Package ‘ADMM’

May 29, 2018

Type Package
Title Algorithms using Alternating Direction Method of Multipliers
Version 0.3.0
Description Provides algorithms to solve popular optimization problems in statistics such as regression or denoising based on Alternating Direction Method of Multipliers (ADMM).
License GPL (>= 3)
Encoding UTF-8
LazyData true
Imports Rcpp, Matrix, Rdpack, stats, doParallel, foreach, parallel
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 6.0.1
RdMacros Rdpack
NeedsCompilation yes
Author Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>), Xiaozhi Zhu [aut]
Maintainer Kisung You <kyou@nd.edu>
Repository CRAN
Date/Publication 2018-05-29 14:24:14 UTC

R topics documented:

ADMM ................................................................. 2
admm.bp ............................................................. 2
admm.enet .......................................................... 4
admm.genlasso ..................................................... 5
admm.lad ............................................................ 7
admm.lasso ........................................................ 9
admm.rpca ........................................................ 11
admm.spca ......................................................... 12
admm.tv .......................................................... 14
### Description

An introduction of Alternating Direction Method of Multipliers (ADMM) method has been a breakthrough in solving complex and non-convex optimization problems in a reasonably stable as well as scalable fashion. Our package aims at providing handy tools for fast computation on well-known problems using the method. For interested users/readers, please visit Prof. Stephen Boyd’s [website](#) entirely devoted to the topic. Below is the list of functions supported,

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>admm.bp</td>
<td>Basis Pursuit</td>
</tr>
<tr>
<td>admm.enet</td>
<td>Elastic Net Regularization</td>
</tr>
<tr>
<td>admm.leglasso</td>
<td>Generalized LASSO</td>
</tr>
<tr>
<td>admm.lad</td>
<td>Least Absolute Deviations</td>
</tr>
<tr>
<td>admm.lasso</td>
<td>Least Absolute Shrinkage and Selection Operator</td>
</tr>
<tr>
<td>admm.rpcap</td>
<td>Robust Principal Component Analysis</td>
</tr>
<tr>
<td>admm.spcap</td>
<td>Sparse Principal Component Analysis</td>
</tr>
<tr>
<td>admm.tv</td>
<td>Total Variation Minimization</td>
</tr>
</tbody>
</table>

#### DESCRIPTION

For an underdetermined system, Basis Pursuit aims to find a sparse solution that solves

\[
\min_x \|x\|_1 \quad \text{s.t.} \quad Ax = b
\]

which is a relaxed version of strict non-zero support finding problem. The implementation is borrowed from Stephen Boyd’s [MATLAB code](#).

#### Usage

admm.bp(A, b, xinit = NA, rho = 1, alpha = 1, abstol = 1e-04, reltol = 0.01, maxiter = 1000)

#### Arguments

- **A**: an \((m \times n)\) regressor matrix
- **b**: a length-\(m\) response vector
- **xinit**: a length-\(n\) vector for initial value
rho  an augmented Lagrangian parameter
alpha an overrelaxation parameter in [1,2]
abstol absolute tolerance stopping criterion
reltol relative tolerance stopping criterion
maxiter maximum number of iterations

Value

a named list containing

\( x \)  a length-\( n \) solution vector

-history dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history recording following fields over iterates,

- objval  object (cost) function value
  - r_norm  norm of primal residual
  - s_norm  norm of dual residual
  - eps_pri feasibility tolerance for primal feasibility condition
  - eps_dual feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both r_norm and s_norm values become smaller than eps_pri and eps_dual, respectively.

Examples

```r
## generate sample data
n = 30;
m = 10;
A = matrix(rnorm(n*m), nrow=m);

x = matrix(rep(0,n))
x[c(3,6,21),] = rnorm(3)
b = A%x

## run example
output = admm.bp(A, b)

## report convergence plot
niter  = length(output$history$s_norm)
par(mfrow=c(1,3))
plot(1:niter, output$history$objval, "b", main="cost function")
plot(1:niter, output$history$r_norm, "b", main="primal residual")
plot(1:niter, output$history$s_norm, "b", main="dual residual")
```
Elastic Net Regularization

Description

Elastic Net regularization is a combination of $\ell_2$ stability and $\ell_1$ sparsity constraint simultaneously solving the following,

$$\min_x \frac{1}{2} \|Ax - b\|_2^2 + \lambda_1 \|x\|_1 + \lambda_2 \|x\|_2^2$$

with nonnegative constraints $\lambda_1$ and $\lambda_2$. Note that if both lambda values are 0, it reduces to least-squares solution.

