In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import itertools
import random
import statistics
```

In [ ]:
```
def kones(n, k):
    result = []
    for bits in itertools.combinations(range(n), k):
        s = [0] * n
        for bit in bits:
            s[bit] = 1
        result.append(s)
    return result
```

In [ ]:
```
def does_commute(op1, op2):
    op_sum = list(np.array(op1) + np.array(op2))
    count = op_sum.count(2)
    return count % 2 == 0
```

In [ ]:
```
def find_logical(stabilizers, error):
    for stab in stabilizers:
        if does_commute(error, stab) == False:
            return None
    return error
```

In [ ]:
```
z_stabilizers = [[1,0,0,0,0,1,0,0,0,0,0],
                 [0,1,0,0,0,1,0,0,0,0,0],
                 [0,0,1,0,0,1,0,0,0,0,0],
                 [0,0,0,1,0,1,0,0,0,0,0],
                 [0,0,0,0,1,1,0,0,0,0,0],
                 [0,0,0,0,0,0,1,0,0,0,1],
                 [0,0,0,0,0,0,0,1,0,0,1],
                 [0,0,0,0,0,0,0,0,1,0,1],
                 [0,0,0,0,0,0,0,0,0,1,1],
                 [0,0,0,0,0,0,0,0,0,0,0]]

x_stabilizers = [[0,1,0,1,0,0,0,1,1,0,0],
                 [0,1,0,1,1,1,0,0,1,1,0],
                 [0,1,0,0,1,0,0,0,0,1,1],
                 [0,1,1,1,0,1,1,0,0,1,0],
                 [0,0,1,1,0,0,1,1,0,1,1],
                 [0,0,1,1,0,0,1,1,1,0,1],
                 [0,1,0,1,1,1,1,1,0,1,1],
                 [0,0,0,1,0,1,0,1,0,1,0],
                 [0,1,0,1,0,0,1,0,1,0,0],
                 [1,1,1,1,1,1,0,0,0,0,0]]
```

In [ ]:
```
# find all potential x logicals
xlogical = []
distanceFound = False
for weight in range(2,8):
    errors = kones(11, weight)
    for error in errors:
        xlog = find_logical(z_stabilizers, error)
        if xlog != None:
            xlogical.append(error)
            if len(xlogical) == 1:
                print(f"distance = {weight}")
            break
```

```
distance = 5
```

In [ ]:
```
xlogical
```

Out[ ]:
```
[[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]]
```

In [ ]:
```
# find all z logicals
zlogical = []
distanceFound = False
for weight in range(2,8):
    errors = kones(11, weight)
    for error in errors:
        zlog = find_logical(x_stabilizers, error)
        if zlog != None:
            for xlog in xlogical:
                if does_commute(zlog, xlog) == False:
                    zlogical.append(zlog)
                    if len(zlogical) == 1:
                        print(f"distance = {weight}")
```

```
distance = 5
```

In [ ]:
```
# remove x logicals that don't anticommute with any z logicals
for xlog in xlogical:
    anticommutes = False
    for zlog in zlogical:
        if does_commute(xlog, zlog) == False:
            anticommutes = True
            break
    if anticommutes == False:
        xlogical.remove(xlog)
```

In [ ]:
```
print(f"Z Logical = {zlogical}")
print(f"X Logical = {xlogical}")
```

```
Z Logical = [[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]]
X Logical = [[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]]
```

In [ ]:
```
errors = []
for weight in range(0,11):
    errors = errors + kones(11, weight)
errors.append([1]*11)

# get all combinations of z and x errors
all_errors = []
for z_err in errors:
    for x_err in errors:
        all_errors.append((z_err, x_err))

syndrome_error_table = dict()
for z_err, x_err in all_errors:
    # get syndromes
    z_synd = [0 if does_commute(z_err, stab) else 1 for stab in x_stabilizers]
    x_synd = [0 if does_commute(x_err, stab) else 1 for stab in z_stabilizers]

    # add syndromes together
    syndrome = list((np.array(z_synd) + np.array(x_synd)) % 2)

    # find integer form of binary list
    syndrome_int = int("".join(str(x) for x in syndrome), 2)

    # add syndrome and error to table
    if syndrome_int not in syndrome_error_table:
        syndrome_error_table[syndrome_int] = (z_err, x_err)
```

In [ ]:
```
print(f"Number of possible errors = {len(all_errors)}")
print(f"Number of unique syndromes = {len(syndrome_error_table)}")
```

```
Number of possible errors = 4194304
Number of unique syndromes = 1024
```

In [ ]:
```
def simulatePerfectDecoding(noiseRate, trials):
    numDecodingErrors = 0
    for i in range(trials):
        z_error = [0]*11
        x_error = [0]*11
        for bit in range(11):
            r = random.random()
            if r <= noiseRate/3:
                x_error[bit] = 1
            elif r <= (2*noiseRate)/3:
                z_error[bit] = 1
            elif r <= noiseRate:
                x_error[bit] = 1
                z_error[bit] = 1
        # get syndrome
        x_synd = [0 if does_commute(x_error, stab) else 1 for stab in z_stabilizers
        z_synd = [0 if does_commute(z_error, stab) else 1 for stab in x_stabilizers
        syndrome = list((np.array(x_synd) + np.array(z_synd)) % 2)

        # find syndrome in syndrome_error_table
        syndrome_int = int("".join(str(x) for x in syndrome), 2)
        if syndrome_int not in syndrome_error_table:
            print("syndrome not found")
        x_corr, z_corr = syndrome_error_table[syndrome_int][1], syndrome_error_tabl

        # apply correction to error
        new_z = list((np.array(z_err) + np.array(z_corr)) % 2)
        new_x = list((np.array(x_err) + np.array(x_corr)) % 2)

        # check for logical failure
        logical_error = 1 if (does_commute(new_z, xlogical[0]) or does_commute(new_

        if logical_error == 1:
            numDecodingErrors += 1
    return numDecodingErrors
```

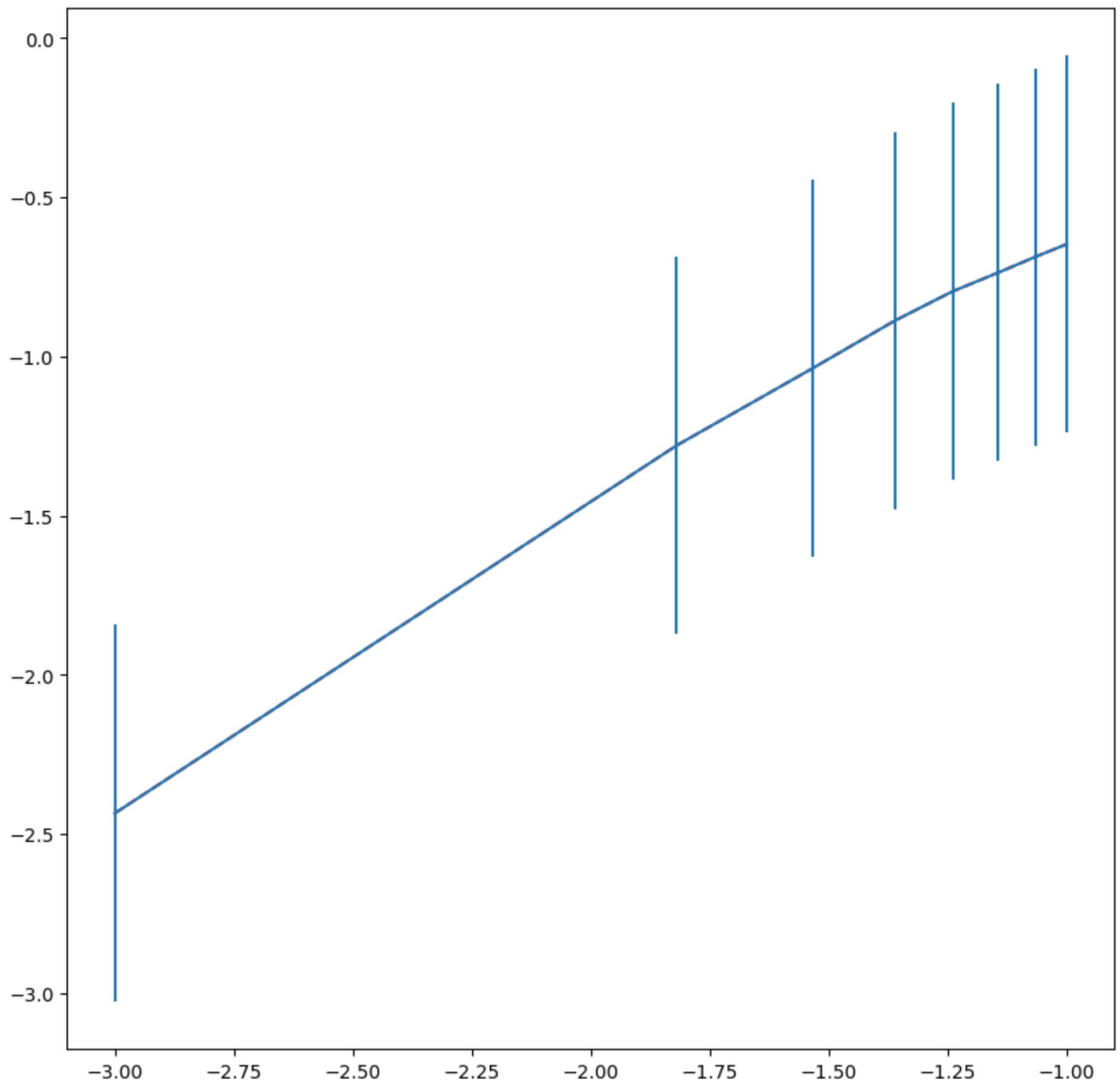
In [ ]:
```
noise_step = ((1e-1) - (1e-3))/7
noise_rates = list(np.arange(1e-3, 1e-1 + noise_step, noise_step))
trials = 10**6
data = []
for p in noise_rates:
    print(p)
    data.append([p, simulatePerfectDecoding(p, trials)/trials])
```

```
0.001
0.015142857142857142
0.029285714285714283
0.04342857142857143
0.057571428571428565
0.0717142857142857
0.08585714285714285
0.09999999999999999
```

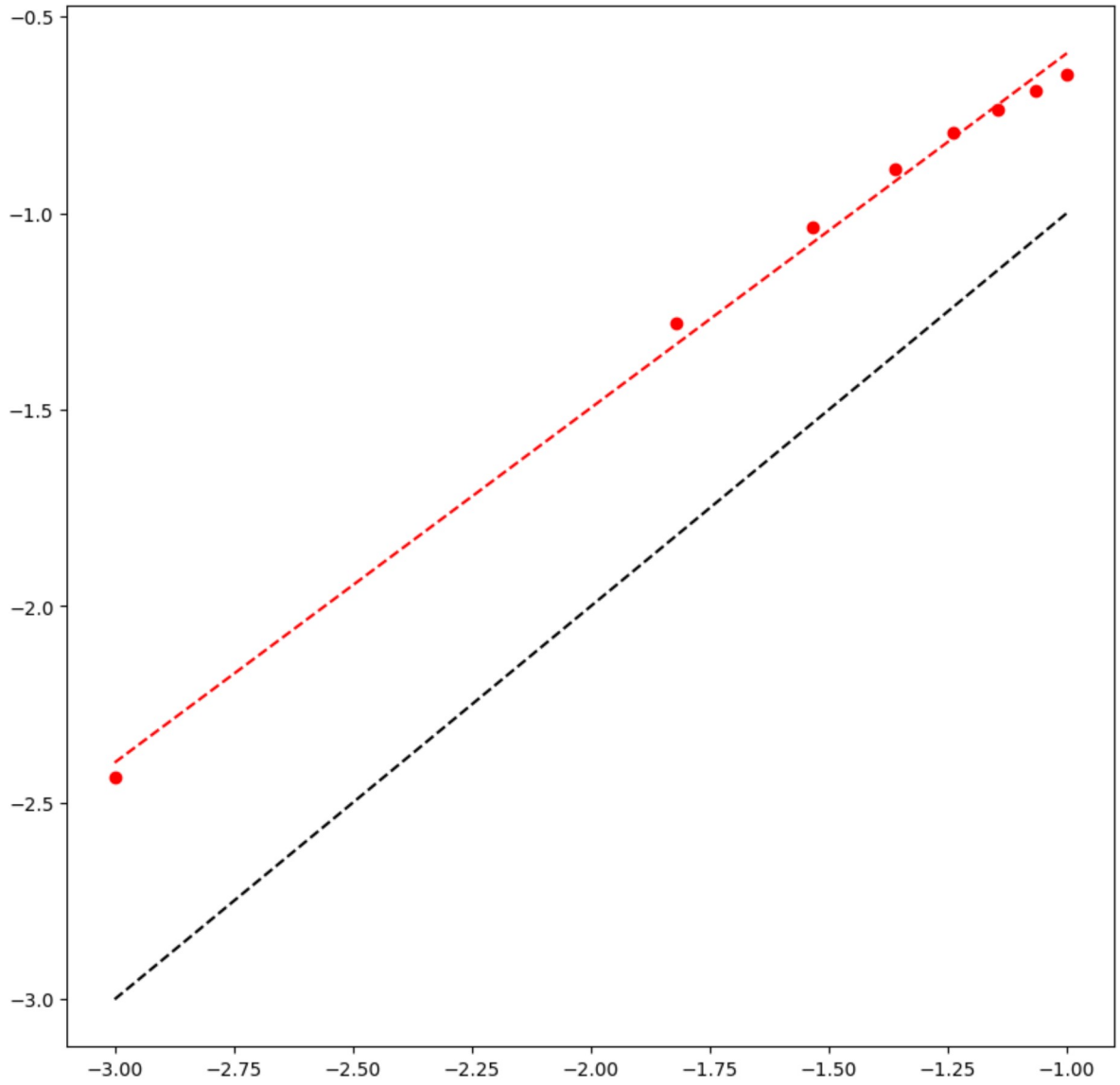
In [ ]:
```
np.array(data)
data = np.log10(data)
print(data)
```

```
[[-3.         -2.43533394]
 [-1.81979217 -1.2795926 ]
 [-1.53334418 -1.0364955 ]
 [-1.36222446 -0.88813285]
 [-1.23979299 -0.79531038]
 [-1.14439432 -0.73660067]
 [-1.06622357 -0.68706139]
 [-1.         -0.64685345]]
```

In [ ]:
```
plt.figure(figsize=(10,10))
plt.plot(data[:,0], data[:,1], 'r--')
plt.errorbar(data[:,0], data[:,1], statistics.stdev(data[:,1]))
plt.show()
```



In [ ]:
```
a,b = np.polyfit(data[:,0], data[:,1], 1)
plt.figure(figsize=(10,10))
plt.plot(data[:,0], data[:,0], 'k--',
        data[:,0], data[:,1], 'ro',
        data[:,0], a*data[:,0]+b, 'r--')
plt.show()
```



In [ ]:
```
# p = 10**(-3)
# trials = 10**5
# errors = simulatePerfectDecoding(p, trials)
# print(f"Errors = {errors}")
# print(f"Trials = {trials}")
# print(f"Error rate = {errors/trials}")
```

In [ ]: