VQEManySamplesExtrap(June 18)-Copy1

June 18, 2020

1 Does Noise Amplification affect VQE Computations?

For Extrapolation to be useful, this condition has to be met.

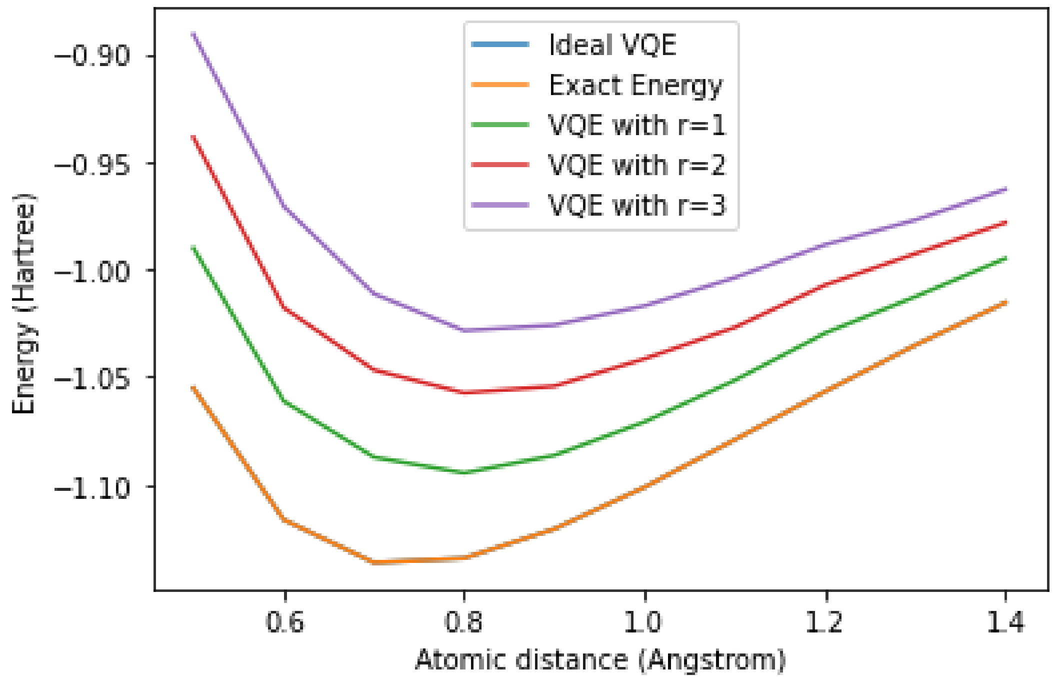
Running on a sample circuit

2 VQE with different noise scalings

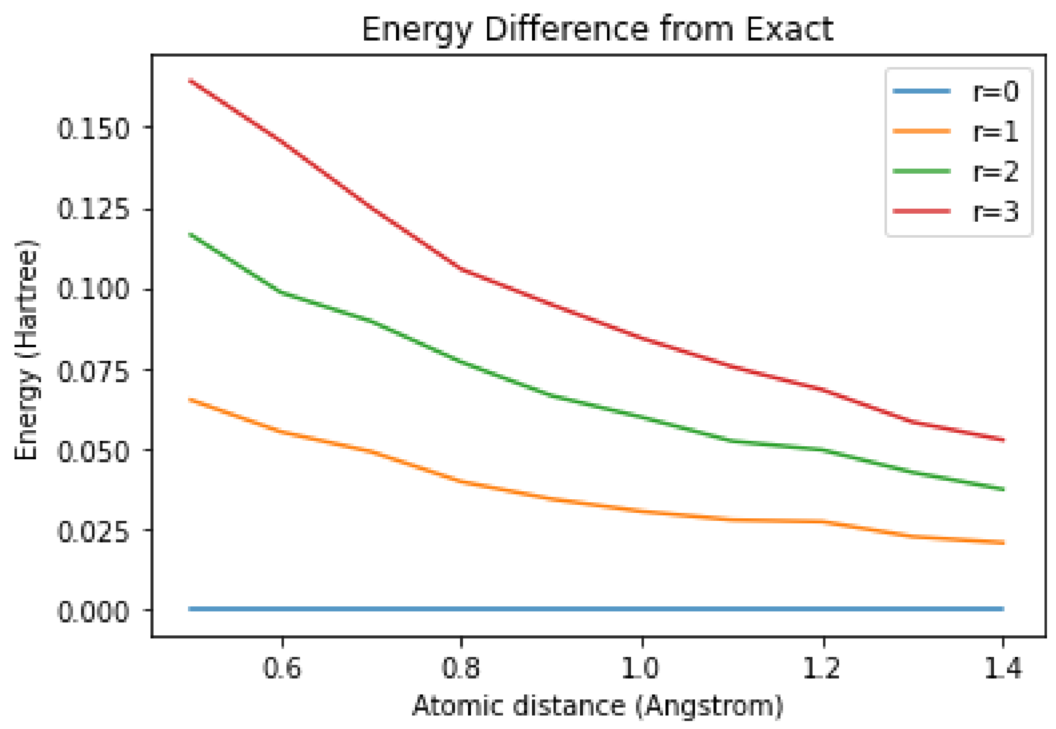
[13]:
```
from scipy import stats

#Getting rid of outliers
vqe_energies_o = []
for s in range(3):
    scale_energies_o = []
    for d in range(10):
        z = np.abs(stats.zscore(vqe_energies[s][d]))
        threshold = 2.5
        outlier_indices = np.where(z>2)[0]
        print(outlier_indices)
        outlier_free= []
        for k in range(len(vqe_energies[s][d])):
            if k not in outlier_indices:
                outlier_free.append(vqe_energies[s][d][k])
        scale_energies_o.append(outlier_free)
    vqe_energies_o.append(scale_energies_o)
#Getting the means
vqe_mean_energies = []
for s in range(3):
    scale_mean_energies = []
    for d in range(10):
        mean = np.mean(vqe_energies[s][d])
        scale_mean_energies.append(mean)
    vqe_mean_energies.append(scale_mean_energies)
#Getting means from exact and vqe_ideal
vqe_ideal = []
exact = []
for d in range(10):
    vqe_ideal.append(vqe_ideal_energies[d][0])
    exact.append(exact_energies[d][0])






# plotting the data
plt.plot(distances, vqe_ideal, label="Ideal VQE")
plt.plot(distances, exact, label="Exact Energy")

plt.plot(distances, vqe_mean_energies[0], label="VQE with r=1")
plt.plot(distances, vqe_mean_energies[1], label="VQE with r=2")
plt.plot(distances, vqe_mean_energies[2], label="VQE with r=3")
plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.legend()
plt.show()
```

```
[11 24]
[16 20 34]
[16 19 45]
[8]
[30 35 42]
[35 47]
[ 7 17 41]
[ 6 26]
[21 23]
[ 2 39]
[11 40]
[27 40]
[22 35]
[ 0  8 23]
[21 43]
[12 17]
[31 49]
[ 9 10 40]
[10 13]
[ 9 41 47]
[ 4 14 47]
[3]
[32 44]
[22 44]
[ 6 37]
[14 28 30 48]
[13 31 47]
[ 3  5 10 37]
```

```
[24 26]
[ 5 14]
```



[16]:
```
energy_difference= []
for i in range(4):
    energy_difference.append([])

for i in range(4):
    if (i ==0):
        for k in range(len(vqe_ideal)):
            energy_difference[i] = energy_difference[i] + [vqe_ideal[k] -␣
 ↪exact[k]]
    else:
        for k in range(len(vqe_ideal)):
            energy_difference[i] = energy_difference[i] +␣
 ↪[vqe_mean_energies[i-1][k] - exact[k]]


plt.plot(distances, energy_difference[0], label="r=0")
plt.plot(distances, energy_difference[1], label="r=1")
plt.plot(distances, energy_difference[2], label="r=2")
plt.plot(distances, energy_difference[3], label="r=3")
plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.title('Energy Difference from Exact')
plt.legend()
plt.show()
```



3 Analysis

The graph suggests that if we scale the depolarizing error higher and higher, our results tend to go away from the exact energy. So noise amplification diverges VQE estimate from actual answer.

4 To do next

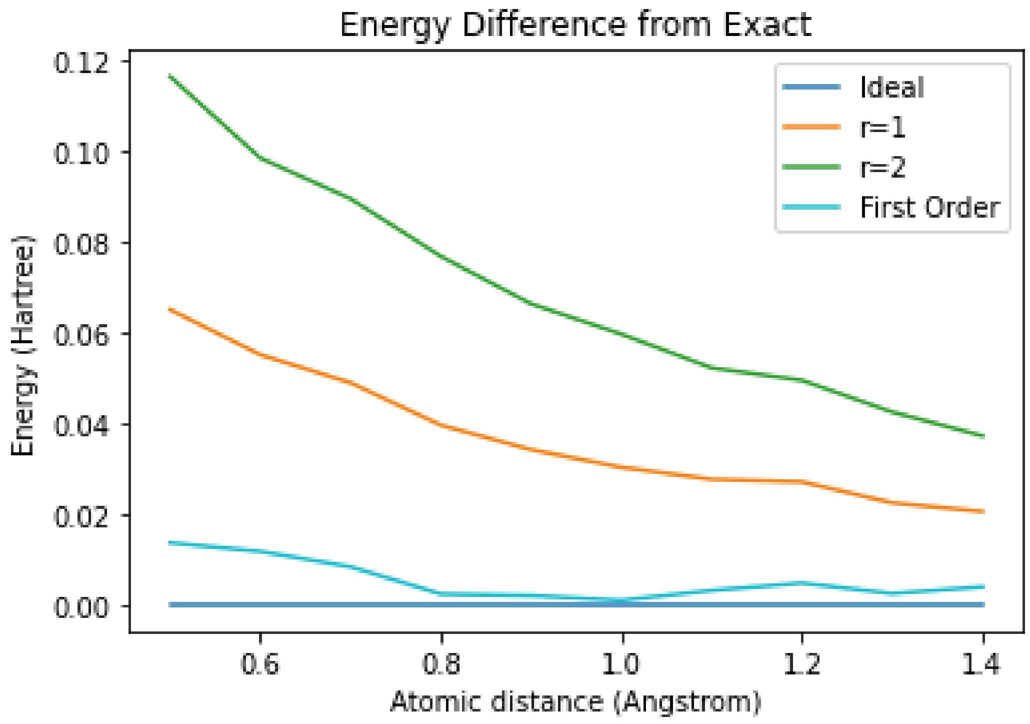
Extrapolate

5 Linear Extrapolation Using Curve Fitting

Instead of using the equations-method (which I used earlier and messed up) as shown in section 3 in supplemental materials of Temme Paperm, here I use a simple curve fitting technique.

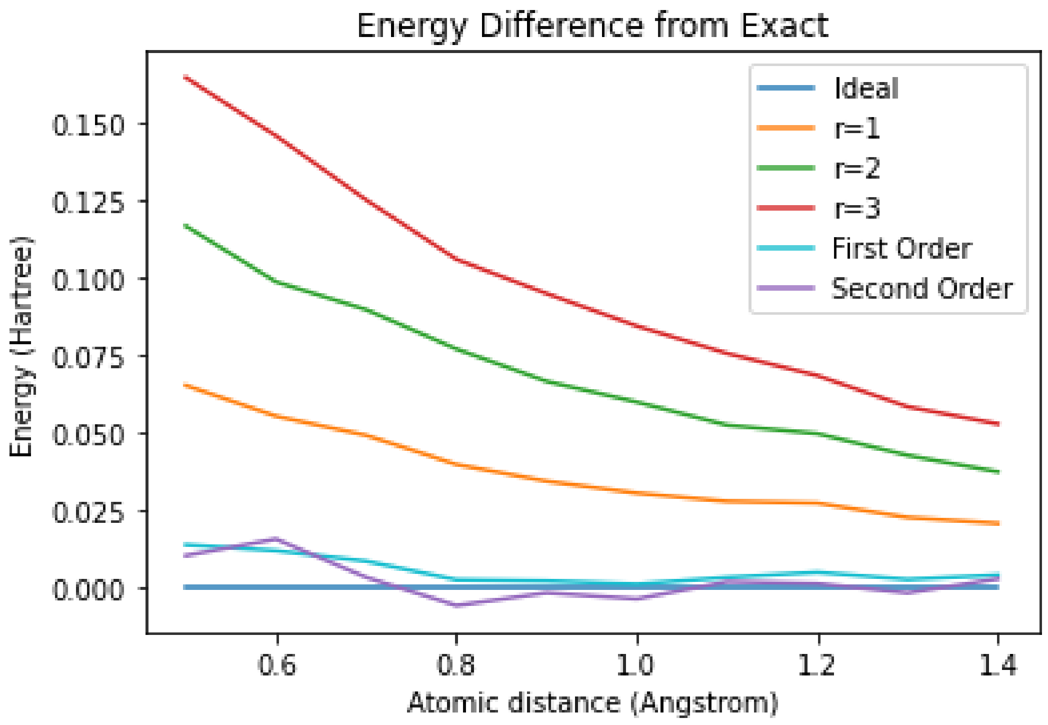
How the curve fitting technique works is shown later in this document.

