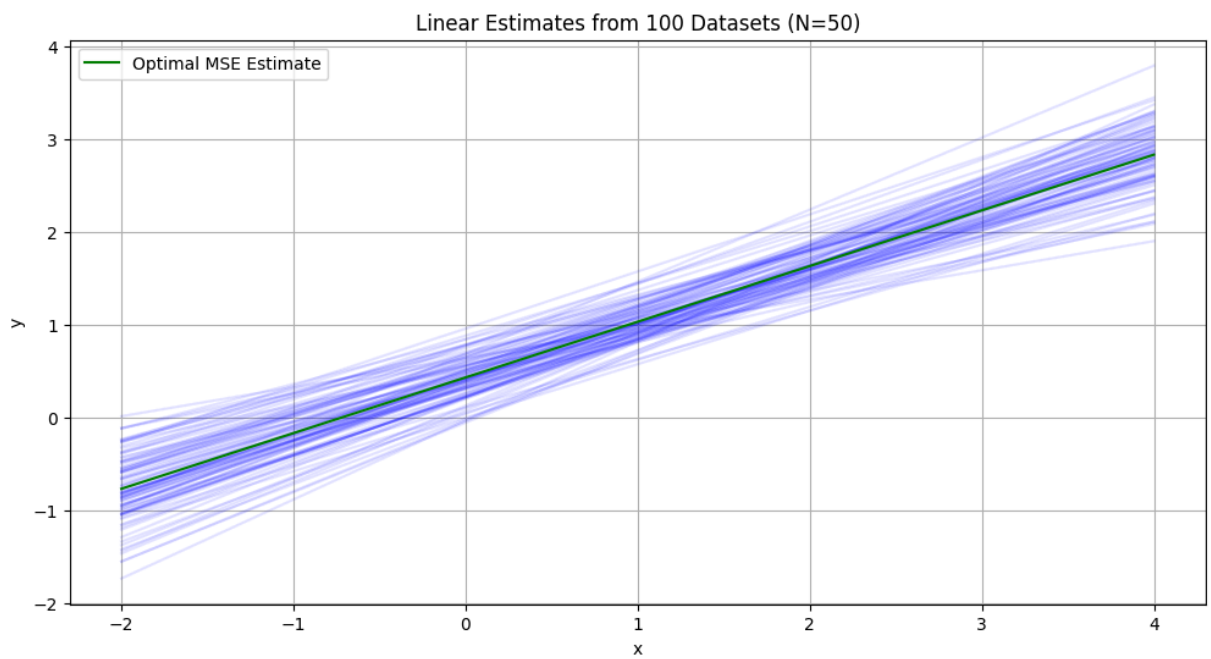
```
# (b) Generate 100 datasets D_i each consisting of N=50 pairs (y_n, x_n)
datasets_50 = []
for _ in range(num_datasets):
    data = np.random.multivariate_normal(mu, sigma, N_50)
    datasets_50.append(data)
```

In [4]:

```python
# (c) Linear estimator f(x; D) using LS criterion
def linear_estimator(data):
    x = data[:, 1]
    y = data[:, 0]
    X = np.vstack([np.ones(len(x)), x]).T
    beta_hat = np.linalg.inv(X.T @ X) @ (X.T @ y)
    return beta_hat

estimators_50 = [linear_estimator(data) for data in datasets_50]
```

In [5]:

