Started 'imessageEnv (Python 3.8.11)' kernel

Python 3.8.11 (default, Aug 6 2021, 08:56:27)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.26.0 -- An enhanced Interactive Python. Type '?' for help.

In [ ]:

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

def get_a(deg_true):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g

    Returns:
    a: (np array of size (deg_true + 1)) coefficients of polynomial g
    """
    return 5 * np.random.randn(deg_true + 1)

def get_design_mat(x, deg):
    """
    Inputs:
    x: (np.array of size N)
    deg: (int) max degree used to generate the design matrix

    Returns:
    X: (np.array of size N x (deg_true + 1)) design matrix
    """
    X = np.array([x ** i for i in range(deg + 1)]).T
    return X

def draw_sample(deg_true, a, N):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g
    a: (np.array of size deg_true) parameter of g
    N: (int) size of sample to draw

    Returns:
    x: (np.array of size N)
    y: (np.array of size N)
    """
    x = np.sort(np.random.rand(N))
    X = get_design_mat(x, deg_true)
    y = X @ a
    return x, y

def draw_sample_with_noise(deg_true, a, N):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g
    a: (np.array of size deg_true) parameter of g
    N: (int) size of sample to draw

    Returns:
    x: (np.array of size N)
    y: (np.array of size N)
    """
    x = np.sort(np.random.rand(N))
    X = get_design_mat(x, deg_true)
    y = X @ a + np.random.randn(N)
    return x, y
```

In [ ]:

```
######################
### PRE PROCESSING ###
######################
a = get_a(2)

x_train, y_train = draw_sample(2, a, 10)
x_test, y_test = draw_sample(2, a, 100)
```

In [ ]:

```
#####################
### PROBLEM SEVEN ###
#####################

def least_square_estimator(X, y):
    # Get rows (N) and columns (d)
    N = X.shape[0]
    d = X.shape[1]

    if d > N:
        print('N must be greater or equal to d')
        return None

    # Otherwise, compute b = (XTX)-1 (XTy)
    XTX_inv = np.linalg.inv(X.T @ X)
    XTy = (X.T) @ y

    b_estimate = XTX_inv @ XTy
    return b_estimate
```

In [ ]:

```
#####################
### PROBLEM EIGHT ###
#####################

def empirical_risk(X, y, b):

    # Get estimated values for y

    y_est = X @ b
    # Compute differences per the L2 Norm
    sq_differences = (y_est - y) ** 2
    sum_sq_differences = np.sum(sq_differences)

    # Compute emp risk
    emp_risk = sum_sq_differences / len(y_est)

    return emp_risk
```

In [ ]:

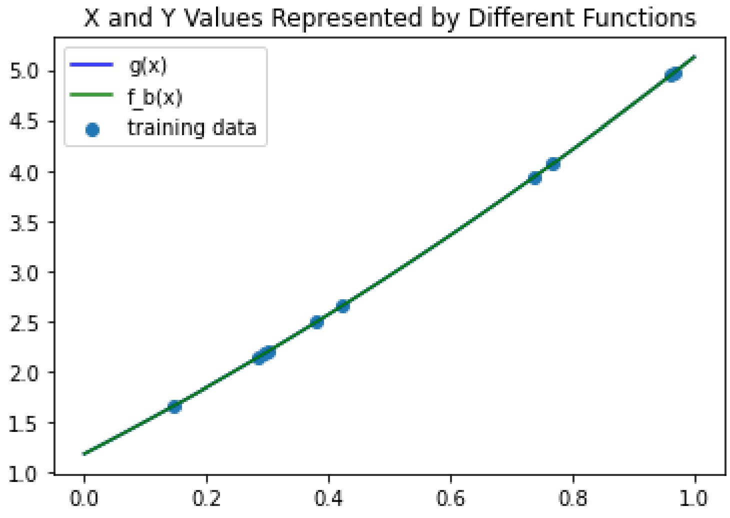
```
####################
### PROBLEM NINE ###
####################
```

```
x_train_9 = get_design_mat(x_train, 2)
x_test_9 = get_design_mat(x_test, 2)

b_est = least_square_estimator(x_train_9, y_train)

print('COMPARISON OF estimated b value and a')
print('estimated b vector: ', b_est)
print('true a vector: ', a)
```

```
COMPARISON OF estimated b value and a
estimated b vector:  [1.18754388 3.12748096 0.81033551]
true a vector:  [1.18754388 3.12748096 0.81033551]
```

In [ ]:

```
x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    fb_x.append(fb_val)

plt.figure(0)
plt.title('X and Y Values Represented by Different Functions')
plt.scatter(x_train, y_train)
plt.plot(x, fb_x, 'b', label = 'f_b(x)')
plt.plot(x, g_x, 'g')
plt.legend(labels=['g(x)', 'f_b(x)', 'training data'])
plt.show()
```



In [ ]:

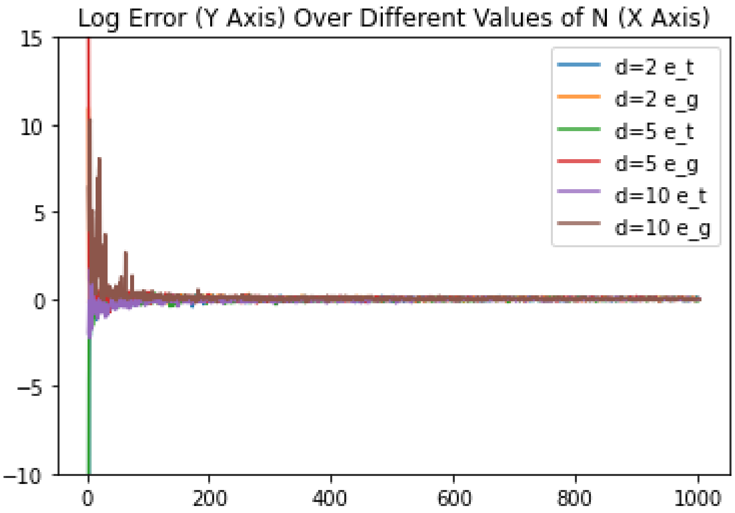
```
###################
### PROBLEM TEN ###
###################

for i in range(1,5):
    d = i

    x_train_10 = get_design_mat(x_train, d)
    x_test_10 = get_design_mat(x_test, d)
    b_est = least_square_estimator(x_train_10, y_train)

    print('d = ', i, ':', empirical_risk(x_test_10, y_test, b_est))

print('The minimum value at which we can get a near perfect fit is d = 2')
```

```
d =  1 : 0.005885449557118774
d =  2 : 2.741050056990792e-28
d =  3 : 6.579271891145255e-24
d =  4 : 2.0329553307404025e-20
The minimum value at which we can get a near perfect fit is d = 2
```

In [ ]:

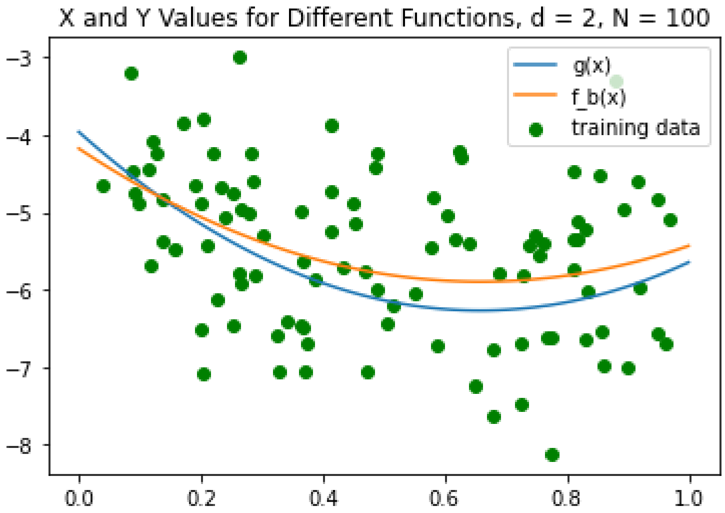
```
######################
### PROBLEM ELEVEN ###
######################

###################
## PLOT TWO MAIN ##
###################

### D = 2 ###

N_2 = []
et_2 = []
eg_2 = []
for i in range(3,1003):
    d = 2
    a = get_a(d)

    x_train11, y_train11 = draw_sample_with_noise(d, a, i)
    x_train11 = get_design_mat(x_train11, d)
    x_test11, y_test11 = draw_sample_with_noise(d, a, i)
    x_test11 = get_design_mat(x_test11, d)

    b_est = least_square_estimator(x_train11, y_train11)

    e_t = np.log(empirical_risk(x_train11, y_train11, b_est))
    e_g = np.log(empirical_risk(x_test11, y_test11, b_est))

    N_2.append(i)
    et_2.append(e_t)
    eg_2.append(e_g)

plt.plot(N_2, et_2)
plt.plot(N_2, eg_2)

### D = 5 ###

N_5 = []
et_5 = []
eg_5 = []
for i in range(6,1006):
    d = 5
    a = get_a(d)

    x_train11, y_train11 = draw_sample_with_noise(d, a, i)
    x_train11 = get_design_mat(x_train11, d)
    x_test11, y_test11 = draw_sample_with_noise(d, a, i)
    x_test11 = get_design_mat(x_test11, d)
    b_est = least_square_estimator(x_train11, y_train11)

    e_t = np.log(empirical_risk(x_train11, y_train11, b_est))
    e_g = np.log(empirical_risk(x_test11, y_test11, b_est))

    N_5.append(i)
    et_5.append(e_t)
    eg_5.append(e_g)

plt.plot(N_2, et_5)
plt.plot(N_2, eg_5)

### D = 10 ###

N_10 = []
et_10 = []
eg_10 = []
for i in range(11,1011):
    d = 10
    a = get_a(d)

    x_train11, y_train11 = draw_sample_with_noise(d, a, i)
    x_train11 = get_design_mat(x_train11, d)
    x_test11, y_test11 = draw_sample_with_noise(d, a, i)
    x_test11 = get_design_mat(x_test11, d)

    b_est = least_square_estimator(x_train11, y_train11)

    e_t = np.log(empirical_risk(x_train11, y_train11, b_est))
    e_g = np.log(empirical_risk(x_test11, y_test11, b_est))

    N_10.append(i)
    et_10.append(e_t)
    eg_10.append(e_g)

plt.plot(N_2, et_10)
plt.plot(N_2, eg_10)

ax = plt.gca()
ax.set_ylim([-10, 15])

plt.title('Log Error (Y Axis) Over Different Values of N (X Axis)')
plt.legend(labels=['d=2 e_t','d=2 e_g','d=5 e_t', 'd=5 e_g', 'd=10 e_t','d=10 e_
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7fad0831a910>
```



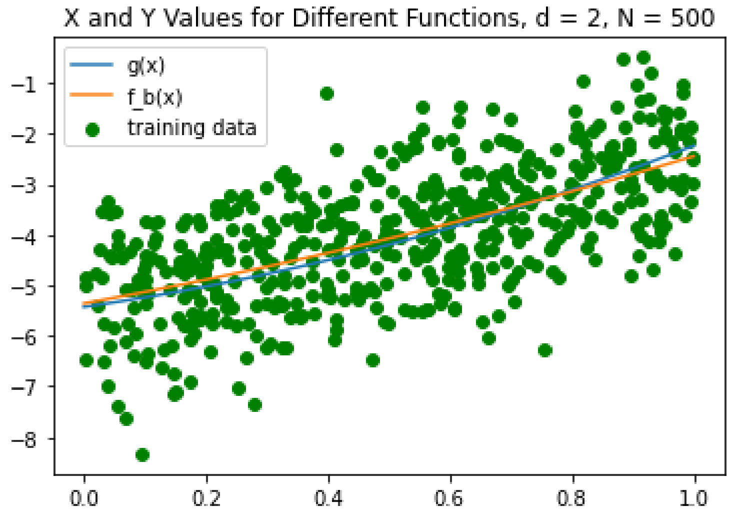
In [ ]:

