1. $EPE(f) = E[(Y - X^T\beta)^2]$

$$= E[(Y-X^T\beta)^T(Y-X^T\beta)]$$

$$= E[Y^TY - Y^TX^T\beta - \beta^TXY + \beta^TXX^T\beta]$$

$$\frac{\partial EPE(f)}{\partial \beta} = 0 \Rightarrow E(-(Y^TX^T)^T - XY + 2XX^T\beta) = 0$$

$$E(X(Y-X^T\beta)) = 0$$

$$E(XY - XX^T\beta) = 0.$$

$$E(XY) = E(XX^T)\beta$$

$$\beta = E(XX^T)^{-1}E(XY)$$

2.

$$E_T[(f(\vec{x}_0) - \hat{y}_0)^2] = E_T[(f - \hat{y}_0)^2]$$

$$= E_T(f^2 - 2f\hat{y}_0 + \hat{y}_0^2)$$

$$= f^2 - 2fE(\hat{y}_0) + E_T(\hat{y}_0^2)$$

$$= f^2 + \sigma^2 + 2fE_T(\hat{y}_0) + E_T(\hat{y}_0)^2$$

$$= (f - E_T(\hat{y}_0))^2 + E_T(\hat{y}_0^2) - E_T(\hat{y}_0)^2 + \sigma^2 \qquad (1)$$

Consider $\vec{Y} = f(\vec{X}) + \epsilon. = X^T\vec{\beta} + \epsilon$ $\quad \vec{\beta}$ known. $\quad \epsilon_i \sim N(0, \sigma^2)$

$\hat{Y} = \vec{X}^T\hat{\beta}$ fitted model.

$$r(\hat{\beta}) = (X\hat{\beta} - X\vec{\beta} - \vec{\epsilon})^T(X\hat{\beta} - X\vec{\beta} - \vec{\epsilon})$$

$$= \underline{\hat{\beta}^TX^TX\hat{\beta} - \hat{\beta}^TX^TX\vec{\beta}} - \vec{\beta}^TX^T\epsilon \underline{- \vec{\beta}^TX^TX\hat{\beta}} + \underline{\vec{\beta}^TX^TX\vec{\beta}} + \vec{\beta}^TX^T\epsilon$$

$$- \epsilon^TX\hat{\beta} + \epsilon^TX\vec{\beta} + \vec{\epsilon}^T\vec{\epsilon}.$$

$$= (\hat{\beta} - \vec{\beta})^TX^TX(\hat{\beta} - \vec{\beta}) - 2\vec{\epsilon}^TX(\hat{\beta} - \vec{\beta}) + \vec{\epsilon}^T\vec{\epsilon}$$

$$\frac{\partial r(\hat{\beta})}{\partial \hat{\beta}} = 0 \Rightarrow 2X^TX(\hat{\beta} - \vec{\beta}) - 2X^T\vec{\epsilon} = 0$$

$$\Rightarrow \hat{\vec{\beta}} = \vec{\beta} + (X^TX)^{-1}X^T\vec{\epsilon}$$

$\forall x_0$ $\quad \hat{y}_0 = \vec{x}_0^T \hat{\vec{\beta}} = \vec{x}_0^T \vec{\beta} + \vec{x}_0^T (X^T X)^{-1} X^T \vec{\epsilon}$ ②

Let $l(\vec{x}_0) = X(X^T X)^{-1} \vec{x}_0$ be the ith element of $X(X^T X)^{-1} x_0$.

By equation ①:

$$E_T[(f(\vec{x}_0) - \hat{y}_0)^2] = \underbrace{E(f - E_T(\hat{y}_0))^2}_{\text{bias}^2} + \underbrace{E_T(\hat{y}_0^2) - E_T(\hat{y}_0)^2}_{} + \sigma^2$$

$$\text{bias}^2 = E(f - E_T(\hat{y}_0))^2 = x_0^T \vec{\beta} - x_0^T \vec{\beta} = 0. \quad \text{By ②.}$$

$$\begin{aligned}
E_T(\hat{y}_0^2) - E(\hat{y}_0)^2 = \text{Var}(\hat{y}_0) &= \text{Var}(\vec{x}_0^T \vec{\beta} + \vec{x}_0^T (X^T X)^{-1} X^T \vec{\epsilon}) \\
&= \text{Var}(\vec{x}_0^T (X^T X)^{-1} X^T \vec{\epsilon}) \\
&= \text{Var}\left(\sum_{i=1}^N l_i(\vec{x}_0) \epsilon_i\right) \\
&= \sum_{i=1}^N l_i(\vec{x}_0) \, \text{Var}(\epsilon_i) \\
&= E_T(x_0^T (X^T X)^{-1} x_0) \sigma^2.
\end{aligned}$$

$$\Rightarrow: \quad EPE(x_0) = E_T x_0^T (X^T X)^{-1} x_0 \sigma^2 + \sigma^2.$$

3.

a) $X = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$ $\quad X^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & \cdots & \cdots & x_n \end{bmatrix}$ $\quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$.

In class, we've proved that $\hat{\beta} = (X^TX)^{-1}X^Ty$.

From the form, we know

$\boxed{\text{In linear regression:}}$ $l_i(x_0;\mathcal{X}) = [1 \quad x_0](X^TX)^{-1}\begin{bmatrix} 1 \\ x_i \end{bmatrix}$ for $1 \le i \le n$.

where $X^TX = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}$

When $\det(X^TX) \neq 0$ we can compute the inverse.

In k-nearest neighbors:

$$l_i(x_0;\mathcal{X}) = \begin{cases} \frac{1}{k} & \text{if } x_i \text{ is one of the nearest } k \text{ points.} \\ 0 & \text{otherwise.} \end{cases}$$

b). X is fixed, y varies.

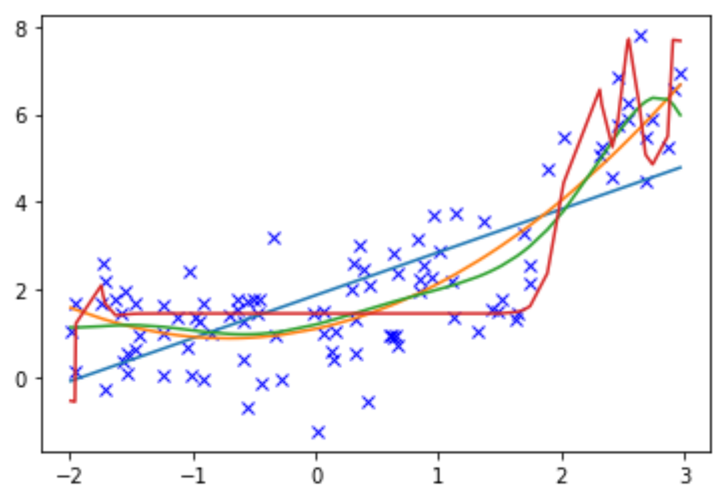
$$E_{y|x}((f(x_0) - \hat{f}(x_0))^2) = f(x_0)^2 - 2f(x_0)E_{y|x}(\hat{f}(x_0)) + E_{y|x}(\hat{f}(x_0)^2)$$

$$= (f(x_0) - E_{y|x}(\hat{f}(x_0)))^2 + E_{y|x}(\hat{f}(x_0)^2) - E_{y|x}(\hat{f}(x_0))^2$$

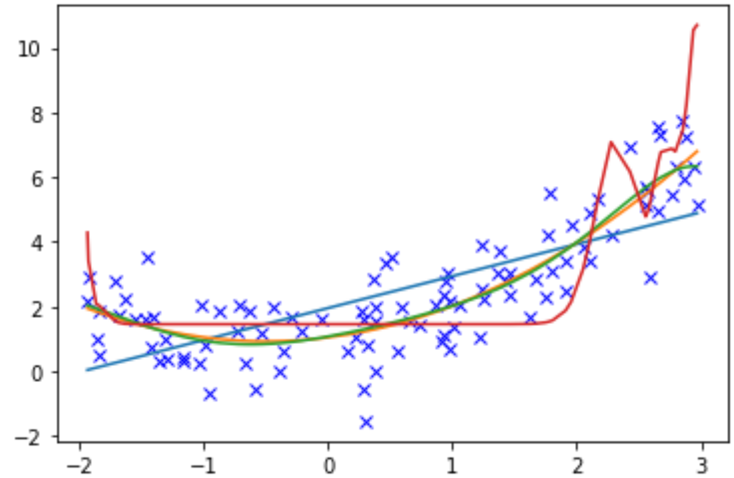
$$= \text{bias}^2 + \text{Var}(\hat{f}(x_0)).$$

