# 5.1 Bias-Variance Analysis

May 31, 2017

## 0.1 Exercise 5.1: Bias-Variance Analysis

In this exercise we explore the bias and variance for different model classes given the simple target function $f(x) = \sin \pi x$. The model classes to investigate are: * Constant models: $h(x) = b$ * Linear models: $h(x) = wx + b$ * Polynomial models: $h(x) = w_d x^d + \ldots w_2 x^2 + w_1 x + b$

You already know how to fit linear models. Polynomial models can be trained the same way if the dataset is augmented with the polynomial features first. Scikit-learn already offers a convenient function to do this for you, but you are welcome to build your own solution for this.

**Task a): Model fitting.** Fill in the functions below for augmenting datasets as well as fitting and evaluating the models to be investigated. Make sure to understand the dimensionality of input and output of each function.

In [1]:
```
import numpy as np

# We will use polynomials of degree 3 in this exercise
poly_degree = 3

# Shape guidelines
# X: (samples, features)
# y: (samples, targets)
# weights: (features, targets)

# A helper function to make sure that we fit and evaluate matrices of
# correct sizes. Remember that the first dimension always indicates
# the samples and the second the features or targets.
# Not all of these constraints are strictly necessary, but they will make
# your life easier when things go wrong.
def verify_shapes(X=None, y=None):
    if X is not None:
        assert X.ndim == 2 and X.shape[1] == 1
    if y is not None:
        assert y.ndim == 2 and y.shape[1] == 1
        if X is not None:
            assert y.shape[0] == X.shape[0]


# The ground-truth that we want to approximate.
# We only care about the domain [-1, 1], so no further preprocessing is required.
```

```python
def target(X):
    verify_shapes(X)
    return np.sin(np.pi * X)


# Fit a constant model h(x) = b.
# Fitting only a bias reduces to the mean over the targets.
# Remember to keep the correct shape!
def fit_constant(X, y):
    verify_shapes(X, y)
    # TODO
    pass


# Evaluate a constant model defined by its bias b.
def eval_constant(X, b):
    verify_shapes(X)
    # TODO
    pass


# Fit a linear model h(x) = wx + b.
# You can use the Moore-penrose pseudoinverse or the
# least squares function in numpy.linalg.
# Remember to augment the dataset to include a column for the bias.
def fit_linear(X, y):
    verify_shapes(X, y)
    # TODO
    pass


# Evaluate a linear model defined by its weights w. Remember to augment X!
def eval_linear(X, w):
    verify_shapes(X)
    # TODO
    pass


# Fit a polynomial model of degree 'poly_degree'.
# Augment the dataset with polynomial features
# and fit a linear model to it.
def fit_polynomial(X, y):
    verify_shapes(X, y)
    # TODO
    pass

# Evaluate a polynomial model defined by its weights w.
# Remember to augment X!
```

```
def eval_polynomial(X, w):
    verify_shapes(X)
    # TODO
    pass
```

Now that we can fit and evaluate these functions, we should create the datasets to evaluate the bias and variance of each model class. Since the bias-variance decomposition is defined as an expectation, we need to create sufficiently many datasets to get a good estimate.

---

** Task b): Model analysis.** Fit 100 models of each class to 100 datasets with only 5 samples each. Determine the average models and calculate the bias and variance of each class.

**Hint**: Remember the sample mean for the integrals: $\mathbb{E}[x] = \int x\, p(x)\, dx \approx \frac{1}{N} \sum_i^N x_i$

We provide you with plotting code for the visualization of the fitted and the average models below the code stub.

In [2]:
```
n_datasets = 100
n_samples = 5

# Bookkeeping
# We can store all the samples efficiently by using a 3d-array
X_all = np.zeros((n_datasets, n_samples, 1))
y_all = np.zeros((n_datasets, n_samples, 1))

# We also need to keep track of every model
constant_models = np.zeros((n_datasets, 1, 1))
linear_models = np.zeros((n_datasets, 2, 1))
polynomial_models = np.zeros((n_datasets, poly_degree + 1, 1))

# Sample the datasets uniformly for x in [-1, 1] and fit the models
# with the previously defined functions.
for i in range(n_datasets):
    # TODO
    pass

# Determine the average model for each class by
# averaging over the first axis
# TODO
average_constant_model = None
average_linear_model = None
average_polynomial_model = None

# In order to evaluate the models over the ground truth, we form
# a linearly spaced grid in the target domain
X_eval = np.linspace(-1, 1, 200).reshape((-1, 1))
y_eval = target(X_eval)

# Calculate the bias and variance for each class with X_eval and y_eval
```

```
        # TODO
        bias_constant_models = np.nan
        variance_constant_models = np.nan

        bias_linear_models = np.nan
        variance_linear_models = np.nan

        bias_polynomial_models = np.nan
        variance_polynomial_models = np.nan

        # Print the decomposition
        bias_info = '''Bias:
            Constant models: {:.3f}
            Linear models: {:.3f}
            Polynomial models: {:.3f}'''
        print(bias_info.format(float(bias_constant_models),
                                            float(bias_linear_models),
                                            float(bias_polynomial_models)))

        variance_info = '''Variance:
            Constant models: {:.3f}
            Linear models: {:.3f}
            Polynomial models: {:.3f}'''
        print(variance_info.format(float(variance_constant_models),
                                                    float(variance_linear_models),
                                                    float(variance_polynomial_models)))
```

```
Bias:
    Constant models: nan
    Linear models: nan
    Polynomial models: nan
Variance:
    Constant models: nan
    Linear models: nan
    Polynomial models: nan
```

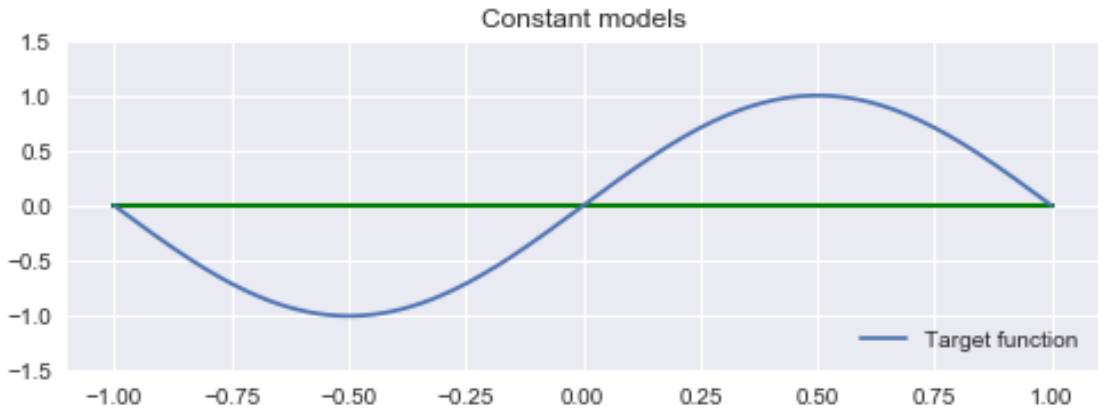
**Q 5.1.1: How and why do the bias and variance differ for the investigated model classes? How do they relate to over- and underfitting terminology?**

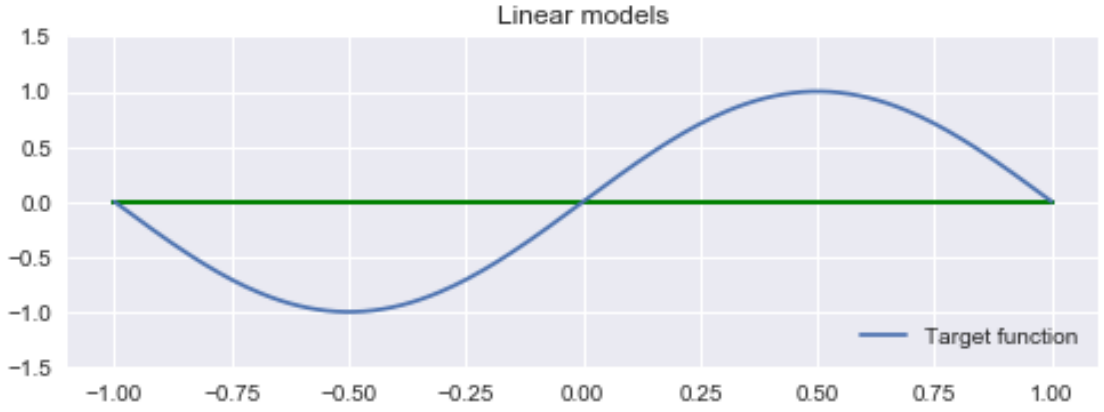
** Q 5.2.2: So which model would you prefer and why? **

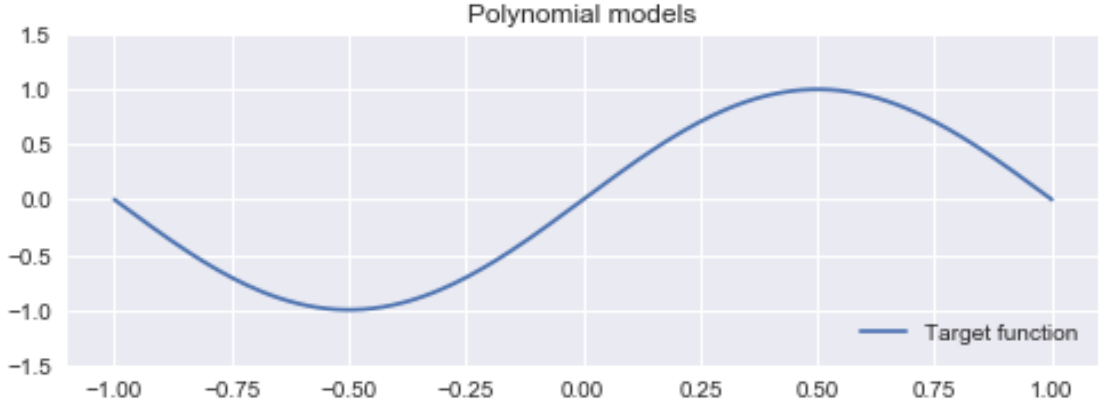
## 0.2 Plotting Code

```
In [3]: %matplotlib inline
        import matplotlib.pyplot as plt
        try: import seaborn as sns
        except ImportError: pass
```

```
# Plotting
fig, axes = plt.subplots(3, 1, figsize=(7, 8))
axes[0].set_title('Constant models')
axes[1].set_title('Linear models')
axes[2].set_title('Polynomial models')

# Plot all the fitted models
for dataset in range(n_datasets):
    constant = constant_models[dataset].flatten()
    axes[0].plot((-1, 1), (constant, constant), c='g', alpha=0.1)

    weights = linear_models[dataset]
    axes[1].plot((-1, 1), (weights[0] - weights[1], weights[0] + weights[1]),
                 c='g', alpha=0.1)
    y_poly = eval_polynomial(X_eval, polynomial_models[dataset])
    if y_poly is not None:
        axes[2].plot(X_eval, y_poly, c='g', alpha=0.1)

# Plot the average models
if average_linear_model is not None:
    axes[1].plot((-1, 1),
        (average_linear_model[0] - average_linear_model[1],
        average_linear_model[0] + average_linear_model[1]),
        c='r', lw=1.3, label='Average model')
if average_constant_model is not None:
    axes[0].plot((-1, 1),
        (average_constant_model.flatten(), average_constant_model.flatten()),
        c='r', lw=1.3, label='Average model')
if average_polynomial_model is not None:
    axes[2].plot(X_eval, eval_polynomial(X_eval, average_polynomial_model),
        c='r', lw=1.3, label='Average model')

for a in axes:
    a.plot(X_eval, target(X_eval), label='Target function')
    a.set_xlim((-1.1, 1.1))
    a.set_ylim((-1.5, 1.5))
    a.legend(loc='lower right')

fig.tight_layout()
```


Constant models
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Target function


Linear models
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Target function


Polynomial models
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Target function