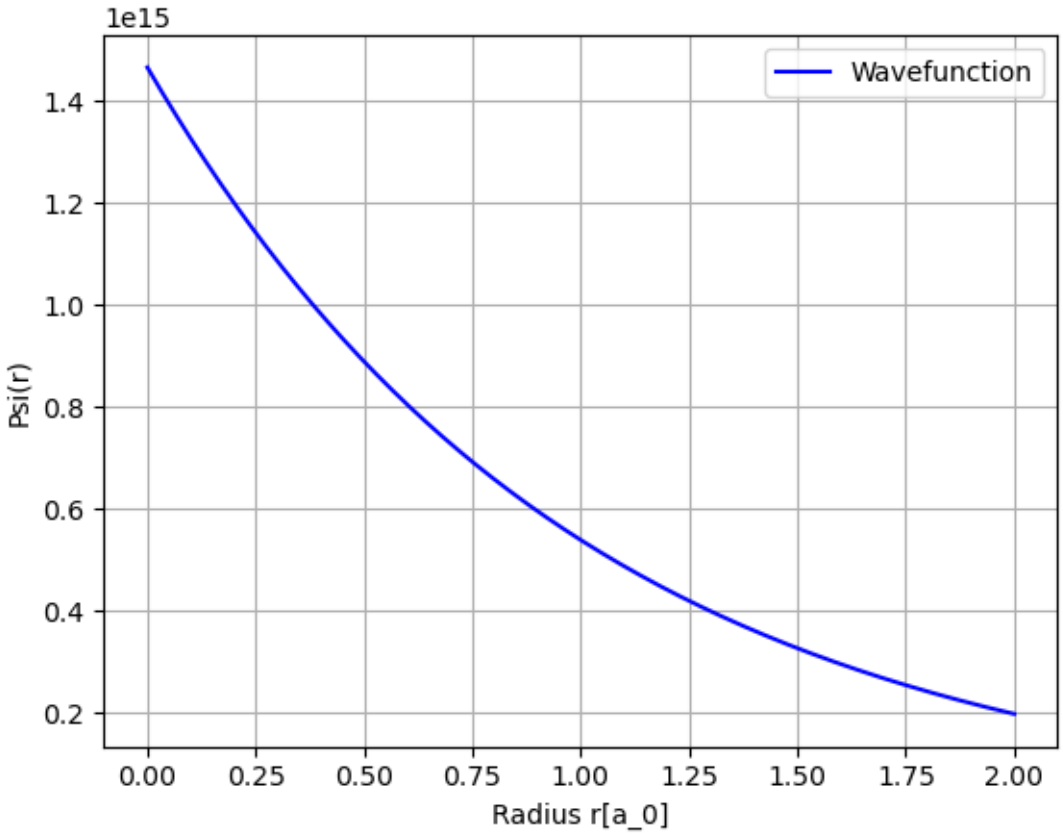
Lez13_Bias_Variance

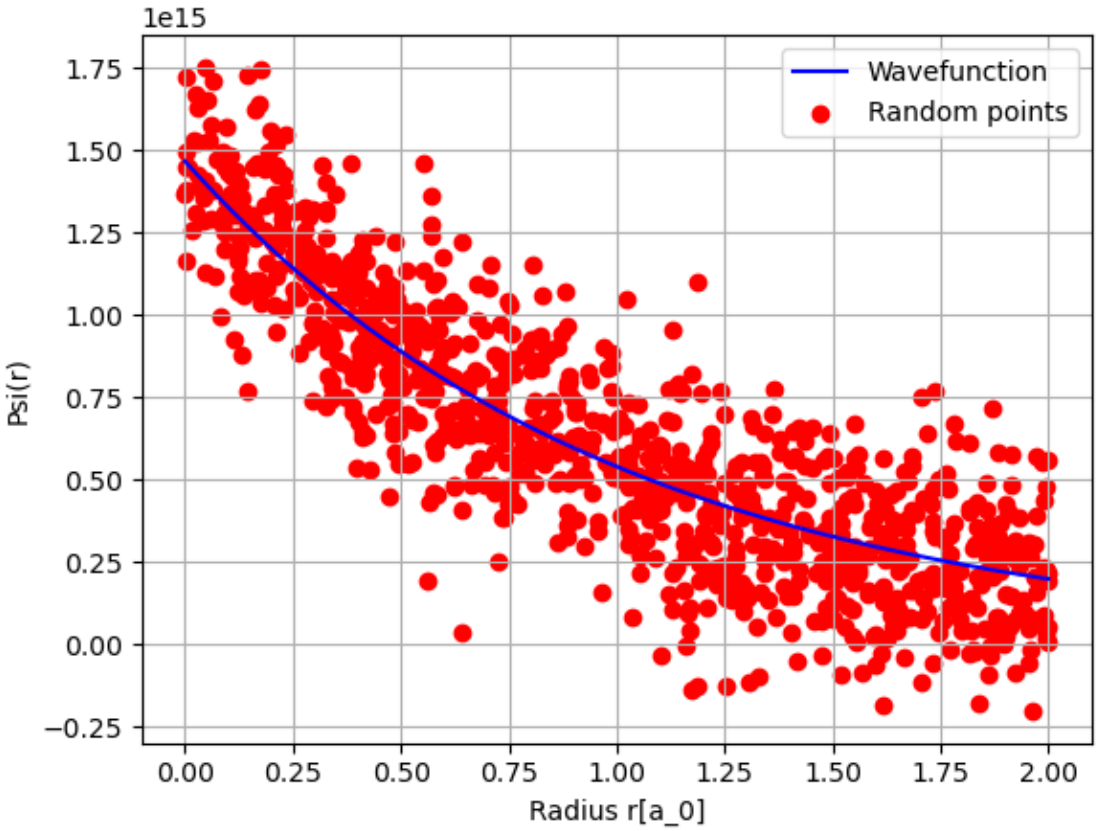
November 28, 2022

[1]:
```
# Goal: Approximate the quantum mechanical wave function of the 1s orbital of␣
 ↪hydrogen. Investigate the effect of different
# proportions for the train and evaluation set on the bias and the variance.

# Imports
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#Variables
Z = 1 # Number of protons for hydrogen
a_0 = 5.2917 * 10**(-11) # Bohr radius


# Define target function
def psi_target(r):
    return 1.0/(np.sqrt(np.pi))*(Z/a_0)**(3.0/2.0)*np.exp(-Z*r/a_0)

# Plot the target function
radius = np.linspace(0, 2*a_0, 1000)

plt.axes(frameon=1)
plt.grid(linestyle='-')
plt.plot(radius/a_0, psi_target(radius), label ='Wavefunction', color = 'blue')
plt.legend()
plt.xlabel('Radius r[a_0]')
plt.ylabel('Psi(r)')
plt.show()
```



[2]:
```
# Generate 1000 random points around the wavefunction, introduce sigma due to␣
 ↪noise
random.seed(42) # Set random seed generator
number_points = 1000
sigma_noise = random.random()*0.3*10**(15) # Sigma random between 0 * 10**(15)␣
 ↪and 0.3*10**(15)

random_points =  np.random.uniform(0, 2*a_0, size = number_points)[:, np.
 ↪newaxis] # Generate points on x-axis
noise = np.random.normal(scale = sigma_noise, size = number_points)[:, np.
 ↪newaxis] # Generate noise
target_points = psi_target(random_points) + noise # Generate points on y-axis

# Plot the points together with the wave function
plt.axes(frameon=1)
plt.grid(linestyle='-')
plt.plot(radius/a_0, psi_target(radius), label ='Wavefunction', color = 'blue')
plt.scatter(random_points/a_0, target_points, label ='Random points', color =␣
 ↪'red')
plt.legend()
```

```
plt.xlabel('Radius r[a_0]')
plt.ylabel('Psi(r)')
plt.show()
plt.close()

print("Sigma: ", sigma_noise)
```




```
Sigma:  191828039537365.12
```

