2.24

a.

$$\bar{g}(x) = \bar{a}x + \bar{b},$$

where $\bar{a}$ and $\bar{b}$ are the mean of a and b, respectively.

b.

1. Generate random uniformly distributed pairs of variables along the interval $[-1, 1]$ and calculate the slope and y-intercept of the line going through each pair.
2. Average the values for slope and y-intercept to obtain $\bar{g}(x)$ above.
3. Use the equations given in Section 2.3.1 to calculate the bias and variance for a test data set.
4. Calculate $E_{out}(g^D(x)) = E_x[(g^D(x) - f(x))^2]$ for the out-of-sample error, with respect to our test data set.

c.

```
from random import uniform
import numpy as np
import matplotlib.pyplot as plt

n=100
#Define arrays used in algorithm#
x_train=np.zeros(shape=(n,2))
a=np.zeros(shape=(n,1))
b=np.zeros(shape=(n,1))
for row in range(0,n):
    x_train[row]=(uniform(-1,1),uniform(-1,1))

#Define target function#
def target(x):
    return np.square(x)

#Calculate the line going through the two entries on each row of x#
for slope in range(0,n):
    a[slope]=(target(x_train[slope][0])-target(x_train[slope][1]))/(x_train[slope][0]-x_train[slope][1])

for intercept in range(0,n):
    b[intercept]=target(x_train[intercept][0])-(a[intercept]*x_train[intercept][0])


del(slope,intercept,row)

#Get the average hypothesis function#
slope=np.mean(a)
y_intercept=np.mean(b)

def g(a,x,b):
    return(a*x+b)

def g_bar(x):
    return (slope*x+y_intercept)

#Get bias and variance of our model#
```

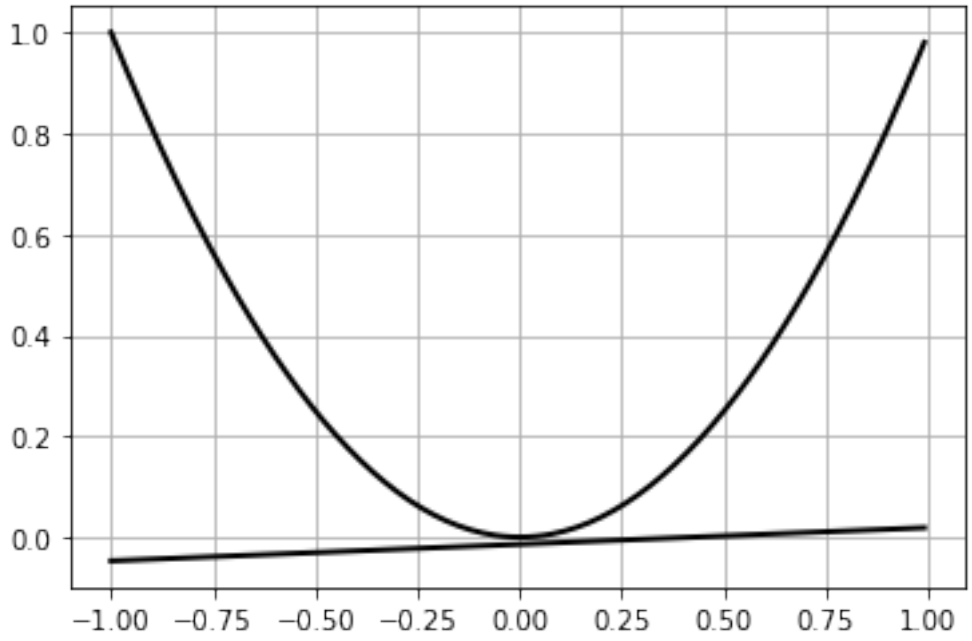

Figure 1: png

```
bias=0
variance=0
sse=0

x_test=np.zeros(shape=(n,2))
for row in range(0,len(x_test)):
    x_test[row]=(uniform(-1,1),uniform(-1,1))

for row in range (0,len(x_test)):
    for col in range(0,1):
        bias += np.square(g_bar(x_test[row][col])-target(x_test[row][col]))/n
        variance += np.square(g(a[row],x_test[row][col],b[row])-g_bar(x_test[row][col]))/n
        sse += np.square(g(a[row],x_test[row][col],b[row])-target(x_test[row][col]))/n

del row
```

```
#Plot g_bar and the target function#
xvals = np.arange(-1,1,.01)
yvals_quadratic=target(xvals)
plt.plot(xvals,yvals_quadratic,'k-',linewidth=2)
yvals_linear = g_bar(xvals)
plt.plot(xvals,yvals_linear,'k-',linewidth=2)
plt.grid()
plt.show()
```

```
print("Bias: ",bias)
print("Variance: ",variance)
print("SSE with respect to x: ", bias+variance)
print("SSE: ", sse)
```

```
Bias:  0.2162917148789125
Variance:  [0.38316872]
SSE with respect to x:  [0.59946044]
SSE:  [0.66266343]
```

I started by comparing 100 training data sets. Each data set is a pair of values along the curve of $f(x) = x^2$ and there are vectors recording the slope and y-intercepts of the line between them. I took the average of my slope and intercept vectors to determing $\bar{g}$. The vector x_test is a new list of data sets that I used to compare g with $\bar{g}$. It's worth nothing that x_test also has 100 pairs. I considered cases down to x_test with size 10, and got comparable results.

Since the data is uniformly generated, it seemed intuitive to me that the $\bar{g}$ be horizontal. The reason being, two randomly generated points along this interval would, on average, be about equidistant from the mean, 0. I also guessed that $b < 0.5$, although I didn't have a reason for it. I ended up being right. After running the code many times, every plot of $\bar{g}$ was roughly horizontal, and it was always very close to $b = 0$.

For both methods of calculating the out-of-sample error, I got $E[E_{out}] \in (0.4, 0.8)$. It seemed to me that the error would converge to 0.5. Since I'm taking the mean of the squared error, I should be no more than half the range away from the true value.

Below, I ran the code a second time, considering 10,000 pairs of data sets, and have displayed the plot and results of what I believe the statistics converge to.

```
from random import uniform
import numpy as np
import matplotlib.pyplot as plt

n=10000
#Define arrays used in algorithm#
x_train=np.zeros(shape=(n,2))
a=np.zeros(shape=(n,1))
b=np.zeros(shape=(n,1))
for row in range(0,n):
    x_train[row]=(uniform(-1,1),uniform(-1,1))

#Define target function#
def target(x):
    return np.square(x)

#Calculate the line going through the two entries on each row of x#
for slope in range(0,n):
    a[slope]=(target(x_train[slope][0])-target(x_train[slope][1]))/(x_train[slope][0]-x_train[slope][1])

for intercept in range(0,n):
    b[intercept]=target(x_train[intercept][0])-(a[intercept]*x_train[intercept][0])


del(slope,intercept,row)

#Get the average hypothesis function#
slope=np.mean(a)
y_intercept=np.mean(b)

def g(a,x,b):
    return(a*x+b)
```

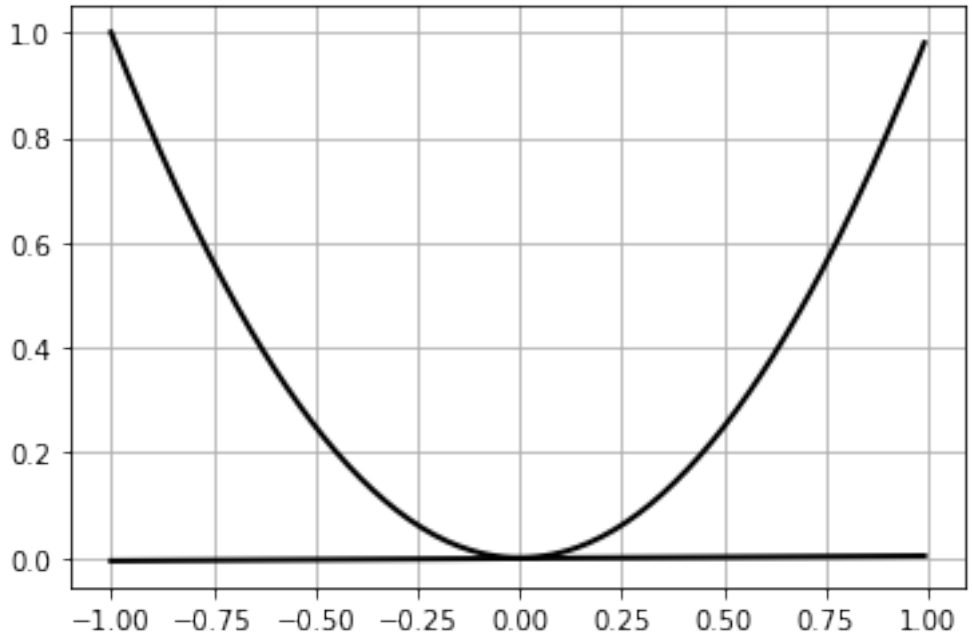

Figure 2: png

```
def g_bar(x):
    return (slope*x+y_intercept)

#Get bias and variance of our model#
bias=0
variance=0
sse=0

x_test=np.zeros(shape=(n,2))
for row in range(0,len(x_test)):
    x_test[row]=(uniform(-1,1),uniform(-1,1))

for row in range (0,len(x_test)):
    for col in range(0,1):
        bias += np.square(g_bar(x_test[row][col])-target(x_test[row][col]))/n
        variance += np.square(g(a[row],x_test[row][col],b[row])-g_bar(x_test[row][col]))/n
        sse += np.square(g(a[row],x_test[row][col],b[row])-target(x_test[row][col]))/n

del row
```

```
#Plot g_bar and the target function#
xvals = np.arange(-1,1,.01)
yvals_quadratic=target(xvals)
plt.plot(xvals,yvals_quadratic,'k-',linewidth=2)
yvals_linear = g_bar(xvals)
plt.plot(xvals,yvals_linear,'k-',linewidth=2)
plt.grid()
plt.show()
```

```
print("Bias: ",bias)
print("Variance: ",variance)
print("SSE with respect to x: ", bias+variance)
print("SSE: ", sse)
```

```
Bias:  0.19705116197492253
Variance:  [0.33428151]
SSE with respect to x:  [0.53133268]
SSE:  [0.53334416]
```

d. Above, we can see that $\bar{g}(x) \to 0$. Using that, we get

$$\begin{aligned} Bias &= E[(\bar{g}(x) - f(x))^2] \\ &= E[(0 - x^2)^2] \\ &= E[x^4] \end{aligned} \qquad \begin{aligned} Variance &= E[(g^{(D)}(x) - \bar{g}(x))^2] \\ &= E[(g^{(D)}(x) - 0)^2] \\ &= E[g^{(D)}(x)^2] \end{aligned}$$

Both of which are less than or equal to 1, because of the range we're considering. So, $E[E_{out}] = E[x^4 + g^{(D)}(x)^2]$

2.22

We have $y(x) = f(x) + \epsilon$, where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$. Note that

$$\begin{aligned} \sigma^2 &= E(\epsilon^2) - E(\epsilon)^2 \\ &= E(\epsilon^2) - 0 \\ &= E(\epsilon^2) \end{aligned}$$

$$\begin{aligned} E_{out}(g^{(D)}(x)) &= E[(g^{(D)}(x) - y(x))^2] \\ &= E[(g^{(D)}(x) - f(x) + f(x) - y(x))^2] \\ &= E[(g^{(D)}(x) - f(x))^2] - 2E[(g^{(D)}(x) - f(x))(f(x) - y(x))] + E[(f(x) - y(x))^2] \\ &= E[(g^{(D)}(x) - f(x))^2] - 2(E[g^{(D)}(x)f(x)] - E[g^{(D)}(x)y(x)] - E[f(x)^2] + E[f(x)y(x)]) + E[\epsilon^2] \\ &= E[(g^{(D)}(x) - f(x))^2] - 2(E[g^{(D)}(x)f(x)] - E[g^{(D)}(x)f(x)] - E[f(x)^2] + E[f(x)^2]) + E[\epsilon^2] \\ &= E[(g^{(D)}(x) - f(x))^2] + E[\epsilon^2] \end{aligned}$$

From the book, we know that $E[(g^{(D)}(x) - f(x))^2] = bias + var$ and from above $E[\epsilon^2] = \sigma^2$. So,

$$E_{out}(g^{(D)}(x)) = bias + var + \sigma^2$$