```
###################
## PLOT 2, d = 2 ##
###################

## N = 100

d = 2
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 100)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 100)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    fb_x.append(fb_val)

plt.figure(1)
plt.title('X and Y Values for Different Functions, d = 2, N = 100')
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.show()
```



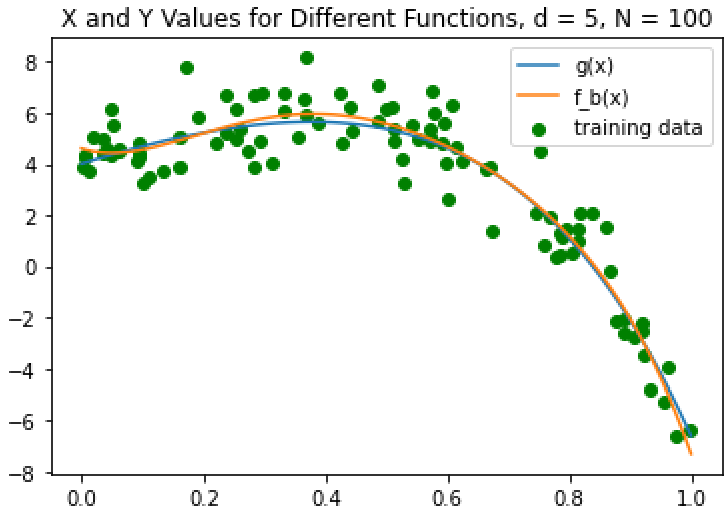
In [ ]:

```
## N = 500

d = 2
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 500)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 500)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2])
    fb_x.append(fb_val)

plt.figure(2)
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.title('X and Y Values for Different Functions, d = 2, N = 500')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.show()
```



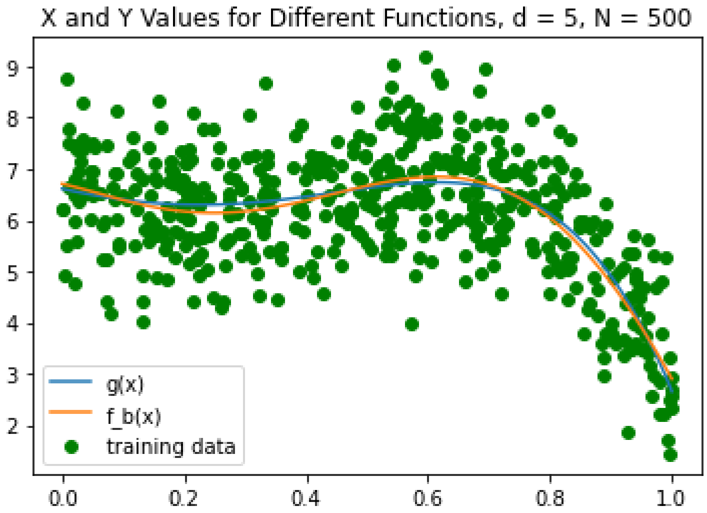
In [ ]:

```
###################
## PLOT 2, d = 5 ##
###################

## N = 100

d = 5
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 100)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 100)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i] ** 4,
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i]
    fb_x.append(fb_val)

plt.figure(3)
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.title('X and Y Values for Different Functions, d = 5, N = 100')
plt.show()
```



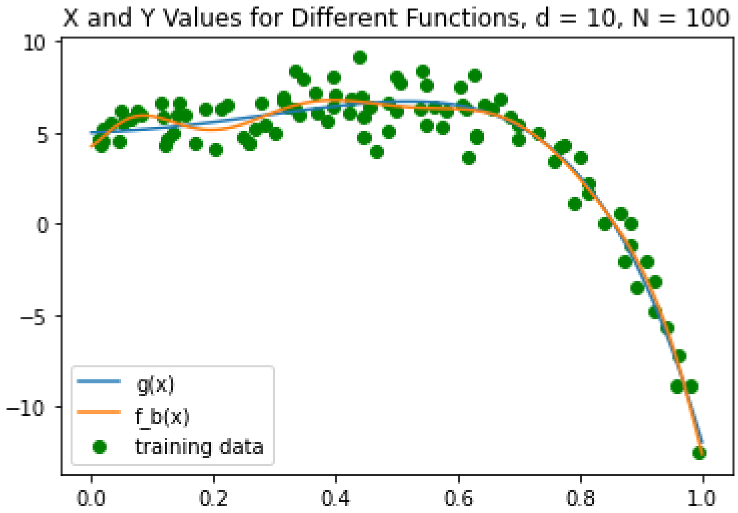
In [ ]:

```
## N = 500

d = 5
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 500)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 500)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i] ** 4,
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i]
    fb_x.append(fb_val)

plt.figure(4)
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.title('X and Y Values for Different Functions, d = 5, N = 500')
plt.show()
```



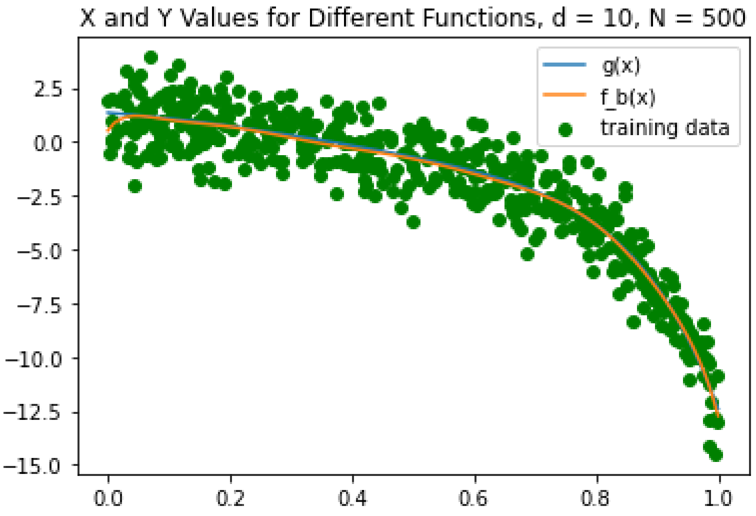
In [ ]:

```
####################
## PLOT 2, d = 10 ##
####################

## N = 100

d = 10
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 100)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 100)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i] ** 4,
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i]
    fb_x.append(fb_val)

plt.figure(5)
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.title('X and Y Values for Different Functions, d = 10, N = 100')
plt.show()
```



In [ ]:

```
## N = 500

d = 10
a = get_a(d)

x_train11, y_train11 = draw_sample_with_noise(d, a, 500)
x_train11_mat = get_design_mat(x_train11, d)
x_test11, y_test11 = draw_sample_with_noise(d, a, 500)
x_test11 = get_design_mat(x_test11, d)

b_est = least_square_estimator(x_train11_mat, y_train11)

x = np.linspace(0, 1, num=100)
g_x = []
fb_x = []
for i in range(100):
    g_val = a @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i] ** 4,
    g_x.append(g_val)

    fb_val = b_est @ np.array([x[i] ** 0, x[i] ** 1, x[i] ** 2, x[i] ** 3, x[i]
    fb_x.append(fb_val)

plt.figure(5)
plt.plot(x, g_x)
plt.plot(x, fb_x)
plt.scatter(x_train11, y_train11, color = 'green')
plt.legend(['g(x)', 'f_b(x)', 'training data'])
plt.title('X and Y Values for Different Functions, d = 10, N = 500')
plt.show()
```

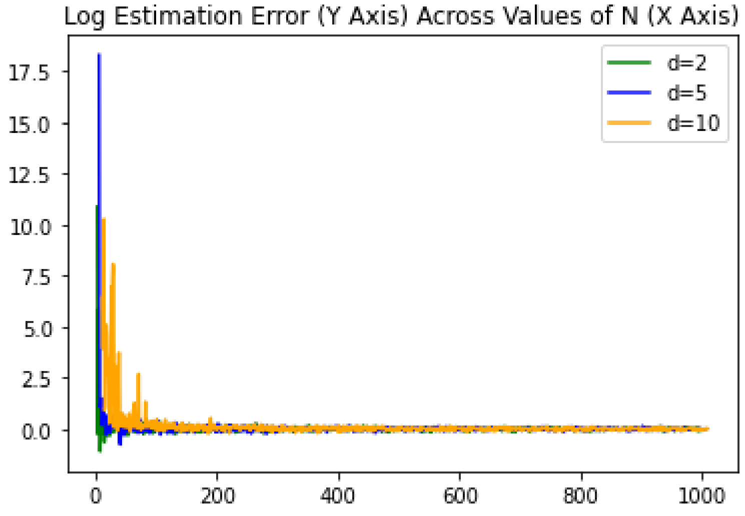


In [ ]:

```
######################
### PROBLEM TWELVE ###
######################

plt.figure(6)
plt.title('Log Estimation Error (Y Axis) Across Values of N (X Axis)')
plt.plot(N_2, eg_2, color = 'green')
plt.plot(N_5, eg_5, color = 'blue')
plt.plot(N_10, eg_10, color = 'orange')
plt.legend(['d=2', 'd=5', 'd=10'])
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7fad18bf0f10>
```



In [ ]:

```
########################
### PROBLEM THIRTEEN ###
########################

'''It appears that raising N allows the estimation error to close in on 0. This
as more data is available to train on, the estimation parameters reach the true
parameters. As such, we expect that increasing N decreases the estimation error
(or almost hits) 0.

It appears that increasing d, in this particular case, creates additional error
```

```
However, once N becomes sufficiently large, it seems that all three values of ul
converge on an error of 0.'''
```

Out[ ]:

```
'It appears that raising N allows the estimation error to close in on 0. This ma
kes sense because\nas more data is available to train on, the estimation paramet
ers reach the true population \nparameters. As such, we expect that increasing N
decreases the estimation error until it hits\n(or almost hits) 0. \n\nIt appears
that increasing d, in this particular case, creates additional error when N is s
mall. \nHowever, once N becomes sufficiently large, it seems that all three valu
es of ultimately\nconverge on an error of 0.'
```

In [ ]:

```
########################
### PROBLEM FOURTEEN ###
########################

'''Optimization error is defined as the difference between a given empirical ris
actual function returned by whatever computation we complete. In this case, as f
the function we have returned optimizes risk in the empirical sense. In other wo
b estimate should be the empirical risk minimizer. Therefore, optimization error
in this case. Given limitation of computing and the packages we used, there may 
is not noteworthy. '''
```

Out[ ]:

```
'Optimization error is defined as the difference between a given empirical risk
minimizer and the\nactual function returned by whatever computation we complete.
In this case, as far as we know,\nthe function we have returned optimizes risk i
n the empirical sense. In other words, our \nb estimate should be the empirical
risk minimizer. Therefore, optimization error should be negligible\nin this cas
e. Given limitation of computing and the packages we used, there may be some, bu
t it \nis not noteworthy. '
```