Homework 5

Jimmy Hickey

Due @ 11:59pm on April 22, 2020

1

Part 1. In this homework we will study two algorithms for iteratively computing the generalized lasso solution.

Let $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{X} \in \mathbb{R}^{n \times p}$ denote a response and design matrix. The generalized lasso is the solution to the following optimization problem.

$$\text{minimize } \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2 + \lambda\|\mathbf{D}\mathbf{b}\|_1,$$

where $\lambda \geq 0$ is a tuning parameter that trades off sparsity in the linear transformation $\mathbf{D}\mathbf{b}$ of $\mathbf{b}$ and the discrepancy between the linear model $\mathbf{X}\mathbf{b}$ and the response $\mathbf{y}$. In this assignment we consider the simpler problem where $\mathbf{X} = \mathbf{I}$.

$$\text{minimize } \frac{1}{2}\|\mathbf{y} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{D}\mathbf{b}\|_1,$$

This is the case, for example, for the fused lasso or trend filtering. The optimization problem is challenging to solve due to the term $\|\mathbf{D}\mathbf{b}\|_1$.

Thus, consider the following equivalent equality constrained problem.

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{y} - \mathbf{b}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1 \\ &\text{subject to } \mathbf{D}\mathbf{b} = \boldsymbol{\theta}. \end{aligned}$$

1. Show that the dual problem is given by

$$\begin{aligned} &\text{maximize } \frac{1}{2}\|\mathbf{y}\|_2^2 - \frac{1}{2}\|\mathbf{y} - \mathbf{D}^{\mathsf{T}}\mathbf{v}\|_2^2 \\ &\text{subject to } \|\mathbf{v}\|_\infty \leq \lambda. \end{aligned}$$

Let's use a Lagrangian to set up our primal problem with our constraint.

$$\begin{aligned} \psi_P(x) &= \frac{1}{2}\left\|y - b\right\|_2^2 + \lambda\left\|\theta\right\|_1 + \nu^T(Db - \theta) \\ &= \frac{1}{2}\left\|y\right\|_2^2 - \frac{2}{2}y^T b + \frac{1}{2}\left\|b\right\|_2^2 + \lambda\left\|\theta\right\|_1 + \nu^T Db - \nu^T\theta \\ &= \frac{1}{2}\left\|y\right\|_2^2 + \frac{1}{2}\left\|b\right\|_2^2 - (-D^T\nu + y)^T b + \lambda\left\|\theta\right\|_1 - \nu^T\theta \end{aligned}$$

We can then complete the square.

$$\begin{aligned}\psi_P(x) &= \frac{1}{2}\|y\|_2^2 + \frac{1}{2}\|b\|_2^2 - (-D^T\nu + y)^T b + \frac{1}{2}\left\|y^T - D^T\nu\right\|_2^2 + \lambda\|\theta\|_1 - \nu^T\theta - \frac{1}{2}\left\|y^T - D^T\nu\right\|_2^2 \\ &= \frac{1}{2}\|y\|_2^2 + \lambda\|\theta\|_1 - \nu^T\theta + \frac{1}{2}\left\|b - (y - D^T\nu)\right\|_2^2 - \frac{1}{2}\left\|y - D^t\nu\right\|_2^2\end{aligned}$$

Then we can optimize with respect to b and θ to find the dual problem. We will first optimize b.

$$\begin{gathered}\nabla_b\psi_P(x) = b + y - D^T\nu \overset{\text{set}}{=} 0 \\ b = y - D^T\nu\end{gathered}$$

Plugging that in gives

$$\frac{1}{2}\|y\|_2^2 + \lambda\|\theta\|_1 - \nu^T\theta - \frac{1}{2}\left\|y - D^T\nu\right\|_2^2.$$

Now we can optimize this with respect to θ by finding the infimum.

$$\begin{aligned}\inf_\theta \frac{1}{2}\|y\|_2^2 + \lambda\|\theta\|_1 - \nu^T\theta - \frac{1}{2}\left\|y - D^T\nu\right\|_2^2 &= -\sup_\theta -\frac{1}{2}\|y\|_2^2 - \lambda\|\theta\|_1 + \nu^T\theta + \frac{1}{2}\left\|y - D^T\nu\right\|_2^2 \\ &= sup_\theta\{\nu^T\theta - \lambda\|\theta\|_1\}\end{aligned}$$

By the Legendre-Fenchel conjugate, $\|\nu\|_\infty \leq \lambda$. Thus, we have our dual problem.

2. What are the KKT conditions for this primal-dual pair of optimization problems?

Primal Feasibility

$$Db - \theta = 0$$

Dual Feasibility

$$\|\nu\|_\infty \leq \lambda$$

Complementary Slackness

There are no inequality conditions.

Stationarity

From above we have

$$b = y - D^T\nu.$$

Taking the gradient with respect to θ gives the following condition.

$$0 \in \nabla_\theta\psi_P = -\nu + \lambda\partial(\|\theta\|_1) \to \nu \in \lambda\partial(\|\theta\|_1)$$

We can rewrite this.

$$\nu_i \in \begin{cases} \lambda & \theta_i > 0 \\ -\lambda & \theta_i < 0 \\ [-\lambda, \lambda] & \theta_i = 0 \end{cases}$$

3. Convert your KKT conditions into a single scalar equation involving a KKT residual. You will use this and the duality gap to evaluate the correctness of your algorithms in part 2.

Since we have strong duality by our constraint conditions, we know that the primal and dual solutions are the same. Thus, solving the dual problem will also solve the primal problem.

From our dual feasbility constraint, we know that $\|\nu\|_\infty \leq \lambda$. Then, our KKT residual will look like

$$\tau(\theta, \nu, i)_i = \begin{cases} |\nu_i - \lambda| & \theta_i > 0 \\ |\nu_i + \lambda| & \theta_i < 0 \\ ||\nu_i| - \lambda|_+ & \theta_i = 0 \end{cases}$$

and we need to check that $\max_i \tau(\theta, \nu, i) \to 0$. We can also check the duality gap.

$$\psi_P(b) - \psi_D(\nu)$$

4. How do you map a dual variable $\mathbf{v}$ to a primal variable $\mathbf{b}$?

We can use the first stationarity condition.

$$b = y - D^T \nu$$

Note that we may equivalently solve the following box constrained least squares problem.

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{y} - \mathbf{D}^\mathsf{T}\mathbf{v}\|_2^2 \\ &\text{subject to } \|\mathbf{v}\|_\infty \leq \lambda. \end{aligned}$$

5. Prove that the projection of $x \in \mathbb{R}$ onto the interval $[-\lambda, \lambda]$ is given by

$$P_{[-\lambda,\lambda]}(x) = \begin{cases} \lambda & x > \lambda \\ -\lambda & x < -\lambda \\ x & |x| \leq \lambda \end{cases}$$

6. Derive a coordinate descent algorithm for solving the dual problem and write out pseudocode for your algorithm.

To find our update rule, we want to find the derivative of our objective function for each ν_i. That is, we need

$$\frac{\partial}{\partial \nu_i}\left[\frac{1}{2}y^T y - y^T D^T \nu + \nu^T D D^T \nu\right].$$

Let's look at a small example to find the update rule. Take $k = 1$ and $n = 4$. Then

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \; D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \end{bmatrix}, \; \nu = \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix}$$

Let's look at the second and third terms of our objective function (because the first has no ν term, so it will have derivative 0).

$$\begin{aligned}
y^T D^T \nu &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \begin{bmatrix} d_{11} & d_{21} & d_{31} \\ d_{12} & d_{22} & d_{32} \\ d_{13} & d_{23} & d_{33} \\ d_{14} & d_{24} & d_{34} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix} \\
&= \begin{bmatrix} y_1 d_{11} + y_2 d_{12} + y_3 d_{13} + y_4 d_{14} & y_1 d_{21} + y_2 d_{22} + y_3 d_{23} + y_4 d_{24} & y_1 d_{31} + y_2 d_{32} + y_3 d_{33} + y_4 d_{34} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix} \\
&= \nu_1(y_1 d_{11} + y_2 d_{12} + y_3 d_{13} + y_4 d_{14}) + \nu_2(y_1 d_{21} + y_2 d_{22} + y_3 d_{23} + y_4 d_{24}) + \nu_3(y_1 d_{31} + y_2 d_{32} + y_3 d_{33} + y_4 d_{34}) \\
&= \sum_{i=1}^{3} \nu_i \sum_{j=1}^{4} y_j d_{ij}
\end{aligned}$$

$$\nu^T DD^T \nu = \begin{bmatrix} \nu_1 & \nu_2 & \nu_3 \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \end{bmatrix} \begin{bmatrix} d_{11} & d_{21} & d_{31} \\ d_{12} & d_{22} & d_{32} \\ d_{13} & d_{23} & d_{33} \\ d_{14} & d_{24} & d_{34} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix}$$

$$\begin{aligned}
&= v_1 \Big(d_{11}(d_{11}v_1 + d_{21}v_2 + d_{31}v_3) + \\
&d_{12}(d_{12}v_1 + d_{22}v_2 + d_{32}v_3) + \\
&d_{13}(d_{13}v_1 + d_{23}v_2 + d_{33}v_3) + \\
&d_{14}(d_{14}v_1 + d_{24}v_2 + d_{34}v_3) \Big) + \\
&v_2 \Big(d_{21}(d_{11}v_1 + d_{21}v_2 + d_{31}v_3) + \\
&d_{22}(d_{12}v_1 + d_{22}v_2 + d_{32}v_3) + \\
&d_{23}(d_{13}v_1 + d_{23}v_2 + d_{33}v_3) + \\
&d_{24}(d_{14}v_1 + d_{24}v_2 + d_{34}v_3) \Big) + \\
&v_3 \Big(d_{31}(d_{11}v_1 + d_{21}v_2 + d_{31}v_3) + \\
&d_{32}(d_{12}v_1 + d_{22}v_2 + d_{32}v_3) + \\
&d_{33}(d_{13}v_1 + d_{23}v_2 + d_{33}v_3) + \\
&d_{34}(d_{14}v_1 + d_{24}v_2 + d_{34}v_3 \Big) \\
&= \sum_{i=1}^{3} \nu_i \sum_{j=1}^{4} d_{ij} \sum_{k=1}^{3} d_{kj} \nu_k
\end{aligned}$$

Now we can take our derivative of these sums and find our update rule

$$\begin{aligned}
\frac{\partial f}{\partial \nu_i} &= \frac{\partial}{\partial \nu_i} \sum_{i=1}^{3} \nu_i \sum_{j=1}^{4} d_{ij} \sum_{k=1}^{3} d_{kj} \nu_k - \sum_{i=1}^{3} \nu_i \sum_{j=1}^{4} y_j d_{ij} \\
&= -\sum_{j=1}^{4} y_j d_{ij} + \nu_i \sum_{j=1}^{4} d_{ij}^2 + \sum_{k \neq i} \nu_j \sum_{j=1}^{4} d_{ij} d_{kj}
\end{aligned}$$

$$\begin{aligned}
0 &\overset{\text{set}}{=} -\sum_{j=1}^{4} y_j d_{ij} + \nu_i \sum_{j=1}^{4} d_{ij}^2 + \sum_{k \neq i} \nu_j \sum_{j=1}^{4} d_{ij} d_{kj} \\
\nu_i^+ &= \frac{\sum_{j=1}^{4} y_j d_{ij} - \sum_{k \neq i} \nu_j \sum_{j=1}^{4} d_{ij} d_{kj}}{\sum_{j=1}^{4} d_{ij}^2}
\end{aligned}$$

In general, our update rule is.

$$\nu_i = \frac{\sum_{j=1}^{n} d_{ij} y_j - \sum_{k \neq i} \nu_k \sum_{j=1}^{n} d_{kj} d_{ij}}{\sum_{j=1}^{n} d_{ij}^2}.$$

```
initialize v0, y, D, step_size

update(v, i)
```

```
{
  numerator = sum( d[i, j] * y[j], {j, 1, n} )
  numerator = numerator - sum( v[k] * sum( d[k,j] * d[i,j], {j, 1, n} ), {k != i})
  denominator = sum( d[i, j]^2, {j, 1, n} )

  return numerator / denominator
}

vnew = v0

repeat

  for i in vnew:
    vnew[i] = update(v, i)

until convergence
```

7. Prove that the dual objective is Lipschitz differentiable with constant $L = \|\mathbf{D}\|_{\text{op}}^2$.

$$\begin{aligned} f(\nu) &= \frac{1}{2}\left\|y - D^T\nu\right\|_2^2 \\ &= \frac{1}{2}y^Ty - y^TD^T\nu + \frac{1}{2}\nu^TDD^T\nu \end{aligned}$$

$$\nabla_\nu f(\nu) = DD^T\nu - Dy$$

$$\nabla_\nu^2 f(\nu) = DD^T$$

$$\begin{aligned} \left\|\nabla_\nu^2 f(\nu)\right\|_2 &= \left\|DD^T\right\|_2 \\ &\le \|D\|_2\left\|D^T\right\|_2 \\ &= \|D\|_2\|D\|_2 \\ &= \|D\|_2^2 \\ &= \|D\|_{\text{op}}^2 \end{aligned}$$

8. Derive a proximal gradient algorithm for solving the dual problem and write out pseudocode for your algorithm.

The gradient of our objective with respect to ν is

$$\nabla_\nu f = DD^T\nu - Dy$$

```
initialize v0, y, D, step_size

projection(v)
{
  if (v > lambda)
    proj = lambda
  elif( v < -lambda)
    proj = -lambda
  else
    proj = v
  return proj
}

gradient_step(v)
{
  return projection( v - step_size * (D * D^T * v  - D * y ) )
}

vnew = v0

repeat
  vnew = gradient_step(vnew)
until convergence
```

2

You can find my code in `homework5/lasso_driver.R` and in my `jhickeyST790/R/generalized_lasso.R`.

Part 2. You will implement a coordinate descent algorithm and proximal gradient algorithm for solving the trend filtering problem.

Here are some helper functions.

```r
#' Compute Lasso primal objective
#'
#' @param y response
#' @param b primal variable
#' @param D differencing matrix
#' @param lambda regularization parameter
#' @export
lasso_primal = function(y, b, D, lambda )
{
 return( 1/2 * norm(y - b, "2")^2 + lambda * norm( D %*% b, '1') )
}


#' Compute Lasso dual objective
#'
#' @param y response
#' @param v dual variable
#' @param D differencing matrix
#' @export
lasso_dual = function(y, v, D)
{
 return(1/2 * norm(y - t(D) %*% v, '2')^2 )
}


#' Compute Lasso dual objective original
#'
#' @param y response
#' @param v dual variable
#' @param D differencing matrix
#' @export
lasso_dual_original = function(y, v, D)
{
  return( 1/2 * norm(y, '2')^2 - 1/2 * norm(y - t(D) %*% v, '2')^2 )
}

#' Compute Lasso dual objective
#'
#' @param y response
#' @param v dual variable
#' @param D differencing matrix
#' @export
lasso_dual_gradient = function(y, v, D)
{
 return( D %*% t(D) %*% v - D %*% y )
}

#' Compute the primal variable b from the dual variable
#'
#' @param y response
#' @param v dual variable
#' @param D differencing matrix
#' @return the primal variable b
#' @export
dual_primal_map_b = function(y, v, D)
{
 return(y - t(D) %*% v)
}


#' Compute the primal variable theta from the dual variable
#'
#' @param y response
#' @param v dual variable
#' @param D differencing matrix
#' @return the primal variable b
#' @export
dual_primal_map_theta = function(y, v, D)
{
 return(D %*% dual_primal_map_b(y, v, D))
}
```

Step 1: Write a function to compute the kth order differencing matrix $\mathbf{D}_n^{(k)}$.

```r
#' Compute kth order differencing matrix
#'
#' @param k order of the differencing matrix
#' @param n Number of time points
#' @param negative Flip signs (default: FALSE)
#'  TRUE:   row one of D^1_n will look like 1 -1 0 ... 0
#'  FALSE:  row one of D^1_n will look like -1 1 0 ... 0
#' @return kth order differencing matrix
myGetDkn <- function(k, n, negative=FALSE)
{
  library(Matrix)
  neg = negative

  ii = 1 + (!neg) * (-2)
  iiplusone = -ii

   # create D1n
  if (k==1)
  {
    D1 = matrix(nrow = n-1, ncol = n)
    zeros_matrix = rep(0, n)

    for(i in 1:n-1)
    {
      D1[i,] = zeros_matrix
      D1[i,i] = ii
      D1[i, i+1] = iiplusone
    } # for
    return(Matrix(D1, sparse = TRUE))
  } # endif
  return(myGetDkn(k=1, n=n - k + 1, neg) %*% myGetDkn(k=k-1, n=n, neg))

}
```

Step 2: Write a function to compute the KKT residual you derived in part 1.

```r
#' Compute KKT residual
#'
#' @param y response
#' @param b primal variable
#' @param theta primal variable
#' @param v dual variable
#' @param D differencing matrix
#' @param lambda regularization parameter
#' @export
kkt_residual <- function(y, b, theta, v, D, lambda) {

 tau = v

 # round theta so that values near 0 are 0
 theta = round(theta, 7)

 tau[theta != 0 ] = abs( abs(v[theta != 0]) - lambda )
 tau[theta == 0 ] = max( abs(v[theta == 0]) - lambda, 0)

 return(max(tau))
}
```

Step 3: Write a function to compute the duality gap.

```r
#' Compute duality gap
#'
#' @param y response
#' @param b primal variable
#' @param v dual variable
#' @param D differencing matrix
#' @param lambda regularization parameter
#' @export
duality_gap <- function(y, b, v, D, lambda) {
 return( lasso_primal(y, b, D, lambda) -lasso_dual_original(y, v, D) )
}
```

Step 4: Write a coordinate descent algorithm for solving the dual problem. Use the relative change in function values as a stopping criterion.

```r
#' Coordinate descent update
#'
#' @param y response
#' @param v dual variable
#' @param lambda regularization parameter
#' @export
coordinate_descent = function(v, lambda, D, y)
{
  v_update = v
  n = length(v)
  bool_array = rep(TRUE, n)

  for (i in 1:n)
  {
    bool_array_i = bool_array
    bool_array_i[i] = FALSE

    # sum( d_ij * y_j, j=1:n)
    numerator = sum( D[i, ] * y)

    # numerator - \sum( v_k * \sum( d_kj* j_ij, j=1:n)  , k != i)
    numerator = numerator + sum( v[bool_array_i] * D[bool_array_i, ] * D[i,] )

    # denominator = sum( d_ij^2 , j: 1, n)
    denominator = sum( D[i, ]^2 )

    v_update[i] = numerator / denominator

    v_update[i] = lasso_proxmap(v_update[i], lambda)
  }

  return(v_update)
}


#' Solve trend-filtering by coordinate descent on dual problem
#'
#' @param y response
#' @param k order of differencing matrix
#' @param v Initial dual variable
#' @param lambda regularization parameter
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
#' @return
#' \itemize{
#'   \item{final_iterate}{The final iterate}
#'   \item{objective_history}{A vector of objective function value from each iteration.}
#'   \item{relative_objective_history}{A vector of relative change in object value between iterations.}
#'   \item{relative_iterate_history}{A vector of relative change in iterate value between iterations.}
#'   \item{kkt_residual_history}{A vector of KKT residual value between iterations.}
#'   \item{duality_gap_history}{A vector of duality gap value between iterations.}
#' }
trend_filter_cd <- function(y, k, v, lambda=0, max_iter=1e2, tol=1e-3) {

  # Create differencing matrix
  n = length(y)
  D = as.matrix(myGetDkn(k, n))

  # Get Lipschitz step size
  t = 1 / norm(D, '2')^2

  b = dual_primal_map_b(y, v, D)
  theta = dual_primal_map_theta(y, v, D)

  # create vectors
  objective_history = c()
  relative_objective_history = c()
  relative_iterate_history = c()
  kkt_residual_history = c()
  duality_gap_history = c()


  # initialize variables
  current_iterate = v
  objective_history[1] = lasso_dual(y, current_iterate, D)
  relative_objective_history[1] = 0
  relative_iterate_history[1] = 0
  kkt_residual_history[1] = kkt_residual(y = y, b = b,
                                         theta = theta, v= current_iterate, D = D, lambda = lambda)
  duality_gap_history[1] = duality_gap(y, b, v, D, lambda)

  # perform gradient descent until either
  #   we have changed less than the tolerance
  #   we have done the maximum number of iterations
  for (i in 2:max_iter)
  {

    # Coordinate Descent step
    new_iterate = coordinate_descent(v = current_iterate,
                                     lambda = lambda,
                                     D = D,
                                   y = y)

    b = dual_primal_map_b(y, new_iterate, D)
    theta = dual_primal_map_theta(y, new_iterate, D)

    objective_history[i] = lasso_dual(y, new_iterate, D)

    relative_objective_history[i] = abs((objective_history[i] - objective_history[i-1]))/(1 + abs(objec
    relative_iterate_history[i] = norm(new_iterate - current_iterate, '2') / (1 + norm(new_iterate, '2'

    kkt_residual_history[i] = kkt_residual(y ,b, theta, new_iterate, D, lambda)
    duality_gap_history[i] = duality_gap(y, b, new_iterate, D, lambda)

    current_iterate = new_iterate

    # break if change less than tolerated amount
    if (duality_gap_history[i] <= tol)
      break

  } # end for

  return_list = list(
    "final_iterate" = current_iterate,
    "objective_history" = objective_history,
    "relative_objective_history" = relative_objective_history,
    "relative_iterate_history" = relative_iterate_history,
    "kkt_residual_history" = kkt_residual_history,
    "duality_gap_history" = duality_gap_history
  )

  return(return_list)
}
```

Your function should return

- The final iterate value
- The objective function values
- The relative change in the function values
- The relative change in the iterate values
- The KKT residual after every iteration
- The duality gap after every iteration

Step 5: Write a proximal gradient algorithm for solving the dual problem. Use the relative change in function values as a stopping criterion. You may either use a fixed step size or write your own backtracking line search functions.

```r
#' Compute Lasso proxmap
#'
#' @param v dual variable
#' @param lambda regularization parameter
#' @export
lasso_proxmap = function(v, lambda){
 return( ifelse(v > lambda, lambda,
                ifelse(v < - lambda, -lambda, v) ))
}

#' Proximal Gradient Step
#'
#' @param proxmap handle to a function that returns the proximal map
#' @param gradf gradient of the objective at x
#' @param x current parameter estimate
#' @param t step-size
#' @param lambda regularization parameter
#' @export
proximal_gradient_step <- function(proxmap, gradf, x, t, lambda) {
 return( proxmap( x - t * gradf, lambda) )
}


#' Solve trend-filtering by proximal gradient on dual problem
#'
#' @param y response
#' @param k order of differencing matrix
#' @param v Initial dual variable
#' @param lambda regularization parameter
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
#' @return
#' \itemize{
#'   \item{final_iterate}{The final iterate}
#'   \item{objective_history}{A vector of objective function value from each iteration.}
#'   \item{relative_objective_history}{A vector of relative change in object value between iterations.}
#'   \item{relative_iterate_history}{A vector of relative change in iterate value between iterations.}
#'   \item{kkt_residual_history}{A vector of KKT residual value between iterations.}
#'   \item{duality_gap_history}{A vector of duality gap value between iterations.}
#' }
trend_filter_pg <- function(y, k, v, lambda=0, max_iter=1e2, tol=1e-3) {

 # Create differencing matrix
 n = length(y)
 D = as.matrix(myGetDkn(k, n))

 # Get Lipschitz step size
 t = 1 / norm(D, '2')^2

 b = dual_primal_map_b(y, v, D)
```

```r
theta = dual_primal_map_theta(y, v, D)

# create vectors
objective_history = c()
relative_objective_history = c()
relative_iterate_history = c()
kkt_residual_history = c()
duality_gap_history = c()


# initialize variables
current_iterate = v
objective_history[1] = lasso_dual(y, current_iterate, D)
relative_objective_history[1] = 0
relative_iterate_history[1] = 0
kkt_residual_history[1] = kkt_residual(y = y, b = b,
                                       theta = theta, v= current_iterate, D = D, lambda = lambda)
duality_gap_history[1] = duality_gap(y, b, v, D, lambda)

# perform gradient descent until either
#   we have changed less than the tolerance
#   we have done the maximum number of iterations
for (i in 2:max_iter)
{

  # Calculate gradient for current x
  gradient_value = lasso_dual_gradient(y, v, D)

  # Gradient step to get new objective iterate value
  new_iterate = proximal_gradient_step(proxmap = lasso_proxmap,
                                       gradf = gradient_value,
                                       x = current_iterate,
                                       t = t,
                                       lambda = lambda)

 b = dual_primal_map_b(y, new_iterate, D)
 theta = dual_primal_map_theta(y, new_iterate, D)

 objective_history[i] = lasso_dual(y, new_iterate, D)

 relative_objective_history[i] = abs((objective_history[i] - objective_history[i-1]))/(1 + abs(objecti
 relative_iterate_history[i] = norm(new_iterate - current_iterate, '2') / (1 + norm(new_iterate, '2'))

 kkt_residual_history[i] = kkt_residual(y ,b, theta, new_iterate, D, lambda)
 duality_gap_history[i] = duality_gap(y, b, new_iterate, D, lambda)

 current_iterate = new_iterate

 # break if change less than tolerated amount
 if (duality_gap_history[i] <= tol)
  break

} # end for
```

```
 return_list = list(
  "final_iterate" = current_iterate,
  "objective_history" = objective_history,
  "relative_objective_history" = relative_objective_history,
  "relative_iterate_history" = relative_iterate_history,
  "kkt_residual_history" = kkt_residual_history,
  "duality_gap_history" = duality_gap_history
 )

 return(return_list)
}
```

Your function should return

- The final iterate value
- The objective function values
- The relative change in the function values
- The relative change in the iterate values
- The KKT residual after every iteration
- The duality gap after every iteration

Step 6: Use your two trend filtering function to smooth some interesting time series data. For example, you might use the tseries R package on CRAN (see the function **get.hist.quote**) to download historical financial data for the daily closing prices of Apple stock over the past two years. You may use the same data used in Homework 4. Try several λ values - different enough to generate noticably different smoothed estimates - and at least two differencing matrix orders, e.g. $\mathbf{D}_n^{(2)}$ and $\mathbf{D}_n^{(3)}$.

For both algorithms (and all λ and all differencing matrices) plot the following

- The noisy data and smoothed estimates.

For both algorithms (and one λ and one differencing matrix) plot the following against the iteration

- The relative change in the function values
- The relative change in the iterate values
- The KKT residual after every iteration
- The duality gap after every iteration

For large λ our dual problem will become essentially unconstrained. So notice the difference in output when we increase λ.

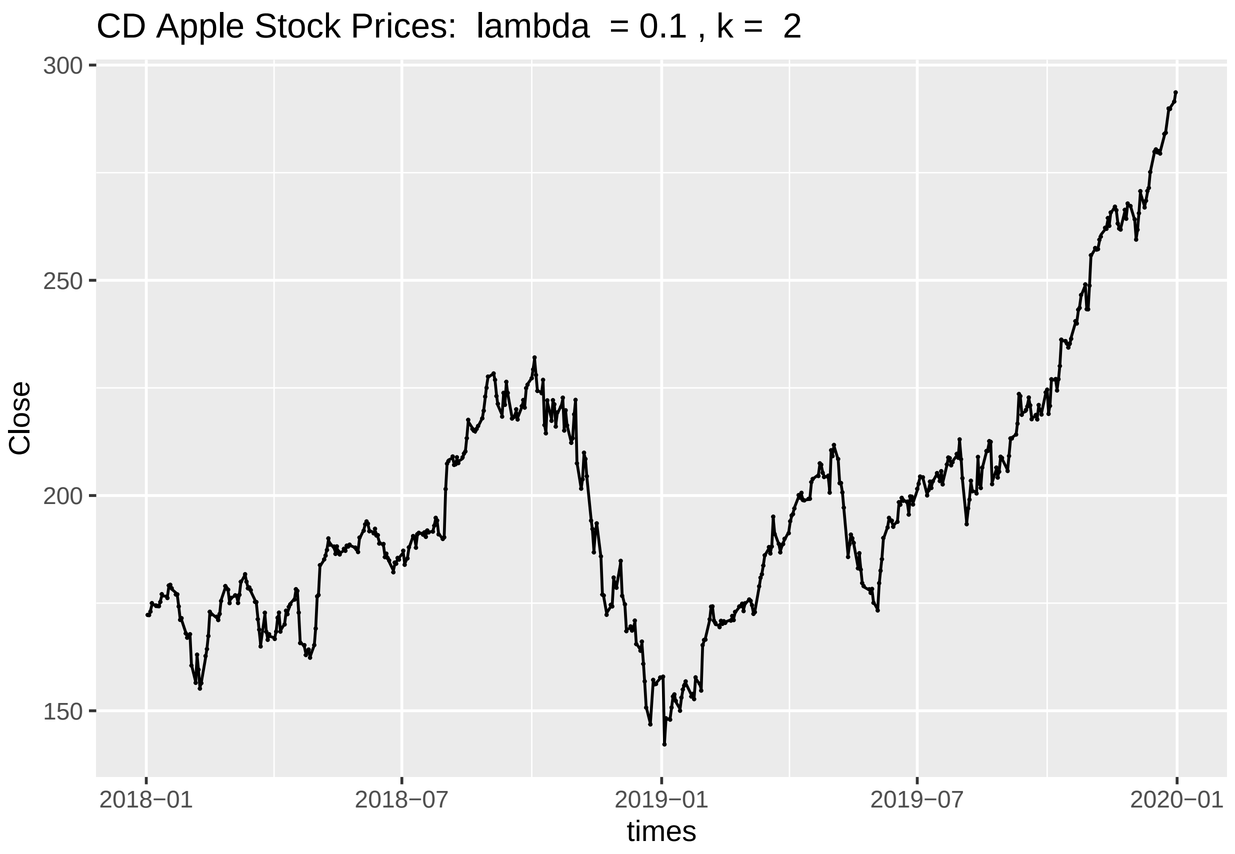
We will look at APPL closing stock prices from 2018-01-01 to 2020-01-01. There is some extra drive and plotting code that you can find in the driver file script and below in the RMarkdown, however I will hide it to save space.

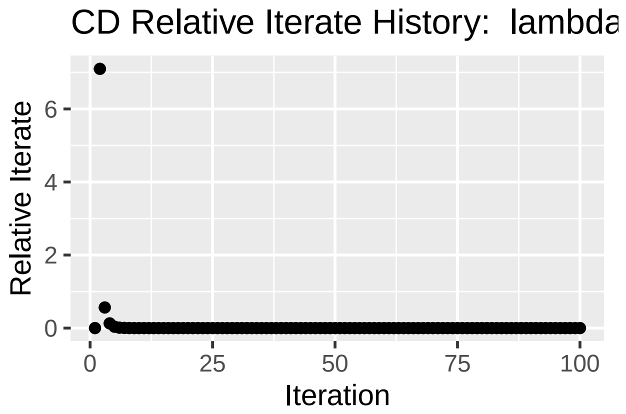
Also, we will use the relative change in duality gap as our stopping condition so that we can try out the new functionality! We will, of course, plot all output.

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

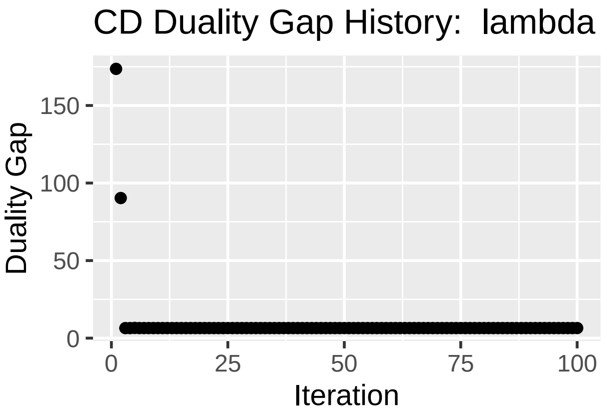
```
## time series starts 2018-01-02
## time series ends   2019-12-31
```

```
k = 2
lambda = 0.1
max_iter = 1e2
tol=1e-3

make_plots(y=y,k=k, lambda = lambda, max_iter=max_iter, tol=tol)
```

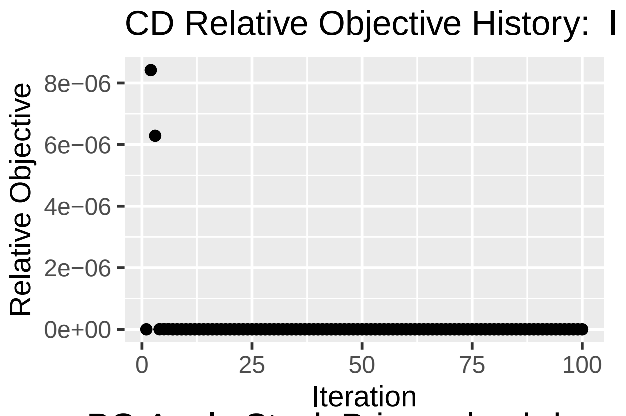




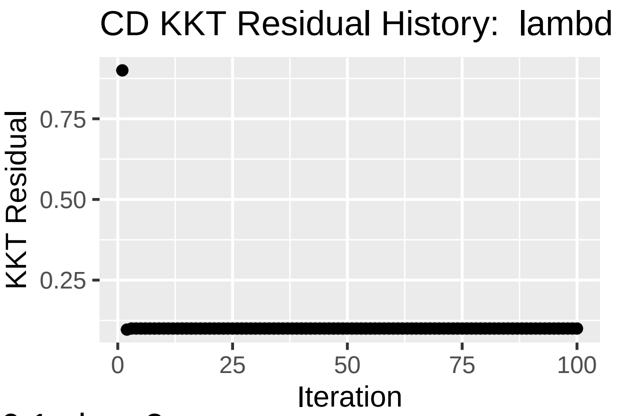
CD Relative Iterate History: lambda
Relative Iterate
Iteration
0
2
4
6
0
25
50
75
100



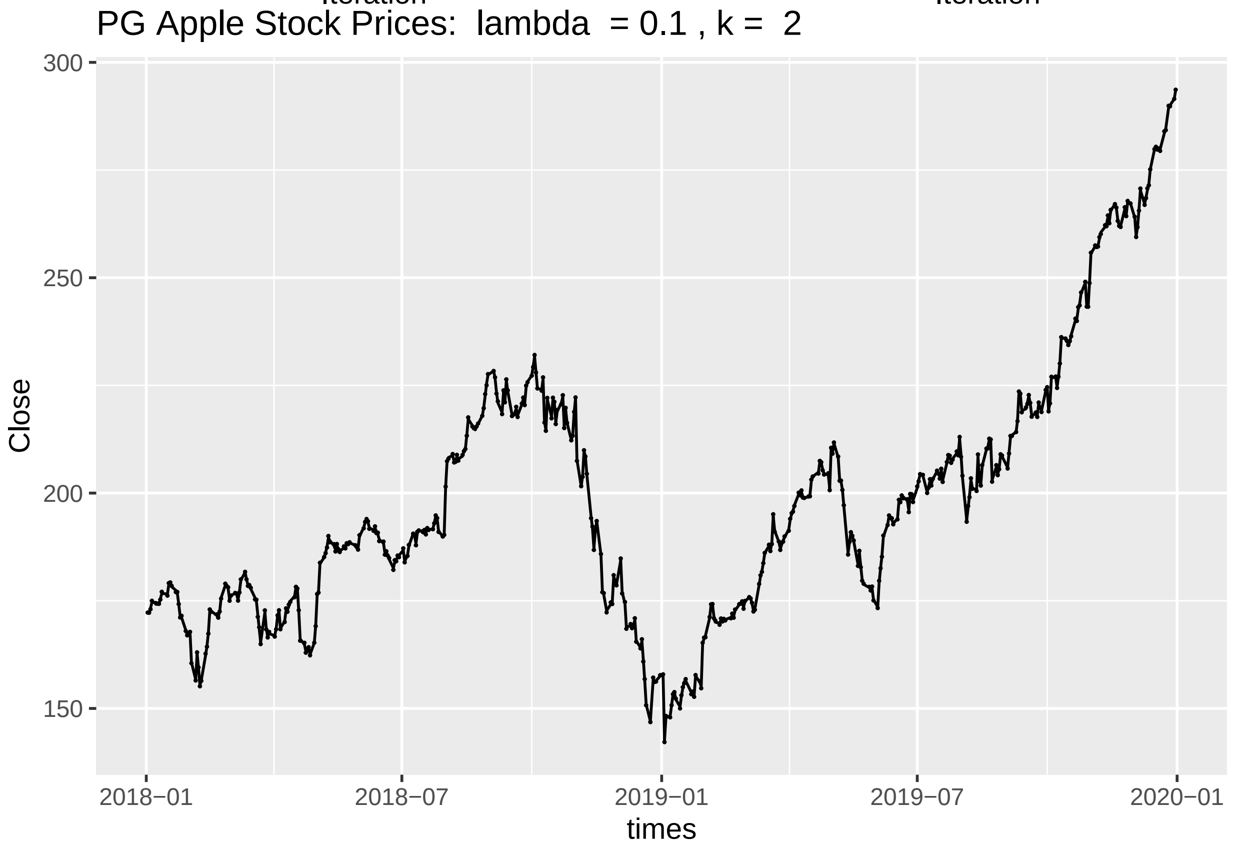
CD Duality Gap History: lambda
Duality Gap
Iteration
0
50
100
150
0
25
50
75
100



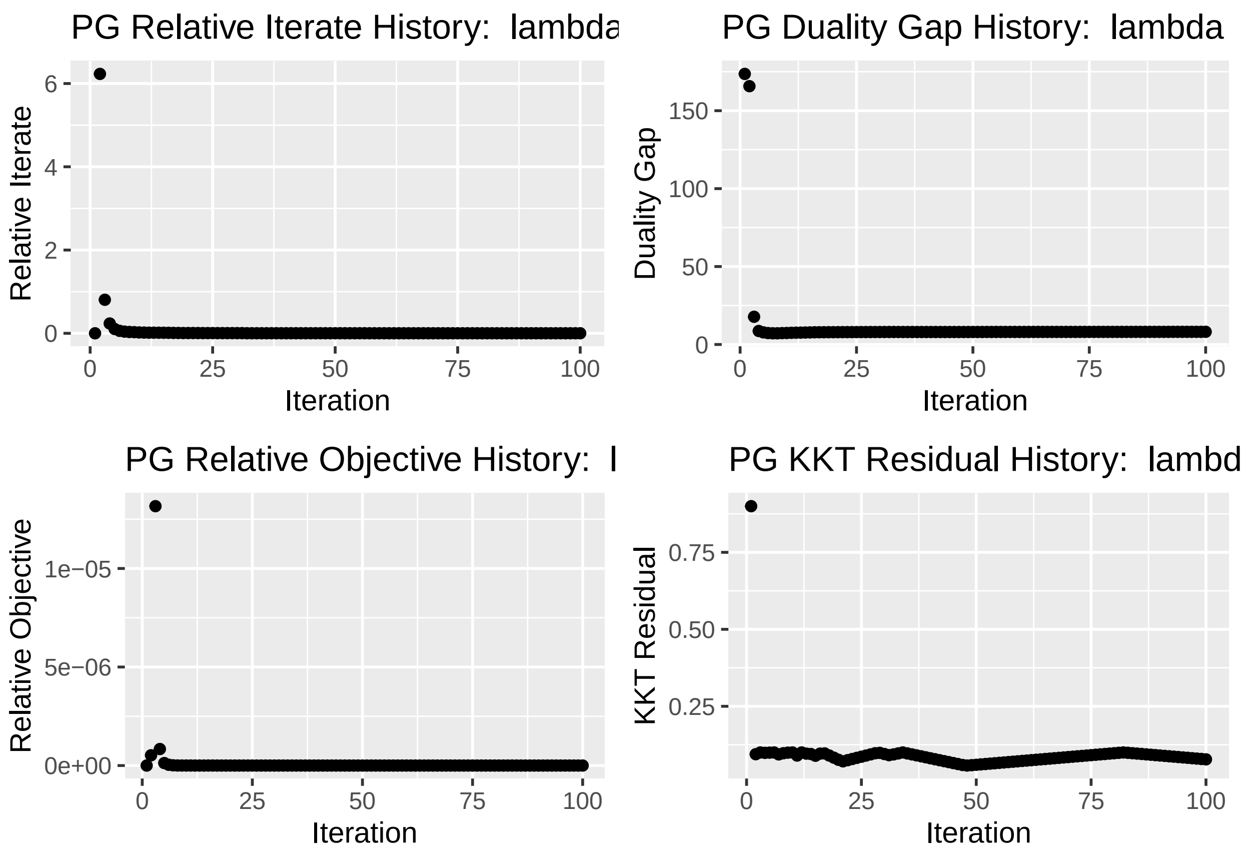
CD Relative Objective History: l
Relative Objective
Iteration
0e+00
2e−06
4e−06
6e−06
8e−06
0
25
50
75
100



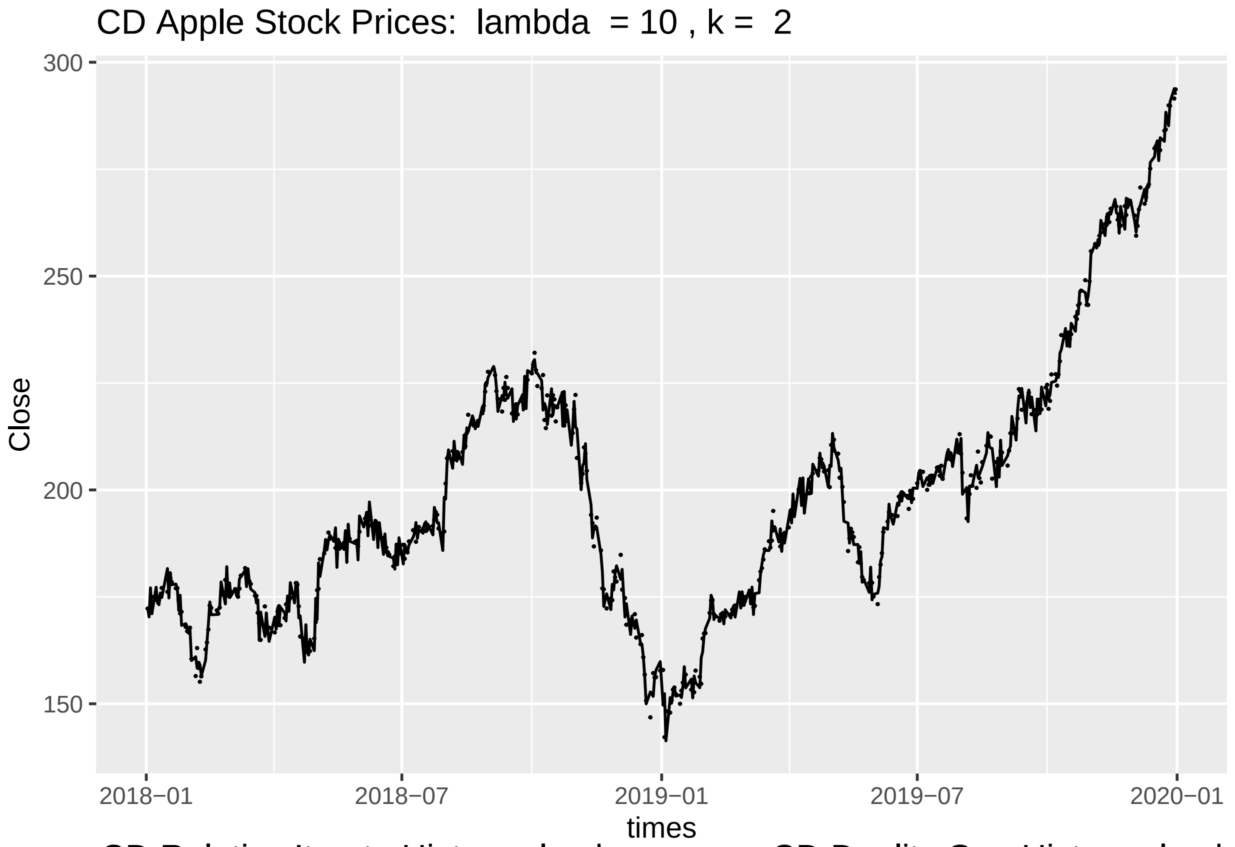
CD KKT Residual History: lambd
KKT Residual
Iteration
0.25
0.50
0.75
0
25
50
75
100

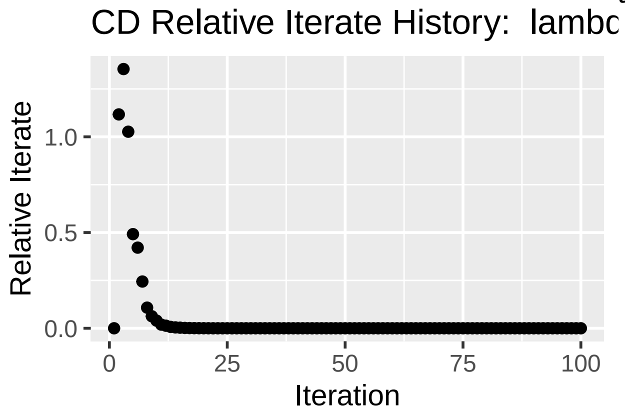


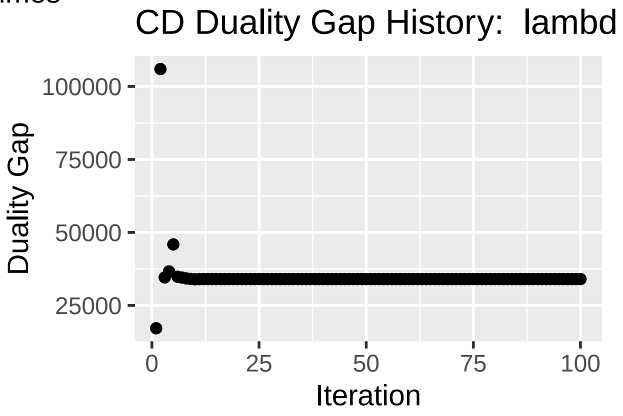
PG Apple Stock Prices: lambda = 0.1 , k = 2
Close
times
150
200
250
300
2018−01
2018−07
2019−01
2019−07
2020−01

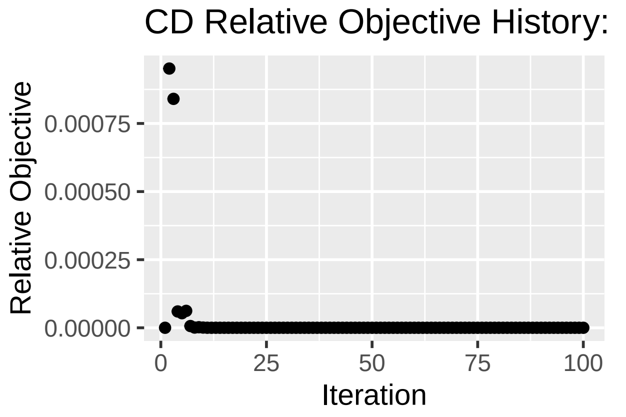


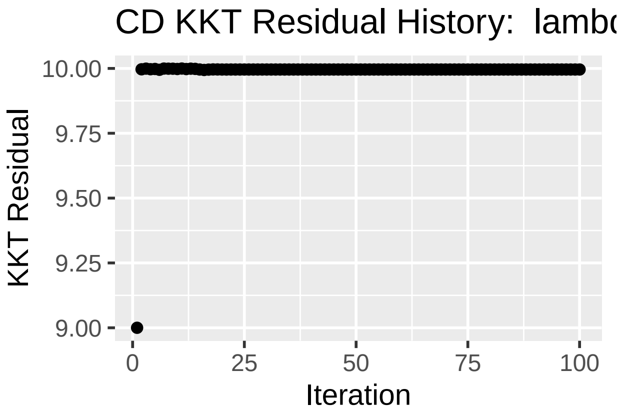
```r
k=2
lambda = 10
make_plots(y=y,k=k, lambda = lambda, max_iter=max_iter, tol=tol)
```

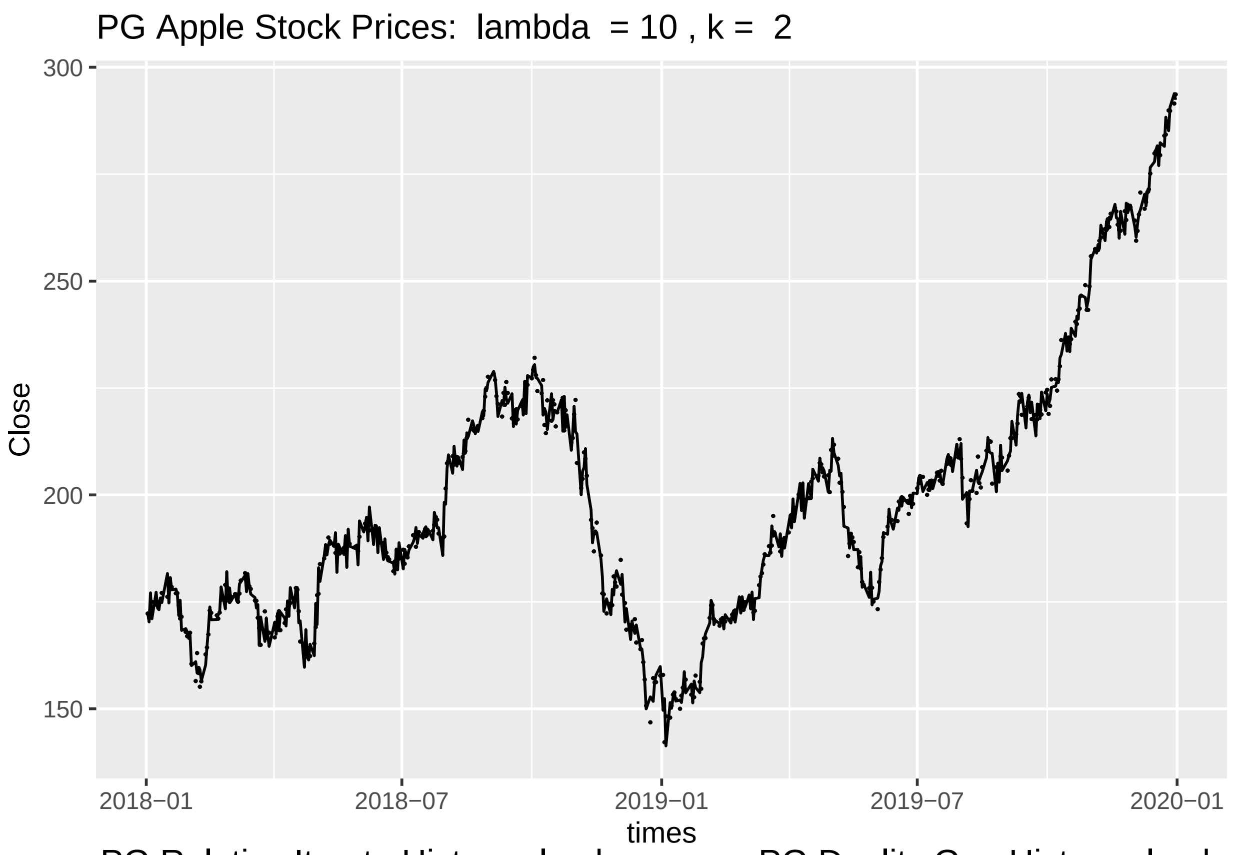

CD Apple Stock Prices: lambda = 10 , k = 2
Close
300
250
200
150
2018−01
2018−07
2019−01
2019−07
2020−01
times

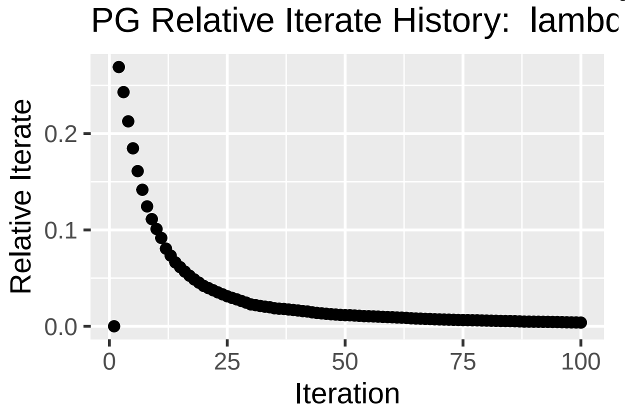

CD Relative Iterate History: lambc
Relative Iterate
1.0
0.5
0.0
0
25
50
75
100
Iteration

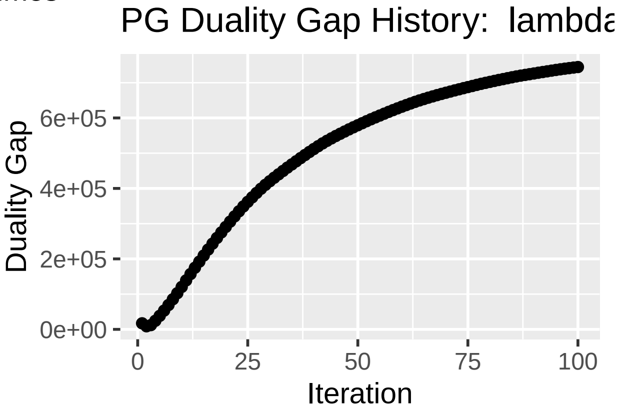

CD Duality Gap History: lambd
Duality Gap
100000
75000
50000
25000
0
25
50
75
100
Iteration

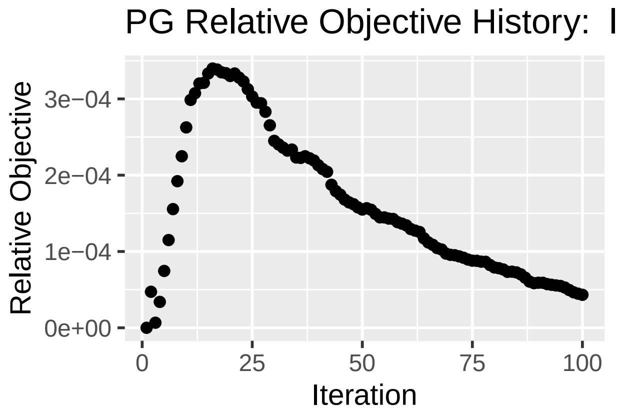

CD Relative Objective History:
Relative Objective
0.00075
0.00050
0.00025
0.00000
0
25
50
75
100
Iteration

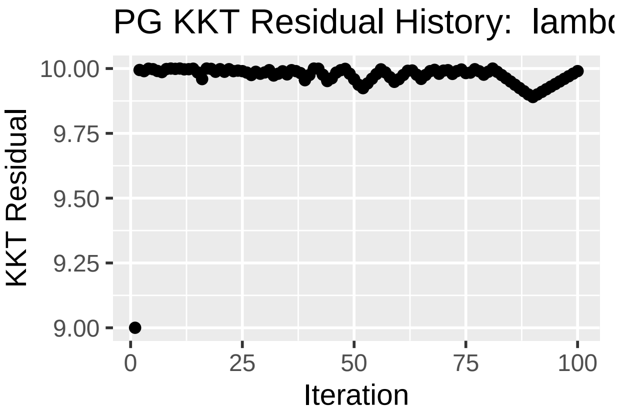

CD KKT Residual History: lambc
KKT Residual
10.00
9.75
9.50
9.25
9.00
0
25
50
75
100
Iteration


PG Apple Stock Prices: lambda = 10 , k = 2
Close
300
250
200
150
2018−01
2018−07
2019−01
2019−07
2020−01
times

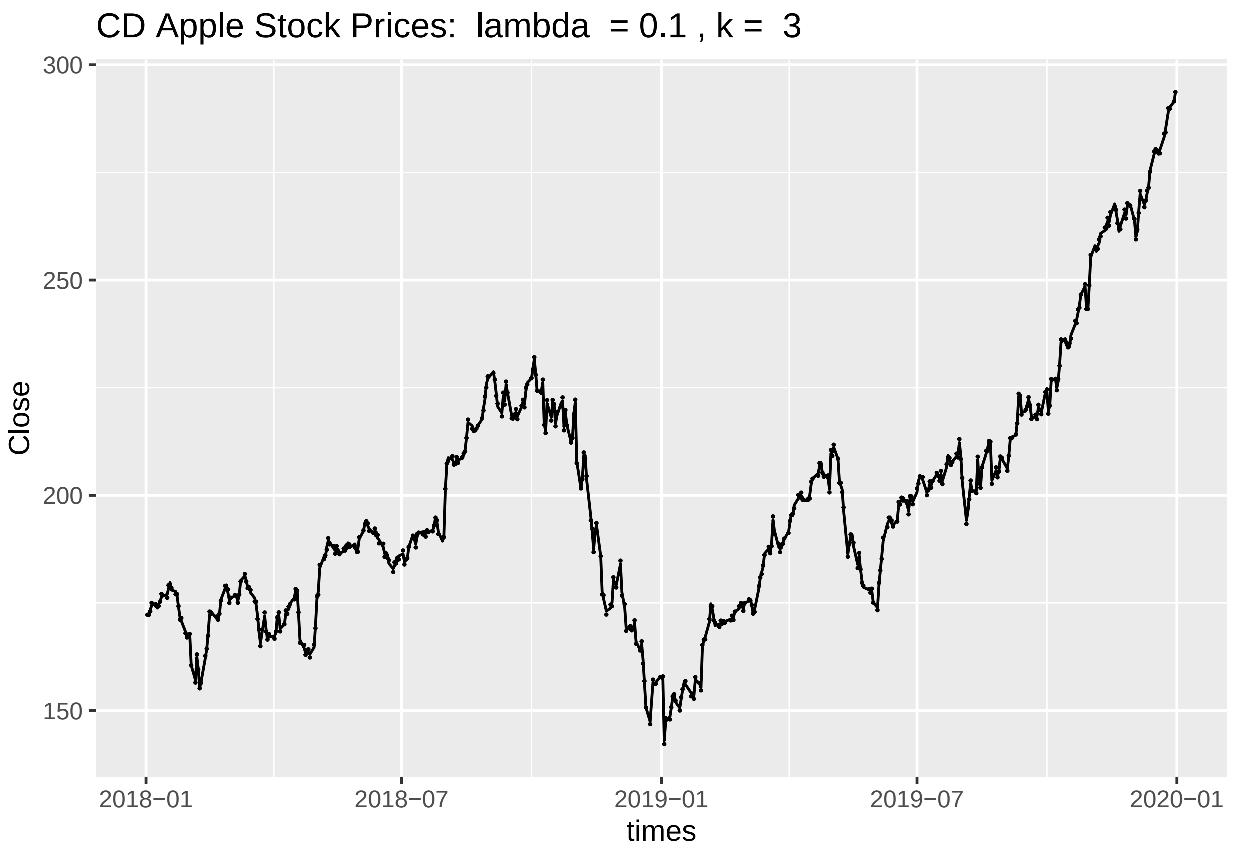

PG Relative Iterate History: lambc
Relative Iterate
0.2
0.1
0.0
0
25
50
75
100
Iteration

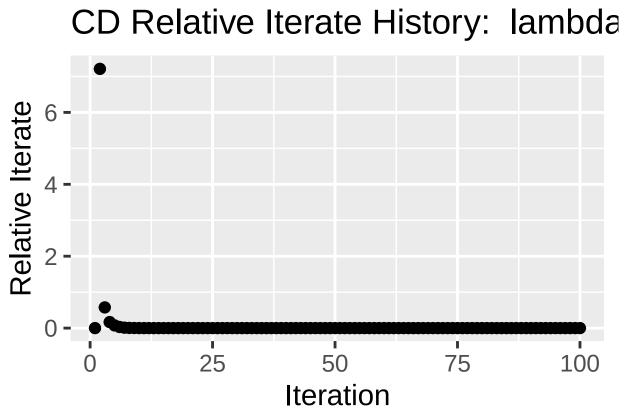

PG Duality Gap History: lambda
Duality Gap
6e+05
4e+05
2e+05
0e+00
0
25
50
75
100
Iteration

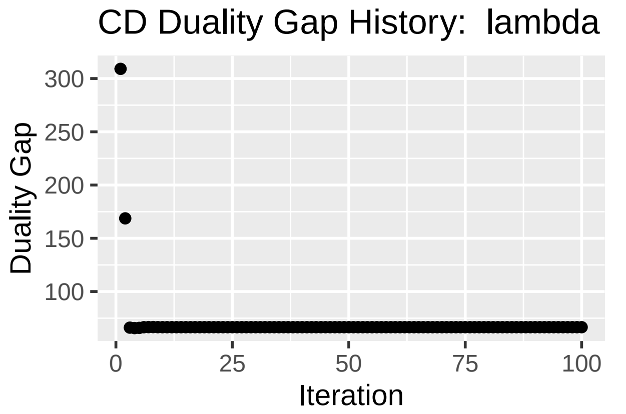

PG Relative Objective History: l
Relative Objective
3e−04
2e−04
1e−04
0e+00
0
25
50
75
100
Iteration

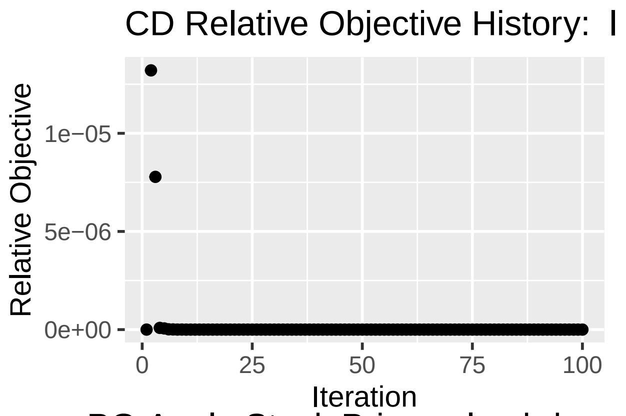

PG KKT Residual History: lambc
KKT Residual
10.00
9.75
9.50
9.25
9.00
0
25
50
75
100
Iteration

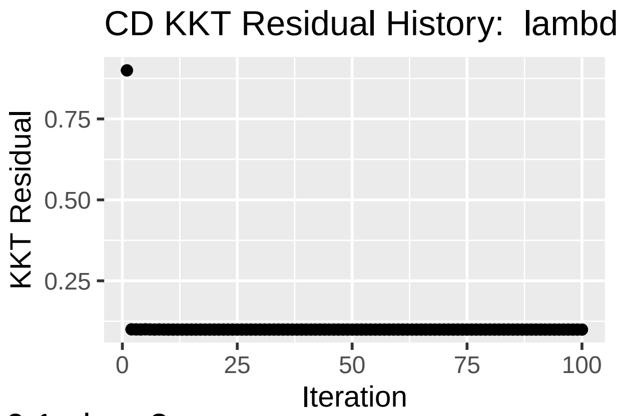
```r
k = 3
lambda = 0.1
max_iter = 1e2
tol=1e-3

make_plots(y=y,k=k, lambda = lambda, max_iter=max_iter, tol=tol)
```

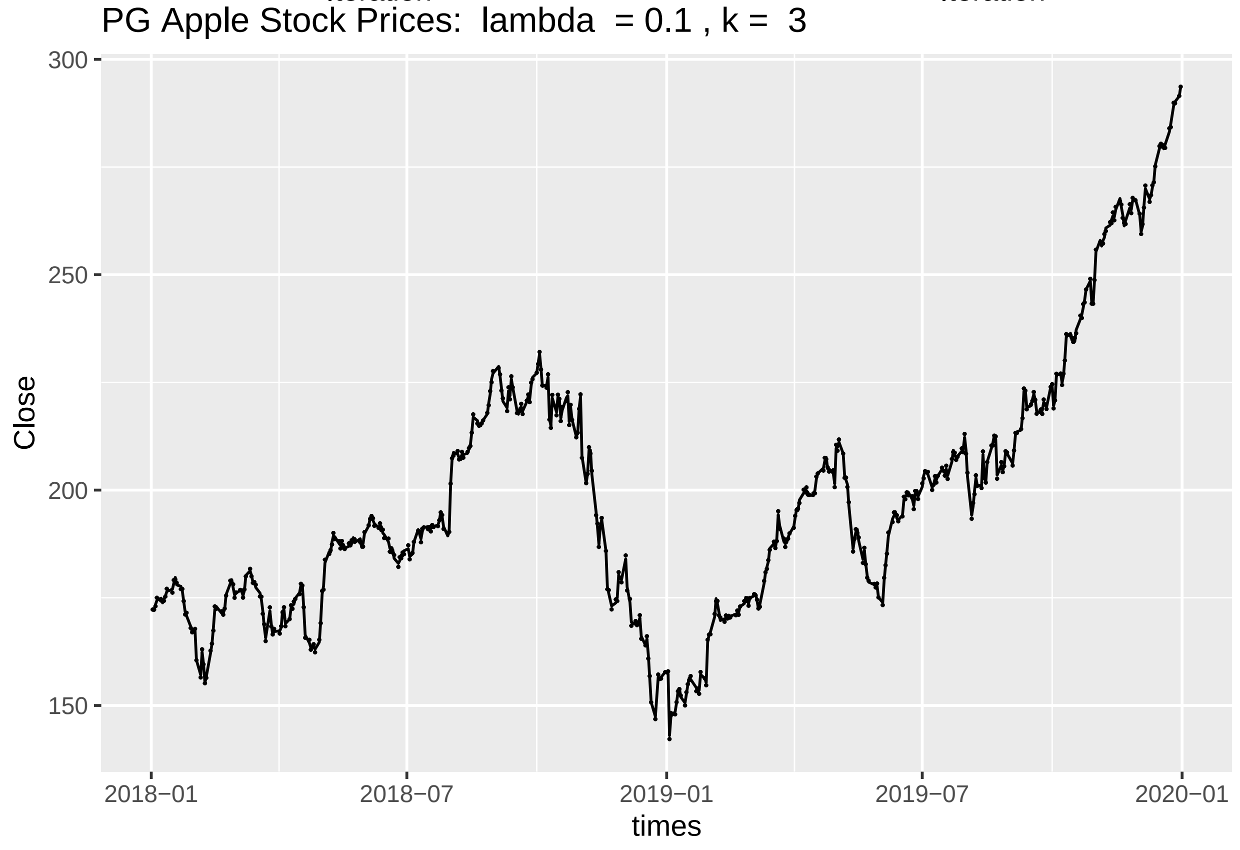


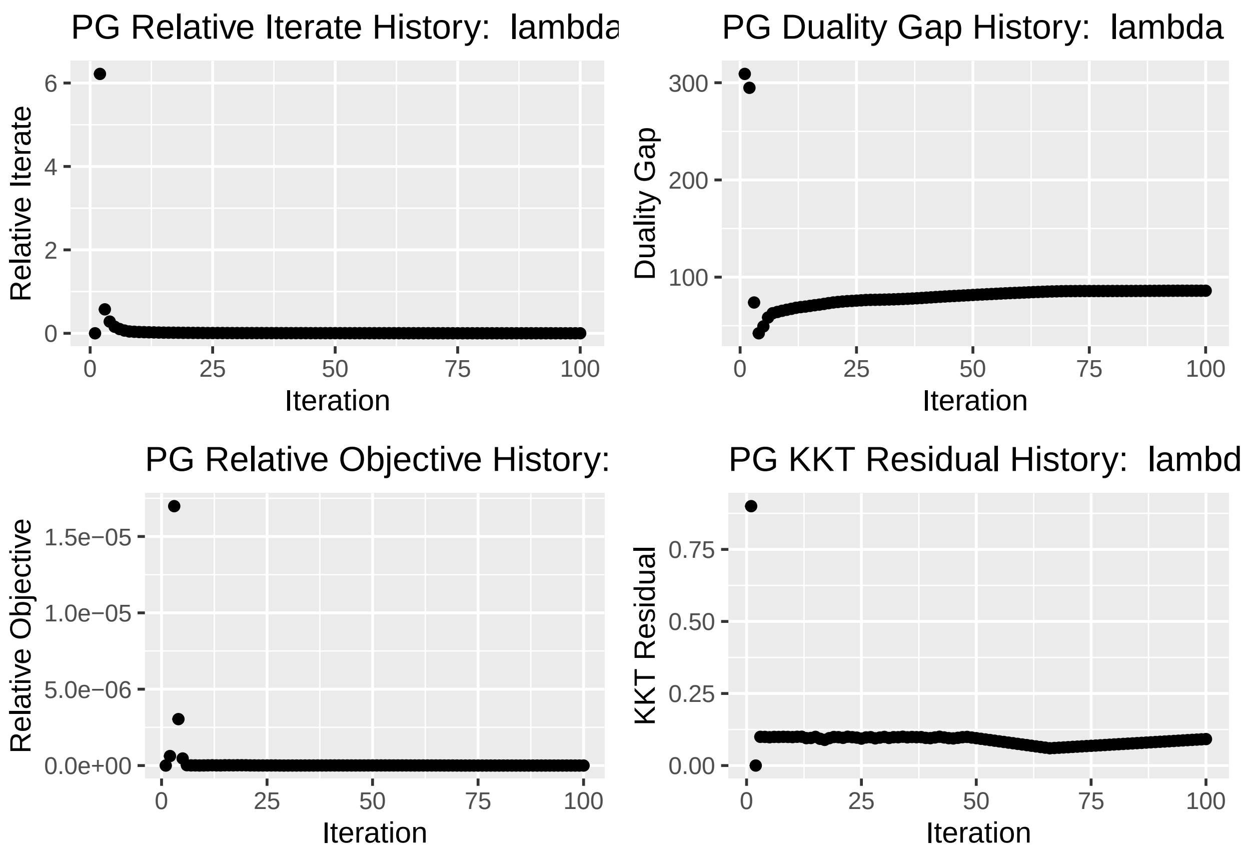

CD Relative Iterate History: lambda
Relative Iterate
Iteration


CD Duality Gap History: lambda
Duality Gap
Iteration

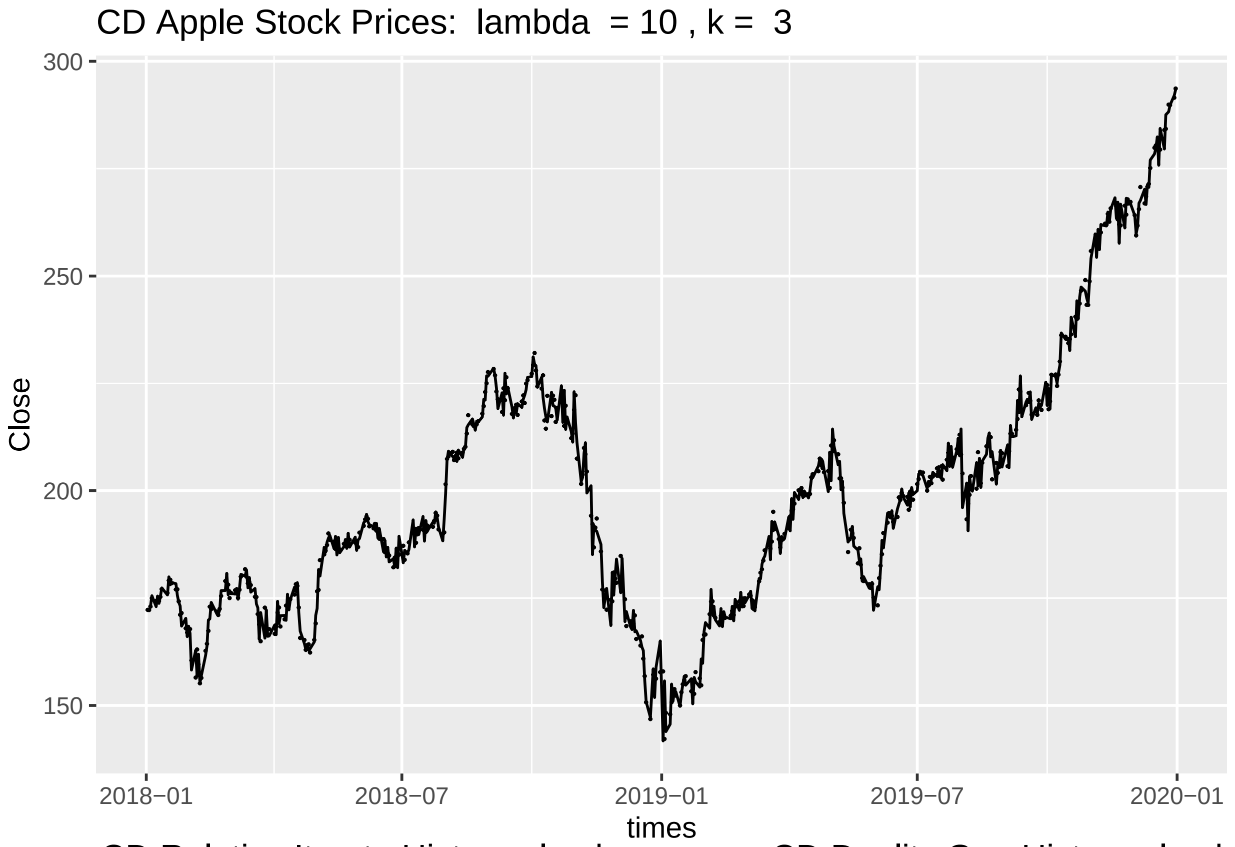

CD Relative Objective History: l
Relative Objective
Iteration

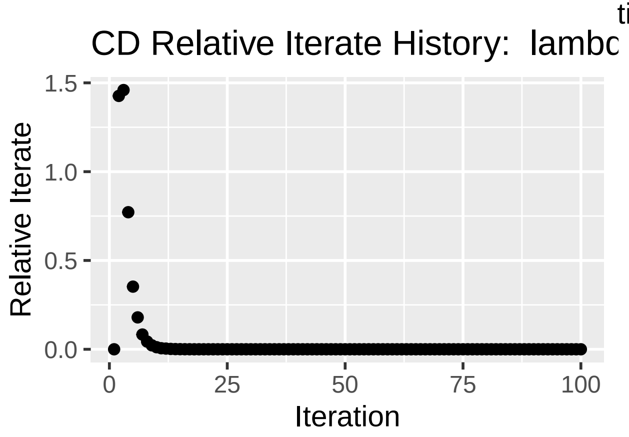

CD KKT Residual History: lambd
KKT Residual
Iteration

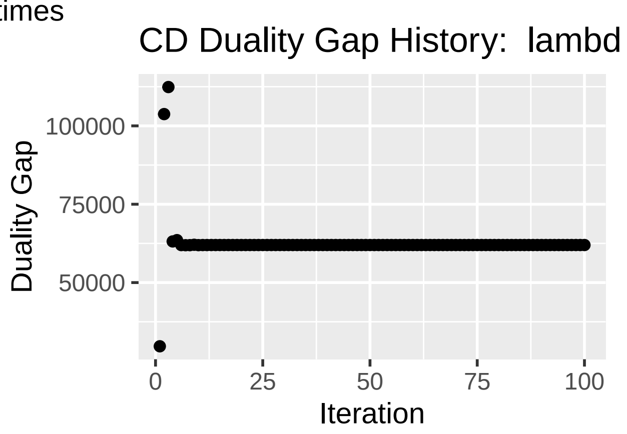

PG Apple Stock Prices: lambda = 0.1 , k = 3
Close
times

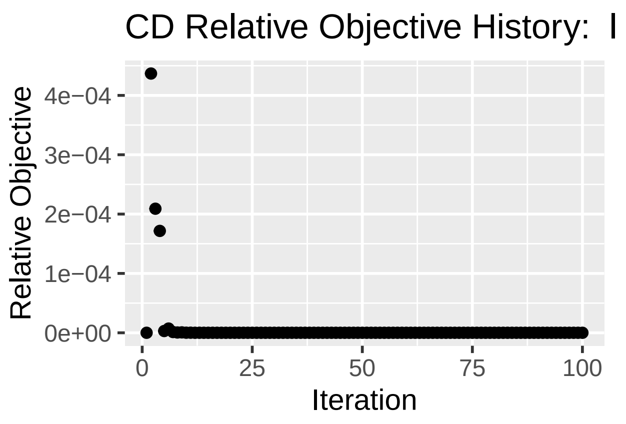


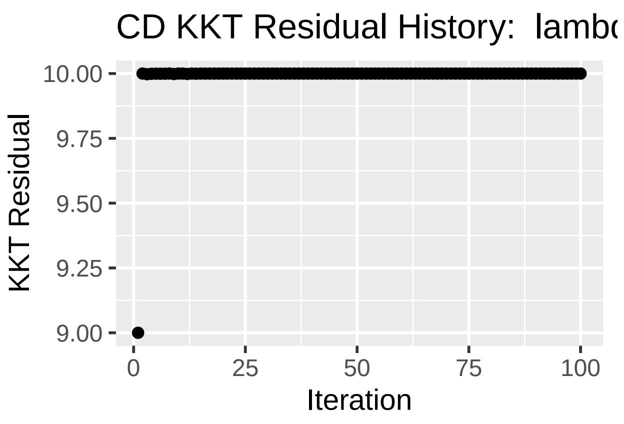
```
k=3
lambda = 10
make_plots(y=y,k=k, lambda = lambda, max_iter=max_iter, tol=tol)
```

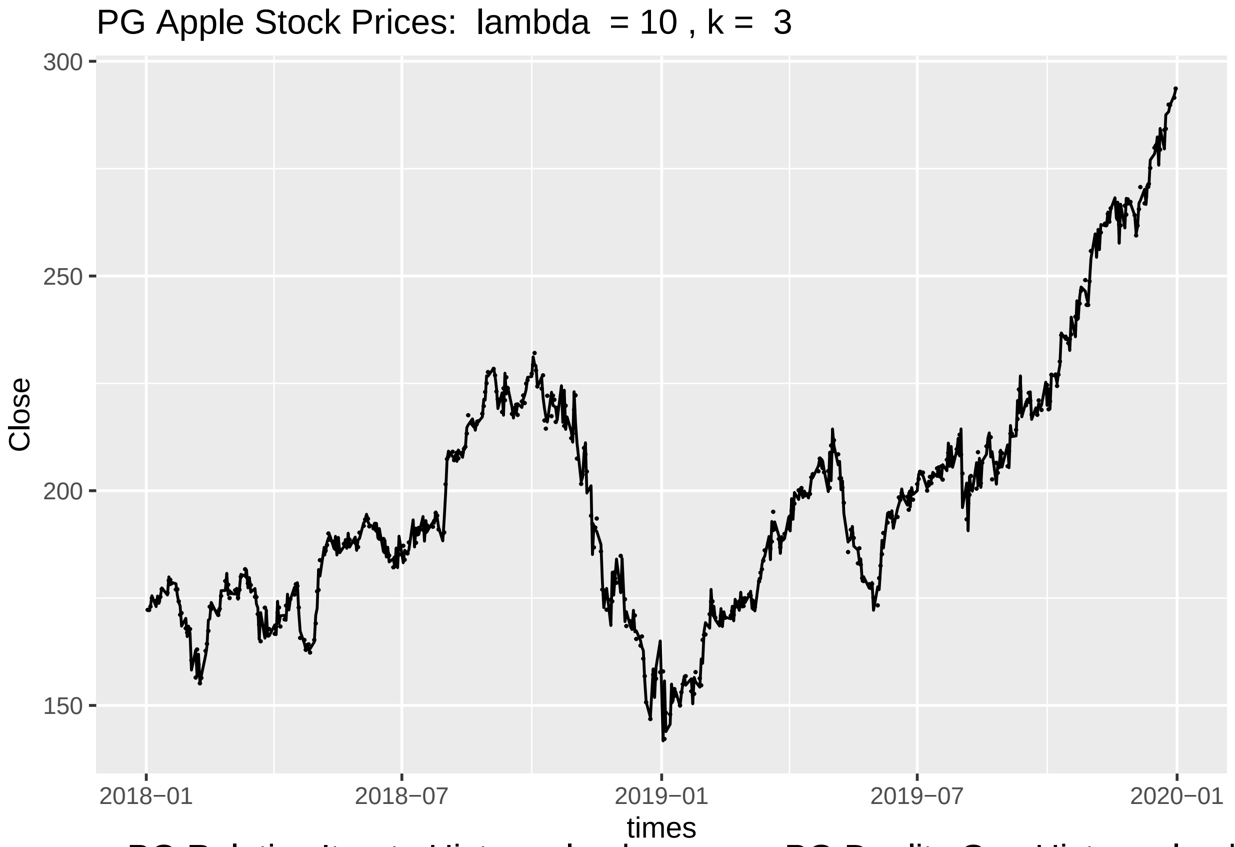

CD Apple Stock Prices: lambda = 10 , k = 3
Close
300
250
200
150
2018−01
2018−07
2019−01
2019−07
2020−01
times


CD Relative Iterate History: lambc
Relative Iterate
1.5
1.0
0.5
0.0
0
25
50
75
100
Iteration

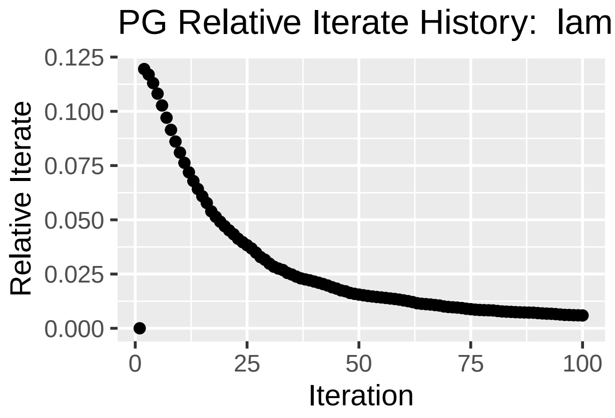

CD Duality Gap History: lambd
Duality Gap
100000
75000
50000
0
25
50
75
100
Iteration


CD Relative Objective History: l
Relative Objective
4e−04
3e−04
2e−04
1e−04
0e+00
0
25
50
75
100
Iteration

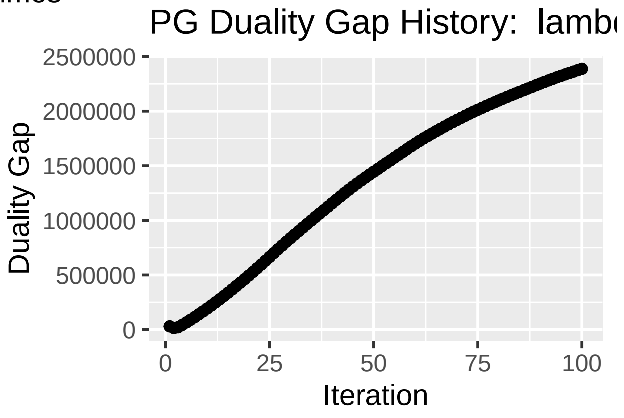

CD KKT Residual History: lambc
KKT Residual
10.00
9.75
9.50
9.25
9.00
0
25
50
75
100
Iteration



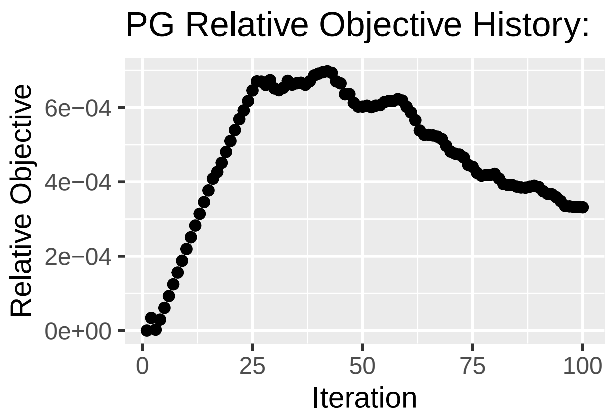
PG Apple Stock Prices: lambda = 10 , k = 3
Close
300
250
200
150
2018−01
2018−07
2019−01
2019−07
2020−01
times



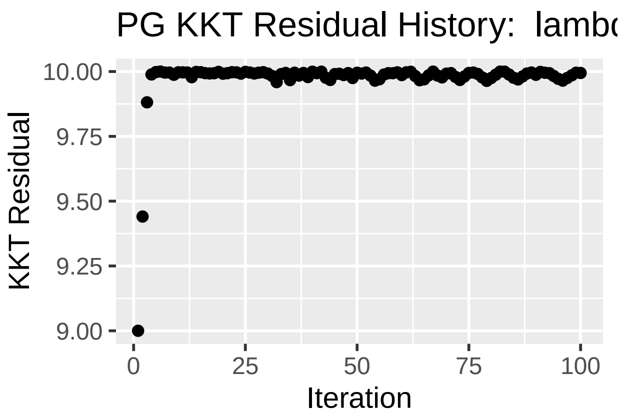
PG Relative Iterate History: laml
Relative Iterate
0.125
0.100
0.075
0.050
0.025
0.000
0
25
50
75
100
Iteration



PG Duality Gap History: lamb
Duality Gap
2500000
2000000
1500000
1000000
500000
0
0
25
50
75
100
Iteration



PG Relative Objective History: l
Relative Objective
6e−04
4e−04
2e−04
0e+00
0
25
50
75
100
Iteration



PG KKT Residual History: lamb
KKT Residual
10.00
9.75
9.50
9.25
9.00
0
25
50
75
100
Iteration