[24]:
```
from scipy.optimize import curve_fit
# extrapolating to 1st order
first_order = []
for k in range(len(vqe_ideal)):
    plot=[]
    x = np.array([])
    y = np.array([])
    for c in range(2):
        x = np.append(x, [c+1])
        y = np.append(y, [vqe_mean_energies[c][k]])
    plot.append(x)
    plot.append(y)
    def fit_func(x, a, b):
        return a*x + b
    params = curve_fit(fit_func, x, y)
    [a, b] = params[0]
    plot.append([a, b])
    first_order.append(params[0][1])
#differences from exact

first_order_diff = []
for k in range(len(vqe_ideal)):
    first_order_diff.append(first_order[k] - exact[k])
#plotting energy differences
plt.plot(distances, energy_difference[0], label="Ideal")
plt.plot(distances, energy_difference[1], label="r=1")
plt.plot(distances, energy_difference[2], label="r=2")
plt.plot(distances, first_order_diff, label="First Order", color= 'tab:cyan')


plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.title('Energy Difference from Exact')
plt.legend()
plt.show()
```

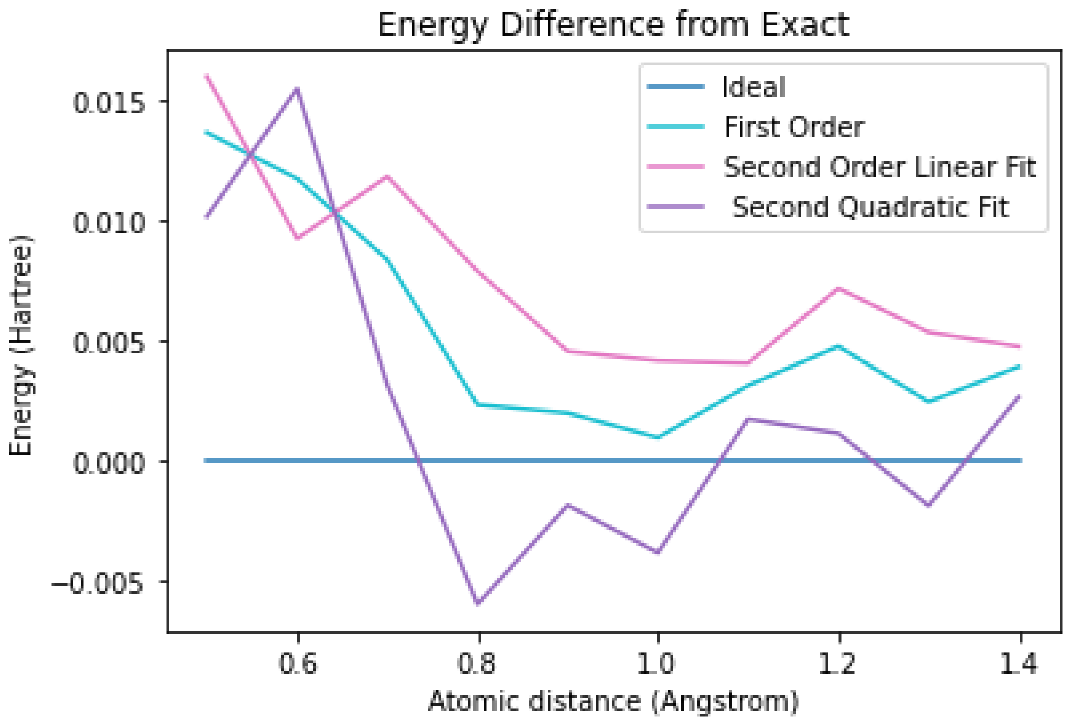


5.1 Onto Second Order

[32]:
```
# extrapolating to 2nd order
second_order = []
for k in range(len(vqe_ideal)):
    x = np.array([])
    y = np.array([])
    for c in range(3):
        x = np.append(x, [c+1])
        y = np.append(y, [vqe_mean_energies[c][k]])
    def fit_func(x, a, b, c):
        return a*(x**2) + b*x + c
    params = curve_fit(fit_func, x, y)
    [a, b, c] = params[0]
    second_order.append(params[0][2])
#differences from exact

second_order_diff = []
for k in range(len(vqe_ideal)):
    second_order_diff.append(second_order[k] - exact[k])
#plotting energy differences
plt.plot(distances, energy_difference[0], label="Ideal")
plt.plot(distances, energy_difference[1], label="r=1")
plt.plot(distances, energy_difference[2], label="r=2")
plt.plot(distances, energy_difference[3], label="r=3")
plt.plot(distances, first_order_diff, label="First Order", color= 'tab:cyan')
plt.plot(distances, second_order_diff, label="Second Order", color= 'tab:purple')

plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.title('Energy Difference from Exact')
plt.legend()
plt.show()
```

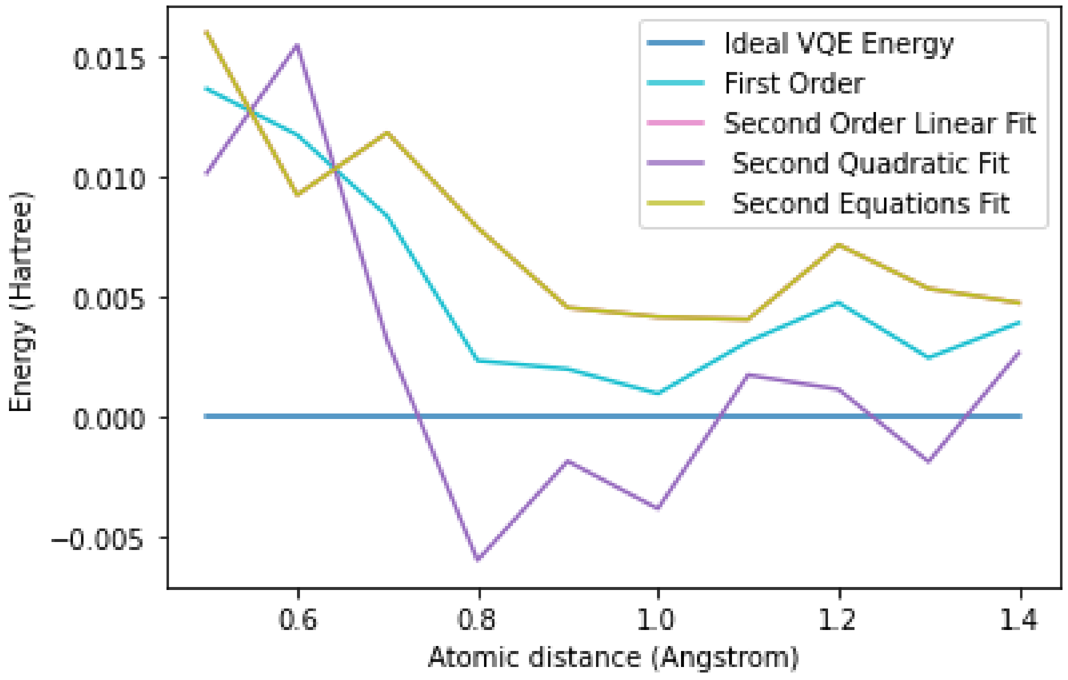


[42]:
```
# extrapolating to 2nd order
second_order_lin = []
for k in range(len(vqe_ideal)):
    x = np.array([])
    y = np.array([])
    for c in range(3):
        x = np.append(x, [c+1])
        y = np.append(y, [vqe_mean_energies[c][k]])
    def fit_func(x, a, b):
        return a*x + b
    params = curve_fit(fit_func, x, y)
    [a, b] = params[0]
    second_order_lin.append(params[0][1])
#differences from exact

second_order_lin_diff = []
for k in range(len(vqe_ideal)):
    second_order_lin_diff.append(second_order_lin[k] - exact[k])
#plotting energy differences
plt.plot(distances, energy_difference[0], label="Ideal")
#plt.plot(distances, energy_difference[1], label="r=1")
#plt.plot(distances, energy_difference[2], label="r=2")
#plt.plot(distances, energy_difference[3], label="r=3")
plt.plot(distances, first_order_diff, label="First Order", color= 'tab:cyan')
plt.plot(distances, second_order_lin_diff, label="Second Order Linear Fit",␣
 ↪color= 'tab:pink')
plt.plot(distances, second_order_diff, label=" Second Quadratic Fit", color=␣
 ↪'tab:purple')

plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.title('Energy Difference from Exact')
plt.legend()
plt.show()
```



6 Doing Second Order Via Equations

[51]:
```
#S
second_order_eq = []
for i in range(len(vqe_ideal)):
    s = ((3*vqe_mean_energies[0][i])) - (3*(vqe_mean_energies[1][i])) +␣
 ↪((vqe_mean_energies[2][i]))
    second_order_eq.append(s)
second_order_eq_diff = []
for k in range(len(vqe_ideal)):
    second_order_eq_diff.append(second_order_lin[k] - exact[k])
# plotting the data

plt.plot(distances, energy_difference[0], label="Ideal VQE Energy")
plt.plot(distances, first_order_diff, label="First Order", color= 'tab:cyan')
plt.plot(distances, second_order_lin_diff, label="Second Order Linear Fit",␣
 ↪color= 'tab:pink')
plt.plot(distances, second_order_diff, label=" Second Quadratic Fit", color=␣
 ↪'tab:purple')
plt.plot(distances, second_order_eq_diff, label=" Second Equations Fit", color=␣
 ↪'tab:olive')
plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy (Hartree)')
plt.legend()
plt.show()
```



[ ]:

```
```