Bayesian Reasoning I

Problem A

$$p\,(B=t)=0.01, p\,(B=f)=0.99$$

$$p\,(G=t)=\frac{1}{1000000}, p\,(G=f)=1-\frac{1}{1000000}$$

$p(B=t)$ denotes the probability that the blood test is positive, and $p(G=t)$ the probability of being guilty. Therefore,

$$p(B=t|G=t)=1$$

The prosecutor argues the following:

$$p(G=t|B=t)=1-p(B=t|G=f)=1-0.01=.99$$

This is not correct because it implies that:

$$p(B|\neg G)=p(\neg G|B)$$

Problem B

The defendent's arguement would be correct if the accused was picked randomly from the city population. However, the accused was a suspect that was likely selected based on other evidence (e.g. eye witness). Hence, the defendent's arguement is not valid.

Problem C

The judge's conclusion is wrong because the probability value can never be larger than 1. The mistake is in the assumed value of $p(B)=\frac{1}{100}$, given that the blood type occurs in 1% of the city population. However, the search to 10 suspects is more than 1% of the new population that have the bloodtype. Hence, $p(B)\geq\frac{1}{100}$ since the guilty person is among the 10 suspects.

Bayesian Reasoning II

Problem A

Using Bayes Theorem,

$$p(D=1|R=1)=\frac{p(R=1|D=1)p(D=1)}{p(R=1)}$$

$$=\frac{p(R=1|D=1)p(D=1)}{p(R=1|D=0)p(D=0)+p(R=1|D=1)p(D=1)}$$

$$=\frac{\theta\alpha}{(1-\theta)(1-\alpha)+\theta\alpha}$$

if $\alpha=0.001$ and $\theta=0.95$, then:

$$=\frac{(0.95)(0.001)}{(1-0.95)(1-0.001)+(0.95)(0.001)}=\frac{19}{1018}=0.0186640471$$

Problem B

$$p(D=1|R_1=1\&R_2=1)=\frac{p(R_1=1\&R_2=1|D=1)p(D=1)}{p(R_1=1\&R_2=1)}$$

$$=\frac{p(R_1=1|D=1)p(R_2=1|D=1)p(D=1)}{p(R_1=1\&R_2=1|D=0)p(D=0)+p(R_1=1\&R_2=1|D=1)p(D=1)}$$

$$=\frac{\theta^2\alpha}{p(R_1=1|D=0)p(R_2=1|D=0)p(D=0)+p(R_1=1|D=1)p(R_2=1|D=1)p(D=1)}$$

$$=\frac{\theta^2\alpha}{(1-\theta)^2(1-\alpha)+\theta^2\alpha}$$

if $\alpha = 0.001$ and $\theta = 0.95$, then:

$$=\frac{0.95^2(0.001)}{(1-0.95)^2(1-0.001)+0.95^2(0.001))}$$

$$=\frac{361}{1360}=0.26544117647$$

Problem C

The question implies the following:

$$\frac{\theta^2\alpha}{(1-\theta)^2(1-\alpha)+\theta^2\alpha}>\frac{\theta\alpha}{(1-\theta)(1-\alpha)+\theta\alpha}$$

$$\theta>\frac{(1-\theta)^2(1-\alpha)+\theta^2\alpha}{(\theta-\theta^2)(1-\alpha)+\theta^2\alpha}$$

$$1>\frac{(1-\theta)^2(1-\alpha)+\theta^2\alpha}{(\theta-\theta^2)(1-\alpha)+\theta^2\alpha}$$

$$(\theta-\theta^2)(1-\alpha)+\theta^2\alpha>(1-\theta)^2(1-\alpha)+\theta^2\alpha$$

$$(\theta-\theta^2)>(1-\theta)^2$$

$$(\theta-\theta^2)>1-2\theta+\theta^2$$

$$3\theta-2\theta^2>1$$

$$3\theta-2\theta^2-1>0$$

$$-3\theta+2\theta^2+1<0$$

$$(2\theta-1)(\theta-1)<0$$

$$\frac{1}{2}<\theta<1$$

Problem D

If the installed cameras are independent of each other then the result would have the same effect as multiple blood tests. That is:

$$p(D=1|R_1=1\&R_2=1)>p(D=1|R=1)$$

However, if the installed cameras are dependent then the result would be the same as having one camera. That is:

$$p(D=1|R_1=1\&R_2=1)=p(D=1|R=1)$$

Linear Algebra

Problem A

Let x_1 be an eigenvector of $X^T X$ with eigenvalue λ_1 . That is:

$$(X^T X)x_1 \ = \lambda_1 x_1$$

Now

$$(X^T X + \lambda I)x_1 \ = X^T X + \lambda I x_1$$
$$= \lambda_1 x_1 \ + \lambda x_1 \ = (\lambda_1 \ + \lambda)x_1$$

Hence, x_1 is an eigenvector of $(X^T X + \lambda I)$ and $(X^T X)$ with an eigen value of $\lambda_1 \ + \lambda$.

Problem B

let assume x is a non-zero vector.

Let us consider the following:

$$x^T(X^T X + \lambda I)x, x \neq 0$$
$$= x^T X^T X x + x^T \lambda I x$$
$$= x^T X^T X x + \lambda x^T x$$
$$= \|Xx\|^2 + \lambda \|x\|^2$$

Hence, it is suffice to say that $\|Xx\|^2$ can be zero but $\lambda\|x\|^2$ can't be zero. Therefore, $(X^T X + \lambda I)$ is strictly positive i.e. $\lambda_1 \ + \lambda \geq \lambda > 0 \quad \text{since} \quad \lambda_1 \ \geq 0 \quad \text{and} \quad \lambda > 0.$

Problem C

Since, $(X^T X + \lambda I)$ is positive definite, it does not have eigenvalue of zero. That is,

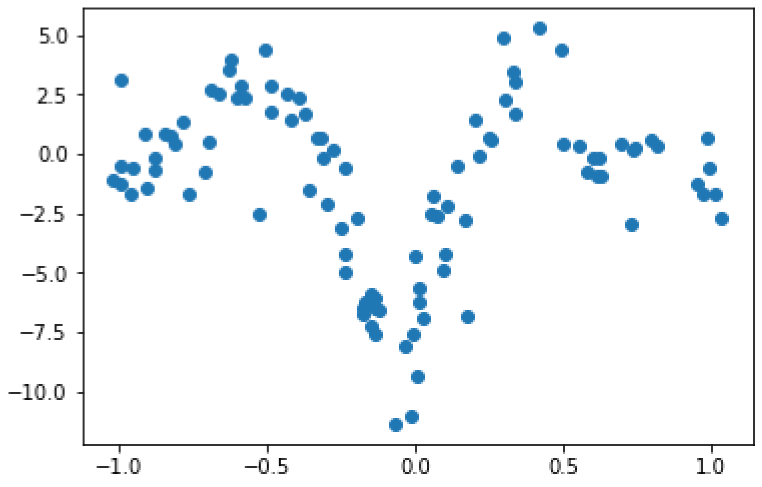
$$det(X^T X + \lambda I) \neq 0$$

Furthermore, the nullspace of $X^T X + \lambda Id$ is empty and it has full rank. Therefore, $(X^T X + \lambda I)$ is invertible.

Linear Regression

Step 1: Load the data

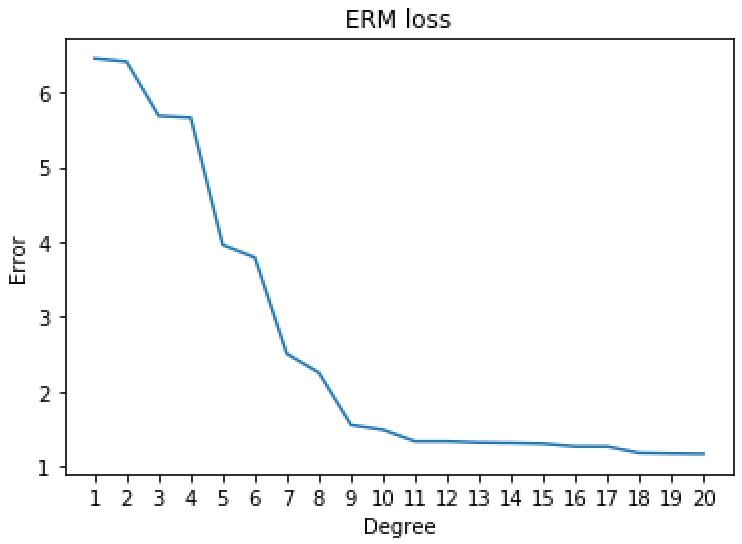
```
In [7]: import matplotlib.pyplot as plt
        import numpy as np
        import math
        from numpy.linalg import inv
        from random import randint
        x = np.loadtxt('dataset1_inputs.txt').reshape(100, 1)
        t = np.loadtxt('dataset1_outputs.txt').reshape(100, 1)
        plt.plot(x, t, 'o')
        plt.show()
```



Step 2: ERM

In [10]:
```
def design_matrix(x, d):
    phi = np.zeros((len(x),d))
    for i in range(len(x)):
        for j in range(d):
            phi[i][j] = x[i][0] ** j
    return phi

def ERM(DM, train_t):
    return np.dot(np.dot(inv(np.dot(np.transpose(DM), DM)),np.transpose(DM)),trai
n_t)

def RLM(DM, train_t, lumda, degree=20):
    return np.dot(np.dot(inv(np.add(np.multiply(np.identity(degree), lumda), np.d
ot(np.transpose(DM), DM))),np.transpose(DM)),train_t)

def calc_error(d_w, val_t):
    return ((d_w - val_t) ** 2)/2

def draw_plot(x, title, ylabel):
    plt.plot(range(1, len(x) + 1), x)
    plt.xticks(np.arange(1, 21, step=1))
    plt.xlabel(ylabel)
    plt.ylabel("Error")
    plt.title(title)
    plt.show()

def fit(train_x, train_t, ftype='ERM', degrees=20, x_val=np.array([]), lumda=None
):
    result = []
    for d in range(1, degrees + 1):
        DM = design_matrix(train_x, d)
        if ftype=='ERM':
            w = ERM(DM, train_t)
        elif lumda != None:
            w = RLM(DM, train_t, lumda=lumda, degree=d)
        else:
            DM = design_matrix(train_x, 20)
            w = RLM(DM, train_t, lumda=math.exp(d*-1), degree=20)

        if x_val.size==0:
            result.append(np.transpose(np.dot(DM, w)).tolist()[0])
        else:
            DS = design_matrix(x_val, d)
            result.append(np.transpose(np.dot(DS, w)).tolist()[0])
    return np.array(result)

def error(train_x,train_t,degrees=20):
    result = []
    for d in range(degrees):
        d_w = train_x[d].reshape(train_x[d].size, 1)
        pow_d = calc_error(d_w, train_t)
        result.append(pow_d.sum() / pow_d.size)
    return np.array(result)
```

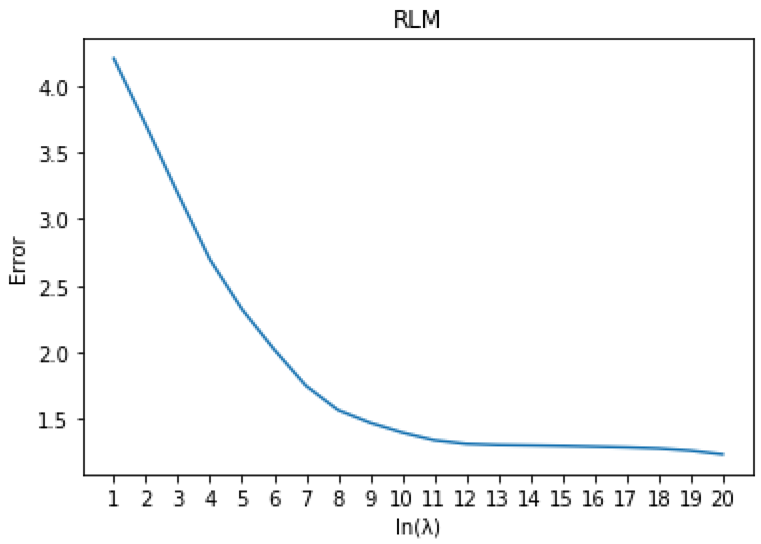
In [3]:
```
result_ERM = fit(x, t)
error_ERM = error(result_ERM, t)
draw_plot(error_ERM, "ERM loss", 'Degree')
```



Polynomial degree of $W \geq 10$ and $W < 13$ would be the most suitable. The loss decreases significantly up to degree 10, followed by slow continues decrease. This would most likely indicate that, for degrees larger than 10, overfitting would likely to occur.

Step 3: RLM

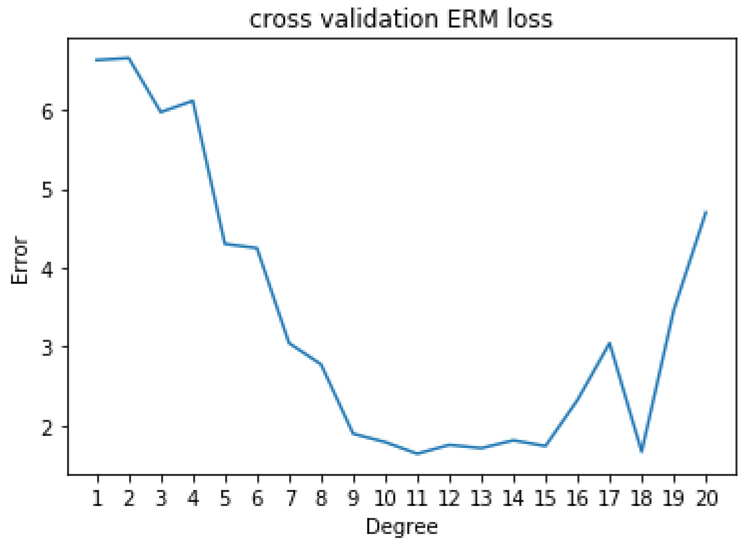
In [12]:
```
result_RLM = fit(x,t,'RLM')
error_RLM = error(result_RLM, t)
draw_plot(error_RLM, 'RLM', 'ln(λ)')
```



While having polynomial degree constant ($W = 20$), we changed the regularizer $ln(\lambda) = -1... - 20$. Compared with the ERM loss figure, the RLM has overall slietly higher error. However, we expect the RLM to be better at preventing for overfitting.

Step 4: Cross Validation

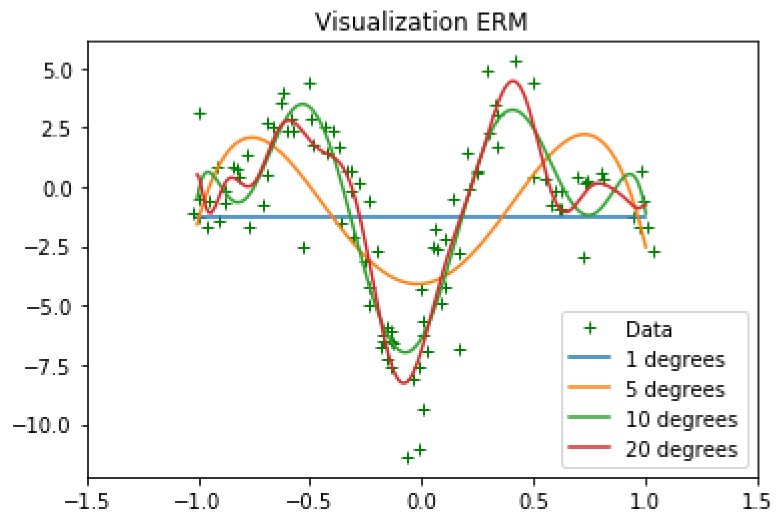
In [5]:
```python
# get random
fold_size = 10
data_perm = np.random.permutation(len(x))
index_split = np.split(data_perm, fold_size)
result_CV = []

for i in range(fold_size):
    train_index_raw = [index_split[ith] for ith in range(fold_size) if ith != i]
    val_index_raw = [index_split[ith] for ith in range(fold_size) if ith == i]
    train_index = np.concatenate(train_index_raw).ravel()
    val_index = np.concatenate(val_index_raw).ravel()
    train_x = np.array([x[ith] for ith in train_index]).reshape(90, 1)
    train_y = np.array([t[ith] for ith in train_index]).reshape(90, 1)
    val_x = np.array([x[ith] for ith in val_index]).reshape(10, 1)
    val_y = np.array([t[ith] for ith in val_index]).reshape(10, 1)

    y_est = fit(train_x, train_y, x_val=val_x)
    result_CV.append(error(y_est, val_y))

result_CV_avg = [np.average(i) for i in zip(*result_CV)]
draw_plot(result_CV_avg, "cross validation ERM loss", 'Degree')
```

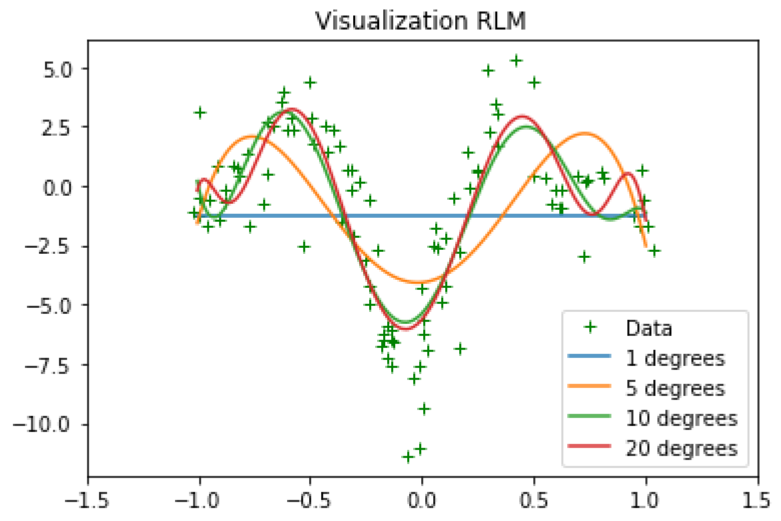


Polynomial degree of $W \geq 10$ and $W < 15$ would be the most suitable. The loss decreases significantly up to degree 9, followed by slow continues decrease. However, at degree $W = 17, 19, and \quad 20$ has higher degree of error compared to the ERM and RLM figures. This could be due to the training data (N=90) were

Step 5: Visualization

In [6]:
```
in1 = x
ou1= t

d_range = [1, 5, 10, 20]
x_ran = 2.01 * np.random.rand(1000, 1) - 1.005
plt.plot(in1, ou1, '+g', label="Data")

for i, d in enumerate(d_range):
    fitted = fit(x, t, degrees=d, x_val=x_ran)
    fitted_dic = dict(zip(x_ran.reshape(1, len(x_ran))[0], fitted[d - 1].reshape(
1, len(fitted[d - 1]))[0]))
    x_fitted, y_fitted = zip(*sorted(fitted_dic.items()))
    plt.plot(x_fitted, y_fitted, label=str(d)+' degrees')

plt.axes().set_title("Visualization ERM")
plt.axes().set_xlim(-1.5, 1.5)
plt.legend(loc='lower right')
plt.show()

plt.plot(in1, ou1, '+g', label="Data")

for i, d in enumerate(d_range):
    fitted = fit(x, t, 'RLM', degrees=d,lumda=.001 ,x_val=x_ran)
    fitted_dic = dict(zip(x_ran.reshape(1, len(x_ran))[0], fitted[d - 1].reshape(
1, len(fitted[d - 1]))[0]))
    x_fitted, y_fitted = zip(*sorted(fitted_dic.items()))
    plt.plot(x_fitted, y_fitted, label=str(d)+' degrees')

plt.axes().set_title("Visualization RLM")
plt.axes().set_xlim(-1.5, 1.5)
plt.legend(loc='lower right')
plt.show()
```





The most suitable degree is 10 for ERM and RLM. However, it is particularly important for ERM because unlike RLM, ERM has higher chance of overfitting. The overfitting could be seen by comparing degree 20 of ERM and RLM. The regularizer prevent it from overfitting. However, making the regularizer drastically big would cause the algorithm to perform worse.