Package ‘quadprog’

October 13, 2022

**Type** Package

**Title** Functions to Solve Quadratic Programming Problems

**Version** 1.5-8

**Date** 2019-11-20

**Author** S original by Berwin A. Turlach &lt;Berwin.Turlach@gmail.com&gt;  
R port by Andreas Weingessel &lt;Andreas.Weingessel@ci.tuwien.ac.at&gt;  
Fortran contributions from Cleve Moler (dpol/LINPACK and  
a modified version of) dpod/LINPACK

**Maintainer** Berwin A. Turlach &lt;Berwin.Turlach@gmail.com&gt;

**Description** This package contains routines and documentation for  
solving quadratic programming problems.

**Depends** R (>= 3.1.0)

**License** GPL (>= 2)

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2019-11-20 08:20:02 UTC

**R topics documented:**

- `solve.QP` .................................................. 1
- `solve.QP.compact` .................................... 3

**Index** 5

---

```r

solve.QP

Solve a Quadratic Programming Problem

Description

This routine implements the dual method of Goldfarb and Idnani (1982, 1983) for solving quadratic  
programming problems of the form \( \min(-d^Tb + 1/2b^TDb) \) with the constraints \( A^Tb \geq b_0 \).
```

1
solve.QP

Usage

```r
solve.QP(Dmat, dvec, Amat, bvec, meq=0, factorized=FALSE)
```

Arguments

- `Dmat` matrix appearing in the quadratic function to be minimized.
- `dvec` vector appearing in the quadratic function to be minimized.
- `Amat` matrix defining the constraints under which we want to minimize the quadratic function.
- `bvec` vector holding the values of $b_0$ (defaults to zero).
- `meq` the first `meq` constraints are treated as equality constraints, all further as inequality constraints (defaults to 0).
- `factorized` logical flag: if `TRUE`, then we are passing $R^{-1}$ (where $D = R^T R$) instead of the matrix $D$ in the argument `Dmat`.

Value

A list with the following components:

- `solution` vector containing the solution of the quadratic programming problem.
- `value` scalar, the value of the quadratic function at the solution.
- `unconstrained.solution` vector containing the unconstrained minimizer of the quadratic function.
- `iterations` vector of length 2, the first component contains the number of iterations the algorithm needed, the second indicates how often constraints became inactive after becoming active first.
- `Lagrangian` vector with the Lagrangian at the solution.
- `iact` vector with the indices of the active constraints at the solution.

References


See Also

`solve.QP.compact`
## Examples

```r
## Assume we want to minimize: -(0 5 0) %*% b + 1/2 b^T b
## under the constraints: A^T b >= b0
## with b0 = (-8,2,0)^T
## and A = (-3 1 -2)
## ( 0 0 1)
## we can use solve.QP as follows:
##
## Dmat <- matrix(0,3,3)
diag(Dmat) <- 1
dvec <- c(0,5,0)
Amat <- matrix(c(-4,-3,0,2,1,0,0,-2,1),3,3)
bvec <- c(-8,2,0)
solve.QP(Dmat,dvec,Amat,bvec=bvec)
```

---

### solve.QP.compact

**Solve a Quadratic Programming Problem**

This routine implements the dual method of Goldfarb and Idnani (1982, 1983) for solving quadratic programming problems of the form

\[
\min \left( -d^T b + \frac{1}{2} b^T D b \right)
\]

with the constraints \( A^T b \geq b_0 \).

#### Usage

```r
solve.QP.compact(Dmat, dvec, Amat, Aind, bvec, meq=0, factorized=FALSE)
```

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dmat</td>
<td>matrix appearing in the quadratic function to be minimized.</td>
</tr>
<tr>
<td>dvec</td>
<td>vector appearing in the quadratic function to be minimized.</td>
</tr>
<tr>
<td>Amat</td>
<td>matrix containing the non-zero elements of the matrix ( A ) that defines the constraints. If ( m_i ) denotes the number of non-zero elements in the ( i )-th column of ( A ) then the first ( m_i ) entries of the ( i )-th column of ( \text{Amat} ) hold these non-zero elements. <em>(If ( \text{maxmi} ) denotes the maximum of all ( m_i ), then each column of ( \text{Amat} ) may have arbitrary elements from row ( m_i + 1 ) to row ( \text{maxmi} ) in the ( i )-th column.)</em></td>
</tr>
<tr>
<td>Aind</td>
<td>matrix of integers. The first element of each column gives the number of non-zero elements in the corresponding column of the matrix ( A ). The following entries in each column contain the indexes of the rows in which these non-zero elements are.</td>
</tr>
<tr>
<td>bvec</td>
<td>vector holding the values of ( b_0 ) (defaults to zero).</td>
</tr>
<tr>
<td>meq</td>
<td>the first ( \text{meq} ) constraints are treated as equality constraints, all further as inequality constraints (defaults to 0).</td>
</tr>
<tr>
<td>factorized</td>
<td>logical flag: if ( \text{TRUE} ), then we are passing ( R^{-1} ) (where ( D = R^T R )) instead of the matrix ( D ) in the argument ( \text{Dmat} ).</td>
</tr>
</tbody>
</table>
Value

a list with the following components:

- **solution** vector containing the solution of the quadratic programming problem.
- **value** scalar, the value of the quadratic function at the solution
- **unconstrained.solution** vector containing the unconstrained minimizer of the quadratic function.
- **iterations** vector of length 2, the first component contains the number of iterations the algorithm needed, the second indicates how often constraints became inactive after becoming active first.
- **Lagrangian** vector with the Lagrangian at the solution.
- **iact** vector with the indices of the active constraints at the solution.

References


See Also

solve.QP

Examples

```r
## Assume we want to minimize: -(0 5 0) %*% b + 1/2 b^T b
## under the constraints: A^T b >= b0
## with b0 = (-8,2,0)^T
## and (-4 2 0)
## A = (-3 1 -2)
## ( 0 0 1)
## we can use solve.QP.compact as follows:
##
## Dmat <- matrix(0,3,3)
diag(Dmat) <- 1
dvec <- c(0,5,0)
Aind <- rbind(c(2,2,2),c(1,1,2),c(2,2,3))
Amat <- rbind(c(-4,2,-2),c(-3,1,1))
bvec <- c(-8,2,0)
solve.QP.compact(Dmat,dvec,Amat,Aind,bvec=bvec)
```
Index

* optimize
  solve.QP, 1
  solve.QP.compact, 3

solve.QP, 1.4
solve.QP.compact, 2.3