sheet06_main

December 5, 2022

1 Sheet 6

[ ]:
```python
import numpy as np

from matplotlib import pyplot as plt
plt.rc('font', family='monospace', size=14, serif='courier')
plt.rc('mathtext', fontset='stix')

import scipy.sparse
from sklearn.linear_model import Ridge, LinearRegression

from IPython.display import Image
```

2 1. Regularization and Basis

[ ]:
```python
Image(filename='Sheet06-1.jpg')
```

[ ]:

1.) Regularization & Basis

Given: $X = (1, x_0, x_1)^T$ & $\beta = (\beta_0, \beta_1, \beta_2)^T$

a.) SSQ Loss fxn $\rightarrow \mathcal{L}_{OLS} = (y - \beta^T x)(y - \beta^T x)^T = \sum_i^N (y_i - \beta x_i)^2$

Ridge loss fxn $\rightarrow \mathcal{L}_{Ridge} = (y - \beta^T x)(y - \beta^T x)^T + \lambda \beta^T \beta$

$$= \|y - \beta^T x\|_2^2 + \lambda \|\beta\|_2^2$$

$$= \underbrace{\sum_i^N \sum_j^p (y_i - \beta_j x_i)^2}_{\sum_i^N (y_i - (\beta_0 + \beta_1 x_1 + \beta_2 x_2))^2} + \underbrace{\lambda . [\sum_j^p (\beta_j)^2]}_{\lambda(\beta_0^2 + \beta_1^2 + \beta_2^2)}$$

— λ (regularization strength) scales β_0.

b.) We can do that in two ways:

1.) By not penalizing the intercept only:

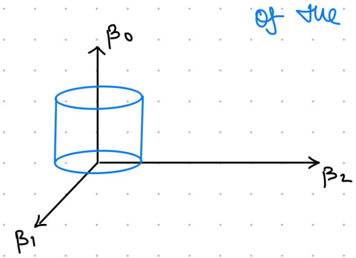
$\therefore \mathcal{L} = SSQ(\beta) + \lambda \|\beta\|_2^2 \longrightarrow \mathcal{L}^* = SSQ(\beta_0, \beta') + \lambda \|\beta'\|_2^2$

where, $\beta' = (\beta_1, \beta_2)^T$

2.) By centering data (Y) about 0.

c.) For (a) case: sphere

For (b) case: cylinder where the circle is in (β_1, β_2) plane & height of the cylinder is given by value β_0 (const.)



3 2. Estimating Parameter Relevance

[ ]:
```
# load the data
with open('data/vostok.txt', 'r') as f:
    lines = f.readlines()

# remove header and split lines
lines = [l.split() for l in lines[2:]]

# filter out lines with missing data
lines = [l for l in lines if len(l) == 4]

# convert to float
lines = np.array(lines).astype(np.float32)
print(f'{lines.shape=}')

features = np.concatenate([lines[:, :1], lines[:, 2:]], axis=1).T
feature_names = 'age', 'CO ', 'dust'
labels = lines[:, 1]
label_name = ' T'

print(f'{features.shape=}, {labels.shape=}')
```

```
lines.shape=(3729, 4)
features.shape=(3, 3729), labels.shape=(3729,)
```

[ ]:
```
# TODO: fit the linear regressor and compute the sum of square deviations\

model =  LinearRegression()   ##first creating a model
model.fit(features.T,labels)   ##this will fit the model with the input␣
 ↪features.T and target variable labels

y_pred = model.predict(features.T) ### this is the predicted labels


def squared_residuls(pred):
    return np.sum(np.square(pred-labels))
square_error_base = squared_residuls(y_pred)
print(" The sum of squared residuals(unperturbed rows) is = "␣
 ↪+str(square_error_base))
```

```
 The sum of squared residuals(unperturbed rows) is = 6362.9375
```

[ ]:
```
# TODO: for each feature, randomly permute it amongst the samples,
print(" The sum of squared residuals(unperturbed rows) is = "␣
 ↪+str(square_error_base))
def permutation(i):
```

```
    """
    Input : takes the three values of the particular row i =[0,1,2]
    output: return the feature matrix with ith row permuted

    """
    permutation_array = np.arange(0,len(labels)) ##this will create the index␣
 ↪array ranging from 0 to N-1

    np.random.shuffle(permutation_array)
    X_i = np.array(features)

    X_i[i,:] = features[i,:][permutation_array]   ##permuting the ith row

    return  X_i

#       refit the regressor and compte sum of squared deviations
square_res_arr = []
for i in range(3):
    X = permutation(i)
    model.fit(X.T,labels)
    y_pred = model.predict(X.T)
    epsilon = squared_residuls(y_pred)
    square_res_arr.append(epsilon)
print("SSQ for only first,second and third row perturbed is respectively ",␣
 ↪square_res_arr  )
```

```
 The sum of squared residuals(unperturbed rows) is = 6362.9375
SSQ for only first,second and third row perturbed is respectively  [6817.659,
19161.434, 6528.004]
```

3.0.1 From the values we see that second feature is most relevant because it posses the large SSQ compared to others. While the third feature is least relevant.

4 3. σ^2 Estimation

[ ]:
```
Image(filename='Sheet06-2.jpg')
```

[ ]:

3.) a.) Maximum Likelihood:

$$\hat{\beta} = \underset{\beta}{\arg\max} \sum_{n=1}^{N} \log \mathcal{N}(y_n | \beta^T x, \sigma^2)$$

$$\mathcal{N}(y_n | \beta^T x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y_n - \beta^T x_n)^2}{2\sigma^2}\right) \quad -①$$

$$\Rightarrow \sum_{n=1}^{N} \log_e\left(\mathcal{N}(y_n | \beta^T x_n, \sigma^2)\right) = \underbrace{-\frac{N}{2}\ln(2\pi\sigma^2)}_{\text{const.}} - \frac{1}{2\sigma^2} \underbrace{\sum_{n=1}^{N} (y_n - \beta^T x_n)^2}_{SSQ(\beta)} \quad -②$$

one way,

$$\therefore \hat{\beta} = \underset{\beta}{\arg\max}\left(-\frac{1}{2\sigma^2} \cdot SSQ(\beta)\right) \quad -③$$

[as const. vanishes in the argmax f^{x^n}]

now, if we take $\sigma = 1$, then eq^n ③ becomes,

$$= \underset{\beta}{\arg\max}\left(-\frac{1}{2} SSQ(\beta)\right) \Rightarrow \boxed{\hat{\beta} = \underset{\beta}{\arg\min}\left(SSQ(\beta)\right)}$$

same sol. for β

another way

if now, $\frac{\partial}{\partial \beta}\left(-\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \beta^T x_n)^2\right) = 0$

$$\Rightarrow \frac{1}{\sigma^2}\sum_{n=1}^{N} x_n^T (y_n - \beta^T x_n) = 0 \Rightarrow \sum_{n=1}^{N} x_n^T y_n - \sum_{n=1}^{N} x_n^T \beta^T x_n = 0$$

$$\Rightarrow X^T Y = X^T \beta^T X$$

$$\Rightarrow \boxed{X \cdot Y^T = X \cdot X^T \cdot \beta}$$

→ normal eq^n sol. which is same for SSQ.

$$\to \hat{\beta} = (X \cdot X^T)^{-1} X \cdot Y^T$$

→ $\hat{\beta}$ is an unbiased estimator of β

```
[ ]: Image(filename='Sheet06-3.jpg')
```

[ ]:

b.) Estimation of σ^2

$$\hat{\sigma}^2 = \underset{\sigma^2}{\operatorname{argmax}} \sum_{n=1}^{N} \log \mathcal{N}(y_n | \hat{\beta}^T x, \sigma^2)$$

$$\sum_{n=1}^{N} \log \mathcal{N}(y_n | \hat{\beta}^T x, \sigma^2) = -\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2$$

$$\frac{\partial}{\partial \sigma^2}\left(\sum_{n=1}^{N} \log \mathcal{N}(y_n | \hat{\beta}^T x, \sigma^2)\right) = 0 \rightarrow \text{for } \hat{\sigma}^2$$

$$\Rightarrow 0 - \frac{N}{2} \cdot \frac{1}{\sigma^2} - \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2 \frac{\partial}{\partial \sigma^2}(1/\sigma^2) = 0$$

$$\Rightarrow \frac{-N}{2\sigma^2} + \frac{1}{2(\sigma^2)^2} \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2 = 0$$

$$\Rightarrow \left(\frac{1}{\sigma^2} \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2 - N\right) \frac{1}{2\sigma^2} = 0$$

$$\Rightarrow \frac{1}{\sigma^2} \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2 = N \Rightarrow \boxed{\hat{\sigma}^2 = \frac{1}{N} \cdot \sum_{n=1}^{N} (y_n - \hat{\beta}^T x_n)^2} \quad \text{Ans}$$

$$= \frac{1}{N} (Y - \hat{Y})$$

$$= \frac{1}{N} [Y(I - H)]$$

$$\boxed{\hat{\sigma}^2 = \frac{1}{N} \mathcal{E}_{SSQ}} \quad \text{Ans}$$

4.1 4 Visualize Regularization Contours

[ ]:
```
# load the data
data = np.load('data/linreg.npz')
x = data['X']
y = data['Y']
print(f'{x.shape} {y.shape}')
```

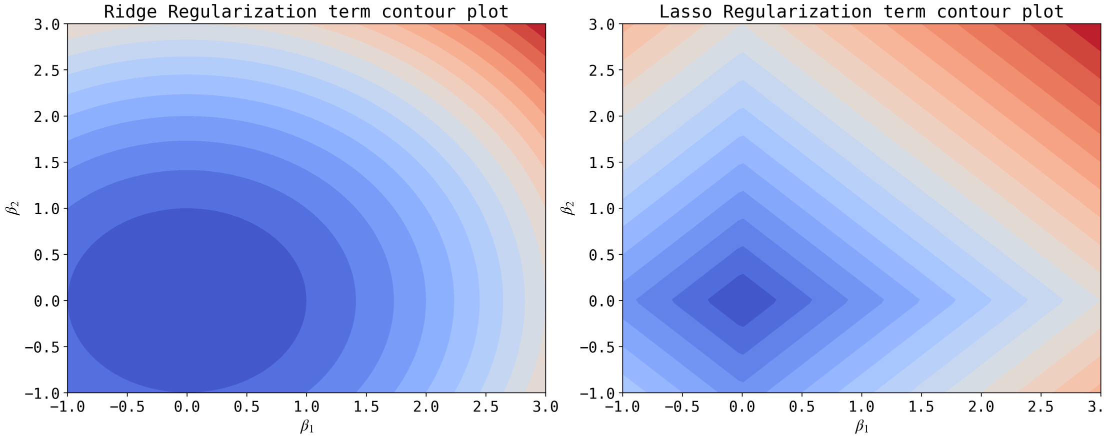
```
(2, 100) (1, 100)
```

[ ]:
```
# TODO: create a grid of points in the parameter space
xlist = np.linspace(-1, 3.0, 100)  ##creating xlists and ylists from -1 to 3␣
 ↪and taking 100 points in between
ylist = np.linspace(-1, 3.0, 100)   ##creating ylists as above
beta1, beta2 = np.meshgrid(xlist, ylist)  ##creating meshgrid
```

(a)

[ ]:
```
# TODO: make coutour plots for ridge and lasso regularization terms
fig,ax=plt.subplots(1,2,figsize=(15,6),dpi=300)
fig.tight_layout(pad=2)

Z_ridge = beta1**2 + beta2**2  ###calculating ridge regularization term

ax[0].contourf(beta1, beta2, Z_ridge,20, cmap = 'coolwarm',)
ax[0].set_title("Ridge Regularization term contour plot")
ax[0].set_xlabel(r'$\beta_1$',size = 15)
ax[0].set_ylabel(r'$\beta_2$',size  = 15)

##plotting for lasso contour

Z_lasso = np.abs(beta1) + np.abs(beta2)   ###calculating ridge regularization␣
 ↪term

ax[1].contourf(beta1,beta2, Z_lasso,20, cmap = 'coolwarm')
ax[1].set_title("Lasso Regularization term contour plot")
ax[1].set_xlabel(r'$\beta_1$',size = 15)
ax[1].set_ylabel(r'$\beta_2$',size  = 15)
```

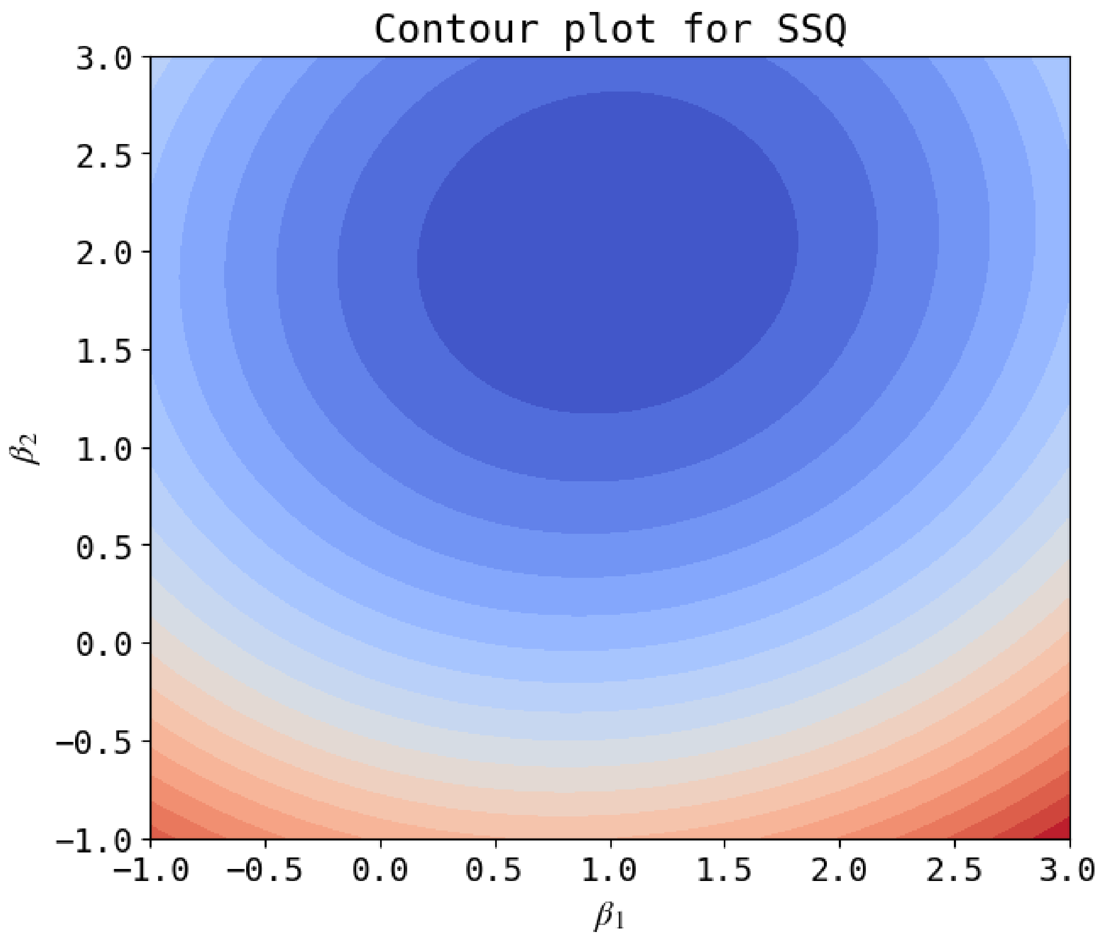
[ ]:
```
Text(2313.5643939393935, 0.5, '$\\beta_2$')
```



(b)

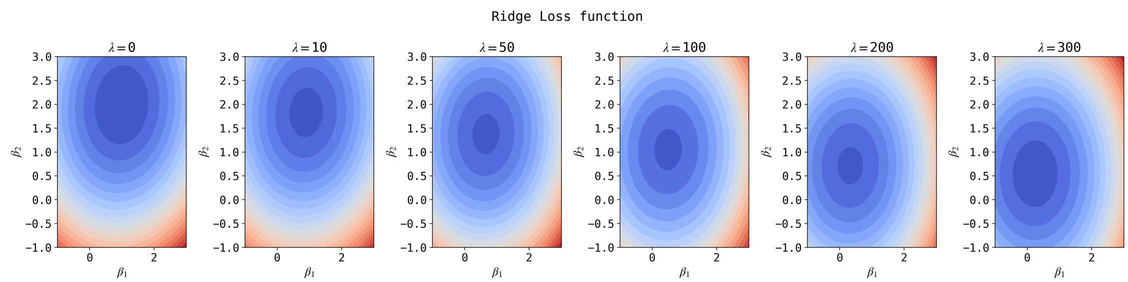
```
[ ]: # TODO: for each combination of parameters, compute the sum of squared␣
     ↪deviations.
     #       do not use loops, but numpy broadcasting!

     ##first we will convert the meshgrid arrays into an array whose first column is␣
     ↪beta1 and second column is beta2

     coordinate_arr = np.array([beta1.reshape(-1),beta2.reshape(-1)]).T  ##.
     ↪rehshape(-1) is used to flatten the array

     ## then we need to multiply it by the X matrix
     predict_y = coordinate_arr@x  ## each column is now a predicted value for y for␣
     ↪a particular value for beta1 and beta2
     ssq = np.sum(np.square((y.flatten()-predict_y)),axis = 1)
     ssq = ssq.reshape(beta1.shape)  ##changing ssq shape to be shame as grid

     # TODO: make a coutour plot for sum of squared deviations
     plt.figure(figsize=(7,6))
     plt.contourf(beta1, beta2,ssq,20, cmap = 'coolwarm',)
     plt.xlabel(r'$\beta_1$',size = 15)
     plt.ylabel(r'$\beta_2$',size  = 15)
     plt.title("Contour plot for SSQ")
```

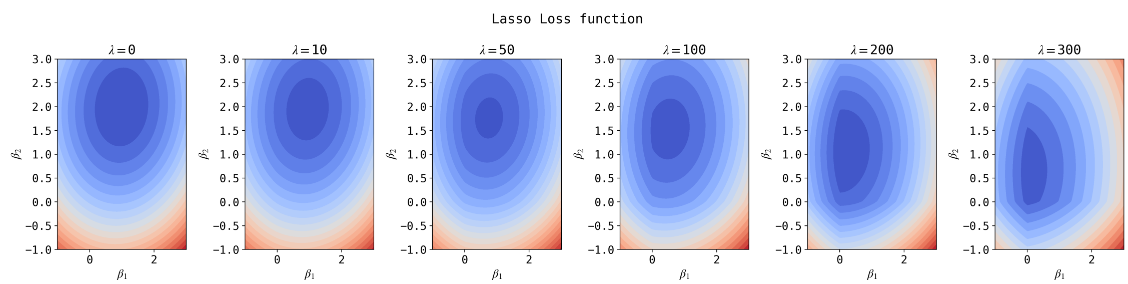
```
[ ]: Text(0.5, 1.0, 'Contour plot for SSQ')
```



4.C

[ ]:
```python
# TODO: for each lambda, plot both ridge regression and lasso loss functions
lambdas = [0, 10, 50, 100, 200, 300]
loss = {"Ridge":Z_ridge,"Lasso":Z_lasso}
for i in loss:
    fig,ax=plt.subplots(ncols = len(lambdas),figsize = (20,5),dpi=300)
    for j in range(len(lambdas)):
        Z = ssq+lambdas[j]*loss[i]
        cp = ax[j].contourf(beta1, beta2, Z,20, cmap = 'coolwarm')
        ax[j].set_title(r'$\lambda = $' + str(lambdas[j]))
        ax[j].set_xlabel(r'$\beta_1$',size = 15)
        ax[j].set_ylabel(r'$\beta_2$',size  = 15)
    plt.suptitle(i+" Loss function")
    plt.tight_layout()
    plt.show()
```





4.2 CT

set up design matrix (run this once to save to disk)

[ ]:
```
# create design matrix
# don't change any of this, just run it once to create and save the design matrix
import os

if not os.path.exists('data/design_matrix.npy'):
    res = (99, 117)
    xs = np.arange(0, res[1]+1) - res[1]/2 # np.linspace(-1, 1, res[1] + 1)
    ys = np.arange(0, res[0]+1) - res[0]/2 #np.linspace(-1, 1, res[0] + 1)

    # rays are defined by origin and direction
    n_parallel_rays = 70
    ray_offset_range = [-res[1]/1.5, res[1]/1.5]
    n_ray_angles = 30
    n_rays = n_parallel_rays * n_ray_angles

    ray_angles = np.linspace(0, np.pi, n_ray_angles, endpoint=False) + np.pi/n_ray_angles

    # offsets for ray_angle = 0, i.e. parallel to x-axis
```

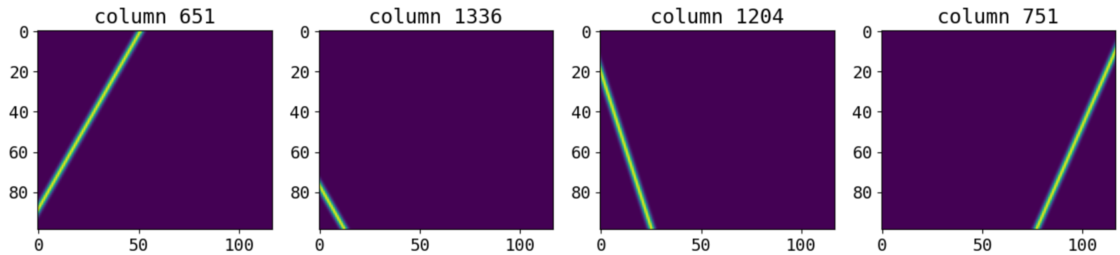
```
    ray_0_offsets = np.stack([np.zeros(n_parallel_rays), np.
 ↪linspace(*ray_offset_range, n_parallel_rays)], axis=-1)
    ray_0_directions = np.stack([np.ones(n_parallel_rays), np.
 ↪zeros(n_parallel_rays)], axis=-1)

    def rot_mat(angle):
        c, s = np.cos(angle), np.sin(angle)
        return np.stack([np.stack([c, s], axis=-1), np.stack([-s, c],␣
 ↪axis=-1)], axis=-1)

    ray_rot_mats = rot_mat(ray_angles)

    ray_offsets = np.einsum('oi,aij->aoj', ray_0_offsets, ray_rot_mats).
 ↪reshape(-1, 2)
    ray_directions = np.einsum('oi,aij->aoj', ray_0_directions, ray_rot_mats).
 ↪reshape(-1, 2)

    sigma = 1
    kernel = lambda x: np.exp(-x**2/sigma**2/2)

    xsc = (xs[1:] + xs[:-1]) / 2
    ysc = (ys[1:] + ys[:-1]) / 2
    b = np.stack(np.meshgrid(xsc, ysc), axis=-1).reshape(-1, 2)
    a = ray_offsets
    v = ray_directions
    v = v / np.linalg.norm(v, axis=-1, keepdims=True)
    p = ((b[None] - a[:, None]) * v[:, None]).sum(-1, keepdims=True) * v[:,␣
 ↪None] + a[:, None]
    d = np.linalg.norm(b - p, axis=-1)
    d = kernel(d)
    design_matrix = d.T

    np.save('data/design_matrix.npy', design_matrix)
    print(f'created and saved design matrix of shape {design_matrix.shape} at␣
 ↪data/design_matrix.npy')
```

```
created and saved design matrix of shape (11583, 2100) at data/design_matrix.npy
```

(a)

[ ]:
```
design_matrix = np.load('data/design_matrix.npy')

# TODO: visualize four random columns as images, using an image shape of (99,␣
 ↪117)
img_shape = (99, 117)

fig, axs = plt.subplots(1, 4, figsize=(16, 4))
```

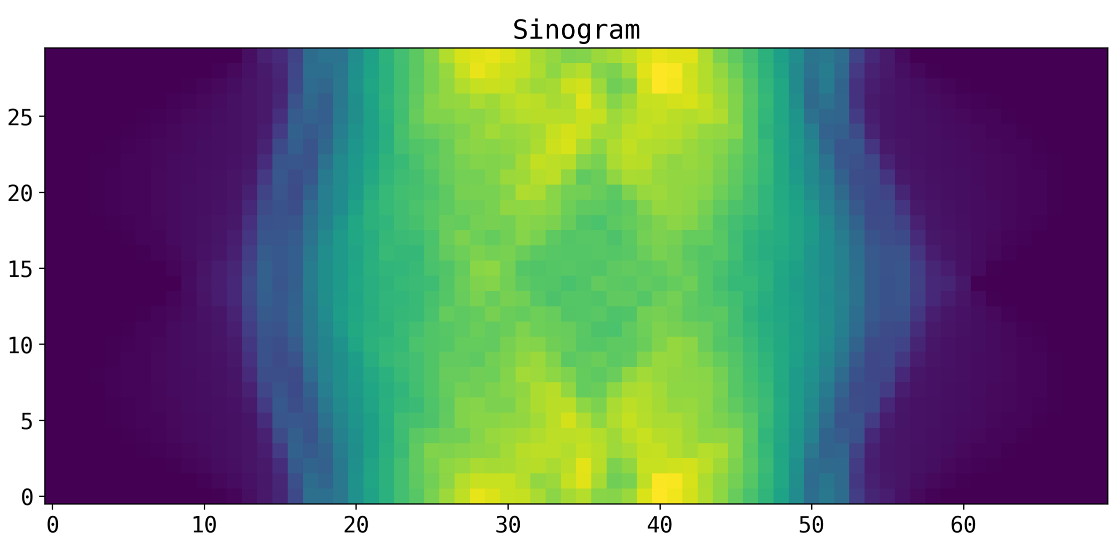
```
for i, ax in zip(np.random.choice(np.arange(design_matrix.shape[1]), 4), axs):
    ax.imshow(design_matrix[:, i].reshape(*res));
    ax.set_title(f'column {i}')
```



Interpretation of the column of the Matrix X To be honest , we could not make any sense from the images but based on the definition given in the exercise we conclude that a particular column represents a detector reading at one specific angle while the object is scanned longitudinally.

[ ]:
```
sino = np.load('data/sino.npy')

# visualize sinogram as image
n_parrallel_rays = 70
n_angles = 30

plt.figure(figsize=(12,6),dpi=300)
plt.imshow(sino.reshape(n_angles, n_parallel_rays), origin='lower');
plt.title('Sinogram')
```

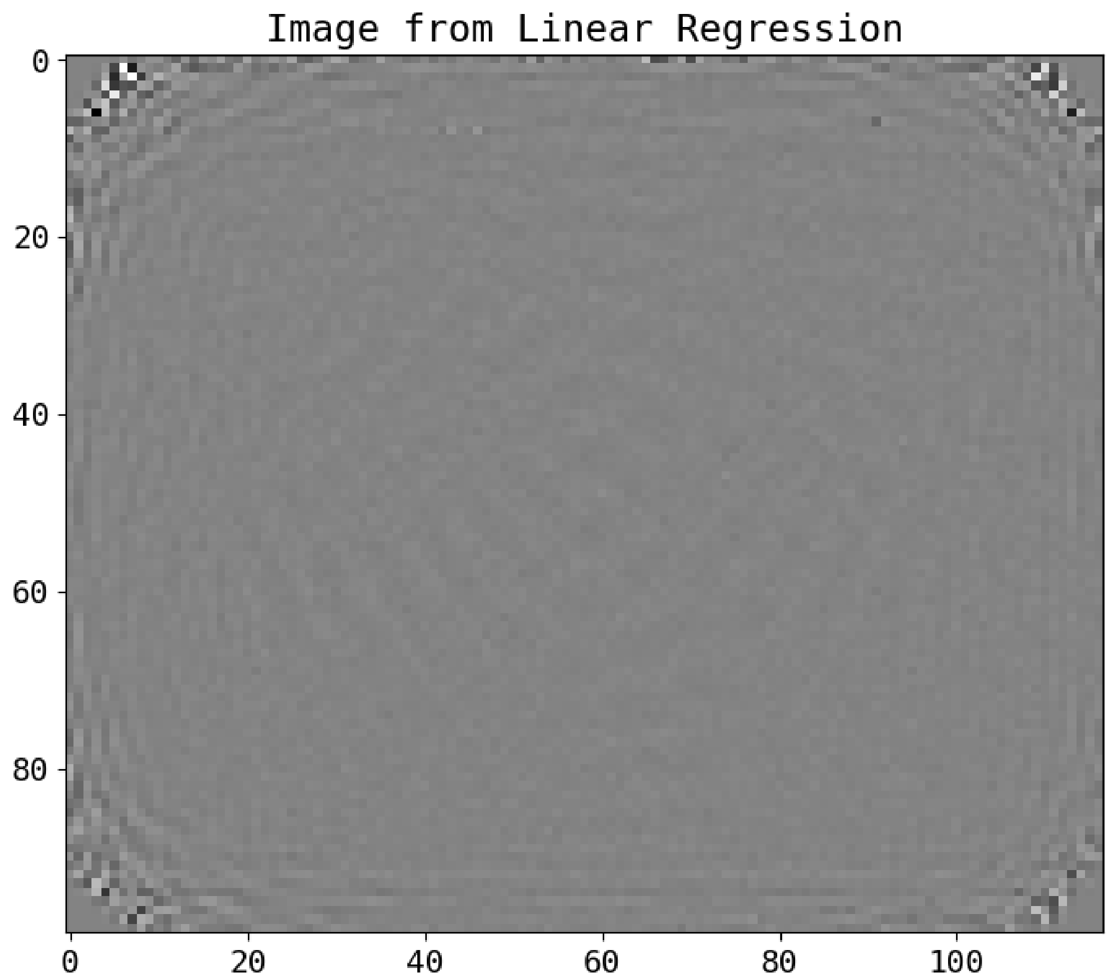
[ ]: Text(0.5, 1.0, 'Sinogram')



(b)

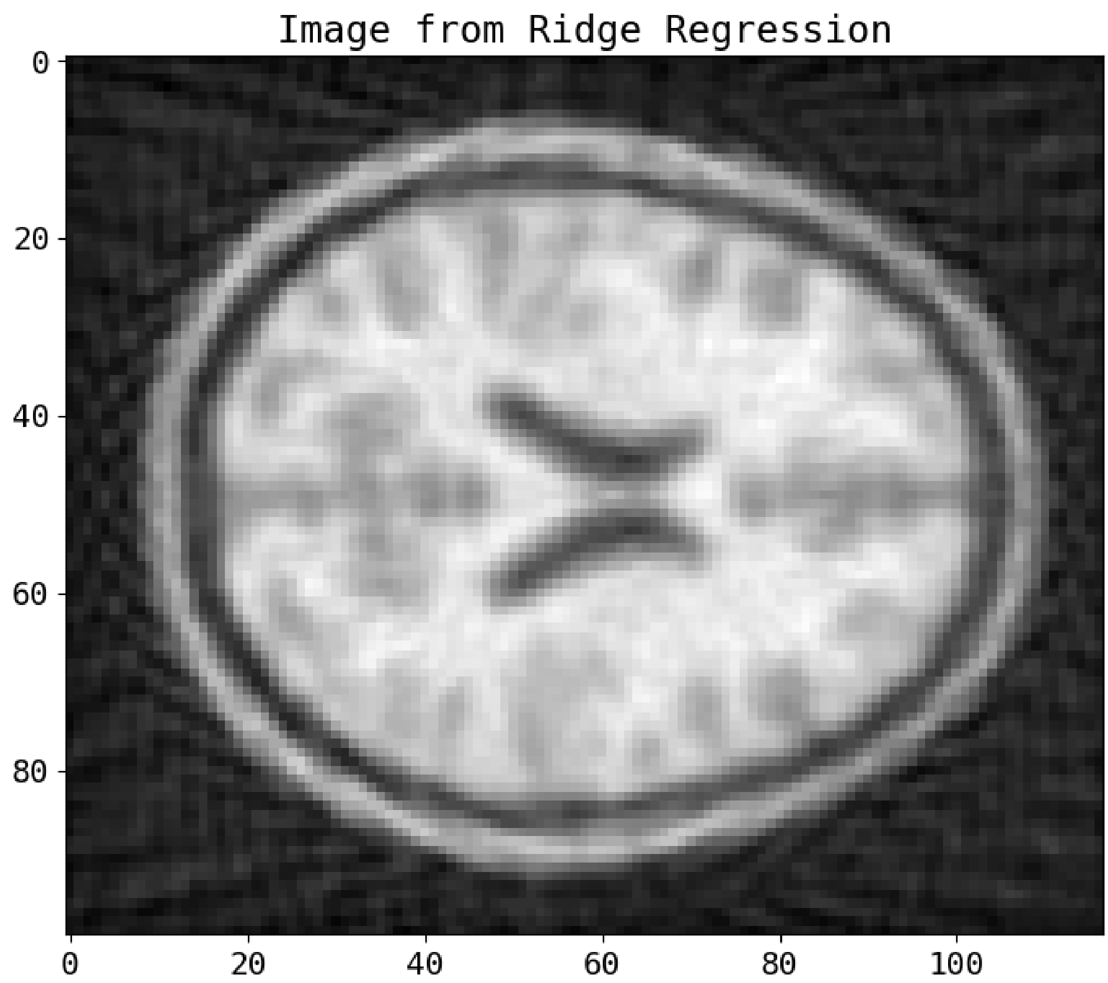
```
[ ]: # TODO: solve the reconstruction with linear regression and visualize the result
     # We are implementing scikit learn Linear Regression for this
     model = LinearRegression()  ##initialing the regression model
     model.fit(design_matrix.T,sino.reshape(-1))   ##fitting the  model for the  data
     image_LR =  model.coef_.reshape(*res)  ##getting the coefficients and␣
      ↪converting their shape to 99*117

     ##plotting the image
     plt.figure(figsize=(10,7))
     plt.imshow(image_LR,cmap = 'gray');
     plt.title("Image from Linear Regression")
```

```
[ ]: Text(0.5, 1.0, 'Image from Linear Regression')
```



[ ]:
```
# TODO: solve the reconstruction with ridge regression and visualize the result
clf = Ridge(alpha=1.0)  ##creating the ridge regression model with strength  = 1
clf.fit(design_matrix.T,sino.reshape(-1))  ##fitting the model with the input␣
 ↪data
image_ridge  =   clf.coef_.reshape(*res)  ##getting the coefficient and␣
 ↪changing their shape as (99,117)

##plotting the image
plt.figure(figsize=(10,7))
plt.imshow(image_ridge,cmap = 'gray');
plt.title("Image from Ridge Regression")
```

[ ]:
```
Text(0.5, 1.0, 'Image from Ridge Regression')
```

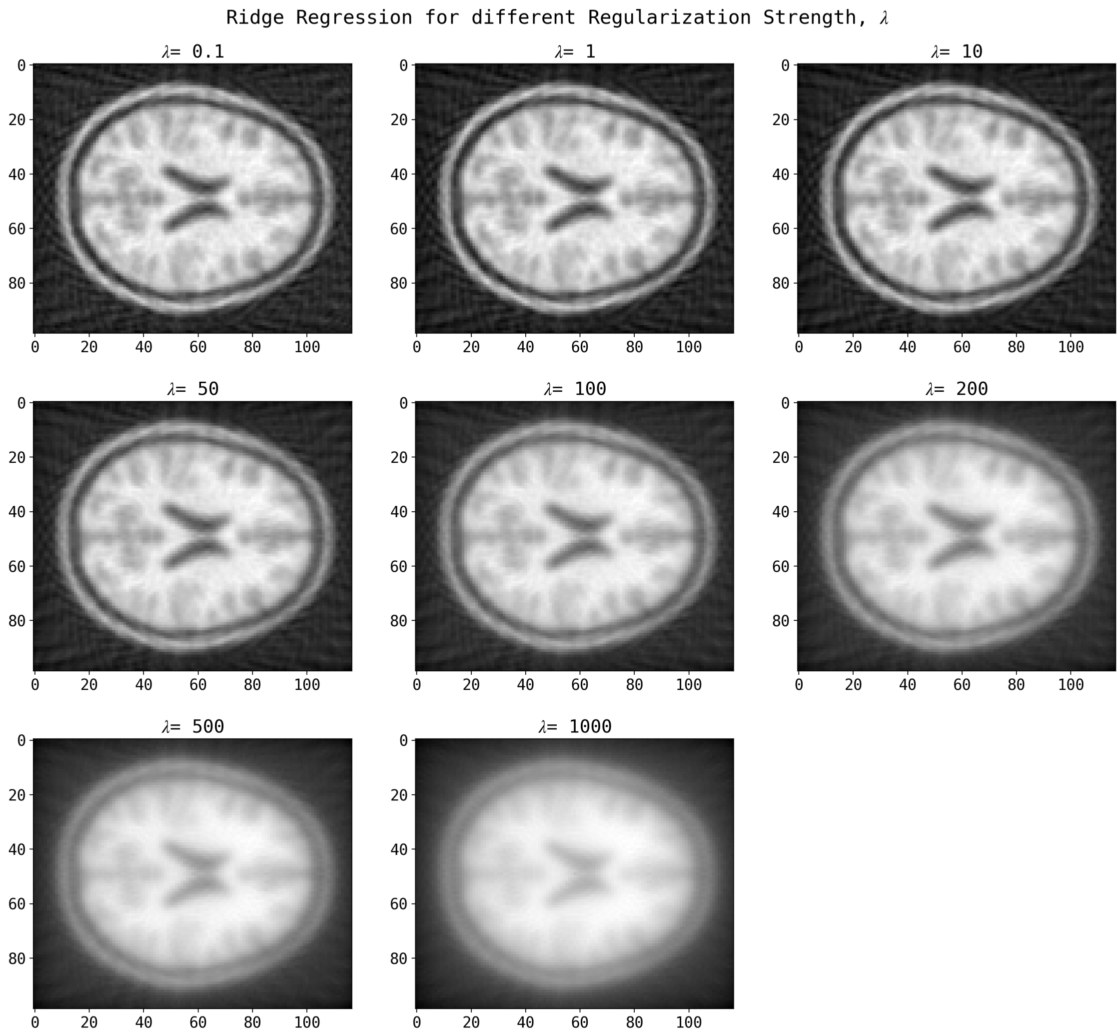


4.2.1 As we clearly see that for the Linear Regression the image we get is completely blurry and it is almost impossible to make any sense of the first image. On the other hand for Ridge regression we see the well defined image for the brain tissues.

4.2.2 But as we increase the regression length the image again turns blurry(see below).

4.3 Ridge Regression for different Regularization Strength λ

[ ]:
```
## plotting the image for different regularization strength

lambdas = [0.1,1,10,50,100,200,500,1000]


plt.figure(figsize=(18,16),dpi=200)
# plt.subplots_adjust(hspace=0.2)
plt.suptitle(f"Ridge Regression for different Regularization Strength,␣
↪$\lambda$", fontsize=18,y=0.92)
plt.tight_layout(pad=2)


#setting no. of rows and columns for subplot
ncols = 3
nrows = len(lambdas) // ncols + (len(lambdas) % ncols > 0) # calculating number␣
↪of rows

for n,i in enumerate(lambdas):
    clf = Ridge(alpha=i)  ##creating the ridge regression model with strength ␣
↪= 1
    clf.fit(design_matrix.T,sino.reshape(-1))  ##fitting the model with the␣
↪input data
    image_ridge = clf.coef_.reshape(*res)  ##getting the coefficient and␣
↪changing their shape as (99,117)
    # plt.figure(figsize = (8,5))  ##initializing the figure

    ax = plt.subplot(nrows, ncols, n + 1)
    ax.imshow(image_ridge,cmap = 'gray') ##plotting the image
    ax.set_title(f'$\lambda $= {i}')
```


Ridge Regression for different Regularization Strength, λ
λ= 0.1
λ= 1
λ= 10
λ= 50
λ= 100
λ= 200
λ= 500
λ= 1000