In [39]:
```
from numpy.linalg import inv
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
%matplotlib inline
for i in range(3):
    print("In the ", i+1, " trial")
    x = np.random.uniform(0,1,(100,1))*5-2###########          o-1??
    mu, sigma = 0, 1 # mean and standard deviation
    s = np.random.normal(mu, sigma, (100,1))
    Y = 0.5*x*x+0.5*x+1+s
    plt.plot(x,Y,'bx')
    xsort = np.sort(x)
    line = np.array(np.sort(x,axis = None)).reshape(100,1)
    for ii in [1,2,7,50]:
        poly = PolynomialFeatures(ii, include_bias=False)
        tran_X = poly.fit_transform(xsort)
        reg = LinearRegression().fit(tran_X, Y)
        line_poly = poly.transform(line)
        y_hat = reg.predict(line_poly)
        mae = mean_absolute_error(Y, y_hat)
        print("p= ", ii, "mean_absolute_error is", mae)
        plt.plot(line, y_hat)
    plt.show()
```

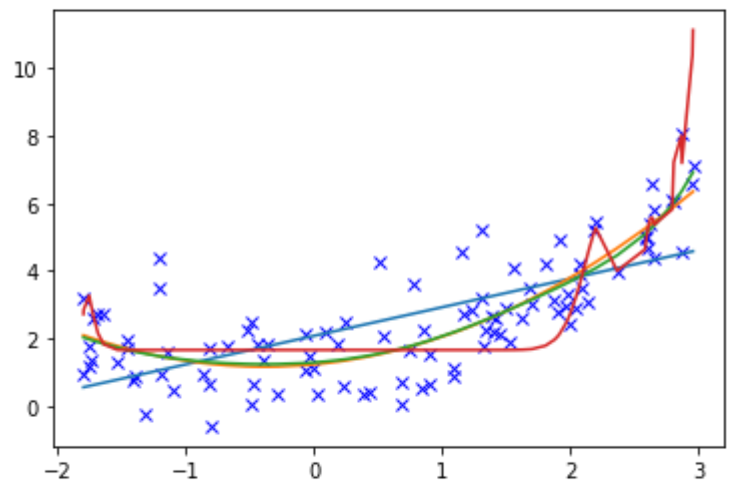
```
In the  1  trial
p=  1 mean_absolute_error is 1.7674505564609846
p=  2 mean_absolute_error is 1.803352402078023
p=  7 mean_absolute_error is 1.811035948348298
p=  50 mean_absolute_error is 1.8273915868630255
```




```
In the  2  trial
p=  1 mean_absolute_error is 1.948030411837093
p=  2 mean_absolute_error is 2.0727259430897997
p=  7 mean_absolute_error is 2.07665520059751
p=  50 mean_absolute_error is 2.0463238109468147
```



```
In the  3  trial
p=  1 mean_absolute_error is 1.7285544609961285
p=  2 mean_absolute_error is 1.8497399443649098
p=  7 mean_absolute_error is 1.852059503268694
p=  50 mean_absolute_error is 1.934517780639168
```



b) From the graph, we see both p=2 and p=3 give us the best fitting result. However, p=7 has downward trend in graph 1, so I pick p = 2 as the best. If I didn't know the true regression function and guess the model from the resulting graph, I'll pick p = 1 as the best beacuse p=1 has the lowest absolute error compared to the last 3 in any trial.

In [ ]: