# Package 'umfpackr'

March 25, 2021

**Type** Package

**Title** Sparse Linear Algebra with UMFPACK

**Version** 0.8.0

**Author** Rob van Harrevelt [aut, cre]

**Maintainer** Rob van Harrevelt <rvanharrevelt@gmail.com>

**Description** Solve linear and non-linear
systems of equations using the sparse linear algebra package UMFPACK.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**LinkingTo** Rcpp

**Imports** Rcpp

**Depends** methods, Matrix

**RoxygenNote** 7.1.1

**Roxygen** list(markdown = TRUE, old_usage = TRUE)

**Suggests** testthat, nleqslv, stringr, knitr, R.rsp

**NeedsCompilation** yes

**SystemRequirements**
For Linux and other non-Windows platforms the libsuitesparse-dev library is required.

**VignetteBuilder** knitr,
R.rsp

## R topics documented:

`umf_solve` *Solves the system of linear equations using UMFPACK*

## Description

This function solved the linear equations of the form $Ax = b$ using UMFPACK.

## Usage

```
umf_solve(a, b, umf_control = list())
```

## Arguments

| | |
|---|---|
| `a` | an object of class `dgCMatrix` (see `dgCMatrix-class`) |
| `b` | the vector $b$ |
| `umf_control` | A named list with control parameters passed to UMFPACK. Currently only a single control parameter can specified: `ordering`, which specifies the ordering method. Allowed values are `"AMD"` (the default), `"CHOLMOD"`, `"METIS"` and `"BEST"`. See the Vignette *UMFPACK interface for R* for more details. For example, to use METIS ordering specify `umf_control = list(ORDERING = "METIS")`. The METIS ordering method can handle larger matrices than the standard AMD method. |

## Value

the solution $x$

## References

Dennis, J.E. Jr and Schnabel, R.B. (1997), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Siam.

Davis, T.A. (2004). A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, **30(2)**, 165–195.

Davis, T.A (2004). Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, **30(2)**, 196–199.

Davis, T.A and Duff, I.S. (1997). An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Applic.*, **18(1)**, 140–158.

Davis, T.A and Duff, I.S (1999). A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw.*, **25(1)**, 1–19..

## See Also

`umf_solve_nl`.

umf_solve_nl *Solves a system of non-linear equations using UMFPACK*

## Description

This function solves a system of non-linear equations $F(x) = 0$ using Newton's method. UMFPACK is employed to solve the linear equations in each Newton iteration. Optionally a cubic line search is used when a Newton step does not yield a sufficient reduction of the function values.

## Usage

```
umf_solve_nl(start, fn, jac, ..., control, global = c("no", "cline"),
  scaling = c("row", "col", "none"), umf_control = list())
```

## Arguments

| | |
|---|---|
| `start` | initial guess of the solution $x$ |
| `fn` | the function $F$ |
| `jac` | a function returning the Jacobian of the function as a `dgCMatrix` object |
| `...` | arguments passed to `fn` and `jac` |
| `control` | a list with control parameters. See Details. |
| `global` | The global strategy. Possible values are `"no"` (no global strategy, the default) and `"cline"` (cubic line search) (cubic line search) |
| `scaling` | Scaling method. Possible values are `"row"`. `"col"` and `"none"`. The default is `"row"`. See Details. |
| `umf_control` | A named list with control parameters passed to UMFPACK. Currently only a single control parameter can specified: `ordering`, which specifies the ordering method. Allowed values are `"AMD"` (the default), `"CHOLMOD"`, `"METIS"` and `"BEST"`. See the Vignette *UMFPACK interface for R* for more details. For example, to use METIS ordering specify `umf_control = list(ORDERING = "METIS")`. The METIS ordering method can handle larger matrices than the standard AMD method. |

## Details

**Control options:** Argument `control` is a named list containing one or more of the following components:

`ftol` The function value tolerance. Convergence is reached if the largest absolute function value is smaller than `ftol`. The default value is `1e-8`.

`xtol` The relative step size tolerance. When the relative step size of all variables is smaller than `xtol`, then the iteration process is stopped with an error. The default value is `1e-8`. The relative step size of variable $x_j$ at iteration $i$ is calculated as $|x_j^i - x_j^{i-1}|/\max(|x_j^i|, 1)$. If column scaling is applied (see argument `scaling`), then $x_i$ is actually the scaled variable.

`maxiter` The maximum number of iterations. The default is 20 if no global strategy is used (argument `global = "no"`), and 150 if cublic line searching is used (argument `global = "cline"`).

`trace` A logical. If `TRUE` then the progress of the iteration is printed. The default is `FALSE`.

`silent` A logical. If `TRUE` then all output is suppressed. The default is `FALSE`.

`cnd_tol` The tolerance for the inverse condition of the jacobian. If the inverse condition is smaller than `cnd_tol`, the solution process is terminated with an error, except if control parameter `allow_singular` is set to `TRUE`. The default is the machine precision (on most platforms about `2e-16`). If the inverse condition is very small but nonzero it may be difficult to find a solution, or the solution may not be meaningful. However, sometimes a good solution can be found even if the condition is quite small. The test for the ill-conditioning of the jacobian can be turned off by setting `cndtol` to 0 or a negative number. However, if the matrix is singular (the inverse condition is exactly zero), it is not possible to continue with the solution process (except if control parameter `allow_singular` is set to `TRUE`). The default value of `cnd_tol` is quite small, in some cases it can be appropiate to use a somewhat larger value (`1e-12`)

`cnd_method` A character vector specifying the method used to estimate the inverse condition number of the jacobian. Possible options are `"umfpack"`(the default), `"condest"` and `"kappa"`. For `"umfpack"` a rough estimate of the condition as computed by UMFPACK is used, using the expression $\min(\text{abs}(\text{diag}(\text{U})))/\max(\text{abs}(\text{diag}(\text{U})))$, where $U$ is the $U$ matrix of the LU factorization of the jacobian. Method `"condest"` employs function `condest` of the `Matrix` package and `kappa` the function `kappa` of the `base` package. Method `condtest` is more accurate than the rough estimate of UMFPACK, but takes more time. `kappa` is essentially exact, but is very slow for large matrices because this function does not use sparse matrices. Method `"condest"` usually gives a reasonable approximation of the inverse condition number. It is recommended to normally use `"umfpack"`, but occasionally use `"condest"` for a more accurate check of the condition number.

`allow_singular` A logical value (default `FALSE`) indicating if a small correction to the Jacobian is applied when it is singular or too ill-conditioned. The method used is similar to a Levenberg-Marquardt correction and is explained in Dennis and Schnabel (1996) on page 151. If `TRUE`, then the correction is applied if the inverse condition is exactly zero or if the inverse condition is smaller than control parameter `cndtol`.

**Scaling of the Jacobian:** For each iteration in the Newton method the linear system $Js = F(x)$ is solved, where the Jacobiab matrix $J$ are the derivatives of the equations with respect to the variables, and $s$ the Newton step. Scaling can improve the condition of the Jacobian. For *row scaling*, the system is transformed to $D^{-1}Js = D^{-1}F(x)$, where $D$ is a diagonal matrix with row scaling factors. Here the scaling factors are the L1 norms of the rows of $J$. For *column scaling*, the system is transformed to $JD^{-1}Ds = F(x)$, where $D$ is a diagonal matrix with column scaling factors, calculated from the L1 norms of the columns of $J$.

The scaling is only used to solve the non-linear equations and has no effect on the convergence of the Newton algorihtm. Thus the iterations are considered to be converged when the maximum value of the unscaled function values $F(x)$ is smaller than `ftol`.

## Value

a list with the following components:

| | |
|---|---|
| `solved` | A logical equal to `TRUE` if convergence of the function values has been achieved. |
| `iter` | the number of iterations |
| `x` | the final values of $x$ |
| `fval` | the function value |

`message` A string equal to `"ok"` if a solution has been found. Otherwise it describes the reason why the iteration was stopped without success

## References

Dennis, J.E. Jr and Schnabel, R.B. (1997), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Siam.

Davis, T.A. (2004). A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, **30(2)**, 165–195.

Davis, T.A (2004). Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, **30(2)**, 196–199.

Davis, T.A and Duff, I.S. (1997). An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Applic.*, **18(1)**, 140–158.

Davis, T.A and Duff, I.S (1999). A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw.*, **25(1)**, 1–19..

## See Also

`umf_solve`.

## Examples

```
library(umfpackr)

dslnex <- function(x, c) {
   y <- numeric(2)
   y[1] <- x[1]^2 + x[2]^2 - c
   y[2] <- exp(x[1]-1) + x[2]^3 - c
   y
}

jacdsln <- function(x, c) {
   n <- length(x)
   Df <- matrix(numeric(n*n),n,n)
   Df[1,1] <- 2*x[1]
   Df[1,2] <- 2*x[2]
   Df[2,1] <- exp(x[1]-1)
   Df[2,2] <- 3*x[2]^2

   return(as(Df, "dgCMatrix"))
}


xstart <- c(2,3)
print(umf_solve_nl(xstart, dslnex, jacdsln, c = 2,
                   control = list(trace = TRUE)))

# now use METIs columns ordering (run this on Linux only)
if (.Platform$OS.type != "windows") {
   print(umf_solve_nl(xstart, dslnex, jacdsln, c = 2,
```

```
                    control = list(trace = TRUE),
                    umf_control = list(ordering = "METIS")))
}
```

# Index