Problem 5:

$$\frac{1}{2}(\hat{\Phi}w-\hat{y})^T(\hat{\Phi}w-\hat{y}) = \frac{1}{2}\begin{pmatrix}\Phi w - y \\ \sqrt{\lambda} I_m w\end{pmatrix}^T \cdot \begin{pmatrix}\Phi w - y \\ \sqrt{\lambda} I_m w\end{pmatrix}$$

$$= \frac{1}{2}(\Phi w - y)^T(\Phi w - y) + \frac{1}{2}(\sqrt{\lambda} I_m w)^T \cdot (\sqrt{\lambda} I_m w)$$

$$= \frac{1}{2}\sum_{i=1}^{N}(\Phi_i w - y_i)^2 + \frac{\lambda}{2}\|w\|_2^2$$

ridge regression

Problem 6: a) with higher degree polynomial, the error always goes down.

b). use validation set, get w from training set and then test on validation set, choose the degree with lowest validation error.

Problem 7:

$$\log p(w,\beta|D) = \log p(w,\beta) + \log(y|\Phi, w, \beta) + \text{constant}$$

$$= \log N(w|m_0, \beta^{-1}S_0)\,\text{Gamma}(\beta|a_0,b_0) + \log\prod_{i=1}^{N} N(y_i|w^T\phi(x_i), \beta)$$

$$= \log\left(\frac{\beta}{2\pi}\right)^{\frac{M}{2}}\exp\left(-\frac{\beta S_0^{-1}}{2}(w-m_0)^T(w-m_0)\right) + \sum_{i=1}^{N}\ln\left[\frac{\beta}{2\pi}\exp\left(-\frac{\beta}{2}(w^T\phi(x_i)-y_i)^2\right)\right]$$

$$+ (a_0-1)\log\beta - b_0\beta + a_0\log b_0 - \log\Gamma(a_0)$$

$$= \frac{M}{2}\log\frac{\beta}{2\pi} - \frac{\beta S_0^{-1}}{2}(w-m_0)^T(w-m_0) + N\ln\frac{\beta}{2\pi} - \frac{\beta}{2}\sum_{i=1}^{N}(w^T\Phi(x_i)-y_i)^2 + (a_0-1)\log\beta$$

$$- b_0\beta + a_0\log b_0 - \log\Gamma(a_0)$$

$$= \left(\frac{M}{2}+N\right)\log\frac{\beta}{2\pi} - \frac{\beta S_0^{-1}}{2}(w^Tw - 2m_0^Tw + m_0^Tm_0) - \frac{\beta}{2}(w^T\Phi^T\Phi w - 2y^T\Phi w + y^Ty) +$$

$$(a_0-1)\log\beta - b_0\beta + a_0\log b_0 - \log\Gamma(a_0) + \text{constant}$$

$$-\frac{\beta S_0^{-1}}{2}m_0^Tm_0$$

$$= \left(\frac{M}{2}+N+a_0-1\right)\log\beta + \beta S_0^{-1}m_0^Tw - \frac{\beta S_0^{-1}}{2}w^Tw - \frac{\beta\Phi^T\Phi}{2}w^Tw + \beta y^T\Phi w - \frac{\beta}{2}y^Ty - b_0\beta$$

$$+ a_0\log b_0 - \log\Gamma(a_0) + \text{constant}$$

$$= \left(\frac{M}{2}+N+a_0-1\right)\log\beta + \left(-\frac{S_0}{2} m_0^T m_0 - \frac{y^T y}{2} - b_0\right)\beta + \left(\beta S_0^{-1} m_0^T + \beta y^T \Phi\right) w - \left(\frac{\cdots}{2} + \frac{\cdots}{2}\right) \cdots$$

$$a_0 \log b_0 - \log\Gamma(a_0) + \text{constant}$$

$\log p(w, \beta \mid D)$

$$= -\frac{\beta S_N^{-1}}{2}(w-m_N)^T(w-m_N) + \frac{M}{2}\log\frac{\beta}{2\pi} + (a_N-1)\log\beta - b_N\beta + a_N\log b_N - \log\Gamma(a_N)$$

$$= \left(\frac{M}{2} + a_N - 1\right)\log\beta + \left(-\frac{S_N^{-1}}{2} m_N^T m_N - b_N\right)\beta + \beta S_N^{-1} m_N^T w - \frac{\beta S_N^{-1}}{2} w^T w + a_N \log b_N - \log\Gamma(a_N)$$

$$\frac{M}{2} + a_N - 1 = \frac{M}{2} + N + a_0 - 1$$

$$a_N = N + a_0$$

$$\begin{cases} -\frac{S_0^{-1}}{2} m_0^T m_0 - \frac{y^T y}{2} - b_0 = -\frac{S_N^{-1}}{2} m_N^T m_N - b_N \\ \beta S_0^{-1} m_0^T + \beta y^T \Phi = \beta S_N^{-1} m_N^T \\ \frac{\beta S_0^{-1}}{2} + \frac{\beta}{2}\Phi^T\Phi = \frac{\beta S_N^{-1}}{2} \end{cases}$$

$$S_0^{-1} + \Phi^T\Phi = S_N^{-1}$$

$$S_N = \left(S_0^{-1} + \Phi^T\Phi\right)^{-1}$$

$$\beta S_0^{-1} m_0^T + \beta y^T \Phi = \beta\left(S_0^{-1} + \Phi^T\Phi\right)^{-1} m_N^T$$

$$m_N = \left(\left(S_0^{-1} + \Phi^T\Phi\right) \cdot \left(S_0^{-1} m_0^T + y^T \Phi\right)\right)^T$$

$$b_N = -\frac{S_N^{-1}}{2} m_N^T m_N + \frac{S_0^{-1}}{2} m_0^T m_0 + \frac{y^T y}{2} + b_0$$

$$a_N = N + a_0$$

Problem 8:

$$E_{ridge}(w) = \frac{1}{2}\sum_{i=1}^{N}(w^T\Phi(x_i) - y_i)^2 + \frac{\lambda}{2}w^Tw$$

$$\nabla_w E_{ridge}(w) = \Phi^T(x)\Phi(x)\cdot w - \Phi^T(x)y + \lambda w$$

$$= (\Phi^T(x)\Phi(x) + \lambda\cdot I)w - \Phi^T(x)y$$

$$\nabla_w E_{ridge}(w) = 0$$

$$(\Phi^T(x)\Phi(x) + \lambda I)w = \Phi^T(x)y$$

$$w = (\Phi^T(x)\Phi(x) + \lambda I)^{-1}\Phi^T(x)y$$

if $N < M$, $\text{rank}(\Phi(x)) \le N$, $\text{rank}(\Phi^T(x)\Phi(x)) \le N$

$\Phi^T(x)\Phi(x) \in R^{M\times M}$ is not invertible.

introducing regularization can make $(\Phi^T(x)\Phi(x) + \lambda I)$ invertible.

Problem 9:

a).

$$\hat{y} = w^{*T}x$$

$$\hat{y} = w_{new}^T x_{new}$$

$$= w_{new}^T a x = w^{*T}x$$

$$w_{new} = \frac{w^*}{a}$$

b).

$$w^* = (X^TX + \lambda I)^{-1}X^Ty$$

$$w_{new}^* = \frac{w^*}{a} = (X_{new}^TX_{new} + \lambda_{new}I)^{-1}X_{new}^Ty$$

$$\frac{(X^T X + \lambda I) \; X^T y}{a} = (a^2 X + \lambda_{new} I)^{-1} a X^T y$$

$$(X^T X + \lambda I)^{-1} X^T y = (X + \frac{\lambda_{new}}{a^2} I)^{-1} X^T y$$

$$\lambda_{new} = a^2 \lambda$$

Programming Task: Linear Regression

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

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
```

Your task

This notebook provides a code skeleton for performing linear regression. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

In the beginning of every function there is docstring which specifies the input and and expected output. Write your code in a way that adheres to it. You may only use plain python and anything that we imported for you above such as numpy functions (i.e. no scikit-learn classifiers).

Load and preprocess the data

In this assignment we will work with the Boston Housing Dataset. The data consists of 506 samples. Each sample represents a district in the city of Boston and has 13 features, such as crime rate or taxation level. The regression target is the median house price in the given district (in $1000's).

More details can be found here: http://lib.stat.cmu.edu/datasets/boston

In [2]:
```
X , y = fetch_california_housing(return_X_y=True)

# Add a vector of ones to the data matrix to absorb the bias term
# (Recall slide #7 from the lecture)
X = np.hstack([np.ones([X.shape[0], 1]), X])
# From now on, D refers to the number of features in the AUGMENTED dataset (i.e. in

# Split into train and test
test_size = 0.9
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
```

Task 1: Fit standard linear regression

In [3]:
```
def fit_least_squares(X, y):
    """Fit ordinary least squares model to the data.

    Parameters
```

```
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Regression targets.

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).

    """
    ### YOUR CODE HERE ###
    return np.linalg.pinv(X) @ y
```

Task 2: Fit ridge regression

In [4]:

```
def fit_ridge(X, y, reg_strength):
    """Fit ridge regression model to the data.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Regression targets.
    reg_strength : float
        L2 regularization strength (denoted by lambda in the lecture)

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).

    """
    ### YOUR CODE HERE ###
    return np.linalg.inv(X.T @ X + reg_strength * np.eye(X.shape[1])) @ X.T @ y
```

Task 3: Generate predictions for new data

In [5]:

```
def predict_linear_model(X, w):
    """Generate predictions for the given samples.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    w : array, shape [D]
        Regression coefficients.

    Returns
    -------
    y_pred : array, shape [N]
```

```
        Predicted regression targets for the input data.

    """
    ### YOUR CODE HERE ###
    return X @ w
```

Task 4: Mean squared error

In [6]:
```
import numpy as np
def mean_squared_error(y_true, y_pred):
    """Compute mean squared error between true and predicted regression targets.

    Reference: `https://en.wikipedia.org/wiki/Mean_squared_error`

    Parameters
    ----------
    y_true : array
        True regression targets.
    y_pred : array
        Predicted regression targets.

    Returns
    -------
    mse : float
        Mean squared error.

    """
    ### YOUR CODE HERE ###
    return np.mean((y_true - y_pred)**2)
```

Compare the two models

The reference implementation produces

- MSE for Least squares $\approx$ **0.5347**
- MSE for Ridge regression $\approx$ **0.5331**

You results might be slightly (i.e. $\pm 1\%$) different from the reference soultion due to numerical reasons.

In [7]:
```
# Load the data
np.random.seed(1234)
X , y = fetch_california_housing(return_X_y=True)
X = np.hstack([np.ones([X.shape[0], 1]), X])
test_size = 0.9
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

# Ordinary least squares regression
w_ls = fit_least_squares(X_train, y_train)
y_pred_ls = predict_linear_model(X_test, w_ls)
mse_ls = mean_squared_error(y_test, y_pred_ls)
```

```python
print('MSE for Least squares = {0}'.format(mse_ls))

# Ridge regression
reg_strength = 1
w_ridge = fit_ridge(X_train, y_train, reg_strength)
y_pred_ridge = predict_linear_model(X_test, w_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print('MSE for Ridge regression = {0}'.format(mse_ridge))
```

```
MSE for Least squares = 0.5347102426013359
MSE for Ridge regression = 0.5912098054500012
```