HW5_arkam2

March 6, 2021

1 STAT 542: Homework 5

1.1 NAME - ARKA MITRA; netid - arkam2

1.1.1 Spring 2021, by Ruoqing Zhu (rqzhu)

1.1.2 Due: Tuesday, Mar 9, 11:59 PM CT

1.1.3 About HW5

Question 1 [40 Points] Lasso solution for fixed λ

Question 2 [40 Points] Path-wise Coordinate Descent

Question 3 [20 Points] Recovering the Original Scale

2 Question 1 (40 Points) Lasso Solution for Fixed λ

For this question, you cannot use functions from any additional library in your algorithm. Following HW4, we use the this version of the objective function:

$$\arg\min_{\beta} \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda\|\beta\|_1$$

The following data is used to fit this model. Note that the MASS package can only be used to generate multivariate normal data. You can consider using similar functions in Python if needed.

```
library(MASS)
 set.seed(10)
 n = 100
 p = 200

 # generate data
 V = matrix(0.3, p, p)
 diag(V) = 1
 X_org = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
 true_b = c(runif(10, -1, 1), rep(0, p-10))
 y_org = X_org %*% true_b + rnorm(n)

 # pre-scale and center X and y
 X = scale(X_org)*sqrt(n/(n-1))
```

```
y = scale(y_org)*sqrt(n/(n-1))
lambda = 0.3
```

We will use the pre-scale and centered data X and y for this question, hence no intercept is needed. Write a Lasso algorithm function `myLasso(X, y, lambda, tol, maxitr)` which will output a vector of β values without the intercept. You need to consider the following while completing this question:

Do not use functions from any additional library

Start with a vector $\boldsymbol{\beta} = \mathbf{0}$

Use the soft-threshold function you developed in HW4.

Use the efficient **r** update algorithm we introduced during the lecture.

Run your coordinate descent algorithm for a maximum of `maxitr = 100` iterations (while each iteration will loop through all variables). However, stop your algorithm if the $\boldsymbol{\beta}$ value of the current iteration is sufficiently similar to the previous one, i.e., $\|\boldsymbol{\beta}^{(k)} - \boldsymbol{\beta}^{(k-1)}\|^2 \leq \text{tol}$. Set `tol = 1e-7`.

After running the algorithm, print out the first 10 variables.

Finally, check and compare your answer to the `glmnet` package using the following code:

```
library(glmnet)
glmnetfit = glmnet(X, y, lambda = 0.3, intercept = FALSE)
glmnetfit$beta[1:10]
```

2.1 Solution:

We replace the use of `MASS` package in R with `scipy.stats.multivariate_normal`, and of `glmnet` with `sklearn.linear_model.Lasso` in Python. We also replace the use of the variable `lambda` with `penalty` as `lambda` is a keyword in Python. In the later question, for symmetry, `lambda_all` is replaced with `allpenalty`.

[3]:
```python
import random
import numpy as np
from scipy.stats import norm
from scipy.stats import uniform
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from scipy.stats import multivariate_normal as mvrnorm

random.seed(100)

X = np.array([random.gauss(0,1) for i in range(100)])
epsilon = np.array([random.gauss(0,1) for i in range(100)])
X = X / np.sqrt(sum(X**2)/100)

Y = X + epsilon
```

```python
# soft thresholding function from HW 4
def soft_th(beta, penalty = 1):
    '''
    Function that given beta_OLS, calculates Lasso Parameter

    Variables:
    beta (Input) :: Estimator from OLS
    penalty (Input) :: Penalty (Lambda from theory)
    beta_lasso - Estimator from lasso
    '''
    if beta > penalty:
        return beta - penalty
    elif beta < (-penalty):
        return beta + penalty
    elif abs(beta) <= penalty:
        return 0
    else:
        print("Input not in range!")
        return -9e9


#Lasso implementation function
def myLasso(X, y, penalty, tol = 1e-7, maxitr = 100):
    '''
    Coordinate descent algorithm for Lasso Regression

    Variables:
    X (Input)     : Covariates (n x p)
    y (Input)      : Outputs (n x 1)
    penalty (Input, optional) : penalty level
    tol (Input, optional)    : tolerance
    maxitr (Input)   : Maximum iterations
    beta (Output)    : Estimated Parameters for Lasso Regression (p x 1)
    '''
    p = X.shape[1]
    beta = np.zeros(p) # All parameters initially set to zero

    #Common denominator = (X_i^T X_i)
    denom = X[:,1].T.dot(X[:,1])

    for count in range(maxitr):
        beta_temp = list(beta)
        r = y - X[:,1:].dot(beta[1:])
        beta_ols = X[:,0].T.dot(r)/denom
        beta[0] = soft_th(beta_ols , penalty)

        # update from 1 to p
```

```
        for j in range(1,p):
            r = r - beta[j-1]*X[:,j-1] + beta[j]*X[:,j] #efficient r update
            beta_ols = X[:,j].T.dot(r)/denom
            beta[j] = soft_th(beta_ols, penalty)

        #Threshold
        if np.linalg.norm(beta - beta_temp) <= tol:
            print("Num. of iterations before convergence = ", count)
            break

    return beta

#Python implementation of data generation code given above
#Covariance matrix
n, p = 100, 200
V = 0.3 * np.ones([p,p])
np.fill_diagonal(V, 1)
mu = np.zeros(p)
#100 covariates from multivariate normal distribution
X_org = mvrnorm.rvs(mean = mu, cov = V, size = n, random_state = 10)
true_beta = np.append(uniform.rvs(loc=-1, scale=2, size=10, random_state = 10),
                      np.zeros(p-10)) #true beta

# y = X*beta + i.i.d noise
y_org = X_org.dot(true_beta) + norm.rvs(size = n, random_state = 10)
# Pre-scaling and Centering X and y
ss = StandardScaler()
X = ss.fit_transform(X_org)
y = ss.fit_transform(y_org.reshape(-1,1)).reshape(n,)

#Using function myLasso
penalty = 0.3
beta_lasso = myLasso(X, y, penalty)
print("First 10 variables from myLasso = ", np.round(beta_lasso[:10], 3))

#Using the Lasso model from scikit-learn library
reg = Lasso(alpha = penalty, fit_intercept = False).fit(X,y)
print("First 10 variables from Python's scikit lasso = ", np.round(reg.coef_[:
 ↪10], 3))
```

```
Num. of iterations before convergence =  16
First 10 variables from myLasso =  [ 0.    -0.284  0.     0.     0.     0.
-0.048  0.    -0.028 -0.09 ]
First 10 variables from Python's scikit lasso =  [ 0.    -0.284  0.    -0.
-0.    -0.    -0.048  0.    -0.028 -0.09 ]
```

Python's sci-kit Lasso and myLasso results are exactly identical, hence, our function is correct.

3 Question 2 (40 Points) Path-wise Coordinate Descent

Let's modify our Lasso code to perform path-wise coordinate descent. The idea is simple: we will solve the solution on a grid of λ values, starting from the largest one. After obtaining the optimal $\boldsymbol{\beta}$ for a given λ, we simply use this solution as the initial value (instead of all zero) for the next (smaller) λ. This is referred to as a warm start in optimization problems. For more details, please watch the lecture video. We will consider the following grid of λ, with the `glmnet` solution of the first 10 variables plotted.

You need to add an additional input argument `lambda_all` to your Lasso function. After finishing your algorithm, output a matrix that records all the fitted parameters on your λ grid.

Provide a plot same as the above `glmnet` solution plot of the first 10 variables.

Which two variables entering (start to have nonzero values) the model first?

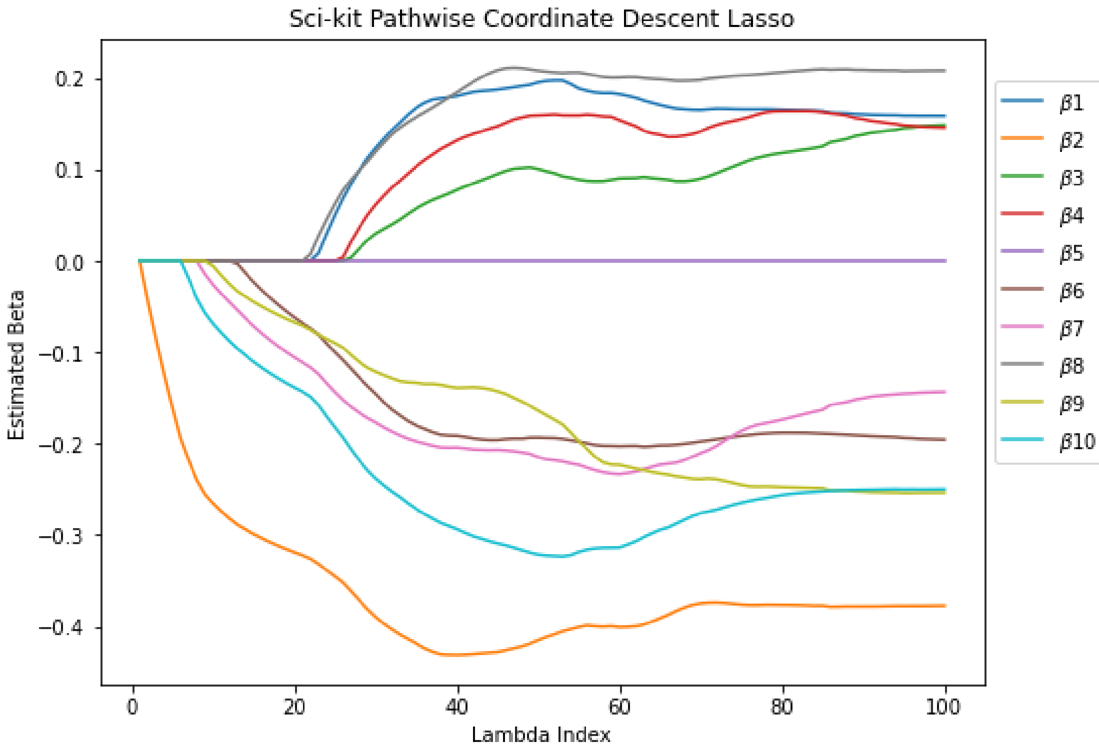
What is the maximum discrepancy between your solution and `glmnet`?

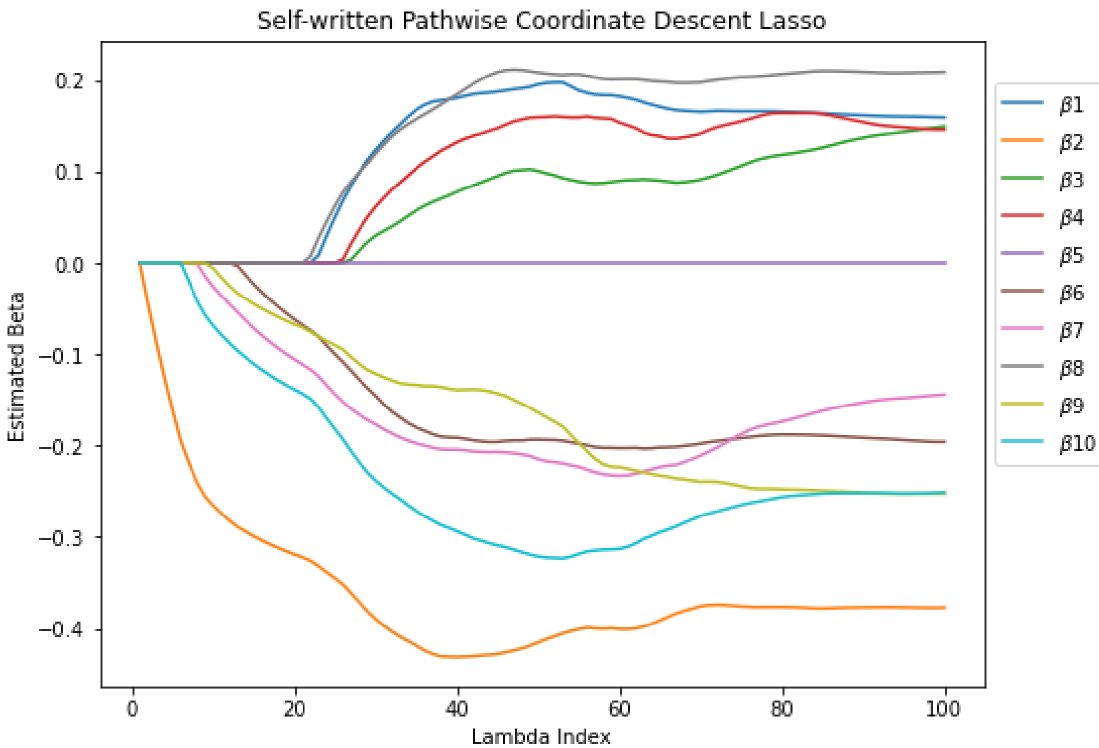
We replace the use of of `glmnet` package in R with `sklearn.linear_model.Lasso` for standard Lasso and `sklearn.linear_model.lasso_path` for pathwise Lasso in Python.

[12]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.linear_model import lasso_path

def myPathwiseLasso(X, y, allpenalty, tol = 1e-7, maxitr = 100):
    '''
    Coordinate descent algorithm for Lasso Regression

    Variables:
    X (Input)     : Covariates (n x p)
    y (Input)      : Outputs (n x 1)
    allpenalty (Input, optional) : Sorted array of penalties
    tol (Input, optional)    : tolerance
    maxitr (Input)   : Maximum iterations
    beta (Output)    : Estimated Parameters for Lasso Regression (p x 1)
    '''
    p = X.shape[1]
    betamat = np.zeros([len(allpenalty), p])

    #Common denominator = (X_i^T X_i)
    denom = X[:,1].T.dot(X[:,1])

    #Start loop for lamda
    for i in range(len(allpenalty)):
        if i==0:
            betamat[i] = np.zeros(p)
            continue
        else:
            beta = list(betamat[i-1])
```

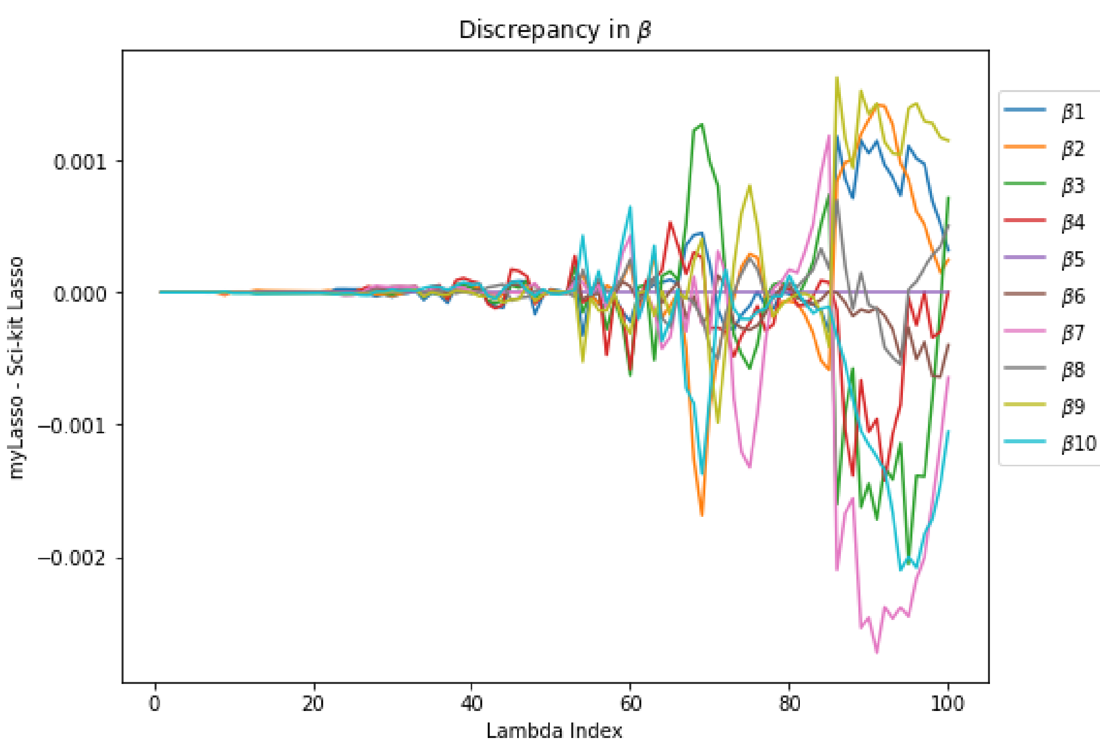
```
        for count in range(maxitr):
            beta_temp = list(beta)

            # beta_0 calculation
            r = y - X[:,1:].dot(beta[1:])
            beta_ols = X[:,0].T.dot(r)/denom
            beta[0] = soft_th(beta_ols, allpenalty[i])

            # beta_i calculations
            for j in range(1,p):
                r = r - beta[j-1]*X[:,j-1] + beta[j]*X[:,j] # efficient r update
                beta_ols = X[:,j].T.dot(r)/denom
                beta[j] = soft_th(beta_ols , allpenalty[i])

            #Threshold
            if np.linalg.norm(np.array(beta) - beta_temp) <= tol:
                break

        betamat[i] = beta

    return betamat.T

# Python pathwise Lasso
allpenalty, betamat, _ = lasso_path(X, y)
plt.figure(figsize = (8,6))
for i in range(10):
    plt.plot(range(1,101), betamat[i], label = r'$\beta$'+str(i+1))
plt.xlabel(r"Lambda Index")
plt.ylabel("Estimated Beta")
plt.legend(bbox_to_anchor=(1, 0.95), loc='upper left')
plt.title(r'Sci-kit Pathwise Coordinate Descent Lasso')
plt.show()

# My pathwise Lasso
betamat_myLasso = myPathwiseLasso(X, y, allpenalty)
plt.figure(figsize = (8,6))
for i in range(10):
    plt.plot(range(1,101), betamat_myLasso[i], label = r'$\beta$'+str(i+1))
plt.xlabel(r"Lambda Index")
plt.ylabel("Estimated Beta")
plt.legend(bbox_to_anchor=(1, 0.95), loc='upper left')
plt.title(r'Self-written Pathwise Coordinate Descent Lasso')
plt.show()


plt.figure(figsize = (8,6))
```

```
for i in range(10):
    plt.plot(range(1,101), betamat_myLasso[i] - betamat[i], label =␣
 ↪r'$\beta$'+str(i+1))
plt.xlabel(r"Lambda Index")
plt.ylabel(r"myLasso - Sci-kit Lasso")
plt.legend(bbox_to_anchor=(1, 0.95), loc='upper left')
plt.title(r'Discrepancy in $\beta$')
plt.show()

print('Maximum discrepancy =', np.min([betamat_myLasso[i] - betamat[i] for i in␣
 ↪range(1,101)]))
```





Self-written Pathwise Coordinate Descent Lasso
Estimated Beta
Lambda Index
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100
β1
β2
β3
β4
β5
β6
β7
β8
β9
β10



Discrepancy in β
myLasso - Sci-kit Lasso
Lambda Index
0.001
0.000
−0.001
−0.002
0
20
40
60
80
100
β1
β2
β3
β4
β5
β6
β7
β8
β9
β10

```
Maximum discrepancy = -0.003927429257725323
```

β_2 and β_{10} are the first 2 variables to enter the model run first.

The maximum discrepancy = -0.0039

4 Question 3 (20 Points) Recovering the Original Scale

The formula provided in HW4 can also be used when there are multiple variables.

$$\frac{Y-\bar{Y}}{\text{sd}_y} =amp; \sum_{j=1}^{p} \frac{X_j-\bar{X}_j}{\text{sd}_j}\gamma_j \tag{1}$$

$$Y =amp; \underbrace{\bar{Y}-\sum_{j=1}^{p}\bar{X}_j\frac{\text{sd}_y\cdot\gamma_j}{\text{sd}_j}}_{\beta_0}+\sum_{j=1}^{p}X_j\underbrace{\frac{\text{sd}_y\cdot\gamma_j}{\text{sd}_j}}_{\beta_j}, \tag{2}$$

Use this formula to recover the original scale of the $\boldsymbol{\beta}$, including the intercept term β_0.

Use the following code of `glmnet()` to obtain a solution path.

```
glmnetfit2 = glmnet(X_org, y_org, lambda = lambda_all*sd(y_org)*sqrt(n/(n-1)))
lassobeta2 = coef(glmnetfit2)[2:11, ]
matplot(t(as.matrix(coef(glmnetfit2)[2:11, ])), type = "l", xlab = "Lambda Index", ylab = "Est
```

After recovering your β values, produce a plot of your solution path.

What is the maximum discrepancy between your solution and `glmnet`?

[Bonus 5 Points] If we do not specify lambda in the following `glmnet()` function, the package will pick a different grid, which lead to a different set of solution.

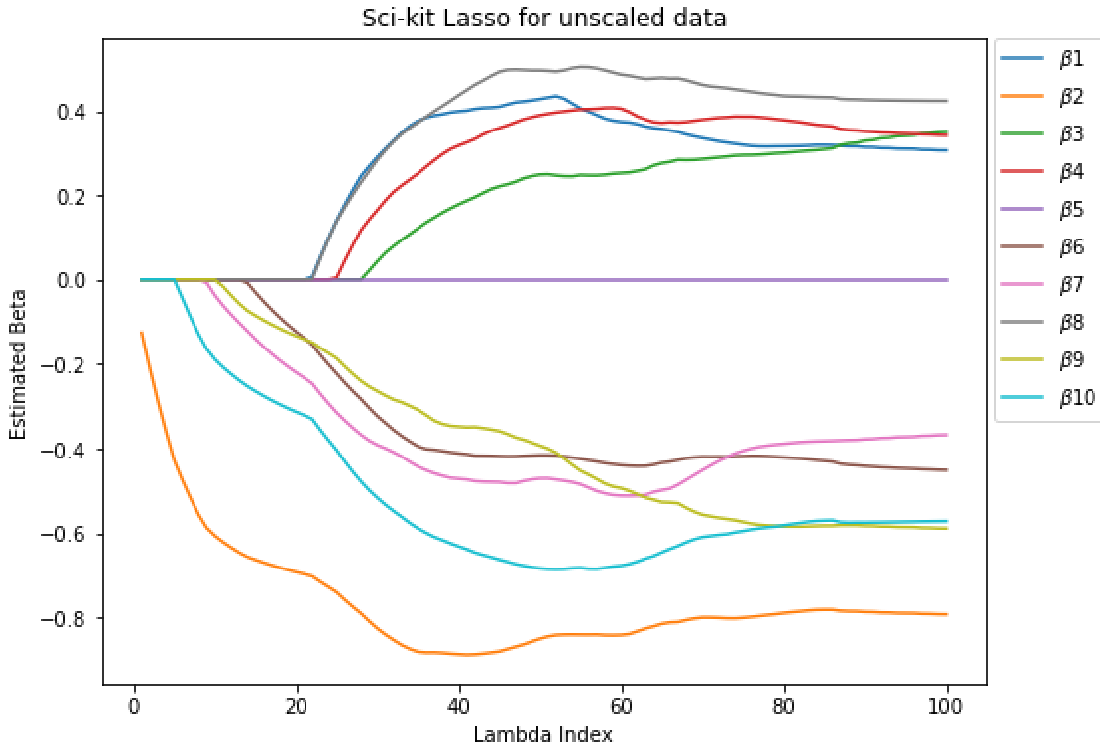
Explain how the lambda values are picked in this case.

What is the largest lambda being considered? and why we don't need to consider a larger lambda value?

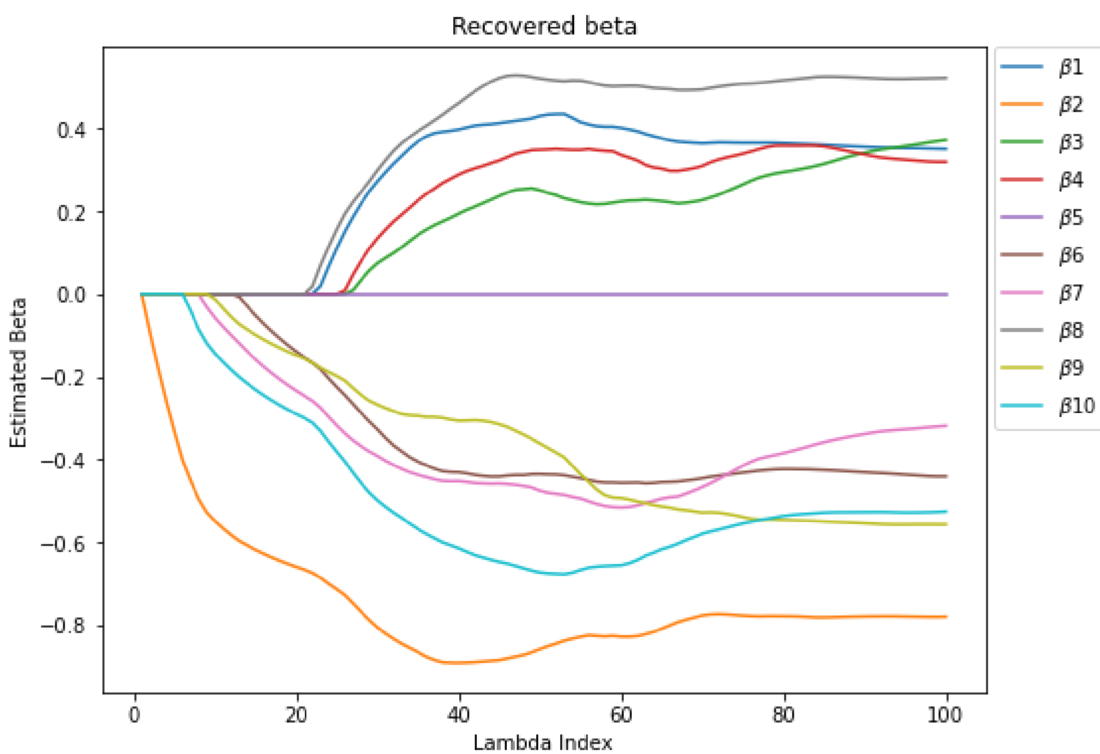
Consider reading the following paper (section 2.5) and the documentation of the `glmnet()` function at the CRAN website. However, please note that the descriptions from these two sources are slightly different, with similar ideas. Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. "Regularization paths for generalized linear models via coordinate descent." Journal of statistical software 33, no. 1 (2010): 1.

[22]:
```
def recover_betavalues(X_org, y_org, betamat):
    betamat_recovered = np.zeros((201,100))
    Xmean, Xstd, ymean, ystd = X_org.mean(axis = 0), X_org.std(axis = 0), y_org.
↪mean(axis = 0), y_org.std(axis = 0)
    beta_rescaled = ystd * betamat / Xstd.reshape(-1,1)
    betamat_recovered[0] = ymean - (beta_rescaled * Xmean.reshape(-1,1)).
↪sum(axis = 0)
    betamat_recovered[1:] = beta_rescaled
```

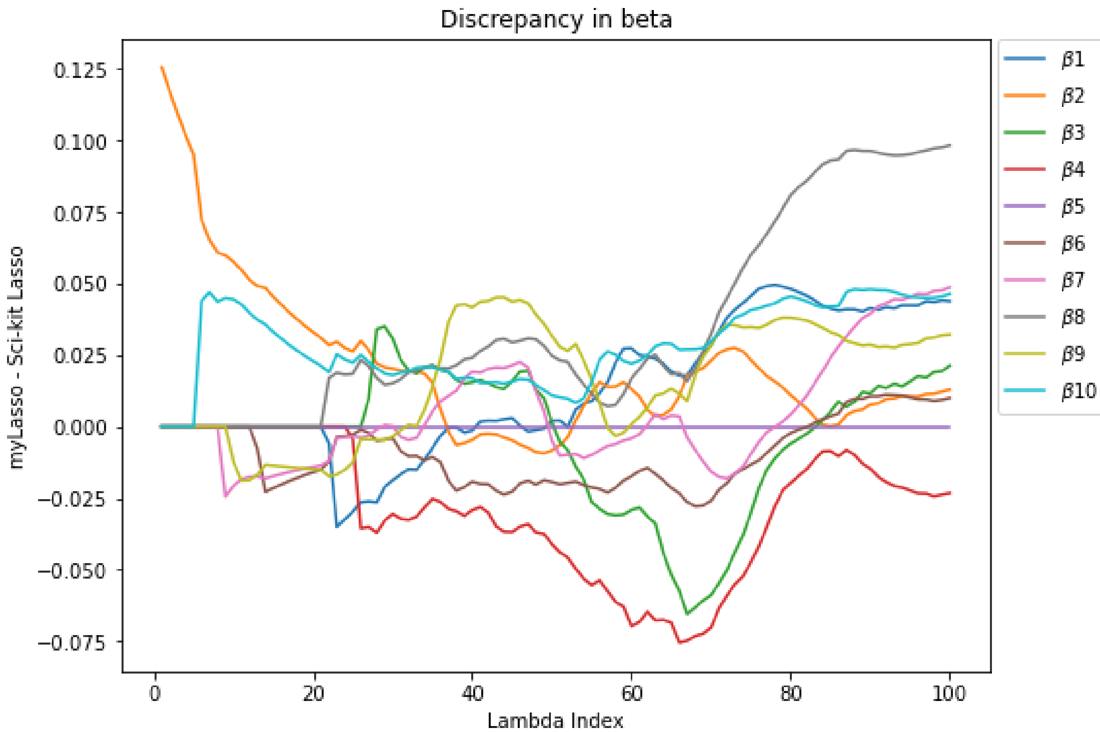
```
    return betamat_recovered

#Inputting the unscaled data
_, betamat3, _ = lasso_path(X_org, y_org, alphas = allpenalty*y_org.std())

plt.figure(figsize = (8,6))
for i in range(10):
    plt.plot(range(1,101) , betamat3[i] , label = r'$\beta$'+str(i+1))
plt.xlabel("Lambda Index")
plt.ylabel("Estimated Beta")
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.title(r'Sci-kit Lasso for unscaled data')
plt.show()

betamat_recovered = recover_betavalues(X_org, y_org, betamat_myLasso)
plt.figure(figsize = (8,6))
for i in range(10):
    plt.plot(range(1,101), betamat_recovered[i+1], label = r'$\beta$'+str(i+1))
plt.xlabel("Lambda Index")
plt.ylabel("Estimated Beta")
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.title(r'Recovered beta')
plt.show()

plt.figure(figsize = (8,6))
for i in range(10):
    plt.plot(range(1,101) , betamat_recovered[i+1] - betamat3[i] , label =␣
 ↪r'$\beta$'+str(i+1))
plt.xlabel("Lambda Index")
plt.ylabel(r"myLasso - Sci-kit Lasso")
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.title(r'Discrepancy in beta')
plt.show()
```



Sci-kit Lasso for unscaled data
Estimated Beta
Lambda Index
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
20
40
60
80
100
β1
β2
β3
β4
β5
β6
β7
β8
β9
β10



Recovered beta
Estimated Beta
Lambda Index
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
20
40
60
80
100
β1
β2
β3
β4
β5
β6
β7
β8
β9
β10



The maximum discrepancy = 0.125

When a grid of λ is not provided, an upper threshold for λ is selected based on the value for which all coefficients, $\beta = 0$.

No greater λ needs to be considered, as for those values, all $\beta = 0$.

[ ]: