Reproducibility README for "Penalized and Constrained Optimization" (PaC)

This README is for the purposes of running the code accompanying the JASA paper submission "Penalized and Constrained Optimization: An Application to High-Dimensional Website Advertising" (Manuscript ID JASA-A&CS-2018-0133.R3) for reproducibiilty. Although we will be formally creating a package through CRAN for the code upon acceptance and publication of the manuscript, this README is designed to help anyone evaluating the reproducibility aspects of the manuscript to more easily follow the logic and functions submitted with the paper.

Usage

Although this README is designed explicitly for the purpose of reproducing the code contained in "Penalized and Constrained Optimization: An Application to High-Dimensional Website Advertising" (hereafter referred to as "PaC"), it does contain information and walkthrough that will be part of the eventual CRAN package for the PaC method. The authors have tried to note where functions are being used that are intended solely for reproducibility purposes rather than distribution to a wider audience.

The code contained in this project is designed to be used for problems in which users wish to optimize both a penalized and constrained criterion. The functions here focus on a common example of such problems, a constrained version of the LASSO penalized criterion. In addition, the functions include minimal examples of the functions used in analyzing the real-world data set used in the PaC case study, i.e. the comScore Media Metrix data. Because this data set is proprietary and cannot be provided itself, the authors provide substitute data designed for a minimal working example that users could apply to their own such data.

Project Directory

Below is a list of all files contained in the *PaC_Code_Reproducibility.zip* file:

1. **PaC_Functions_Revised.R**: This file contains the functions created as part of the PaC paper. It includes functions that are both included purely for reproducibility purposes (such as *compare.to.lasso()*) and the main PaC functions that will be released as a CRAN package upon acceptance of the PaC manuscript (such as *lasso.c()*). See the README section "PaC Functions" below for full details of the functions contained in this file. It is sourced by *PaC_Simple_Table2.R*, *PaC_Tables_2_and_3.R, PaC_Timings_Fig2.R*, *PaC_Full_Reproducibility.R*, and *Minimal_Working_Example_Revised.R*.

2. **PaC_Full_Reproducibility.R**: This file should be run in its entirety to get the full reproducible results for Section 5 of the PaC paper. However, the authors recognize this code is unwieldy and difficult to parse for understanding. To that end, we have also provided three separate files for the results in Section 5:

- *PaC_Simple_Table2.R*: a more simplistic, commented version of the code to show the calculations done to achieve the first line of Table 2 in Section 5.1 of the PaC paper. (Since Tables 2 and 3 are calculated from the same general code.)
- *PaC_Tables_2_and_3.R*: the complete code to get the results in Tables 2 and 3.
- *PaC_Timings_Fig2.R*: the code to run a version of Figure 2 of Section 5.3. This is separated from the other reproducibility code due to its long runtime.

3. **PaC_Simple_Table2.R**: This file contains the code necessary to recreate only the first line of Table 2 in Section 5.1 of the PaC paper. Because the full reproducibility code is difficult to parse, this code file is intended to give users a simple, clear walkthrough of how the authors have created the results of Table 2 without having to run the complete *PaC_Full_Reproducibility.R* code file. It sources the *PaC_Functions_Revised.R* function file.

4. **PaC_Tables_2_and_3.R**: This file contains the code necessary to recreate the full Table 2 and Table 3 results in Sections 5.1 and 5.2 of the PaC paper respectively. Because the full reproducibility code takes a long time to run due to the timings code included at the end (for Figure 2 in Section 5.3), this code file is intended to give a faster way to calculate all Table 2 and 3 results. It sources the *PaC_Functions_Revised.R* function file.

5. **PaC_Timings_Fig2.R**: This file contains the code necessary to recreate a version of Figure 2 in Section 5.3 of the PaC paper. Because this code takes considerable time to run due to the changing values of the number of coefficients, this code is separated out for easier walkthrough. It sources the *PaC_Functions_Revised.R* function file.

6. **Minimal_Working_Example_Revised.R**: This file contains the code necessary to recreate the minimal working example demonstrated in the ACS Author form submitted with the PaC paper. Due to the proprietary nature of the data presented in the case study, the authors cannot provide the actual data. However, this file uses the example data given in *Page_View_Matrix_Example.csv* and *500_Site_Info_Example.csv* to demonstrate how the case study results were created in Section 6. It also sources the *PaC_Functions_Revised.R* function file.

