

# Package ‘splitFeas’

October 14, 2022

**Title** Multi-Set Split Feasibility

**Version** 0.1.0

**Description** An implementation of the majorization-minimization (MM) algorithm introduced by Xu, Chi, Yang, and Lange (2017) <[arXiv:1612.05614](#)> for solving multi-set split feasibility problems. In the multi-set split feasibility problem, we seek to find a point  $x$  in the intersection of multiple closed sets and whose image under a mapping also must fall in the intersection of several closed sets.

**Depends** compiler, corpcor, matrixStats

**License** MIT + file LICENSE

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Eric C. Chi [aut, cre, cph],  
Jason Xu [ctb],  
Meng Yang [ctb],  
Kenneth Lange [ctb]

**Maintainer** Eric C. Chi <[ecchi1105@gmail.com](mailto:ecchi1105@gmail.com)>

**Repository** CRAN

**Date/Publication** 2018-04-11 08:15:34 UTC

## R topics documented:

backtrack . . . . .	2
ddg . . . . .	2
dg . . . . .	3
mmqn_step . . . . .	4
nmsfp_mm . . . . .	4
nmsfp_mmqn . . . . .	5
nmsfp_sap . . . . .	6
nmsfp_sap_one_step . . . . .	7
project_ball . . . . .	7
project_cube . . . . .	8
project_halfspace . . . . .	8

project_square . . . . .	9
proximity . . . . .	10
qnamm . . . . .	10
softmax . . . . .	11
split_feasibility . . . . .	11
wood_inv_solve . . . . .	12

<b>Index</b>	<b>13</b>
--------------	-----------

---

backtrack	<i>Backtracking Line Search</i>
-----------	---------------------------------

---

### Description

Given a descent direction backtrack computes a step size that ensures sufficient decrease in an objective.

### Usage

```
backtrack(x, dx, f, df, alpha = 0.01, beta = 0.8)
```

### Arguments

x	Current iterate
dx	Descent direction
f	objective function
df	gradient of objective function
alpha	sufficient decrease parameter
beta	sufficient decrease parameter

---

ddg	<i>Compute the approximate Hessian of the majorization.</i>
-----	---

---

### Description

ddg computes the Hessian of the majorization of the proximity function.

### Usage

```
ddg(x, v, w, hgrad)
```

**Arguments**

x	non-anchor point
v	weights for first set of constraints
w	weights for second set of constraints
hgrad	Handle for output mapping Jacobian

**Examples**

```

set.seed(12345)
n <- 10
p <- 2
x <- matrix(rnorm(p),p,1)
v <- 1
w <- 1
A <- matrix(rnorm(n*p),n,p)
hgrad <- function(x) {return(t(A))}
sol <- ddg(x,v,w,hgrad)

```

---

 dg

---

*Compute the gradient of the majorization.*


---

**Description**

dg computes the gradient of the majorization of the proximity function.

**Usage**

```
dg(x, xa, v, w, plist1, plist2, h, hgrad)
```

**Arguments**

x	non-anchor point
xa	Anchor point
v	weights for first set of constraints
w	weights for second set of constraints
plist1	list of projection functions for first set of constraints; each takes a single point and returns its projection
plist2	list of projection functions for second set of constraints; each takes a single point and returns its projection
h	Function handle for output mapping
hgrad	Handle for output mapping Jacobian

---

mmqn_step	<i>MM-quasi-Newton step</i>
-----------	-----------------------------

---

### Description

mmqn\_step computes a single step.

### Usage

```
mmqn_step(x, v, w, plist1, plist2, f, df, h, hgrad, woodbury = TRUE)
```

### Arguments

x	Current iterate
v	weights for first set of constraints
w	weights for second set of constraints
plist1	list of projection functions for first set of constraints; each takes a single point and returns its projection
plist2	list of projection functions for second set of constraints; each takes a single point and returns its projection
f	objective function
df	gradient of objective function
h	Function handle for output mapping
hgrad	Handle for output mapping Jacobian
woodbury	Boolean: TRUE to use the Woodbury inversion formula

---

nmsfp_mm	<i>MM algorithm for nonlinear multiple-sets split feasibility problem</i>
----------	---

---

### Description

nmsfp\_mm uses quasi-Newton updates to solve the nonlinear multiple-sets split feasibility problem.

### Usage

```
nmsfp_mm(x0, v, w, plist1, plist2, f, df, h, hgrad, tol = 1e-10,
max_iter = 1000)
```

**Arguments**

<code>x0</code>	Initial iterate
<code>v</code>	weights for first set of constraints
<code>w</code>	weights for second set of constraints
<code>plist1</code>	list of projection functions for first set of constraints; each takes a single point and returns its projection
<code>plist2</code>	list of projection functions for second set of constraints; each takes a single point and returns its projection
<code>f</code>	objective function
<code>df</code>	gradient of objective function
<code>h</code>	Function handle for output mapping
<code>hgrad</code>	Handle for output mapping Jacobian
<code>tol</code>	Stopping tolerance
<code>max_iter</code>	Maximum number of iterations

**See Also**

`mmqn_step`

---

<code>nmsfp_mmqn</code>	<i>MM algorithm (accelerated) for nonlinear multiple-sets split feasibility problem</i>
-------------------------	---

---

**Description**

`nmsfp_mmqn` uses quasi-Newton updates to solve the nonlinear multiple-sets split feasibility problem.

**Usage**

```
nmsfp_mmqn(x0, v, w, plist1, plist2, f, df, h, hgrad, qn = 5, tol = 1e-10,
max_iter = 1000)
```

**Arguments**

<code>x0</code>	Initial iterate
<code>v</code>	weights for first set of constraints
<code>w</code>	weights for second set of constraints
<code>plist1</code>	list of projection functions for first set of constraints; each takes a single point and returns its projection
<code>plist2</code>	list of projection functions for second set of constraints; each takes a single point and returns its projection

f	objective function
df	gradient of objective function
h	Function handle for output mapping
hgrad	Handle for output mapping Jacobian
qn	number of secants
tol	convergence tolerance
max_iter	maximum number of iterations

**See Also**

mmqn\_step, qnamm

---

nmsfp_sap	<i>Self-adaptive projection-type method algorithm for nonlinear multiple-sets split feasibility problem</i>
-----------	---

---

**Description**

nmsfp\_sap performs the self-adaptive projection-type method of Li et al.

**Usage**

```
nmsfp_sap(x0, v, w, plist1, plist2, proj, f, df, h, hgrad, delta = 0.3,
mu = 0.7, beta0 = 1, tol = 1e-10, max_iter = 1000)
```

**Arguments**

x0	Initial iterate
v	weights for first set of constraints
w	weights for second set of constraints
plist1	list of projection functions for first set of constraints; each takes a single point and returns its projection
plist2	list of projection functions for second set of constraints; each takes a single point and returns its projection
proj	handle to projection operation.
f	objective function
df	gradient of objective function
h	Function handle for output mapping
hgrad	Handle for output mapping Jacobian
delta	step-size parameter
mu	step-size parameter
beta0	initial
tol	Tolerance
max_iter	Maximum number of iterations

---

nmsfp\_sap\_one\_step      *One step of self-adaptive projection-type method for the NMSFP*

---

### Description

nmsfp\_sap\_one\_step performs the self-adaptive projection-type method of Li et al.

### Usage

nmsfp\_sap\_one\_step(x, delta, mu, tau, gamma, df, proj)

### Arguments

x	current iterate
delta	step-size parameter
mu	step-size parameter
tau	step-size parameter
gamma	step-size parameter
df	handle to gradient of objective function
proj	handle to projection operation.

### References

Li, Han, and Zhang. (2013) A self-adaptive projection-type method for nonlinear multiple-sets split feasibility problem, *Inverse Problems in Science and Engineering*, 21(1):155-170.

---

project\_ball      *Projection onto a ball*

---

### Description

project\_ball computes the Euclidean projection of a point onto a ball.

### Usage

project\_ball(x, center, r)

### Arguments

x	Point to project
center	Center of the sphere
r	Radius of the sphere

**Examples**

```
set.seed(12345)
p <- 3
center <- rnorm(p)
r <- runif(1)
x <- rnorm(p)
y <- project_ball(x, center, r)
```

---

project\_cube            *Project onto a cube*

---

**Description**

project\_cube computes the Euclidean projection of a point onto a cube.

**Usage**

```
project_cube(x, center, r)
```

**Arguments**

x	Point to project
center	Center of the square
r	Half the length of a side

**Examples**

```
set.seed(12345)
p <- 3
center <- matrix(rnorm(p),p,1)
r <- runif(1)
x <- matrix(rnorm(p),p,1)
y <- project_cube(x,center,r)
```

---

project\_halfspace        *Projection onto a halfspace*

---

**Description**

project\_halfspace computes the Euclidean projection of a point onto a closed half-space. The function returns the projection onto the set

**Usage**

```
project_halfspace(x, a, b)
```



**Arguments**

x	Point to project
a	is the normal vector
b	is the threshold

**Examples**

```
set.seed(12345)
p <- 3
a <- matrix(rnorm(p),p,1)
a <- a/norm(a,'f')
b <- runif(1)
x <- matrix(rnorm(p),p,1)
y <- project_halfspace(x,a,b)
```

---

project\_square      *Project onto a square*

---

**Description**

project\_square computes the Euclidean projection of a point onto a square.

**Usage**

```
project_square(x, center, r)
```

**Arguments**

x	Point to project
center	Center of the square
r	Half the length of a side

**Examples**

```
set.seed(12345)
p <- 2
center <- matrix(rnorm(p),p,1)
r <- runif(1)
x <- matrix(rnorm(p),p,1)
y <- project_square(x,center,r)
```

---

proximity	<i>Proximity function</i>
-----------	---------------------------

---

**Description**

proximity computes the proximity function.

**Usage**

```
proximity(x, v, w, plist1, plist2, h)
```

**Arguments**

x	Current iterate
v	weights for first set of constraints
w	weights for second set of constraints
plist1	list of projection functions for first set of constraints; each takes a single point and returns its projection
plist2	list of projection functions for second set of constraints; each takes a single point and returns its projection
h	Function handle for output mapping

---

qnamm	<i>Quasi-Newton acceleration of MM algorithm</i>
-------	--

---

**Description**

qnamm performs Quasi-Newton acceleration of an MM algorithm.

**Usage**

```
qnamm(x, fx_mm, qn, fx_obj, max_iter = 100, tol = 1e-06, ...)
```

**Arguments**

x	initial iterate
fx_mm	MM algorithm map
qn	number of secants
fx_obj	handle to objective function
max_iter	maximum number of iterations
tol	convergence tolerance
...	Additional arguments to pass to fx_mm

**References**

H Zhou, D Alexander, and K Lange. (2011) A quasi-Newton acceleration method for high-dimensional optimization algorithms, *Statistics and Computing*, 21(2):261-273.

---

softmax	<i>Compute soft-max</i>
---------	-------------------------

---

**Description**

softmax returns the soft maximum of a collection of reals.

**Usage**

```
softmax(x, a = 100)
```

**Arguments**

x	input
a	scaling factor

**Examples**

```
set.seed(12345)
n <- 10
x <- rnorm(n)
softmax(x)
```

---

split_feasibility	<i>split_feasibility</i>
-------------------	--------------------------

---

**Description**

split\_feasibility

---

wood_inv_solve	<i>Compute the inverse approximate Hessian of the majorization using the Woodbury inversion formula. wood_inv_solve computes the inverse of the Hessian term of the majorization of the proximity function using the Woodbury formula. The function mmqn_step invokes wood_inv_solve instead of ddg if the argument woodbury=TRUE. This should be used when <math>p \ll n</math>.</i>
----------------	---

---

### Description

Compute the inverse approximate Hessian of the majorization using the Woodbury inversion formula. wood\_inv\_solve computes the inverse of the Hessian term of the majorization of the proximity function using the Woodbury formula. The function mmqn\_step invokes wood\_inv\_solve instead of ddg if the argument woodbury=TRUE. This should be used when  $p \ll n$ .

### Usage

```
wood_inv_solve(x, v, w, hgrad, df)
```

### Arguments

x	non-anchor point
v	weights for first set of constraints
w	weights for second set of constraints
hgrad	Handle for output mapping Jacobian
df	Right hand side

# Index

backtrack, [2](#)

ddg, [2](#)

dg, [3](#)

mmqn\_step, [4](#)

nmsfp\_mm, [4](#)

nmsfp\_mmqn, [5](#)

nmsfp\_sap, [6](#)

nmsfp\_sap\_one\_step, [7](#)

project\_ball, [7](#)

project\_cube, [8](#)

project\_halfspace, [8](#)

project\_square, [9](#)

proximity, [10](#)

qnamm, [10](#)

softmax, [11](#)

split\_feasibility, [11](#)

split\_feasibility-package  
    (split\_feasibility), [11](#)

wood\_inv\_solve, [12](#)