Usage

```r
admm.enet(A, b, lambda1 = 1, lambda2 = 1, rho = 1, abstol = 1e-04, reltol = 0.01, maxiter = 1000)
```

Arguments

- `A` an $(m \times n)$ regressor matrix
- `b` a length-$m$ response vector
- `lambda1` a regularization parameter for $\ell_1$ term
- `lambda2` a regularization parameter for $\ell_2$ term
- `rho` an augmented Lagrangian parameter
- `abstol` absolute tolerance stopping criterion
- `reltol` relative tolerance stopping criterion
- `maxiter` maximum number of iterations

Value

a named list containing

- `x` a length-$n$ solution vector
- `history` dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history recording following fields over iterates,

- `objval` object (cost) function value
- `r_norm` norm of primal residual
- `s_norm` norm of dual residual
- `eps_pri` feasibility tolerance for primal feasibility condition
- `eps_dual` feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both `r_norm` and `s_norm` values become smaller than `eps_pri` and `eps_dual`, respectively.
Generalized LASSO

Description

Generalized LASSO is solving the following equation,

\[
\min_x \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|Dx\|_1
\]

where the choice of regularization matrix \(D\) leads to different problem formulations.
Usage

```r
admm.genlasso(A, b, D = diag(length(b)), lambda = 1, rho = 1, alpha = 1, abstol = 1e-04, reltol = 0.01, maxiter = 1000)
```

Arguments

- `A`: an `m x n` regressor matrix
- `b`: a length-
- `D`: a regularization matrix of `n` columns
- `lambda`: a regularization parameter
- `rho`: an augmented Lagrangian parameter
- `alpha`: an overrelaxation parameter in [1,2]
- `abstol`: absolute tolerance stopping criterion
- `reltol`: relative tolerance stopping criterion
- `maxiter`: maximum number of iterations

Value

A named list containing

- `x`: a length-
- `history`: dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history recording following fields over iterates,

- `objval`: object (cost) function value
- `r_norm`: norm of primal residual
- `s_norm`: norm of dual residual
- `eps_pri`: feasibility tolerance for primal feasibility condition
- `eps_dual`: feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both `r_norm` and `s_norm` values become smaller than `eps_pri` and `eps_dual`, respectively.

Author(s)

Xiaozhi Zhu
**References**


**Examples**

```r
## generate sample data
m = 100
n = 200
p = 0.1  # percentange of non-zero elements

x0 = matrix(Matrix::rsparsesparse(n,1,p))
A = matrix(rnorm(m*n),nrow=m)
for (i in 1:ncol(A)){
  A[,i] = A[,i]/sqrt(sum(A[,i]*A[,i]))
}

b = A*x0 + sqrt(0.001)*matrix(rnorm(m))
D = diag(n);

## set regularization lambda value
regval = 0.1*Matrix::norm(t(A)*%*%b, 'l')

## solve LASSO via reducing from Generalized LASSO
output = admm.genlasso(A,b,D,lambda=regval) # set D as identity matrix

## visualize
## report convergence plot
niter = length(output$history$s_norm)
par(mfrow=c(1,3))
plot(1:niter, output$history$objval, "b", main="cost function")
plot(1:niter, output$history$r_norm, "b", main="primal residual")
plot(1:niter, output$history$s_norm, "b", main="dual residual")
```

---

**Least Absolute Deviations (LAD)**

Least Absolute Deviations (LAD) is an alternative to traditional Least Squares by using cost function

\[
\min_x \|Ax - b\|_1
\]

to use \(\ell_1\) norm instead of square loss for robust estimation of coefficient.
Usage

```r
admm.lad(A, b, xinit = NA, rho = 1, alpha = 1, abstol = 1e-04,
  reltol = 0.01, maxiter = 1000)
```

Arguments

- `A` an \((m \times n)\) regressor matrix
- `b` a length-\(m\) response vector
- `xinit` a length-\(n\) vector for initial value
- `rho` an augmented Lagrangian parameter
- `alpha` an overrelaxation parameter in \([1,2]\)
- `abstol` absolute tolerance stopping criterion
- `reltol` relative tolerance stopping criterion
- `maxiter` maximum number of iterations

Value

a named list containing

- `x` a length-\(n\) solution vector
- `history` dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history recording following fields over iterates,

- `objval` object (cost) function value
- `r_norm` norm of primal residual
- `s_norm` norm of dual residual
- `eps_pri` feasibility tolerance for primal feasibility condition
- `eps_dual` feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both `r_norm` and `s_norm` values become smaller than `eps_pri` and `eps_dual`, respectively.

Examples

```r
## Not run:
## generate data
m = 1000
n = 100
A = matrix(rnorm(m*n),nrow=m)
x = 10*matrix(rnorm(n))
b = A%x

## add impulsive noise to 10% of positions
```
idx = sample(1:m, round(m/10))
b[idx] = b[idx] + 100*rnorm(length(idx))

## run the code
output = admm.lad(A,b)

## report convergence plot
niter = length(output$history$s_norm)
par(mfrow=c(1,3))
plot(1:niter, output$history$objval, "b", main="cost function")
plot(1:niter, output$history$r_norm, "b", main="primal residual")
plot(1:niter, output$history$s_norm, "b", main="dual residual")

## End(Not run)

---

**admm.lasso**

*Least Absolute Shrinkage and Selection Operator*

**Description**

LASSO, or L1-regularized regression, is an optimization problem to solve

\[
\min_x \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1
\]

for sparsifying the coefficient vector \(x\). The implementation is borrowed from Stephen Boyd’s MATLAB code.

**Usage**

\[
\text{admm.lasso}(A, b, \lambda = 1, \rho = 1, \alpha = 1, \text{abstol} = 1e-04, \text{reltol} = 0.01, \text{maxiter} = 1000)
\]

**Arguments**

- **A**: an \((m \times n)\) regressor matrix
- **b**: a length-\(m\) response vector
- **lambda**: a regularization parameter
- **rho**: an augmented Lagrangian parameter
- **alpha**: an overrelaxation parameter in \([1,2]\)
- **abstol**: absolute tolerance stopping criterion
- **reltol**: relative tolerance stopping criterion
- **maxiter**: maximum number of iterations
Value

a named list containing

x  a length-n solution vector

history  dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history
recording following fields over iterates,

objval  object (cost) function value

r_norm  norm of primal residual

s_norm  norm of dual residual

eps_pri  feasibility tolerance for primal feasibility condition

eps_dual  feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both r_norm and s_norm values become smaller
than eps_pri and eps_dual, respectively.

References

jstor.org/stable/2346178.

Examples

```r
## Not run:
## generate sample data
m = 50
n = 100
p = 0.1  # percentage of non-zero elements

x0 = matrix(Matrix::rsparsematrix(n,1,p))
A = matrix(rnorm(m*n),nrow=m)
for (i in 1:ncol(A)){
}
b = A*x0 + sqrt(0.001)*matrix(rnorm(m))

## set regularization lambda value
lambda = 0.1*Matrix::norm(t(A)*x0, 'I')

## run example
output = admm.lasso(A, b, lambda)

## report convergence plot
niter = length(output$history$s_norm)
par(mfrow=c(1,3))
```
Description

Given a data matrix $M$, it finds a decomposition

$$\min \|L\|_* + \lambda \|S\|_1 \quad \text{s.t.} \quad L + S = M$$

where $\|L\|_*$ represents a nuclear norm for a matrix $L$ and $\|S\|_1 = \sum |S_{i,j}|$, and $\lambda$ a balancing/regularization parameter. The choice of such norms leads to impose low-rank property for $L$ and sparsity on $S$.

Usage

```r
admm.rpca(M, lambda = 1/sqrt(max(nrow(M), ncol(M))), mu = 1, tol = 1e-07, maxiter = 1000)
```

Arguments

- **M**: an $(m \times n)$ data matrix
- **lambda**: a regularization parameter
- **mu**: an augmented Lagrangian parameter
- **tol**: relative tolerance stopping criterion
- **maxiter**: maximum number of iterations

Value

a named list containing

- **L**: an $(m \times n)$ low-rank matrix
- **S**: an $(m \times n)$ sparse matrix
- **history**: dataframe recording iteration numerics. See the section for more details.

Iteration History

For RPCA implementation, we chose a very simple stopping criterion

$$\|M - (L_k + S_k)\|_F \leq tol \ast \|M\|_F$$

for each iteration step $k$. So for this method, we provide a vector of only relative errors,

- **error**: relative error computed
admm.spca

References


Examples

```r
## generate data matrix from standard normal
X = matrix(rnorm(20*5),nrow=5)

## try different regularization values
out1 = admm.rpca(X, lambda=0.01)
out2 = admm.rpca(X, lambda=0.1)
out3 = admm.rpca(X, lambda=1)

## visualize sparsity
par(mfrow=c(1,3))
image(out1$S, main="lambda=0.01")
image(out2$S, main="lambda=0.1")
image(out3$S, main="lambda=1")
```

---

**Sparse PCA**

**Description**

Sparse Principal Component Analysis aims at finding a sparse vector by solving

$$\max_{x} x^T \Sigma x \quad \text{s.t.} \quad \|x\|_2 \leq 1, \quad \|x\|_0 \leq K$$

where \(\|x\|_0\) is the number of non-zero elements in a vector \(x\). A convex relaxation of this problem was proposed to solve the following problem,

$$\max_{X} < \Sigma, X > \quad \text{s.t.} \quad \text{Tr}(X) = 1, \quad \|X\|_0 \leq K^2, \quad X \geq 0, \quad \text{rank}(X) = 1$$

where \(X = xx^T\) is a \((p \times p)\) matrix that is outer product of a vector \(x\) by itself, and \(X \geq 0\) means the matrix \(X\) is positive semidefinite. With the rank condition dropped, it can be restated as

$$\max_{X} < \Sigma, X > - \rho \|X\|_1 \quad \text{s.t.} \quad \text{Tr}(X) = 1, \quad X \geq 0.$$

After acquiring each principal component vector, an iterative step based on Schur complement deflation method is applied to regress out the impact of previously-computed projection vectors. It should be noted that those sparse basis may not be orthonormal.

**Usage**

```r
admm.spca(Sigma, numpc, mu = 1, rho = 1, abstol = 1e-04, reltol = 0.01, maxiter = 1000)
```
Arguments

Sigma a \((p \times p)\) (sample) covariance matrix.
numpc number of principal components to be extracted.
mu an augmented Lagrangian parameter.
rho a regularization parameter for sparsity.
abstol absolute tolerance stopping criterion.
reltol relative tolerance stopping criterion.
maxiter maximum number of iterations.

Value

a named list containing

basis a \((p \times \text{numpc})\) matrix whose columns are sparse principal components.
history a length-numpc list of dataframes recording iteration numerics. See the section for more details.

Iteration History

For SPCA implementation, main computation is sequentially performed for each projection vector. The history field is a list of length numpc, where each element is a data frame containing iteration history recording following fields over iterates,

r_norm norm of primal residual
s_norm norm of dual residual
eps_pri feasibility tolerance for primal feasibility condition
eps_dual feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both r_norm and s_norm values become smaller than eps_pri and eps_dual, respectively.

References


Examples

```r
## generate a random matrix and compute its sample covariance
X = matrix(rnorm(1000*5),nrow=1000)
covX = cov(X)

## compute 3 sparse basis
output = admm.spca(covX, 3)
```
Total Variation Minimization

Description

1-dimensional total variation minimization - also known as signal denoising - is to solve the following
\[
\min_x \frac{1}{2} \|x - b\|_2^2 + \lambda \sum_i |x_{i+1} - x_i|
\]
for a given signal \(b\). The implementation is borrowed from Stephen Boyd’s MATLAB code.

Usage

admm.tv(b, lambda = 1, xinit = NA, rho = 1, alpha = 1, abstol = 1e-04, reltol = 0.01, maxiter = 1000)

Arguments

- \(b\): a length-\(m\) response vector
- \(\lambda\): regularization parameter
- \(xinit\): a length-\(m\) vector for initial value
- \(\rho\): an augmented Lagrangian parameter
- \(\alpha\): an overrelaxation parameter in \([1, 2]\)
- \(\text{abstol}\): absolute tolerance stopping criterion
- \(\text{reltol}\): relative tolerance stopping criterion
- \(\text{maxiter}\): maximum number of iterations

Value

A named list containing

- \(x\): a length-\(m\) solution vector
- \(\text{history}\): dataframe recording iteration numerics. See the section for more details.

Iteration History

When you run the algorithm, output returns not only the solution, but also the iteration history following fields over iterates,

- \(\text{objval}\): object (cost) function value
- \(\text{r_norm}\): norm of primal residual
- \(\text{s_norm}\): norm of dual residual
- \(\text{eps_pri}\): feasibility tolerance for primal feasibility condition
- \(\text{eps_dual}\): feasibility tolerance for dual feasibility condition

In accordance with the paper, iteration stops when both \(\text{r_norm}\) and \(\text{s_norm}\) values become smaller than \(\text{eps_pri}\) and \(\text{eps_dual}\), respectively.
Examples

```r
## generate sample data
x1 = as.vector(sin(1:100)+0.1*rnorm(100))
x2 = as.vector(cos(1:100)+0.1*rnorm(100)+5)
x3 = as.vector(sin(1:100)+0.1*rnorm(100)+2.5)
xsignal = c(x1, x2, x3)

## run example
output = admm.tv(xsignal)

## visualize
par(mfrow=c(1,2))
plot(1:300, xsignal, "1", main="original signal")
plot(1:300, output$x, "1", main="denoised signal")
```
Index

ADMM, 2
ADMM-package (ADMM), 2
admm.bp, 2, 2
admm.enet, 2, 4
admm.genlasso, 2, 5
admm.lad, 2, 7
admm.lasso, 2, 5, 9
admm.rpca, 2, 11
admm.s pca, 2, 12
admm.tv, 2, 14