Unit 2 Lecture 2: Bias-variance tradeoff

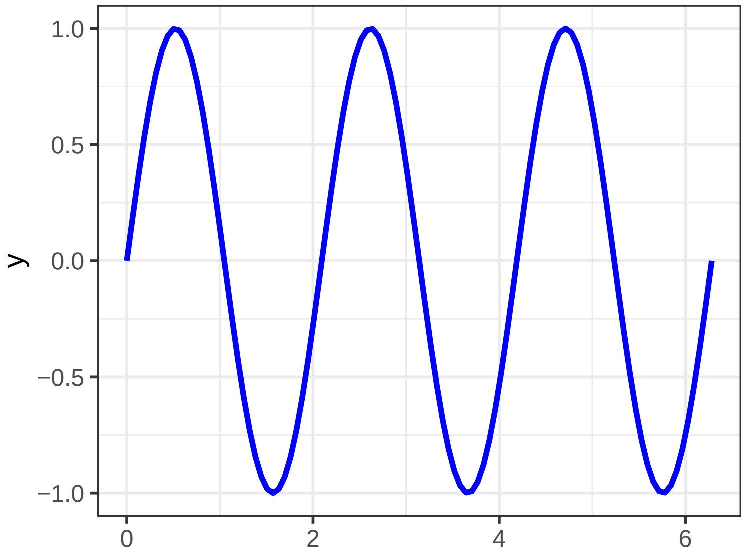
September 20, 2022

In this R demo, we will explore the bias-variance tradeoff in the context of natural spline fits. We'll need the following R packages:

```
library(tidyverse)
library(splines)  # for ns()
library(cowplot)  # for plot_grid()
library(stat471)  # for generate_spline_data(), fit_spline_models(),
                  #  compute_bias_variance_ETE(), bias_variance_ETE_plot()
```

As our true function f, let us use the following sine curve:

```
f <- function(x)(sin(3*x))
ggplot() +
  stat_function(fun = f, colour = "blue", size = 1, xlim = c(0, 2*pi))
```



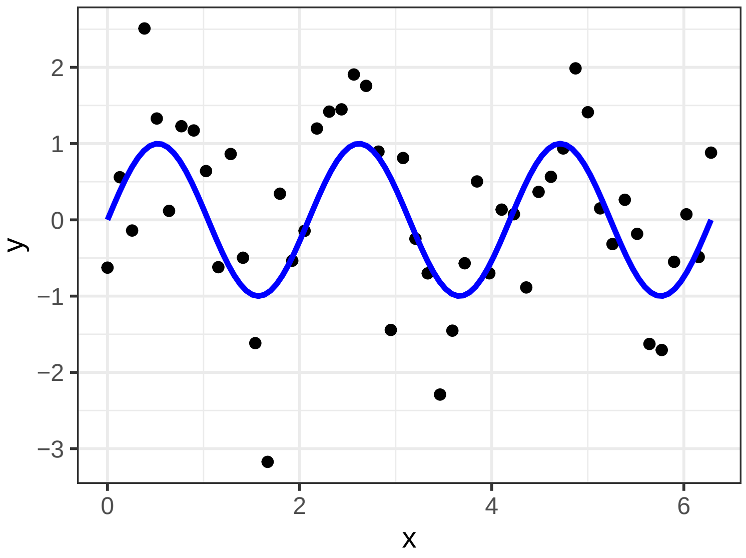
Training and testing

Let us start with the function `f`. Let us create a training set

```
set.seed(1)
n <- 50
sigma <- 1
train_data <- tibble(
  x = seq(0, 2 * pi, length.out = n),
  y = f(x) + rnorm(n, sd = sigma)
)
train_data
```

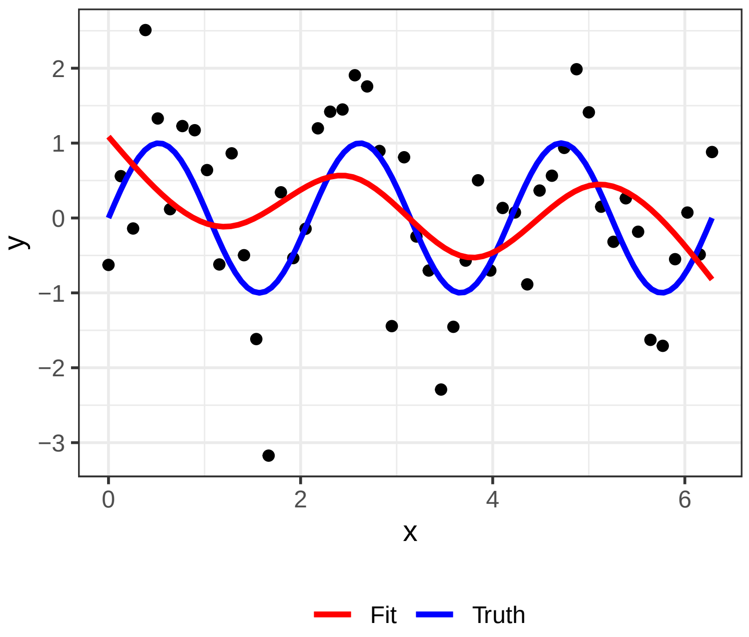
```
## # A tibble: 50 x 2
##         x      y
##     <dbl>  <dbl>
##   1 0     -0.626
##   2 0.128  0.559
##   3 0.256 -0.140
##   4 0.385  2.51
##   5 0.513  1.33
##   6 0.641  0.118
##   7 0.769  1.23
##   8 0.898  1.17
##   9 1.03   0.640
##  10 1.15  -0.620
## # ... with 40 more rows
```

```
train_data %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  stat_function(fun = f, colour = "blue", size = 1)
```



Let us train a natural spline fit with 5 degrees of freedom on this data:

```
spline_fit <- lm(y ~ ns(x, df = 5), data = train_data)
train_data %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  stat_function(fun = f, aes(colour = "Truth"), size = 1) +
  geom_smooth(method = "lm",
              formula = "y ~ ns(x, df = 5)",
              aes(colour = "Fit"), se = FALSE) +
  scale_colour_manual(values = c("red", "blue"))
```



Next, we compute the training and test error for this fit:

```
### training error
y_hat_train <- predict(spline_fit, newdata = train_data)
head(y_hat_train)
```

```
##         1         2         3         4         5         6
## 1.0863047 0.8955427 0.7090961 0.5312804 0.3664109 0.2188031
```

```
train_data %>%
    mutate(y_hat_train = y_hat_train) %>%
    summarise(training_error = mean((y_hat_train-y)^2))
```

```
## # A tibble: 1 x 1
##   training_error
##            <dbl>
## 1           1.16
```

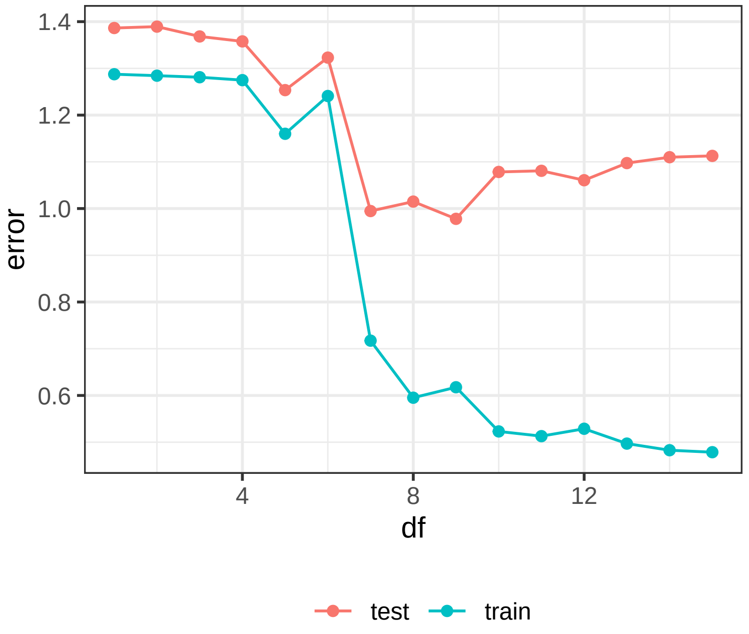
```
### test error

# create a large test set
N <- 50000
test_data <- tibble(x = seq(0, 2*pi, length.out = N),
                   y = f(x) + rnorm(n, sd = sigma))

# compute test error
y_hat_test <- predict(spline_fit, newdata = test_data)
test_data %>%
    mutate(y_hat_test = y_hat_test) %>%
    summarise(test_error = mean((y_hat_test-y)^2))
```

```
## # A tibble: 1 x 1
##   test_error
##        <dbl>
## 1       1.25
```

Varying the degrees of freedom

Next let's do the same thing, but for `df` varying between 1 and 15.

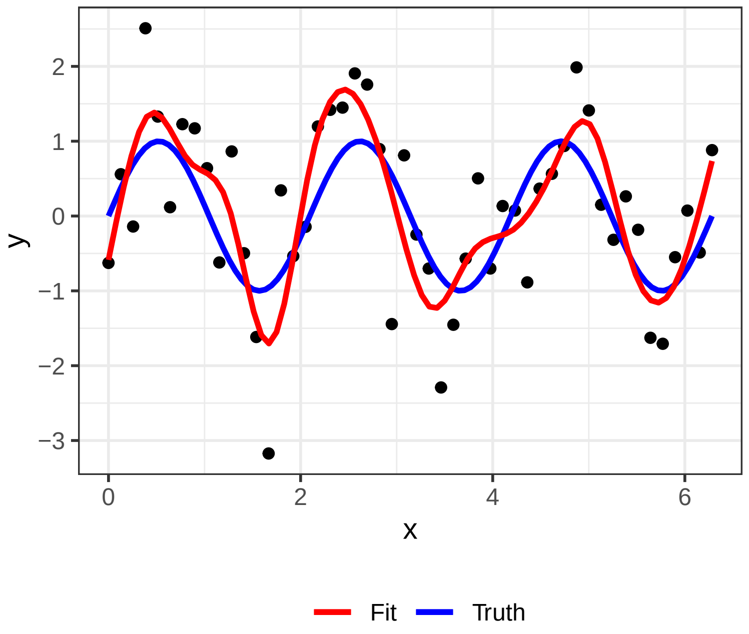
```r
# loop over df = 1,...,15
max_df <- 15
error_test <- numeric(max_df)
error_train <- numeric(max_df)
for(df in 1:max_df){
  formula <- sprintf("y ~ ns(x, df = %d)", df)
  spline_fit <- lm(formula = formula, data = train_data)
  y_hat_train <- predict(spline_fit, newdata = train_data)
  y_hat_test <- predict(spline_fit, newdata = test_data)
  error_train[df] <- train_data %>%
    mutate(y_hat_train = y_hat_train) %>%
    summarise(mean((y_hat_train-y)^2)) %>%
    pull()
  error_test[df] <- test_data %>%
    mutate(y_hat_test = y_hat_test) %>%
    summarise(mean((y_hat_test-y)^2)) %>%
    pull()
}

# create the plot
tibble(df = 1:max_df, error_train, error_test) %>%
  pivot_longer(cols = -df, names_to = "set",
               names_prefix = "error_",values_to = "error") %>%
  ggplot(aes(x = df, y = error, color = set)) +
  geom_point() +
  geom_line()
```



What is the best choice for `df`?

Let's visualize a few of these fits to see if these train and test errors make sense:

```r
train_data %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  stat_function(fun = f, aes(colour = "Truth"), size = 1) +
  geom_smooth(method = "lm",
              formula = "y ~ ns(x, df = 15)",
              aes(colour = "Fit"), se = FALSE) +
  scale_colour_manual(values = c("red", "blue"))
```



Does the best choice of `df` from before make sense?

Fitting splines to many random training sets (numerical simulation)

Recall that the expected test error, bias, and variance are quantities averaged over the randomness in the training data. Therefore, let us repeatedly generate the training data to compute them in the above example. This process is called a *numerical simulation.*

```r
resamples <- 100
train_data_resamples <- generate_spline_data(f, sigma, n, resamples)
train_data_resamples
```

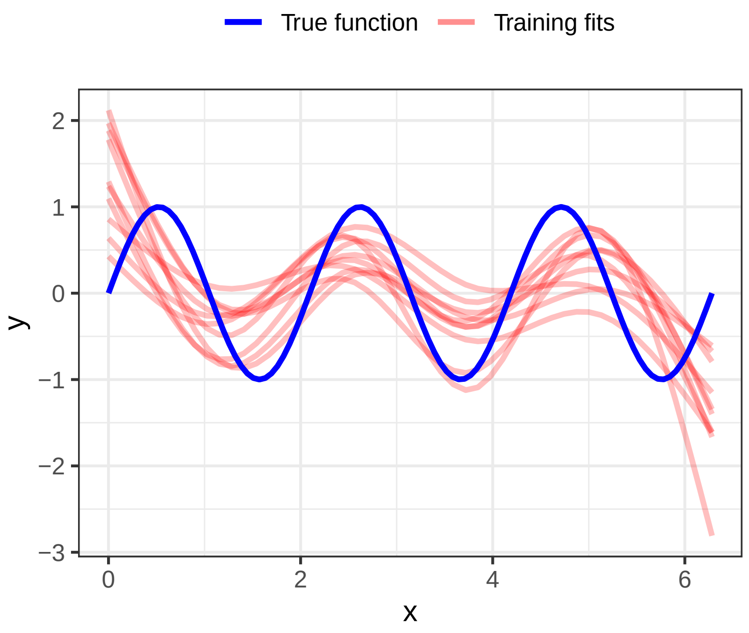
```
## # A tibble: 5,000 x 3
##         x      y resample
##     <dbl>  <dbl>    <int>
##   1 0     -0.620        1
##   2 0.128  0.417        1
##   3 0.256 -0.215        1
##   4 0.385  1.07         1
##   5 0.513  0.345        1
##   6 0.641  2.71         1
##   7 0.769  1.46         1
##   8 0.898  1.34         1
##   9 1.03   0.448        1
##  10 1.15   1.37         1
## # ... with 4,990 more rows
```

Next, let us fit the natural spline model with 5 degrees of freedom to each resampled dataset:

```r
training_results <- fit_spline_models(train_data_resamples, df_values = 5)
training_results
```

```
## # A tibble: 5,000 x 5
##        x       y resample    df    pred
##    <dbl>   <dbl>    <int> <dbl>   <dbl>
##  1 0     -3.07         74     5  0.267
##  2 0.128 -0.939        74     5  0.0722
##  3 0.256  1.37         74     5 -0.118
##  4 0.385  2.51         74     5 -0.298
##  5 0.513  0.0205       74     5 -0.464
##  6 0.641  0.508        74     5 -0.611
##  7 0.769 -0.882        74     5 -0.733
##  8 0.898 -0.213        74     5 -0.826
##  9 1.03   0.425        74     5 -0.886
## 10 1.15  -1.10         74     5 -0.907
## # ... with 4,990 more rows
```

Let's plot the first ten of these fits:



Bias, variance, and expected test error

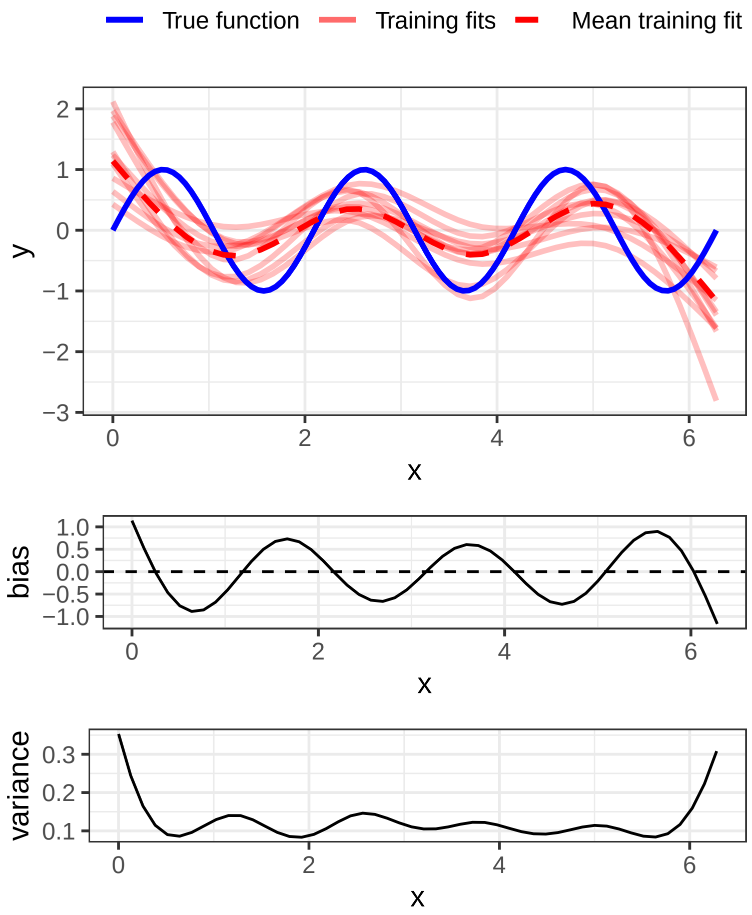
Separately for each data point

Let's compute the mean prediction, bias, and variance for each value of `x` by averaging over the resamples:

```r
training_results_summary <- training_results %>%
  mutate(true_fit = f(x)) %>%
  group_by(x) %>%
  summarise(mean_pred = mean(pred),
            bias = mean(pred - true_fit),
            variance = var(pred))
training_results_summary
```

```
## # A tibble: 50 x 4
##        x mean_pred    bias variance
##    <dbl>     <dbl>   <dbl>    <dbl>
##  1 0         1.14    1.14     0.353
##  2 0.128     0.902   0.527    0.244
##  3 0.256     0.669  -0.0265   0.165
##  4 0.385     0.446  -0.468    0.115
##  5 0.513     0.238  -0.761    0.0902
##  6 0.641     0.0502 -0.888    0.0861
##  7 0.769    -0.113  -0.853    0.0960
##  8 0.898    -0.247  -0.681    0.113
##  9 1.03     -0.345  -0.410    0.130
## 10 1.15     -0.405  -0.0895   0.140
## # ... with 40 more rows
```

Let us plot these and see what we get:



How do we interpret the bias in terms of the first plot above? When is it above zero and when is it below zero?

How do we interpret the variance in terms of the first plot above? Why is it larger at the edges of the data?

Overall bias, variance, and ETE

To get the mean squared bias and mean variance, we average across data points:

```
bias_variance <- training_results_summary %>%
  summarize(mean_sq_bias = mean(bias^2),
            mean_variance = mean(variance),
            irreducible_error = sigma^2)
bias_variance
```

```
## # A tibble: 1 x 3
##   mean_sq_bias mean_variance irreducible_error
##          <dbl>         <dbl>             <dbl>
## 1        0.328         0.126                 1
```

Based on this information, how do we compute expected test error?

```
bias_variance %>% mutate(expected_test_error = ???)
```

Sanity check 1: does the variance match the formula?

What is the formula for mean squared variance of the fit for a linear regression model?

```
# formula for mean variance:
variance_formula = ???
```

Does this match the mean squared variance obtained above?

Sanity check 2: does ETE match the definition?

Let us calculate the ETE from its definition by generating test points:

```
training_results %>%
  mutate(y_test = f(x) + rnorm(n*resamples, sd = sigma)) %>%
  group_by(resample) %>%
  summarise(test_error = mean((pred - y_test)^2)) %>%
  summarise(expected_test_error = mean(test_error))
```

```
## # A tibble: 1 x 1
##   expected_test_error
##                 <dbl>
## 1                1.45
```

Does this quantity match the ETE computed above?

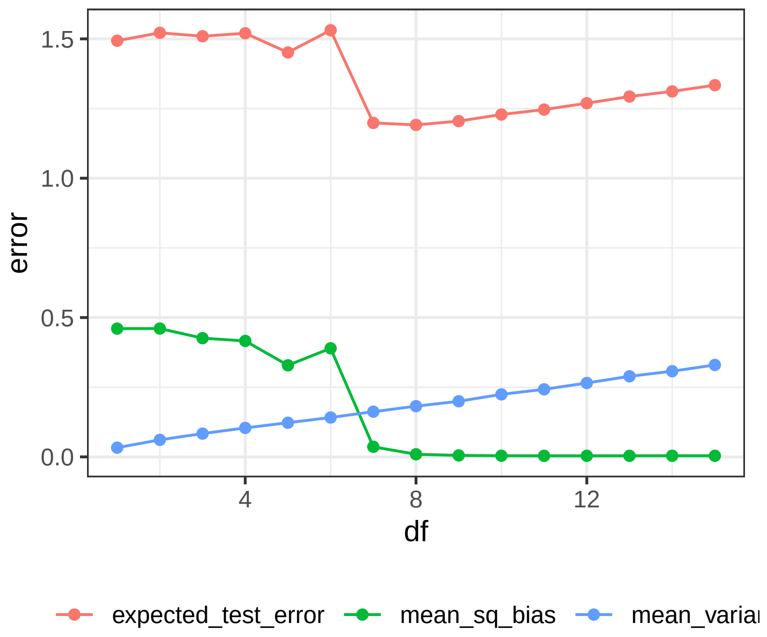
Varying the degrees of freedom

Next let's do the same thing, but for `df` varying between 1 and 15. We wrap everything in a function for convenience:

```
bias_variance_tradeoff <- function(f, sigma, n, resamples){
  # set seed for reproducibility
  set.seed(1)

  # create bias variance trade-off plot
  generate_spline_data(f, sigma, n, resamples) %>%  # generate training datasets
    fit_spline_models(df_values = 1:15) %>% # fit natural splines to training data
    compute_bias_variance_ETE(f,sigma) %>% # compute bias, variance, and ETE
    bias_variance_ETE_plot() %>%  # plot the bias, variance, and ETE
    plot()
}
```

Let's try out this function on the example from before:

```
f <- function(x)(sin(3*x))
sigma <- 1
n <- 50
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```

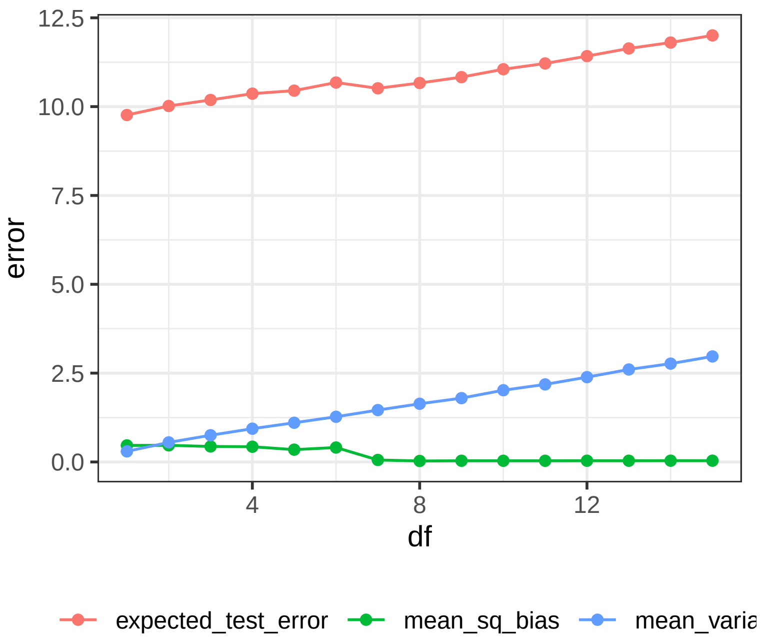


What trends do we observe in this plot? What appears to be the best degrees of freedom? Why does the variance curve appear linear? What does it mean that the bias is zero from `df` = 8 onwards?

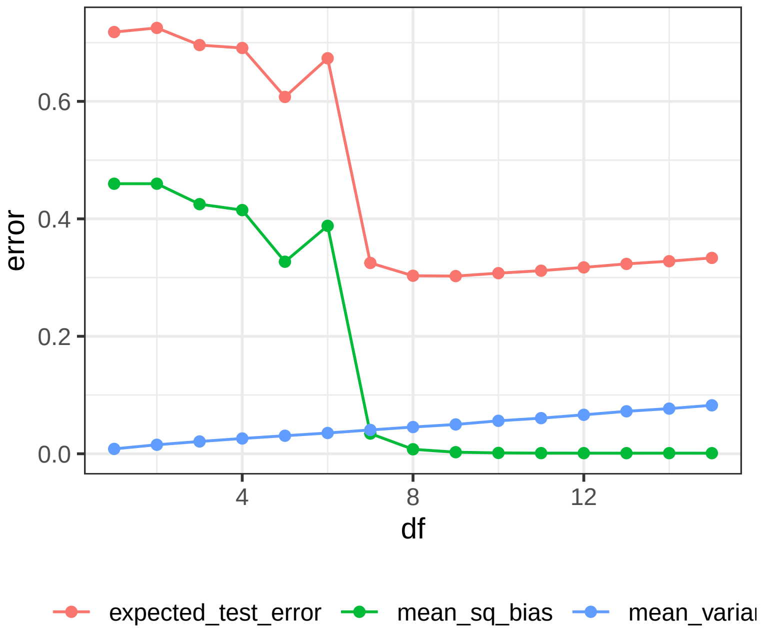
Varying the noise level

What happens if we increase the noise standard deviation? What happens if we decrease it?

```
# try increasing noise level
f <- function(x)(sin(3*x))
sigma <- 3
n <- 50
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```



```
# try decreasing noise level
f <- function(x)(sin(3*x))
sigma <- 0.5
n <- 50
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```

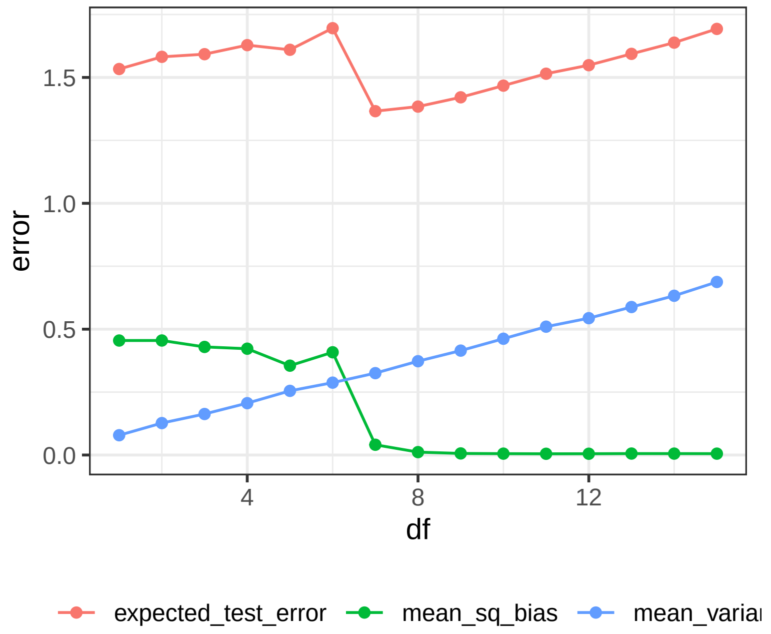


Varying the sample size

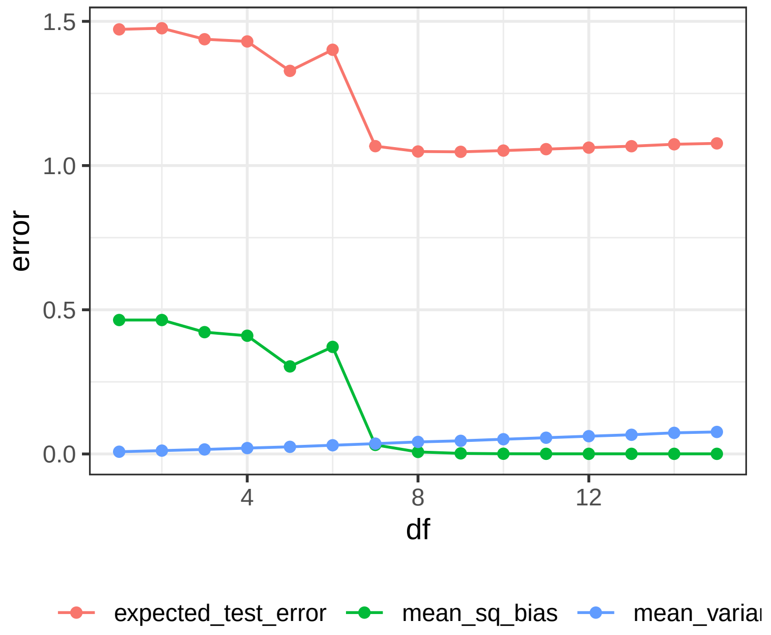
What happens if we decrease the sample size? What happens if we increase it?

```
# try decreasing sample size
f <- function(x)(sin(3*x))
```

```r
sigma <- 1
n <- 25
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```



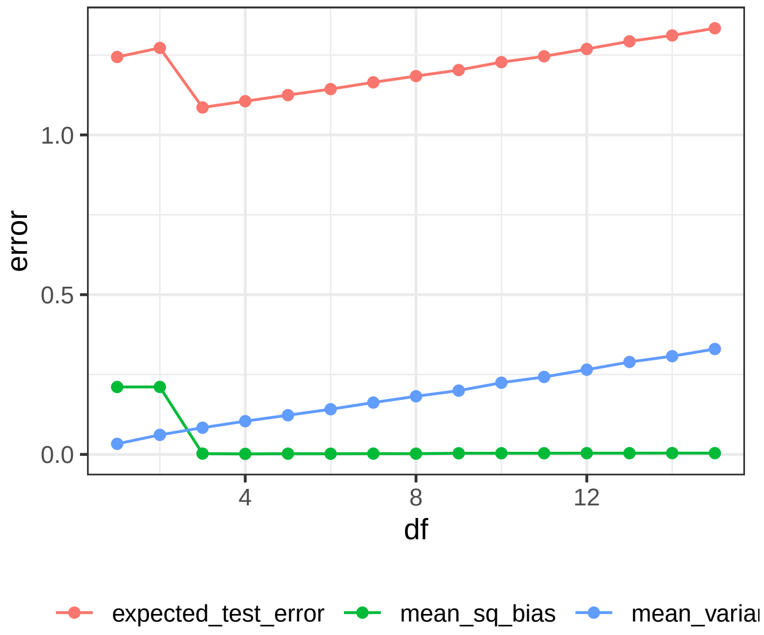
```r
# try increasing sample size
f <- function(x)(sin(3*x))
sigma <- 1
n <- 200
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```



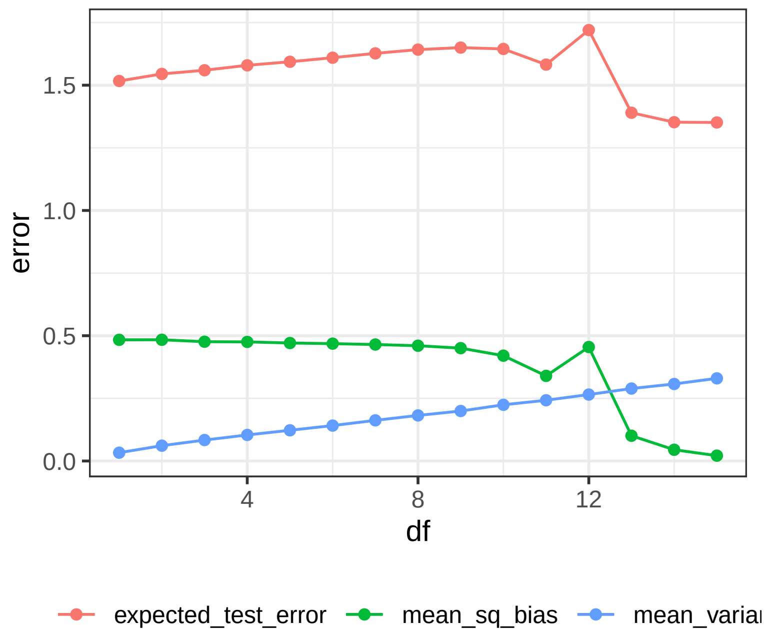
Varying the complexity of the underlying function

What happens if we decrease the complexity of the underlying function `f`? What happens if we increase it? [Hint: Consider `f <- function(x)(sin(k*x))` for different `k`. Higher `k` leads to more wiggles, i.e. increased complexity.]

```
# try decreasing complexity of f
f <- function(x)(sin(1*x))
sigma <- 1
n <- 50
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```



```
# try increasing complexity of f
f <- function(x)(sin(6*x))
sigma <- 1
n <- 50
resamples <- 100
bias_variance_tradeoff(f, sigma, n, resamples)
```


error
1.5
1.0
0.5
0.0
4
8
12
df
expected_test_error
mean_sq_bias
mean_variai