[3]:
```
# Split the data into train and evaluation set with different proportions
X_train_1, X_evaluate_1, Y_train_1, Y_evaluate_1 =␣
 ↪train_test_split(random_points, target_points, test_size = 0.1)
X_train_2, X_evaluate_2, Y_train_2, Y_evaluate_2 =␣
 ↪train_test_split(random_points, target_points, test_size = 0.2)
X_train_3, X_evaluate_3, Y_train_3, Y_evaluate_3 =␣
 ↪train_test_split(random_points, target_points, test_size = 0.3)
X_train_4, X_evaluate_4, Y_train_4, Y_evaluate_4 =␣
 ↪train_test_split(random_points, target_points, test_size = 0.4)
X_train_5, X_evaluate_5, Y_train_5, Y_evaluate_5 =␣
 ↪train_test_split(random_points, target_points, test_size = 0.5)
```

```
# Train the models, use linear approximation
regressor = LinearRegression()
regressor.fit(X_train_1, Y_train_1)
regressor.fit(X_train_2, Y_train_2)
regressor.fit(X_train_3, Y_train_3)
regressor.fit(X_train_4, Y_train_4)
regressor.fit(X_train_5, Y_train_5)

# Define functions to compute bias and variance
def compute_bias(target_values_array, predict_values_array, target_value):

    expectation_value = mean_squared_error(target_values_array,␣
 ↪predict_values_array)

    return expectation_value - target_value

def compute_variance(target_values_array, predict_values_array):

    return mean_squared_error(target_values_array - ␣
 ↪mean_squared_error(target_values_array, predict_values_array),␣
 ↪predict_values_array)**2

# Compute bias and variance

#Bias, compute mean over sample
bias_mean_1 = 0.0
bias_mean_2 = 0.0
bias_mean_3 = 0.0
bias_mean_4 = 0.0
bias_mean_5 = 0.0

for i in range(np.size(X_evaluate_1)):
    bias_mean_1 += compute_bias(psi_target(X_evaluate_1), Y_evaluate_1,␣
 ↪psi_target(X_evaluate_1[i]))

for i in range(np.size(X_evaluate_2)):
    bias_mean_2 += compute_bias(psi_target(X_evaluate_2), Y_evaluate_2,␣
 ↪psi_target(X_evaluate_2[i]))

for i in range(np.size(X_evaluate_3)):
    bias_mean_3 += compute_bias(psi_target(X_evaluate_3), Y_evaluate_3,␣
 ↪psi_target(X_evaluate_3[i]))

for i in range(np.size(X_evaluate_4)):
    bias_mean_4 += compute_bias(psi_target(X_evaluate_4), Y_evaluate_4,␣
 ↪psi_target(X_evaluate_4[i]))
```

```
for i in range(np.size(X_evaluate_5)):
    bias_mean_5 += compute_bias(psi_target(X_evaluate_5), Y_evaluate_5,␣
 ↪psi_target(X_evaluate_5[i]))


bias_mean_1 /= np.size(X_evaluate_1)
bias_mean_2 /= np.size(X_evaluate_2)
bias_mean_3 /= np.size(X_evaluate_3)
bias_mean_4 /= np.size(X_evaluate_4)
bias_mean_5 /= np.size(X_evaluate_5)


#Variance
variance_1 = compute_variance(psi_target(X_evaluate_1), Y_evaluate_1)
variance_2 = compute_variance(psi_target(X_evaluate_2), Y_evaluate_2)
variance_3 = compute_variance(psi_target(X_evaluate_3), Y_evaluate_3)
variance_4 = compute_variance(psi_target(X_evaluate_4), Y_evaluate_4)
variance_5 = compute_variance(psi_target(X_evaluate_5), Y_evaluate_5)

# Print bias and variance for each model

print("Model 1:  Bias: ", bias_mean_1, ", Variance: ", variance_1)
print("Model 2:  Bias: ", bias_mean_2, ", Variance: ", variance_2)
print("Model 3:  Bias: ", bias_mean_3, ", Variance: ", variance_3)
print("Model 4:  Bias: ", bias_mean_4, ", Variance: ", variance_4)
print("Model 5:  Bias: ", bias_mean_5, ", Variance: ", variance_5)
```

```
Model 1:  Bias:  [3.62447074e+28] , Variance:  1.725752023538223e+114
Model 2:  Bias:  [4.68286562e+28] , Variance:  4.8089114608205996e+114
Model 3:  Bias:  [3.34889801e+28] , Variance:  1.2577886855812395e+114
Model 4:  Bias:  [3.40985539e+28] , Variance:  1.3518977445678806e+114
Model 5:  Bias:  [3.75601181e+28] , Variance:  1.9902507511941036e+114
```

[ ]: