

# Package ‘ADMM’

August 8, 2021

**Type** Package

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

**Version** 0.3.3

**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](https://doi.org/10.1561/22000000016)> for complete introduction to the method.

**License** GPL (>= 3)

**Encoding** UTF-8

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

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.1.1

**RdMacros** Rdpack

**NeedsCompilation** yes

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

**Maintainer** Kisung You <kisungyou@outlook.com>

**Repository** CRAN

**Date/Publication** 2021-08-08 04:20:08 UTC

## R topics documented:

ADMM	2
admm.bp	2
admm.enet	4
admm.genlasso	6
admm.lad	8
admm.lasso	10
admm.rpca	12
admm.sdp	13
admm.sPCA	15
admm.tv	17

**Index****20**


---

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.

---

admm.bp	<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) # design matrix
x = c(stats::rnorm(3), rep(0, n-3)) # coefficient
x = base::sample(x)
b = as.vector(A*%x) # response

## run example
output = admm.bp(A, b)
niter = length(output$history$s_norm)
history = output$history

## report convergence plot
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(1:niter, history$objval, "b", main="cost function")
plot(1:niter, history$r_norm, "b", main="primal residual")
plot(1:niter, history$s_norm, "b", main="dual residual")
par(opar)
```

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, Hastie T (2005). "Regularization and variable selection via the elastic net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **67**(2), 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)
niter = length(output$history$s_norm)
history = output$history

## report convergence plot
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(1:niter, history$objval, "b", main="cost function")
```

```

plot(1:niter, history$r_norm, "b", main="primal residual")
plot(1:niter, history$s_norm, "b", main="dual residual")
par(opar)

```

---

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, Taylor J (2011). “The solution path of the generalized lasso.” *The Annals of Statistics*, **39**(3), 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), 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 # 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))
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
niter = length(output$history$s_norm)
history = output$history

## report convergence plot
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(1:niter, history$objval, "b", main="cost function")
plot(1:niter, history$r_norm, "b", main="primal residual")
plot(1:niter, history$s_norm, "b", main="dual residual")
par(opar)
```

---

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**

```
## 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)
niter = length(output$history$s_norm)
history = output$history

## report convergence plot
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(1:niter, history$objval, "b", main="cost function")
plot(1:niter, history$r_norm, "b", main="primal residual")
plot(1:niter, history$s_norm, "b", main="dual residual")
par(opar)
```

---

 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), 267–288. ISSN 00359246.

### Examples

```
## 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*base::norm(t(A)%%b, "F")

## run example
output = admm.lasso(A, b, lambda)
niter = length(output$history$s_norm)
history = output$history

## report convergence plot
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(1:niter, history$objval, "b", main="cost function")
plot(1:niter, history$r_norm, "b", main="primal residual")
plot(1:niter, history$s_norm, "b", main="dual residual")
par(opar)
```

admm.rpca

*Robust Principal Component Analysis***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**

<b>M</b>	an $(m \times n)$ data matrix
<b>lambda</b>	a regularization parameter
<b>mu</b>	an augmented Lagrangian parameter
<b>tol</b>	relative tolerance stopping criterion
<b>maxiter</b>	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, Wright J (2011). “Robust principal component analysis?” *Journal of the ACM*, **58**(3), 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
opar <- par(no.readonly=TRUE)
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")
par(opar)
```

---

 admm.sdp

*Semidefinite Programming*


---

## Description

We solve the following standard semidefinite programming (SDP) problem

$$\begin{aligned} \min_X \quad & \text{tr}(CX) \\ \text{s.t.} \quad & A(X) = b, \quad X \geq 0 \end{aligned}$$

with  $A(X)_i = \text{tr}(A_i^\top X) = b_i$  for  $i = 1, \dots, m$  and  $X \geq 0$  stands for positive-definiteness of the matrix  $X$ . In the standard form, matrices  $C, A_1, A_2, \dots, A_m$  are symmetric and solution  $X$  would be symmetric and positive semidefinite. This function implements alternating direction augmented Lagrangian methods.

## Usage

```
admm.sdp(
  C,
  A,
  b,
  mu = 1,
  rho = 1,
  abstol = 1e-10,
  maxiter = 496,
  print.progress = FALSE
)
```

**Arguments**

<code>C</code>	an $(n \times n)$ symmetric matrix for cost.
<code>A</code>	a length- $m$ list of $(n \times n)$ symmetric matrices for constraint.
<code>b</code>	a length- $m$ vector for equality condition.
<code>mu</code>	penalty parameter; positive real number.
<code>rho</code>	step size for updating in $(0, \frac{1+\sqrt{5}}{2})$ .
<code>abstol</code>	absolute tolerance stopping criterion.
<code>maxiter</code>	maximum number of iterations.
<code>print.progress</code>	a logical; TRUE to show the progress, FALSE to go silent.

**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

**eps\_pri** feasibility tolerance for primal feasibility condition

**eps\_dual** feasibility tolerance for dual feasibility condition

**gap** gap between primal and dual cost function.

We use the stopping criterion which breaks the iteration when all `eps_pri`, `eps_dual`, and `gap` become smaller than `abstol`.

**Author(s)**

Kisung You

**References**

Wen Z, Goldfarb D, Yin W (2010). "Alternating direction augmented Lagrangian methods for semidefinite programming." *Mathematical Programming Computation*, **2**(3-4), 203–230. ISSN 1867-2949, 1867-2957, doi: [10.1007/s1253201000171](https://doi.org/10.1007/s1253201000171).

**Examples**

```

## a toy example
# generate parameters
C = matrix(c(1,2,3,2,9,0,3,0,7),nrow=3,byrow=TRUE)
A1 = matrix(c(1,0,1,0,3,7,1,7,5),nrow=3,byrow=TRUE)
A2 = matrix(c(0,2,8,2,6,0,8,0,4),nrow=3,byrow=TRUE)

A = list(A1, A2)
b = c(11, 19)

# run the algorithm
run = admm.sdp(C,A,b)
hst = run$history

# visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(hst$objval, type="b", cex=0.25, main="objective value")
plot(hst$eps_pri, type="b", cex=0.25, main="primal feasibility")
plot(hst$eps_dual, type="b", cex=0.25, main="dual feasibility")
plot(hst$gap, type="b", cex=0.25, main="primal-dual gap")
par(opar)

## Not run:
## comparison with CVXR's result
require(CVXR)

# problems definition
X = Variable(3,3,PSD=TRUE)
myobj = Minimize(sum_entries(C*X)) # objective
mycon = list( # constraint
  sum_entries(A[[1]]*X) == b[1],
  sum_entries(A[[2]]*X) == b[2]
)
myp = Problem(myobj, mycon) # problem

# run and visualize
res = solve(myp)
Xsol = res$getValue(X)

opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(run$X, axes=FALSE, main="ADMM result")
image(Xsol, axes=FALSE, main="CVXR result")
par(opar)

## End(Not run)

```

**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.
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

Ma S (2013). “Alternating Direction Method of Multipliers for Sparse Principal Component Analysis.” *Journal of the Operations Research Society of China*, **1**(2), 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 = stats::cov(X)

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

### 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
opar <- par(no.readonly=TRUE)
plot(1:300, xsignal, type="l", main="TV Regularization")
lines(1:300, output$x, col="red", lwd=2)
par(opar)
```

# Index

ADMM, [2](#)  
ADMM-package (ADMM), [2](#)  
admm.bp, [2](#)  
admm.enet, [4](#)  
admm.genlasso, [6](#)  
admm.lad, [8](#)  
admm.lasso, [5](#), [10](#)  
admm.rpca, [12](#)  
admm.sdp, [13](#)  
admm.spca, [15](#)  
admm.tv, [17](#)