

# Package ‘ADMM’

September 28, 2018

**Type** Package

**Title** Algorithms using Alternating Direction Method of Multipliers

**Version** 0.3.1

**Description** Provides algorithms to solve popular optimization problems in statistics such as regression or denoising based on Alternating Direction Method of Multipliers (ADMM).  
See Boyd et al (2010) <doi:10.1561/22000000016> for complete introduction to the method.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**Imports** Rcpp, Matrix, Rdpack, stats, doParallel, foreach, parallel

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 6.1.0

**RdMacros** Rdpack

**NeedsCompilation** yes

**Author** Kisung You [aut, cre] (<<https://orcid.org/0000-0002-8584-459X>>),  
Xiaozhi Zhu [aut]

**Maintainer** Kisung You <[kyou@end.edu](mailto:kyou@end.edu)>

**Repository** CRAN

**Date/Publication** 2018-09-28 18:00:52 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

---

ADMM	<i>ADMM : Algorithms using Alternating Direction Method of Multipliers</i>
------	--

---

### 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,

FUNCTION	Algorithm
<a href="#">admm.bp</a>	Basis Pursuit
<a href="#">admm.enet</a>	Elastic Net Regularization
<a href="#">admm.genlasso</a>	Generalized LASSO
<a href="#">admm.lad</a>	Least Absolute Deviations
<a href="#">admm.lasso</a>	Least Absolute Shrinkage and Selection Operator
<a href="#">admm.rpca</a>	Robust Principal Component Analysis
<a href="#">admm.spca</a>	Sparse Principal Component Analysis
<a href="#">admm.tv</a>	Total Variation Minimization

---

<a href="#">admm.bp</a>	<i>Basis Pursuit</i>
-------------------------	----------------------

---

### 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

```
## 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")
```

admm.enet

*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**

```
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.

**Author(s)**

Xiaozhi Zhu

**References**

Zou H and Hastie T (2005). "Regularization and variable selection via the elastic net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **67**(2), pp. 301–320. ISSN 1369-7412, 1467-9868, doi: [10.1111/j.14679868.2005.00503.x](https://doi.org/10.1111/j.14679868.2005.00503.x).

**See Also**[admm.lasso](#)**Examples**

```
## generate underdetermined design matrix
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)){
  A[,i] = A[,i]/sqrt(sum(A[,i]*A[,i]))
}
b = A%%x0 + sqrt(0.001)*matrix(rnorm(m))

## run example with both regularization values = 1
output = admm.enet(A, b, lambda1=1, lambda2=1)

## 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")
```

admm.genlasso

*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**

```
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 $(m \times n)$ regressor matrix
b	a length- $m$ response vector
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- $n$  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.

**Author(s)**

Xiaozhi Zhu

**References**

Tibshirani RJ and Taylor J (2011). “The solution path of the generalized lasso.” *The Annals of Statistics*, **39**(3), pp. 1335–1371. ISSN 0090-5364, doi: [10.1214/11AOS878](https://doi.org/10.1214/11AOS878).

Zhu Y (2017). “An Augmented ADMM Algorithm With Application to the Generalized Lasso Problem.” *Journal of Computational and Graphical Statistics*, **26**(1), pp. 195–204. ISSN 1061-8600, 1537-2715, doi: [10.1080/10618600.2015.1114491](https://doi.org/10.1080/10618600.2015.1114491).

**Examples**

```

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

x0 = matrix(Matrix::rsparsematrix(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, 'I')

## 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")

```

admm.lad

*Least Absolute Deviations***Description**

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**

```
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

```
## 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**

```

admm.lasso(A, b, lambda = 1, 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
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

Tibshirani R (1996). “Regression Shrinkage and Selection via the Lasso.” *Journal of the Royal Statistical Society. Series B (Methodological)*, **58**(1), pp. 267–288. ISSN 00359246.

### Examples

```
## 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)){
  A[,i] = A[,i]/sqrt(sum(A[,i]*A[,i]))
}
b = A%%x0 + sqrt(0.001)*matrix(rnorm(m))

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

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

## 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)
```

**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**

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

**Arguments**

<code>M</code>	an $(m \times n)$ data matrix
<code>lambda</code>	a regularization parameter
<code>mu</code>	an augmented Lagrangian parameter
<code>tol</code>	relative tolerance stopping criterion
<code>maxiter</code>	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

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

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

**error** relative error computed

**References**

Candès EJ, Li X, Ma Y and Wright J (2011). “Robust principal component analysis?” *Journal of the ACM*, **58**(3), pp. 1–37. ISSN 00045411, doi: [10.1145/1970392.1970395](https://doi.org/10.1145/1970392.1970395).

**Examples**

```
## 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$$, main="lambda=0.01")
image(out2$$, main="lambda=0.1")
image(out3$$, main="lambda=1")
```

admm.spca

*Sparse PCA***Description**

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

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

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,

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

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

$$\max_X \langle \Sigma, X \rangle - \rho \|X\|_1 \quad \text{s.t.} \quad \text{Tr}(X) = 1, X \geq 0.$$

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**

```
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.

<code>rho</code>	a regularization parameter for sparsity.
<code>abstol</code>	absolute tolerance stopping criterion.
<code>reltol</code>	relative tolerance stopping criterion.
<code>maxiter</code>	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

Ma S (2013). “Alternating Direction Method of Multipliers for Sparse Principal Component Analysis.” *Journal of the Operations Research Society of China*, **1**(2), pp. 253–274. ISSN 2194-668X, 2194-6698, doi: [10.1007/s4030501300169](https://doi.org/10.1007/s4030501300169).

### Examples

```
## 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)
```

admm.tv

*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**

<code>b</code>	a length- $m$ response vector
<code>lambda</code>	regularization parameter
<code>xinit</code>	a length- $m$ vector for initial value
<code>rho</code>	an augmented Lagrangian parameter
<code>alpha</code>	an overrelaxation parameter in $[1, 2]$
<code>abstol</code>	absolute tolerance stopping criterion
<code>reltol</code>	relative tolerance stopping criterion
<code>maxiter</code>	maximum number of iterations

**Value**

a named list containing

`x` a length- $m$  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**

```
## 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,"l",main="original signal")
plot(1:300,output$x,"l",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.spca, [2](#), [12](#)  
admm.tv, [2](#), [14](#)