7. **Page_View_Matrix_Example.csv**: An example matrix of website page view data used in the minimial working example of *Minimal_Working_Example_Revised.R*.

8. **500_Site_Info_Example.csv**: An example matrix of website information data used in the minimial working example of *Minimal_Working_Example_Revised.R*.

Necessary Libraries

In order to run the reproducible code, you should first install and load the lars, quadprog, and limSolve libraries into R. The code should be compatible with any version of R after

3.1.0 (released April 2014), provided the versions of the packages used are also compatible with the installed version of R.

The lars, quadprog, and limSolve libraries are used in the actual running of the PaC code functions; the MASS and MBESS libraries are used in creating the data and correlation matrix generation of the reproducible results.

```
install.packages(c("lars", "quadprog", "limSolve","MASS","MBESS"))
library(lars)
library(quadprog)
library(limSolve)
```

The versions suggested for the **PaC_Functions_Revised.R** code are as follows:

1. lars, version 1.2
2. quadprog, version 1.5-5
3. limSolve, version 1.5.5.3

The versions suggested for the reproducubility in both the **PaC_Full_Reproducibility.R** and the subset code files (**PaC_Simple_Table2.R**, **PaC_Tables_2_and_3.R**, and **PaC_Timings_Fig2.R**) are as follows:

1. MASS, any version 7.3-32 or later
2. MBESS, any version 4.0.0 or later

Functions contained in PaC_Functions_Revised.R

Below is a list of the 11 functions provided in the main **PaC_Functions_Revised.R** source file, including a short description of their intended use. Further details on each function can be found at the end of this README file, as well as added before each function in the **PaC_Functions_Revised.R** source file. In addition, the authors note ***in bold*** which functions are intended solely for reproducibility purposes vs. functions intended to be distributed as part of the CRAN package upon publication of the manuscript:

1. ***compare.to.lasso***: This function is only used for reproducibility. This is the main function to call in order to compare the PaC and the unconstrained LASSO for simulated data. To do so, it calls ***generate.data*** to create the simulated data set according to given values of observations (*n*), variables (*p*), and constraints (*m*). It then passes this data to the *lasso.c* function to calculate the PaC coefficient paths and the *lars* function (part of the lars package) to calculate corresponding LASSO coefficient paths.

2. ***generate.data***: This function is only used for reproducibility. This function is called by ***compare.to.lasso*** to generate random data.

3. *lars.c*: This function computes the PaC constrained LASSO coefficient paths following the methodology laid out in the PaC paper. This function could be called directly as a standalone function, but the authors recommend using *lasso.c* for any implementation.

This is because *lasso.c* has additional checks for errors across the coefficient paths and allows for users to go forwards and backwards through the paths if the paths are unable to compute in a particular direction for a particular run.

4. *lasso.c*: This is a wrapper function for the *lars.c* PaC constrained Lasso function. *lasso.c* controls the overall path, providing checks for the path and allowing the user to control how the path is computed (and what to do in the case of a stopped path).

5. *lin.int*: This function is called internally by *lars.c* to get the linear programming initial fit if the user requests implementation of the algorithm starting at the largest lambda value and proceeding backwards.

6. *quad.int*: This function is called internally by *lars.c* to get the quadratic programming fit if the user requests implementation of the algorithm starting at the smallest lambda value and proceeding forwards.

7. *transform.data*: This function is called internally by *lars.c* to compute the transformed versions of the X, Y, and constraint matrix data, as shown in the PaC paper.

8. ***pred.error***: This function is only used for reproducibility. Its purpose is to compute the prediction error (SSE) for a given combination of data and estimated coefficients.

9. *ELMSO*: This function is used in the calculation of the case study data of Section 6. This function implements the calculation of the ELMSO method given in Paulson et al. (2018) for calculating reach on page view data for Internet website advertising problems.

10. *reach.calc*: This function calculates the reach of an advertising model given a set of allocation coefficients and a matrix of views across a set of unique advertising channels/opportunities. In the case study presented in the PaC paper, these are Internet websites and views by users to those sites.

11. *compare.to.ELMSO*: This function is used in the calculation of the case study data of Section 6. This function implements the PaC calculation in comparison to the ELMSO method given in Paulson et al. (2018) for calculating reach on page view data for Internet website advertising problems.

Simulation Reproducibility

The files provided for reproducibility are intended to stand alone as files to be run in their entirety to reproduce the results of Section 5 in the PaC paper. However, we provide some details here to indicate what the code is doing as it runs to explain the general process, given the complex code files.

In order to reproduce the results of the PaC paper, you should first source the PAC functions file, **PAC_Functions_Revised.R**, and set the number of iterations (*iter*) you would like to run. Each iteration is a separately-generated simulated data set for which PaC,

LASSO, relaxed PaC, and relaxed LASSO coefficients and error rates are calculated. To reproduce the results of the paper directly, you should set *iter=100*, as is the default value provided in both **PaC_Full_Reproducibility.R** and the subset code files (**PaC_Simple_Table2.R**, **PaC_Tables_2_and_3.R**, and **PaC_Timings_Fig2.R**).

```
source("PaC_Functions_Revised.R")
iter=100
```

For reproducibility, you should also set a seed value and then sample for the number of iterations. Again, to reproduce the results in the paper directly, you should set the seed to 1234 and sample 100 values without replacement from 1 to 10000, as is the default provided in both **PaC_Full_Reproducibility.R** and the subset code files (**PaC_Simple_Table2.R**, **PaC_Tables_2_and_3.R**, and **PaC_Timings_Fig2.R**).

```
set.seed(1234)
seed=sample(1:10000, iter, replace=F)
```

Running PaC and LASSO fits (with Approximate Runtimes)

To reproduce the results for Table 2, you would need to run four set of coefficients (PaC, relaxed PaC, LASSO, and relaxed LASSO) for two correlation settings (no correlation and a dying-off correlation structure) for three simulation settings (with varying numbers of observations, variables, and constraints). In addition, the results of Table 3 are computed similarly, but with no correlation structure (and increasing error rates).

Running the complete **PaC_Full_Reproducibility.R** file will reproduce these two tables exactly, though it may take considerable time. For convenience and an easier walkthrough of the code, we have included the **PaC_Simple_Table2.R** file to run for only the first row of Table 2, because the code called for Tables 2 and 3 is so similar. We have commented each line of **PaC_Simple_Table2.R** to indicate what it is calling at each step. The setting considered in the first row of Table 2 is for a simulation of 100 data sets, where the data sets exhibit no correlation structure, and each data set has 100 observations, 50 variables, and 5 constraints. We discuss some general details of the code implementation here.

First, the reproducibility code sets the number of points to use in the test data set (10,000 in the PaC paper), as well as *s* (the number of non-zero random uniform components, set to 5 in the paper) and the value of *sigma* (set to 1 in the paper to give a standard normal distribution).

```
n.test=10000
s=5
sigma=1
```

Next, the code will set the appropriate values for the setting under consideration. In this setting, we have 100 observations (n), 50 variables (p), and 5 constraints (m):

```
n=100
p=50
m=5
```

To run the complete set of fits for PaC, use the ***compare.to.lasso*** function in the **PaC_Functions_Revised.R** file. Note that this function is used primarily for reproducibility. In the actual CRAN package released with the article, this function would be eliminated, as users would likely want to call *lasso.c()* directly rather than generate data and compare to Lasso. But for reproducibility, this function can be called for each iteration after setting a list for storage like so:

```
fits=vector("list", iter)
for (i in 1:iter) {

  fits[[i]]=compare.to.lasso(n=n,p=p,m=m,cov.mat=NULL,sigma=sigma,trace=F,see
d=seed[i])

}
```

The *n*, *p*, *m*, and *sigma* values are given as above. In addition, the function contains a "trace" option for running the code with verbose tracing to help troubleshoot issues, and the "seed" and "cov.mat" functions as options for exactly reproducing the randomly-generated data and including a correlation structure respectively.

Note that each run of compare.to.lasso() will run relatively quickly. Here are the approximate runtimes for a single iteration of the above code for each setting in Table 2 (using a moderately powerful computer utilizing an Intel Core i7-2600 processor @ 3.40GHz):

- Setting 1 (above; no correlation): approximately 0.50 seconds per iteration
- Setting 1 (above; correlation structure): approximately 0.60 seconds per iteration
- Setting 2 (n=50, p=500, m=10; no correlation): approximately 15.50 seconds per iteration
- Setting 2 (n=50, p=500, m=10; correlation structure): approximately 16.00 seconds per iteration
- Setting 3 (n=50, p=100, m=60; no correlation): approximately 4.50 seconds
- Setting 3 (n=50, p=100, m=60; correlation structure): approximately 5.20 seconds per iteration

The runtime for the reproducible example of the first row of Table 2 in **PaC_Simple_Table2.R** will take only approximately 1 minute to run across all 100 reproduced iterations, while the overall runtime of the full Tables 2 and 3 code in **PaC_Tables_2_and_3.R** will likely take approximately 2 hours to run in its entirety across all 100 reproduced iterations (note this is an estimate; in trials, this code has taken anywhere from 1 hour to 3.5 hours to run completely).

PaC_Full_Reproducibility.R, which includes the code for Tables 2 and 3 in addition to the code necessary to run the timings plot presented in Section 5.3 (also included in **PaC_Timings_Fig2.R**), will require several hours to run due to the timings and measuring those timings at complex values of the number of coefficients, *p*.

Note that the same code as used in the Table 2 example above can be used for the "Violations of Constraints" code in Table 3 of the PaC paper. For example, for an alpha (error) level of 0.25, we can use the "err" argument of the compare.to.lasso code:

```
fits.test1.25=vector("list", iter)
for (i in 1:iter) {

  error.vec=runif(m,0,.25)
  fits.test1.25[[i]]=compare.to.lasso(err=error.vec,n=n,p=p,m=m,sigma=1,seed=
seed[i],backwards=F)

}
```

The only change here is the addition of the "err" argument to indicate we want to generate data with an average error rate of 1.25 times the *b* constraint vector.

Constrained Lasso PaC Function Arguments

Here is a list of arguments used by the constrained Lasso PaC functions. Since many of the functions use the same arguments, the full list is presented here in alphabetical order for easy lookup. The following sections shows the function calls for each function in the PaC function source file and their corresponding outputs. In the CRAN package that will be released with the published PaC article, each function will have its own help file with individual inputs and outputs for completeness, but we separate them here to follow through more easily for reproducibility.

Argument List

b: constraint vector b

backwards: used in *lasso.c*. If backwards = F (default), then the algorithm starts at 10^l.min and works forward. If the algorithm gets stuck, it switches to 10^l.max and moves backwards. If this is T, then the algorithm just starts at 10^l.max and moves backwards (without the forward step).

beta0: initial guess for beta coefficient vector. Used in *lasso.c* and *lars.c*. If this is NULL, the function automatically computes an estimate using either quadratic programming (for forward method) or linear programming (for backwards method).

betas: a vector of coefficient values (of length *p*)

C.full: complete constraint matrix C

cov.mat: a covariance matrix applied in the generation of data to impose a correlation structure. Default is NULL (no correlation).

err: error introduced in generation of coefficient vector beta2 values to test robustness of algorithm. When generating the data (see generate.data), this creates beta2 values from a perturbed b vector (b is multiplied by (1+err)). Default is no error (err=0).

forwards: if forwards = F (default), then the algorithm starts at 10^l.max and moves backwards (without the forward step). If forwards = T, algorithm starts at 10^l.min and works forward.

glasso: should the generalized Lasso be used (TRUE) or standard Lasso (FALSE). Default is FALSE.

intercept: used in *lasso.c* and *lars.c*. Should intercept be included in modeling or not (that is, should function subtract the mean from Y and x, the equivalent to assuming an intercept). Default is TRUE.

l.min: lowest value of lambda to consider (used as 10^l.min). Default is -2.

l.max: largest value of lambda to consider (used as 10^l.max). Default is 6.

lambda: value of lambda to use

m: number of constraints to generate. Default is 5.

max.it: maximum number of times step size is halved before the algorithm terminates and gives a warning. Default is 12.

n: number of observations to generate. Default is 1000.

normalize: used in *lasso.c* and *lars.c*. Should X data be normalized (that is, should each column of x be divided by its standard deviation). The default is TRUE, but note that the returned coefficients are transformed back to the original scale.

p: number of predictors to generate. Default is 10.

plots: used only in ***compare.to.lasso***. If this is F (default), then no plots of the coefficients and method are produced. If this is T, then the algorithm also produces plots of the paths and errors associated with the solutions as it runs.

s: number of non-zero elements in coefficient vector beta1. Default is 5.

seed: option to choose a number to reproduce results given by the function. Default is NULL.

sigma: standard deviation of noise in response

step: step size increase in lambda attempted at each iteration (by a factor of 10^step). Default is 0.2

trace: should function print output as algorithm runs. Default is FALSE.

verbose: should function print output at each iteration (TRUE) or not (FALSE). Default is FALSE.

x: independent variable matrix of data to be used in calculating PaC coefficient paths

y: response vector of data to be used in calculating PaC coefficient paths

Constrained Lasso PaC Functions

- ***compare.to.lasso***

Function call:

compare.to.lasso <- function(err = NULL, n = 1000, p = 10, m = 5, s = 5, sigma = 1, trace = F, seed = NULL, l.min = -2, l.max = 6, backwards = F, plots = F)

Output:

data: randomly-generated data

constrained.fit: fit of model (output from *lasso.c*)

lars.error: sum of squared errors for standard lasso at each lambda

constrained.error: sum of squared errors for constrained lasso at each lambda

- ***generate.data* function**

Function call:

generate.data <- function(n = 1000, p = 10, m = 5, cov.mat = cov.mat, s = 5, sigma = 1, err = 0)

Output:

x: generated x data

y: generated response y vector

C.full: generated full constraint matrix

b: generated constraint vector b

b.run: if error was included, the error-adjusted value of b

beta: the complete beta vector, including generated beta1 and beta2

- ***lars.c* function**

Function call:

lars.c <- function(x, y, C.full, b, l.min = -2, l.max = 6, step = 0.2, beta0 = NULL, verbose = F, max.it = 12, intercept = T, normalize = T, forwards = T)

Output:

coefs: A p by length(lambda) matrix with each column corresponding to the beta estimate for that lambda

lambda: the grid of lambdas used to calculate the coefficients on the coefficient path

intercept: the intercept value

error: did the algorithm terminate due to too many iterations (TRUE or FALSE)

b2index: the index of the beta2 values identified by the algorithm at each lambda

- ***lasso.c* function**

Function call:

lasso.c <- function(x, y, C.full, b, l.min = -2, l.max = 6, step =0.2, beta0 = NULL, verbose = F, max.it = 12, intercept = T, normalize = T, backwards = F)

Output:

coefs: A p by length(lambda) matrix with each column corresponding to the beta estimate for that lambda

lambda: vector of values of lambda that were fit

intercept: vector with each element corresponding to beta 0 for corresponding lambda

error: Indicator of whether the algorithm terminated early because max.it was reached

- ***lin.int* function**

Function call:

lin.int <- function(C.full, b)

Output:

beta: the initial beta vector of coefficients to use for the PaC algorithm

- ***quad.int* function**

Function call:

quad.int <- function(X, Y, C.full, b, lambda, d=10^-7)

Output:

beta: the initial beta vector of coefficients to use for the PaC algorithm

- ***pred.error* function**

Function call:

pred.error <- function(x, y, betas)

Output:

error: the SSE calculated from the actual y and predicted y (x*betas)

- ***transform.data* function**

Function call:

transform.data <- function(x, y, C.full, b, lambda, beta0, eps=10^-8)

Output:

x: transformed x data to be used in the PaC algorithm

y: transformed y data to be used in the PaC algorithm

Y_star: transformed Y* value to be used in the PaC algorithm

a2: index of A used in the calculation of beta2 (the non-zero coefficients)

beta1: beta1 values

beta2: beta2 values

C: constraint matrix

C2: subset of constraint matrix corresponding to non-zero coefficients

active.beta: index of non-zero coefficient values

beta2.index: index of non-zero coefficient values

Other Functions: Case Study Minimal Working Example

ELMSO, ***compare.to.ELMSO***, and ***reach.calc*** functions:

Function calls:

ELMSO <- function(z, beta = 1, gamma, step = 0.05, size = 100, q = NULL)

compare.to.ELMSO <- function(z, gamma, step = 0.05, size = 100, q = NULL)

reach.calc <- function(w, z, gamma, q = NULL)

Arguments:

z: matrix of views to each advertising channel (e.g. page view matrix to Internet websites)

beta: parameter in *ELMSO* only to control the shape of budget allocation curve. Default is 1, indicating each allocation channel has a linear slope for budget allocation (each additional dollar of revenue has a constant buying effect).

gamma: vector of gamma values (the fraction of all ads purchased at website j for every dollar spent by a campaign)

step: step size in lambda for calculation of coefficient path. Default is 0.05.

size: total number of budget points that coefficients should be calculated for. Default is 100.

q: vector of clickthrough rates. Default is NULL, indicating reach calculation.

Output:

w: A p by length(lambda) matrix with each column corresponding to the coefficient estimate for that lambda

w.sum: a total sum of all coefficients at each lambda (used as budget calculation for advertising example)

lam: grid of lambda values used in the estimation

click.est: vector of click rates if q is not NULL