HW 3

- Authors: Marc Brooks, Jack McCarthy, Michael Sarkis
- NetID: mgb45, ___ , ms939

1.) (Hard-thresholding)

(a)

$$\begin{aligned} \mathbb{E}Z1_{|Z|\leq\gamma} &= \int_{-\infty}^{\infty} \frac{z}{\tau\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{z-\omega}{\tau}\right)^2} 1_{|Z|\leq\gamma} dz \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\gamma}^{\gamma} \frac{z}{\tau} e^{-\frac{1}{2}\left(\frac{z-\omega}{\tau}\right)^2} dz \end{aligned}$$

Let $u = \frac{z-\omega}{\tau}$, so $z = u\tau + \omega$ and $du = \frac{1}{\tau}dz$. Substituting these in we get the following:

$$\begin{aligned} \frac{1}{\sqrt{2\pi}} \int_{-\gamma}^{\gamma} \frac{z}{\tau} e^{-\frac{1}{2}\left(\frac{z-\omega}{\tau}\right)^2} dz &= \frac{1}{\sqrt{2\pi}} \left[\int (u\tau + \omega) e^{-\frac{1}{2}u^2} du \right]_{-\frac{\gamma+\omega}{\tau}}^{\frac{\gamma-\omega}{\tau}} \\ &= \frac{1}{\sqrt{2\pi}} \left[\tau \int u e^{-\frac{1}{2}u^2} du + \omega \int e^{-\frac{1}{2}u^2} du \right]_{-\frac{\gamma+\omega}{\tau}}^{\frac{\gamma-\omega}{\tau}} \\ &= \frac{1}{\sqrt{2\pi}} \left[-\tau e^{-\frac{1}{2}u^2} + \frac{\omega\sqrt{\pi}}{\sqrt{2}} \operatorname{erf}\left(\frac{u}{\sqrt{2}}\right) \right]_{-\frac{\gamma+\omega}{\tau}}^{\frac{\gamma-\omega}{\tau}} \\ &= \left[-\frac{\tau}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} + \frac{\omega}{2} \operatorname{erf}\left(\frac{u}{\sqrt{2}}\right) \right]_{-\frac{\gamma+\omega}{\tau}}^{\frac{\gamma-\omega}{\tau}} \end{aligned}$$

Which evaluated gives:

$$\begin{aligned} \mathbb{E}Z1_{|Z|\leq\gamma} = \frac{\omega}{2} \left[\operatorname{erf}\left(\frac{\gamma-\omega}{\tau\sqrt{2}}\right) - \operatorname{erf}\left(\frac{-(\gamma+\omega)}{\tau\sqrt{2}}\right) \right] \\ - \frac{\tau}{\sqrt{2\pi}} \left[\exp\left\{ -\frac{1}{2}\left(\frac{\gamma-\omega}{\tau}\right)^2 \right\} - \exp\left\{ -\frac{1}{2}\left(\frac{-(\gamma+\omega)}{\tau}\right)^2 \right\} \right] \end{aligned}$$

(b)

$$\begin{aligned} \mathbb{E}\left(X - \hat{\mu}_n^H\right)^2 &= \mathbb{E}\left((X-\mu) + (\mu - \hat{\mu}_n^H)\right)^2 \\ &= \mathbb{E}(X-\mu)^2 + \mathbb{E}(\mu - \hat{\mu}_n^H)^2 \\ &= \sigma^2 + \mathbb{E}(\mu - \hat{\mu}_n^H) \\ &= \sigma^2 + \mathbb{E}(\mu^2 - 2\mu\hat{\mu}_n^H + \hat{\mu}_n^{H^2}) \\ &= \sigma^2 + \mu^2 - 2\mu(\mathbb{E}\bar{X}_n - \mathbb{E}\bar{X}_n 1_{|\bar{X}_n|\leq\alpha}) + \mathbb{E}\hat{\mu}_n^{H^2} \\ &= \sigma^2 + \mu^2 - 2\mu^2 + 2\mu\mathbb{E}\bar{X}_n 1_{|\bar{X}_n|\leq\alpha} + \mathbb{E}\hat{\mu}_n^{H^2} \\ &= \sigma^2 - \mu^2 + 2\mu\mathbb{E}\bar{X}_n 1_{|\bar{X}_n|\leq\alpha} + \mathbb{E}\hat{\mu}_n^{H^2} \end{aligned}$$

We can plug the closed form from (a) in for $\mathbb{E}\bar{X}_n 1_{|\bar{X}_n|\le\alpha}$ and evaluate the second moment of $\bar{X}_n 1_{|\bar{X}_n|\ge\alpha}$ to obtain the final expression. To solve for α^{opt}, take the derivative with respect to α and set the resulting expression $= 0$. One can also perform a computational search across a range of α values to identify an approximate optimal value.

(c)

A possible estimator of α^{opt} would be

$$\hat{\alpha}_n = \arg\min_{\alpha}(\text{closed form of } \mathbb{E}\left(X - \hat{\mu}_n^H\right)^2)$$

where μ and σ^2 are replaced with the sample mean and variance.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2. (Great British Bake-off)

In [2]:

```
# importing required packages
import numpy as np
import pandas as pd
import itertools

from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LassoLarsIC, LassoLarsCV, RidgeCV, LinearRegression,
from sklearn.model_selection import cross_val_score, LeaveOneOut

import warnings
warnings.filterwarnings("ignore")
```

(a)

First, since ϵ and X are independent we can see that:

$$\begin{align}
\mathrm{Var}(Y) &= \mathrm{Var}(\mathrm{X}^T\beta^* + \epsilon) \tag{1}\\
&= \mathrm{Var}(\mathrm{X}^T\beta^*) + \mathrm{Var}(\epsilon) \tag{2}\\
&= \beta^{*T}\Sigma(\rho)\beta^* + \sigma^2 \tag{3}
\end{align}$$

Furthermore, we recognize $P(Y - \mathrm{X}^T\beta^*)^2 = P\epsilon^2$ as the expected squared residuals. Since $P\epsilon = 0$ then $P\epsilon^2 = \sigma^2$ and $P(Y - \mathrm{X}^T\beta^*)^2 = \sigma^2$.

Thus,

$$R^2 = 1 - \frac{P(Y - \mathrm{X}^T\beta^*)^2}{\mathrm{Var}(Y)} \tag{4}$$

$$= 1 - \frac{\sigma^2}{\beta^{*T}\Sigma(\rho)\beta^* + \sigma^2} \tag{5}$$

(b)

Setting $R^2 = .8$ we can solve for σ^2 as follows.

$$\begin{aligned}
.8 &= 1 - \frac{\sigma^2}{\beta^{*T}\Sigma(\rho)\beta^* + \sigma^2} &(6)\\
.2 &= \frac{\sigma^2}{\beta^{*T}\Sigma(\rho)\beta^* + \sigma^2} &(7)\\
\sigma^2 &= .2(\beta^{*T}\Sigma(\rho)\beta^* + \sigma^2) &(8)\\
.8\sigma^2 &= .2(\beta^{*T}\Sigma(\rho)\beta^*) &(9)\\
\sigma^2 &= .25(\beta^{*T}\Sigma(\rho)\beta^*) &(10)
\end{aligned}$$

In [3]:

```
labels = ['lasso']*3 + ['adpt lasso']*3 + ['ridge']*3 + ['adpt ridge']*3
tunings = ['AIC', 'BIC', 'LOO-CV']*4
labs = [labels[i] + ' ' + tunings[i] for i in range(len(labels))]
```

In [6]:

```
emse_table = pd.DataFrame(itertools.product(['sparse', 'dense'],
                                            labs,
                                             [10,25,50],
                                             [0,.25,.5]),
             columns=['signal','Method-Tuning', 'p', 'rho'])

emse_table['mse'] = None
```

In [34]:

```
def calc_bic(y_hat, y, D, lamb):
    df_l = np.sum(D**2/(D**2 + lamb))
    rss = np.sum((y - y_hat)**2)
    bic = np.log(rss) + df_l*np.log(len(y))/len(y)
    return bic
```

In [35]:

```
def calc_aic(y_hat, y, D, lamb):
    df_l = np.sum(D**2/(D**2 + lamb))
    rss = np.sum((y.ravel() - y_hat.ravel())**2)
    aic = np.log(rss) + df_l*2/len(y)
    return aic
```

In [36]:

```
def lambda_ic(lambs, X, Y, X_star, Y_star, ic='aic'):
    U, D, V = np.linalg.svd(X, full_matrices=False)
    y_pred = lambda l: (U @ np.diag(D**2/(D**2 + l)) @ U.T) @ Y
    if ic=='bic':
        bic_vals = np.array([calc_bic(y_hat = y_pred(l), y=Y, D=D, lamb=l) for l in la
        min_i = np.argmin(bic_vals)
    else:
        aic_vals = np.array([calc_aic(y_hat = y_pred(l), y=Y, D=D, lamb=l) for l in la
        min_i = np.argmin(aic_vals)
    mse = mean_squared_error(Y, y_pred(lambs[min_i]))
    #beta_hat = np.linalg.inv(X.T @ X + lambs[min_i]*np.eye(p)) @ X.T @ Y
    ridge = Ridge(alpha=lambs[i], fit_intercept=False)
    #omse = mean_squared_error(Y_star, X_star @ beta_hat)
    ridge.fit(X,Y)
    omse = mean_squared_error(Y_star, ridge.predict(X_star))
    return mse, lambs[min_i], omse
```

In [40]:

```
EMSES = np.zeros((24,9))
```

In [64]:

```
from tqdm import tqdm

datasets = 25 #number of loop iterations
n = 100
count = 0
test_size = 1000

# Sparse Signal
lasso_aic_mse_spr = np.zeros(9)
lasso_bic_mse_spr = np.zeros(9)
lasso_loocv_mse_spr = np.zeros(9)
adp_lasso_aic_mse_spr = np.zeros(9)
adp_lasso_bic_mse_spr = np.zeros(9)
adp_lasso_loocv_mse_spr = np.zeros(9)

ridge_aic_mse_spr = np.zeros(9)
ridge_bic_mse_spr = np.zeros(9)
ridge_loocv_mse_spr = np.zeros(9)
adp_ridge_aic_mse_spr = np.zeros(9)
adp_ridge_bic_mse_spr = np.zeros(9)
adp_ridge_loocv_mse_spr =np.zeros(9)

# Dense Signal
lasso_aic_mse_dns = np.zeros(9)
lasso_bic_mse_dns = np.zeros(9)
lasso_loocv_mse_dns = np.zeros(9)
adp_lasso_aic_mse_dns = np.zeros(9)
adp_lasso_bic_mse_dns = np.zeros(9)
adp_lasso_loocv_mse_dns = np.zeros(9)

ridge_aic_mse_dns = np.zeros(9)
ridge_bic_mse_dns = np.zeros(9)
ridge_loocv_mse_dns = np.zeros(9)
adp_ridge_aic_mse_dns = np.zeros(9)
adp_ridge_bic_mse_dns = np.zeros(9)
adp_ridge_loocv_mse_dns = np.zeros(9)

for i,(p,rho) in enumerate(itertools.product([10, 25, 50],[0, 0.25, 0.5])):
    print(p, rho)

    # Generate data
    beta_spr = np.fromfunction(lambda j, _: (2 / np.sqrt(n)) * (j + 1 <= np.sqrt(p)),
    beta_dns = np.fromfunction(lambda j, _: (5 / ((j + 1) * np.sqrt(n))), (p, 1))

    Sigma_p = np.fromfunction(lambda i, j: rho**(np.abs(i - j)), (p, p))
    sigma2_spr = 0.25 * (beta_spr.T @ Sigma_p @ beta_spr)
    sigma2_dns = 0.25 * (beta_dns.T @ Sigma_p @ beta_dns)


    #Generate Test sets for given Rho and p
    test_epsilon_spr = np.random.normal(loc=0, scale=np.sqrt(sigma2_spr))
    test_epsilon_dns = np.random.normal(loc=0, scale=np.sqrt(sigma2_dns))

    test_X = np.random.multivariate_normal(mean=np.zeros(p), cov=Sigma_p, size=test_s

    test_Y_spr =  test_X @ beta_spr + test_epsilon_spr
    test_Y_dns =  test_X @ beta_dns + test_epsilon_dns


    # For LOO-CV
    loo = LeaveOneOut()
```

```
    for _ in range(datasets):

        #To divide out mse sums later
        count += 1

        # Create Dataset
        epsilon_spr = np.random.normal(loc=0, scale=np.sqrt(sigma2_spr))
        epsilon_dns = np.random.normal(loc=0, scale=np.sqrt(sigma2_dns))

        X = np.random.multivariate_normal(mean=np.zeros(p), cov=Sigma_p, size=n)

        Y_spr =  X @ beta_spr + epsilon_spr
        Y_dns =  X @ beta_dns + epsilon_dns

        # For adaptive regressions2.289212      26.952930       0.177174
        lm_spr = LinearRegression().fit(X, Y_spr)
        lm_dns = LinearRegression().fit(X, Y_dns)

        # Adpative Lasso
        # Here we are compute X^T (D^-1)^T
        adX_sprL = X @ np.diagflat(np.abs(lm_spr.coef_))
        adX_dnsL = X @ np.diagflat(np.abs(lm_dns.coef_))

        # Adpative Ridge
        adX_sprR = X @ np.diagflat((lm_spr.coef_**2)**(1/2))
        adX_dnsR = X @ np.diagflat((lm_dns.coef_**2)**(1/2))

        # Adpative Lasso
        # Here we are compute X^T (D^-1)^T
        test_adX_sprL = test_X @ np.diagflat(np.abs(lm_spr.coef_))
        test_adX_dnsL = test_X @ np.diagflat(np.abs(lm_dns.coef_))

        # Adpative Ridge
        test_adX_sprR = test_X @ np.diagflat((lm_spr.coef_**2)**(1/2))
        test_adX_dnsR = test_X @ np.diagflat((lm_dns.coef_**2)**(1/2))


        # Calculate Lassos

        # AIC

        # Sparse
        lasso_aic_spr = LassoLarsIC(criterion='aic')
        lasso_aic_spr.fit(X, Y_spr)
        lasso_aic_mse_spr[i] += mean_squared_error(test_Y_spr, lasso_aic_spr.predict(

        # Dense
        lasso_aic_dns = LassoLarsIC(criterion='aic')
        lasso_aic_dns.fit(X, Y_dns)
        lasso_aic_mse_dns[i] += mean_squared_error(test_Y_dns, lasso_aic_dns.predict(

        #BIC

        # Sparse
        lasso_bic_spr = LassoLarsIC(criterion='bic')
        lasso_bic_spr.fit(X, Y_spr)
        lasso_bic_mse_spr[i] += mean_squared_error(test_Y_spr, lasso_bic_spr.predict(

        # Dense
        lasso_bic_dns = LassoLarsIC(criterion='bic')
        lasso_bic_dns.fit(X, Y_dns)
        lasso_bic_mse_dns[i] += mean_squared_error(test_Y_dns, lasso_bic_dns.predict(

        #LOO-CV
```

```
        # Sparse
        lasso_loo_spr = LassoLarsCV(cv=loo)
        lasso_loo_spr.fit(X, Y_spr)
        lasso_loocv_mse_spr[i] += mean_squared_error(test_Y_spr, lasso_loo_spr.predict

        # Dense
        lasso_loo_dns = LassoLarsCV(cv=loo)
        lasso_loo_dns.fit(X, Y_dns)
        lasso_loocv_mse_dns[i] += mean_squared_error(test_Y_dns, lasso_loo_dns.predict

        # Adaptive Lasso

        # AIC

        # Sparse
        adp_lasso_aic_spr = LassoLarsIC(criterion='aic')
        adp_lasso_aic_spr.fit(adX_sprL, Y_spr)
        adp_lasso_aic_mse_spr[i] += mean_squared_error(test_Y_spr, adp_lasso_aic_spr.p

        # Dense
        adp_lasso_aic_dns = LassoLarsIC(criterion='aic')
        adp_lasso_aic_dns.fit(adX_dnsL, Y_dns)
        adp_lasso_aic_mse_dns[i] += mean_squared_error(test_Y_dns, adp_lasso_aic_dns.p

        # BIC

        # Sparse
        adp_lasso_bic_spr = LassoLarsIC(criterion='bic')
        adp_lasso_bic_spr.fit(adX_sprL, Y_spr)
        adp_lasso_bic_mse_spr[i] += mean_squared_error(test_Y_spr, adp_lasso_bic_spr.p

        # Dense
        adp_lasso_bic_dns = LassoLarsIC(criterion='bic')
        adp_lasso_bic_dns.fit(adX_dnsL, Y_dns)
        adp_lasso_bic_mse_dns[i] += mean_squared_error(test_Y_dns, adp_lasso_bic_dns.p

        # LOO-CV

        # Sparse
        adp_lasso_loo_spr = LassoLarsCV(cv=loo)
        adp_lasso_loo_spr.fit(adX_sprL, Y_spr)
        adp_lasso_loocv_mse_spr[i] += mean_squared_error(test_Y_spr, adp_lasso_loo_spr

        # Dense
        adp_lasso_loo_dns = LassoLarsCV(cv=loo)
        adp_lasso_loo_dns.fit(adX_dnsL, Y_dns)
        adp_lasso_loocv_mse_dns[i] += mean_squared_error(test_Y_dns, adp_lasso_loo_dns

        # Calculate Ridge
        # AIC

        lambda_rng = np.linspace(0,100,num = 200)

        #Sparse
        _,__,mse = lambda_ic(lambda_rng, X, Y_spr, test_X, test_Y_spr, ic='aic')
        ridge_aic_mse_spr[i] += mse

        #Dense
        _,__,mse = lambda_ic(lambda_rng, X, Y_dns, test_X, test_Y_dns, ic='aic')
        ridge_aic_mse_dns[i] += mse

        #BIC
        #Sparse
        _,__,mse = lambda_ic(lambda_rng, X, Y_spr, test_X, test_Y_spr, ic='bic')
        ridge_bic_mse_spr[i] += mse

        #Dense
```

```
        _,__,mse = lambda_ic(lambda_rng, X, Y_dns, test_X, test_Y_dns, ic='bic')
        ridge_bic_mse_dns[i] += mse

        #LOO-CV

        #Sparse
        ridge_loo_spr = RidgeCV(cv = loo)
        ridge_loo_spr.fit(X,Y_spr)
        ridge_loocv_mse_spr[i] += mean_squared_error(test_Y_spr, ridge_loo_spr.predict

        #Dense
        ridge_loo_dns = RidgeCV(cv = loo)
        ridge_loo_dns.fit(X,Y_dns)
        ridge_loocv_mse_dns[i] += mean_squared_error(test_Y_dns, ridge_loo_dns.predict

        #Adaptive Ridge
        # AIC

        #Sparse
        _,__,mse = lambda_ic(lambda_rng,  adX_sprR, Y_spr, test_adX_sprR, test_Y_spr,
        adp_ridge_aic_mse_spr[i] += mse

        #Dense
        _,__,mse = lambda_ic(lambda_rng,  adX_dnsR, Y_dns, test_adX_dnsR, test_Y_dns,
        adp_ridge_aic_mse_dns[i] += mse

        #BIC
        #Sparse
        _,__,mse = lambda_ic(lambda_rng,  adX_sprR, Y_spr, test_adX_sprR, test_Y_spr,
        adp_ridge_bic_mse_spr[i] += mse.

        #Dense
        _,__,mse = lambda_ic(lambda_rng,  adX_dnsR, Y_dns, test_adX_dnsR, test_Y_dns,
        adp_ridge_bic_mse_dns[i] += mse

        # LOO-CV

        #Sparse
        adp_ridge_loo_spr = RidgeCV(cv = loo)
        adp_ridge_loo_spr.fit(adX_sprR,Y_spr)
        adp_ridge_loocv_mse_spr[i] += mean_squared_error(test_Y_spr, adp_ridge_loo_spr

        #Dense
        adp_ridge_loo_dns = RidgeCV(cv = loo)
        adp_ridge_loo_dns.fit(adX_dnsR,Y_dns)
        adp_ridge_loocv_mse_dns[i] += mean_squared_error(test_Y_dns, adp_ridge_loo_dns
```

```
10 0
10 0.25
10 0.5
25 0
25 0.25
25 0.5
50 0
50 0.25
50 0.5
```

In [65]:

```
# Sparse Signal
EMSE = pd.DataFrame(np.r_[lasso_aic_mse_spr[None, :] ,
                            lasso_bic_mse_spr[None, :]       ,
                            lasso_loocv_mse_spr[None, :]     ,
                            adp_lasso_aic_mse_spr[None, :]   ,
                            adp_lasso_bic_mse_spr[None, :]   ,
                            adp_lasso_loocv_mse_spr[None, :] ,
```

```
                             ridge_aic_mse_spr[None, :] ,
                             ridge_bic_mse_spr[None, :] ,
                             ridge_loocv_mse_spr[None, :] ,
                             adp_ridge_aic_mse_spr[None, :] ,
                             adp_ridge_bic_mse_spr[None, :] ,
                             adp_ridge_loocv_mse_spr[None, :] ,
                             # Dense Signal

                             lasso_aic_mse_dns[None, :],
                             lasso_bic_mse_dns[None, :] ,
                             lasso_loocv_mse_dns[None, :] ,
                             adp_lasso_aic_mse_dns[None, :] ,
                             adp_lasso_bic_mse_dns[None, :] ,
                             adp_lasso_loocv_mse_dns[None, :] ,

                             ridge_aic_mse_dns[None, :] ,
                             ridge_bic_mse_dns[None, :] ,
                             ridge_loocv_mse_dns[None, :] ,
                             adp_ridge_aic_mse_dns[None, :] ,
                             adp_ridge_bic_mse_dns[None, :] ,
                             adp_ridge_loocv_mse_dns[None, :]])
```

In [66]:

```
EMSE =  EMSE / datasets
```

In [67]:

```
EMSE.columns = itertools.product([10, 25, 50],[0, 0.25, 0.5])
```

In [68]:

```
EMSE['methods'] = labs*2
```

In [69]:

```
EMSE['signal'] = ['sparse']*12 + ['dense']*12
```

In [70]:

```
EMSE
```

Out[70]:

	(10, 0)	(10, 0.25)	(10, 0.5)	(25, 0)	(25, 0.25)	(25, 0.5)	(50, 0)	(50, 0.25)	(50, 0.5)	methods	sig
0	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	lasso AIC	spa
1	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	lasso BIC	spa
2	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	lasso LOO-CV	spa
3	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	adpt lasso AIC	spa
4	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	adpt lasso BIC	spa
5	0.045534	0.045366	0.075209	0.038493	0.157569	0.117799	0.079710	0.101904	0.277960	adpt lasso LOO-CV	spa
6	0.045061	0.008903	0.031994	0.031058	0.096670	0.043465	0.076087	0.070933	0.210572	ridge AIC	spa
7	0.045061	0.008903	0.031994	0.031058	0.096670	0.043465	0.076087	0.070933	0.210572	ridge BIC	spa
8	0.045543	0.045366	0.075212	0.038485	0.157596	0.117792	0.079679	0.101913	0.277980	ridge LOO-CV	spa
9	0.045061	0.007307	0.031451	0.035534	0.100360	0.043506	0.065929	0.071748	0.086604	adpt ridge AIC	spa

10	0.045061	0.007307	0.031451	0.035534	0.100360	0.043506	0.065929	0.071748	0.086604	adpt ridge BIC	spa
11	0.045817	0.045453	0.075283	0.038487	0.158108	0.117735	0.079800	0.102080	0.277995	adpt ridge LOO-CV	spa
12	0.093783	0.186878	0.373486	0.209297	0.341901	0.478866	0.164044	1.049691	0.338204	lasso AIC	de
13	0.156623	0.286608	0.444831	0.245186	0.400262	0.575146	0.188067	1.097441	0.401711	lasso BIC	de
14	0.087560	0.182817	0.355997	0.160310	0.282328	0.388704	0.103584	0.964066	0.208506	lasso LOO-CV	de
15	0.093783	0.186878	0.373486	0.209297	0.341901	0.478866	0.164044	1.049691	0.338204	adpt lasso AIC	de
16	0.156623	0.286608	0.444831	0.245186	0.400262	0.575146	0.188067	1.097441	0.401711	adpt lasso BIC	de
17	0.087560	0.182817	0.355997	0.160310	0.282328	0.388704	0.103584	0.964066	0.208506	adpt lasso LOO-CV	de
18	0.019132	0.092645	0.281906	0.136210	0.290032	0.164148	0.081073	0.837686	0.201780	ridge AIC	de
19	0.019132	0.092645	0.281906	0.136210	0.290032	0.164148	0.081073	0.837686	0.201780	ridge BIC	de
20	0.087561	0.182820	0.355985	0.160296	0.282305	0.388730	0.103578	0.963970	0.208525	ridge LOO-CV	de
21	0.019132	0.093408	0.293699	0.141134	0.308437	0.197099	0.070249	0.839941	0.161903	adpt ridge AIC	de
22	0.019132	0.093408	0.293699	0.141134	0.308437	0.197099	0.070249	0.839941	0.161903	adpt ridge BIC	de
23	0.088762	0.183744	0.356297	0.164725	0.287393	0.389671	0.110757	0.972016	0.220899	adpt ridge LOO-CV	de

In [56]:

```
EMSE.to_csv('./EMSES.csv', index=False)
```

(c.)

Yes we noticed general patterns. We can see that ridge methods perform better than lasso methods on dense data. While it is difficult to distinguish ridge from lasso on sparse data theoretically we know that lasso should outperform ridge. Due to time limitations we simulated 25 datasets rather than 1000 datasets. It is possible that the smaller number of datasets used to estimate our expected MSE has added noise to our results.

Another thing to note is that as the number of predictors increases the MSE for adaptive ridge increases by a much smaller rate than other methods.

In [ ]:

```

```