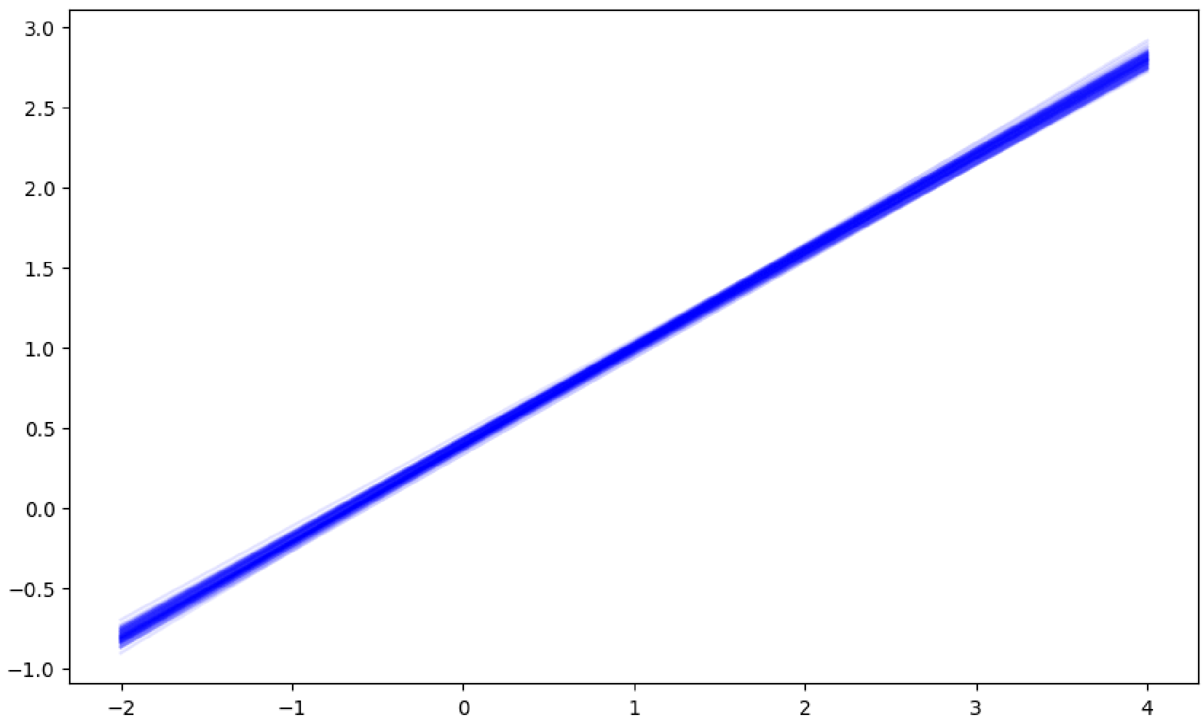
```python
# (d) Plot the estimates and optimal MSE estimate
plt.figure(figsize=(12, 6))
for beta_hat in estimators_50:
    x_fit = np.linspace(-2, 4, 100)
    y_fit = beta_hat[0] + beta_hat[1] * x_fit
    plt.plot(x_fit, y_fit, color='blue', alpha=0.1)

# Optimal MSE estimate
beta_optimal = linear_estimator(np.vstack(datasets_50))
x_fit = np.linspace(-2, 4, 100)
y_fit_optimal = beta_optimal[0] + beta_optimal[1] * x_fit
plt.plot(x_fit, y_fit_optimal, color='green', label='Optimal MSE Estimate')
plt.title("Linear Estimates from 100 Datasets (N=50)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```

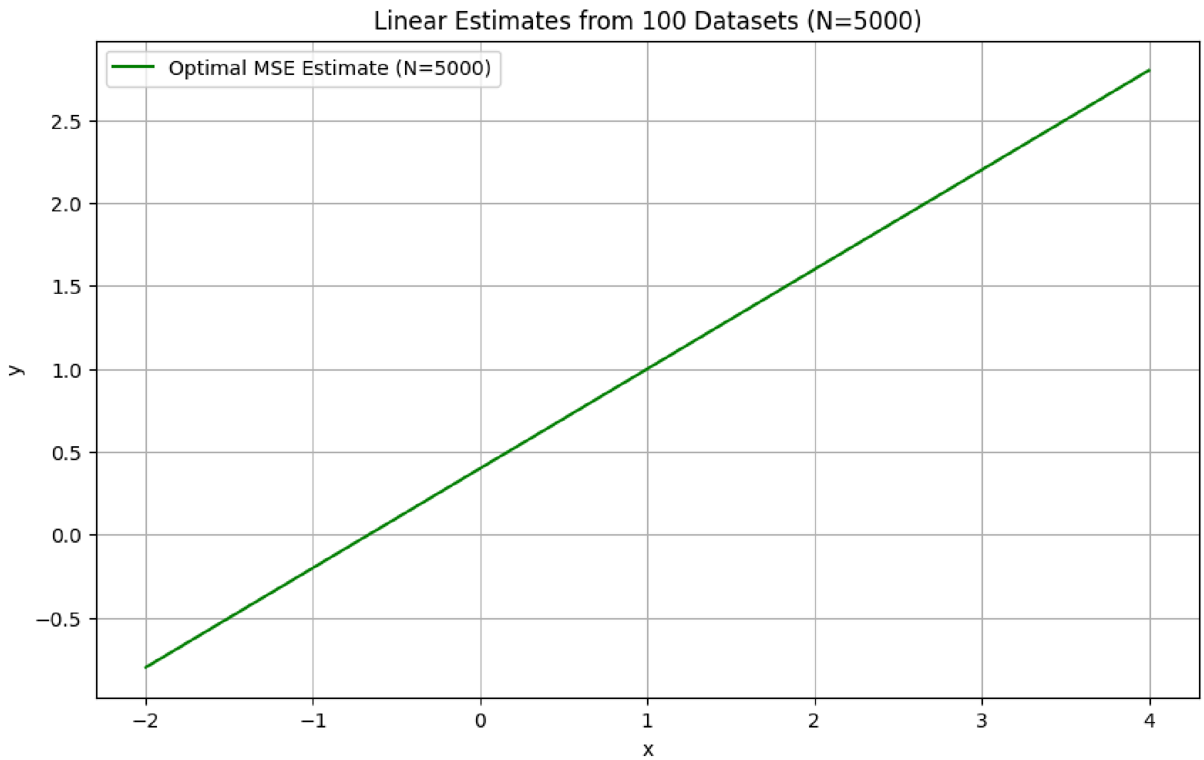



In [7]:

```python
# (e) Repeat steps (b)-(d) for N=5000 points
datasets_5000 = []
for _ in range(num_datasets):
    data = np.random.multivariate_normal(mu, sigma, N_5000)
    datasets_5000.append(data)
```

```python
estimators_5000 = [linear_estimator(data) for data in datasets_5000]

# Plotting for N=5000
plt.figure(figsize=(10, 6))
for beta_hat in estimators_5000:
    x_fit = np.linspace(-2, 4, 100)
    y_fit = beta_hat[0] + beta_hat[1] * x_fit
    plt.plot(x_fit, y_fit, color='blue', alpha=0.1)

# Optimal MSE estimate
beta_optimal_5000 = linear_estimator(np.vstack(datasets_5000))
x_fit = np.linspace(-2, 4, 100)
y_fit_optimal_5000 = beta_optimal_5000[0] + beta_optimal_5000[1] * x_fit
plt.figure(figsize=(10, 6))
plt.plot(x_fit, y_fit_optimal_5000, color='green', label='Optimal MSE Estimate (N=5
plt.title("Linear Estimates from 100 Datasets (N=5000)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```






The figures illustrate that increasing the sample size N from 50 to 5000 leads to a notable reduction in the variance of the estimator. Additionally, since the model for f(.) aligns with that of the optimal regressor, the term $(E[f(x;D)] - E[y|x])^2$ also decreases.

**Exercise 23 (python code + text):**

Consider the set-up of exercise 21 and recall the $E[y|\boldsymbol{x}]$ determined there.

(a) Generate a single data set $D$ of 100 pairs $(y_n, x_n)$, $n = 1, \dots, 100$ from $p(y, x)$.

(b) Determine the linear estimate $f(x; D)$ that minimizes the MSE criterion, based on $D$.

(c) Generate randomly a set $D'$ of additional 50 points $(y'_n, x'_n)$, $n = 1, \dots, 50$. For each $x'_n$ determine the estimate $y'_n = f(x_n; D')$ (50 numbers (estimates) should be finally computed).

(d) Again, for the 50 $x'_n$ 's determine the associated estimates $\hat{y} = E[y|\boldsymbol{x}]$.

(e) Based on the previous derived estimates for the 50 points from both $f(x_n; D)$ and $E[y|\boldsymbol{x}]$, propose and use a (practical) way for quantifying the performance of the two estimators $f(x_n; D')$ and $E[y|\boldsymbol{x}]$.

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt

# Step (a): Generate a single dataset D of 100 pairs (y_n, x_n)
def generate_data(num_samples):
    x = np.random.uniform(0, 1, num_samples)  # Uniform distribution for x
    y = np.random.uniform(x**3, 1, num_samples)  # y in (x^3, 1)
    return x, y
```

```
# Generate dataset D
D_x, D_y = generate_data(100)
```

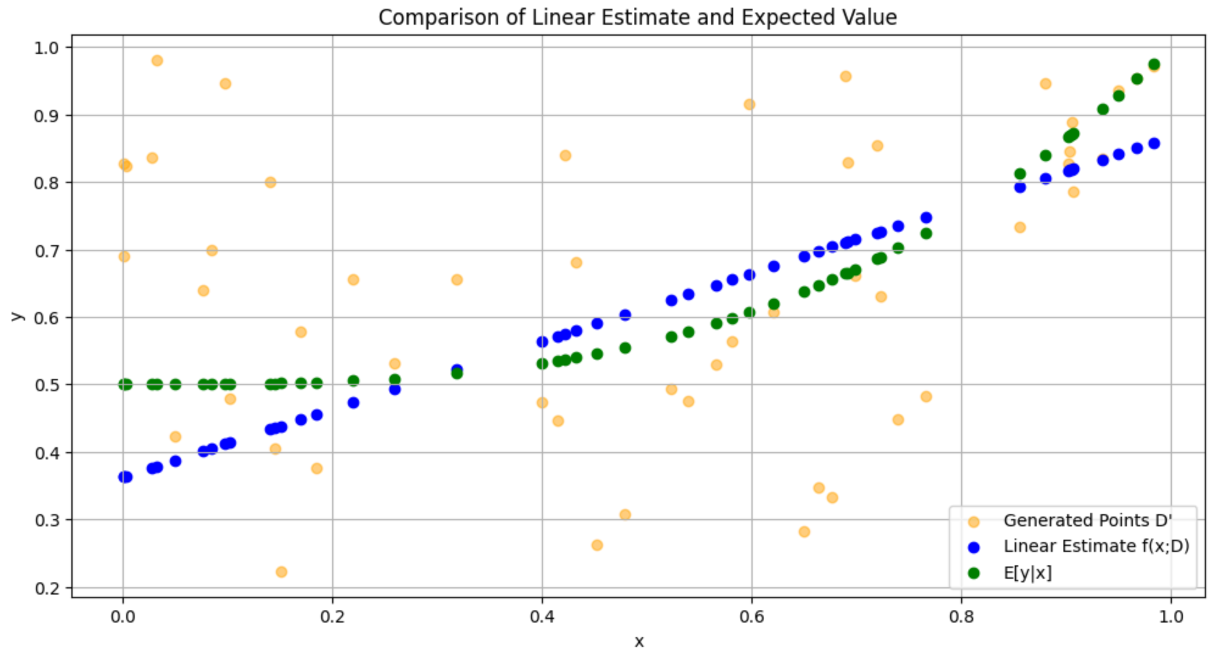
In [13]:
```
# Step (b): Determine the linear estimate f(x;D) that minimizes MSE
A = np.vstack([D_x, np.ones(len(D_x))]).T  # Design matrix for linear regression
m, c = np.linalg.lstsq(A, D_y, rcond=None)[0]  # Linear regression coefficients

# Define the linear estimate function
def linear_estimate(x):
    return m * x + c
```

In [14]:
```
# Step (c): Generate randomly a set D’ of additional 50 points
D_prime_x = np.random.uniform(0, 1, 50)
D_prime_y = np.random.uniform(D_prime_x**3, 1, 50)

# Compute estimates for D’ using the linear model
D_prime_estimates = linear_estimate(D_prime_x)
```

In [15]:
```
# Step (d): Calculate associated estimates ŷ = E[y|x]
def expected_value_y_given_x(x):
    return (1 + x**3) / 2

# Compute E[y|x] for D’ x values
E_y_given_x = expected_value_y_given_x(D_prime_x)
```

In [16]:
```
# Step (e): Quantify the performance using MSE
mse_linear = np.mean((D_prime_y - D_prime_estimates) ** 2)
mse_expected = np.mean((D_prime_y - E_y_given_x) ** 2)

# Output MSE results
print(f'MSE for Linear Estimate: {mse_linear}')
print(f'MSE for Expected Value Estimate: {mse_expected}')

# Plotting results
plt.figure(figsize=(12, 6))
plt.scatter(D_prime_x, D_prime_y, label="Generated Points D'", color='orange', alph
plt.scatter(D_prime_x, D_prime_estimates, label='Linear Estimate f(x;D)', color='bl
plt.scatter(D_prime_x, E_y_given_x, label='E[y|x]', color='green')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Comparison of Linear Estimate and Expected Value')
plt.legend()
plt.grid()
plt.show()
```

```
MSE for Linear Estimate: 0.059488868120379686
MSE for Expected Value Estimate: 0.041930267349554275
```




**<u>Exercise 24</u>** <u>(python code + text):</u> Consider the setup of exercise 21. Generate a set $D$ of $N = 100$ data pairs $\boldsymbol{z}_n = (y_n, x_n)$.

(a) For each $x_n$ compute the optimal MSE estimate (use the results of exercise 3).

(b) Compute $\boldsymbol{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} x_n \\ \frac{1}{N}\sum_{n=1}^{N} y_n \end{bmatrix}$ and $\Sigma = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{\mu} - \boldsymbol{z}_n)(\boldsymbol{\mu} - \boldsymbol{z}_n)^T$.

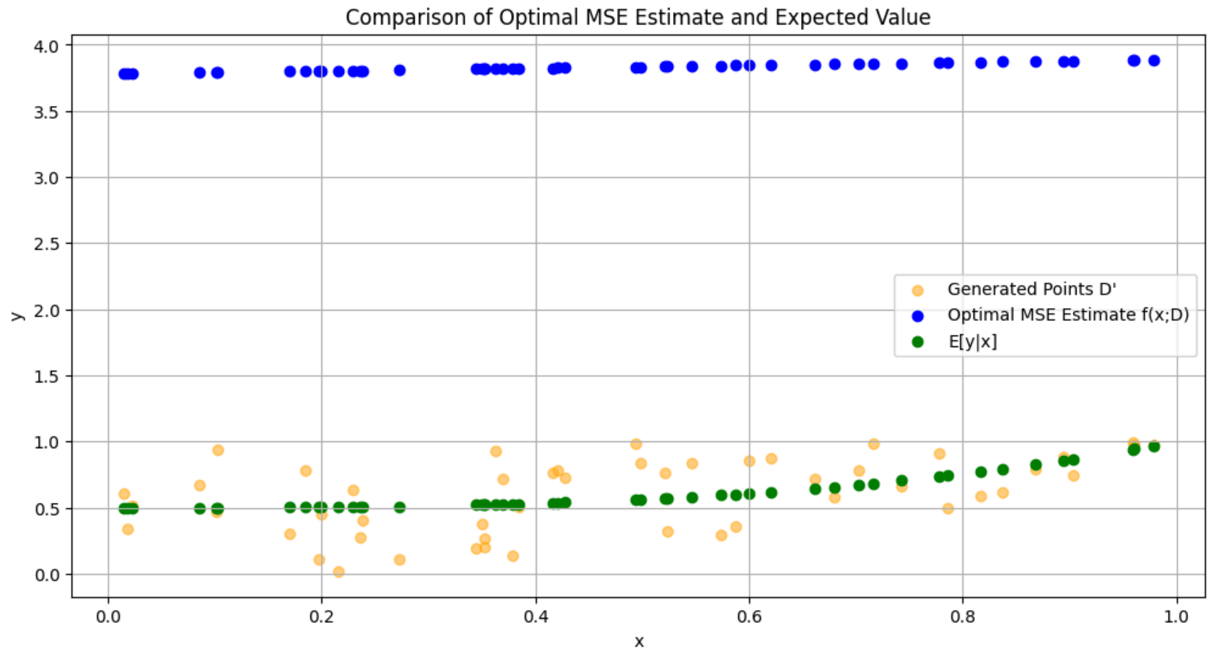
(c) Pretend that you do not know the true distribution that generates the data and you (erroneously) assume that the joint pdf of x and y is a normal one with mean and covariance matrix those computed in (b). Derive the optimum MSE estimate for this case and compute the MSE estimate for each one of the 100 $x_n$ 's.

Discuss the results obtained from (a) and (c).

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

# Step (a): Generate a single dataset D of 100 pairs (y_n, x_n)
def generate_data(num_samples):
    x = np.random.uniform(0, 1, num_samples)  # Uniform distribution for x
    y = np.random.uniform(x**3, 1, num_samples)  # y in (x^3, 1)
    return x, y

# Generate dataset D
D_x, D_y = generate_data(100)

# Step (b): Define the optimal MSE estimate for g(x) based on Exercise 3 results
mu_y, mu_x = 4, 2
sigma_x2, sigma_yx, sigma_y2 = 9, 1, 4

# Optimal MSE estimate function for E[y|x] using the Gaussian assumptions from Exer
def optimal_mse_estimate(x):
```

```python
    return mu_y + (sigma_yx / sigma_x2) * (x - mu_x)

# Compute the optimal MSE estimate for each x in D
D_estimates = optimal_mse_estimate(D_x)

# Step (c): Generate randomly a set D’ of additional 50 points
D_prime_x, D_prime_y = generate_data(50)

# Compute optimal MSE estimates for D’ using the optimal MSE function
D_prime_estimates = optimal_mse_estimate(D_prime_x)

# Step (d): Calculate the associated estimates E[y|x] directly
def expected_value_y_given_x(x):
    return (1 + x**3) / 2

# Compute E[y|x] for D’ x values
E_y_given_x = expected_value_y_given_x(D_prime_x)

# Step (e): Quantify the performance using MSE
mse_optimal = np.mean((D_prime_y - D_prime_estimates) ** 2)
mse_expected = np.mean((D_prime_y - E_y_given_x) ** 2)

# Output MSE results
print(f'MSE for Optimal MSE Estimate: {mse_optimal}')
print(f'MSE for Expected Value Estimate: {mse_expected}')

# Plotting results
plt.figure(figsize=(12, 6))
plt.scatter(D_prime_x, D_prime_y, label="Generated Points D'", color='orange', alph
plt.scatter(D_prime_x, D_prime_estimates, label='Optimal MSE Estimate f(x;D)', colo
plt.scatter(D_prime_x, E_y_given_x, label='E[y|x]', color='green')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Comparison of Optimal MSE Estimate and Expected Value')
plt.legend()
plt.grid()
plt.show()
```

```
MSE for Optimal MSE Estimate: 10.491983457794731
MSE for Expected Value Estimate: 0.055783198966685814
```



Comparison of Optimal MSE Estimate and Expected Value
Generated Points D'
Optimal MSE Estimate f(x;D)
E[y|x]
y
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0


In [ ]: