Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else
/*********************************************************
    *****************/
/*                                factor.c
                     */
/*********************************************************
    *****************/
/*
                     */
/* incomplete FACTORization for the type qmatrix
                     */
/*
                     */
/* Copyright (C) 1992-1995 Tomas Skalicky. All rights res
    erved.              */
/*
                     */
/*********************************************************
    *****************/
/*
                     */
/*        ANY USE OF THIS CODE CONSTITUTES ACCEPTANCE OF TH
    E TERMS             */
/*              OF THE COPYRIGHT NOTICE (SEE FILE copyrght.h
    )               */
/*
                     */
/*********************************************************
    *****************/

#include <string.h>

#include "laspack/factor.h"
#include "laspack/errhandl.h"
#include "laspack/qmatrix.h"
#include "laspack/copyrght.h"
```

```
#define PEN_FACT 1e-4

QMatrix *ILUFactor(QMatrix *Q)
/* returns matrix which contains the incomplete factorized
    matrix Q */
{
    QMatrix *QRes;

    char *QILUName;
    size_t MaxLen, Dim, RoC, RoC_, Len, Len_, ElCount, ElCo
    unt_;
    size_t LDim, i, j, k;
    size_t *IndexMapp;
    Boolean AllocOK, ElFound;
    ElType *PtrEl, *PtrEl_;
    double **L;

    Q_Lock(Q);

    if (LASResult() == LASOK) {
        if (!(*Q->ILUExists)) {
            Q_Constr(Q->ILU, "", Q->Dim, Q->Symmetry, Q->
    ElOrder, Normal, True);
            /* copy entries, detemine maximum len of rows
    or columns */
            Dim = Q->ILU->Dim;
            MaxLen = 0;
            for (RoC = 1; RoC <= Dim; RoC++) {
                Len = Q->Len[RoC];
                Q_SetLen(Q->ILU, RoC, Len);
                if (LASResult() == LASOK)
                    memcpy((void *)Q->ILU->El[RoC], (void *
    )Q->El[RoC],
                        Len * sizeof(ElType));
                if (Len > MaxLen)
                    MaxLen = Len;
            }
            *Q->ILUExists = True;

            /* sort elements, allocate diagonal elements an
    d compute thier inverse */
```

```
        Q_SortEl(Q->ILU);
        Q_AllocInvDiagEl(Q->ILU);

  if (LASResult() == LASOK && (Q->UnitRightKer || Q->
RightKerCmp != NULL)) {
      /* regularization of  the matrix by increasing
of diagonal entries */
            for (RoC = 1; RoC <= Dim; RoC++)
    (*Q->ILU->DiagEl[RoC]).Val *= 1.0 + PEN_FACT;
            /* compute inverse of modified diagonal ele
ments */
            Q_AllocInvDiagEl(Q->ILU);
  }

        if (LASResult() == LASOK && *Q->ILU->ElSorted &
& !(*Q->ILU->ZeroInDiag)) {
            /* allocate an auxiliary vector for index
mapping */
            AllocOK = True;
            IndexMapp = (size_t *)malloc((Dim + 1) * si
zeof(size_t));
            if (IndexMapp == NULL) {
                AllocOK = False;
            } else {
                /* initialization */
                for(i = 1; i <= Dim; i++)
                    IndexMapp[i] = 0;
            }
            /* allocate a dense matrix L for elements
which have influence
               on new elements arising during the
factorization */
            L = (double **)malloc((MaxLen + 1) * sizeof
(double *));
            if (L == NULL) {
                AllocOK = False;
            } else {
                for (j = 0; j <= MaxLen; j++) {
                    L[j] = (double *)malloc((MaxLen + 1
) * sizeof(double));
                    if (L[j] == NULL)
```

```
                        AllocOK = False;
                }
            }

            if (AllocOK) {
                /* incomplete factorization */
                if (LASResult() == LASOK && Q->ILU->Sy
mmetry && Q->ILU->ElOrder == Rowws) {
                    /*
                     *  incomplete Cholesky factoriza
tion
                     *      (Q->ILU ~ (D + U)^T D^(-1)
(D + U))
                     */
                    for (RoC = Dim; RoC >= 1 && LASRes
ult() == LASOK; RoC--) {
                        Len = Q->ILU->Len[RoC];
                        /* set index mapping */
                        PtrEl = Q->ILU->El[RoC] + Len -
 1;
                        LDim = 0;
                        for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos >= RoC;
                            ElCount++) {
                            IndexMapp[(*PtrEl).Pos] =
ElCount + 1;
                            PtrEl--;
                            LDim++;
                        }

                        /* initialization of L */
                        for (j = 1; j <= LDim; j++)
                            for (k = 1; k <= LDim; k++)
                                L[j][k] = 0.0;


                        /* fill matrix L with elements
which have influence
                           on the factorization */
                        PtrEl = Q->ILU->El[RoC] + Len -
 1;
```

```
                        for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos >= RoC;
                            ElCount++) {
                            /* for row or column RoC_ *
/
                            RoC_ = (*PtrEl).Pos;
                            Len_ = Q->ILU->Len[RoC_];
                            PtrEl_ = Q->ILU->El[RoC_] +
 Len_ - 1;
                            for (ElCount_ = 0; ElCount_
 < Len_ && (*PtrEl_).Pos >= RoC;
                                ElCount_++) {
                                L[ElCount + 1][IndexM
app[(*PtrEl_).Pos]] = (*PtrEl_).Val;
                                PtrEl_--;
                            }
                            PtrEl--;
                        }

                        /* factorize L */
                        for (j = 1; j < LDim; j++)
                            for (k = j + 1; k < LDim;
k++)
                                L[LDim][k] -= L[LDim][
j] * L[k][j] / L[j][j];
                        for (j = 1; j < LDim; j++)
                            L[LDim][LDim] -= L[LDim][j]
 * L[LDim][j] / L[j][j];
                        if (IsZero(L[LDim][LDim]))
                            LASError(LASZeroPivotErr, "
ILUFactor", Q_GetName(Q), NULL, NULL);

                        /* set back factorized elements
 */
                        PtrEl = Q->ILU->El[RoC] + Len -
 1;
                        for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos >= RoC;
                            ElCount++) {
                            (*PtrEl).Val = L[LDim][Ind
exMapp[(*PtrEl).Pos]];
```

```
                                PtrEl--;
                            }

                            /* reset index mapping */
                            PtrEl = Q->ILU->El[RoC] + Len -
 1;
                            for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos >= RoC;
                                ElCount++) {
                                IndexMapp[(*PtrEl).Pos] = 0
.0;
                                PtrEl--;
                            }
                        }
                    }
                    if (LASResult() == LASOK && ((Q->ILU->
Symmetry && Q->ILU->ElOrder == Clmws)
                        || (!Q->ILU->Symmetry))) {
                        /*
                         *  incomplete Cholesky factoriza
tion
                         *      (Q->ILU ~ (D + U)^T D^(-1)
(D + U))
                         *  and incomplete LU factorization
                         *      (Q->ILU ~ (D + L) D^(-1) (
D + U))
                         *  respectively
                         */
                        for (RoC = 1; RoC <= Dim && LASRes
ult() == LASOK; RoC++) {
                            Len = Q->ILU->Len[RoC];
                            /* set index mapping */
                            PtrEl = Q->ILU->El[RoC];
                            LDim = 0;
                            for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos <= RoC;
                                ElCount++) {
                                IndexMapp[(*PtrEl).Pos] =
ElCount + 1;
                                PtrEl++;
                                LDim++;
```

```
                        }

                        /* initialization of L */
                        for (j = 1; j <= LDim; j++)
                            for (k = 1; k <= LDim; k++)
                                L[j][k] = 0.0;


                        /* fill matrix L with elements
which have influence
                           on the factorization */
                        PtrEl = Q->ILU->El[RoC];
                        for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos <= RoC;
                            ElCount++) {
                            /* for row or column RoC_ *
/
                            RoC_ = (*PtrEl).Pos;
                            Len_ = Q->ILU->Len[RoC_];
                            PtrEl_ = Q->ILU->El[RoC_];
                            for (ElCount_ = 0; ElCount_
 < Len_ && (*PtrEl_).Pos <= RoC;
                                ElCount_++) {
                                L[ElCount + 1][IndexM
app[(*PtrEl_).Pos]] = (*PtrEl_).Val;
                                PtrEl_++;
                            }
                            PtrEl++;
                        }

                        /* factorize L */
                        if (Q->ILU->Symmetry) {
                            for (j = 1; j < LDim; j++)
                                for (k = j + 1; k < LD
im; k++)
                                    L[LDim][k] -= L[LD
im][j] * L[k][j] / L[j][j];
                            for (j = 1; j < LDim; j++)
                                L[LDim][LDim] -= L[j][
LDim] * L[j][LDim] / L[j][j];
                        } else {
```

```
                        for (j = 1; j < LDim; j++)
                            for (k = j + 1; k < LD
im; k++)
                                L[LDim][k] -= L[LD
im][j] * L[j][k] / L[j][j];
                        for (j = 1; j < LDim; j++)
                            for (k = j + 1; k < LD
im; k++)
                                L[k][LDim] -= L[j][
LDim] * L[k][j] / L[j][j];
                        for (j = 1; j < LDim; j++)
                            L[LDim][LDim] -= L[j][
LDim] * L[LDim][j] / L[j][j];
                    }
                    if (IsZero(L[LDim][LDim]))
                        LASError(LASZeroPivotErr, "
ILUFactor", Q_GetName(Q), NULL, NULL);

                    /* set back factorized elements
 */
                    PtrEl = Q->ILU->El[RoC];
                    for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos <= RoC;
                        ElCount++) {
                        (*PtrEl).Val = L[LDim][Ind
exMapp[(*PtrEl).Pos]];
                        PtrEl++;
                    }
                    if (!Q->ILU->Symmetry) {
                        PtrEl = Q->ILU->El[RoC];
                        for (ElCount = 0; ElCount <
 Len && (*PtrEl).Pos < RoC;
                            ElCount++) {
                            /* for row or column Ro
C_ */
                            RoC_ = (*PtrEl).Pos;
                            Len_ = Q->ILU->Len[RoC_
];
                            PtrEl_ = Q->ILU->El[Ro
C_];
                            ElFound = False;
```

```
                                for (ElCount_ = 0; ElCo
unt_ < Len_ && (*PtrEl_).Pos <= RoC;
                                    ElCount_++) {
                                    if ((*PtrEl_).Pos =
= RoC) {

                                        (*PtrEl_).Val =
 L[ElCount + 1][LDim];

                                        ElFound = True;
                                    }
                                    PtrEl_++;
                                }
                                if (!ElFound)
                                    LASError(LASILU
StructErr, "ILUFactor", Q_GetName(Q), NULL, NULL);
                                PtrEl++;
                            }
                        }

                        /* reset index mapping */
                        PtrEl = Q->ILU->El[RoC];
                        for (ElCount = 0; ElCount < Len
 && (*PtrEl).Pos <= RoC;
                            ElCount++) {
                            IndexMapp[(*PtrEl).Pos] = 0
.0;

                            PtrEl++;
                        }
                    }
                }

                /* invert diagonal elements */
                *Q->ILU->DiagElAlloc = False;
                Q_AllocInvDiagEl(Q->ILU);
            } else {
                LASError(LASMemAllocErr, "ILUFactor",
Q_GetName(Q), NULL, NULL);
            }

            if (IndexMapp != NULL)
                free(IndexMapp);
            if (L != NULL) {
```

```
                    for (j = 0; j <= MaxLen; j++) {
                  if (L[j] != NULL)
                            free(L[j]);
              }
                    free(L);
          }
            } else {
                if (LASResult() == LASOK && !(*Q->ILU->ElS
orted))
                    LASError(LASElNotSortedErr, "ILUFactor"
, Q_GetName(Q), NULL, NULL);
                if (LASResult() == LASOK && *Q->ILU->Zero
InDiag)
                    LASError(LASZeroInDiagErr, "ILUFactor",
 Q_GetName(Q), NULL, NULL);
            }
        }

        if (LASResult() == LASOK) {
      QILUName = (char *)malloc((strlen(Q_GetName(Q)) + 10
) * sizeof(char));
      if (QILUName != NULL) {
          sprintf(QILUName, "ILU(%s)", Q_GetName(Q));
Q_SetName(Q->ILU, QILUName);

                /* element ordering of matrix Q which can
be modified by Transp_Q
                   is valid for Q->ILU */
                Q->ILU->ElOrder = Q->ElOrder;

                /* check for multipliers of the matrix Q */
                if (Q->MultiplU == Q->MultiplD && Q->Multip
lL == Q->MultiplD) {
                    /* multipliers of matrix Q are valid fo
r Q->ILU too */
                    Q->ILU->MultiplD = Q->MultiplD;
                    Q->ILU->MultiplU = Q->MultiplU;
                    Q->ILU->MultiplL = Q->MultiplL;
                    QRes = Q->ILU;
                } else {
                    LASError(LASILUStructErr, "ILUFactor",
```

```
    Q_GetName(Q), NULL, NULL);
                    QRes = NULL;
                }

                free(QILUName);
      } else {
                LASError(LASMemAllocErr, "ILUFactor", Q_Get
    Name(Q), NULL, NULL);
                QRes = NULL;
      }
        } else {
            QRes = NULL;
        }
    } else {
        QRes = NULL;
    }

    Q_Unlock(Q);

    return(QRes);
}

#endif //PremiaCurrentVersion
```

References