```
   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
/************************************************************
    ******************/
/*                                   precond.c
                     */
/************************************************************
    ******************/
/*
                     */
/* PRECONDitioners for iterative solvers of systems of line
    ar equations     */
/*
                     */
/* 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 <math.h>

#include "laspack/precond.h"
#include "laspack/errhandl.h"
#include "laspack/qmatrix.h"
#include "laspack/operats.h"
#include "laspack/factor.h"
```

```
#include "laspack/itersolv.h"
#include "laspack/copyrght.h"

Vector *JacobiPrecond(QMatrix *A, Vector *y, Vector *c,
    double Omega)
/* Jacobi preconditioner */
{
    Q_Lock(A);
    V_Lock(y);
    V_Lock(c);

    /*
     *  Diag(A) / Omega y = c
     *
     *  y = Omega Diag(A)^(-1) c
     */

    Asgn_VV(y, Mul_SV(Omega, MulInv_QV(Diag_Q(A), c)));

    Q_Unlock(A);
    V_Unlock(y);
    V_Unlock(c);

    return(y);
}

Vector *SSORPrecond(QMatrix *A, Vector *y, Vector *c,
    double Omega)
/* SSOR preconditioner */
{
    Q_Lock(A);
    V_Lock(y);
    V_Lock(c);

    /*
     *  1 / (2 - Omega) * (Diag(A) / Omega + Lower(A)) * (
    Diag(A) / Omega)^(-1)
     *      * (Diag(A) / Omega + Upper(A)) y = c
     *
     *  y = (2 - Omega) / Omega * (Diag(A) / Omega + Upper(
    A))^(-1) * Diag(A)
```

```
     *          * (Diag(A) / Omega + Lower(A))^(-1) c
     */

    Asgn_VV(y, Mul_SV((2.0 - Omega) / Omega,
        MulInv_QV(Add_QQ(Mul_SQ(1.0 / Omega, Diag_Q(A)), Up
    per_Q(A)),
        Mul_QV(Diag_Q(A),
        MulInv_QV(Add_QQ(Mul_SQ(1.0 / Omega, Diag_Q(A)),
    Lower_Q(A)), c)))));

    Q_Unlock(A);
    V_Unlock(y);
    V_Unlock(c);

    return(y);
}

Vector *ILUPrecond(QMatrix *A, Vector *y, Vector *c,
    double Omega)
/* incomplete factorization preconditioner */
{
    Q_Lock(A);
    V_Lock(y);
    V_Lock(c);

    /*
     *  if Q symmetric with rowwise stored elements
     *
     *    (Diag(C) + Upper(C)) * Diag(C)^(-1) * (Diag(C) +
    Lower(C)) y = c
     *    y = (Diag(C) + Upper(C))^(-1) * Diag(C) * (Diag(
    C) + Lower(C))^(-1) c
     *
     *  otherwise
     *
     *    (Diag(B) + Lower(B)) * Diag(B)^(-1) * (Diag(B) +
    Upper(B)) y = c
     *    y = (Diag(B) + Upper(B))^(-1) * Diag(B) * (Diag(
    B) + Lower(B))^(-1) c
     *
     *  B denotes the incomplete factorized matrix A
```

```
     */

    if (Q_GetSymmetry(A) && Q_GetElOrder(A) == Rowws)
        Asgn_VV(y, MulInv_QV(Add_QQ(Diag_Q(ILUFactor(A)),
    Lower_Q(ILUFactor(A))),
            Mul_QV(Diag_Q(ILUFactor(A)),
            MulInv_QV(Add_QQ(Diag_Q(ILUFactor(A)), Upper_Q(
    ILUFactor(A))), c))));
    else
        Asgn_VV(y, MulInv_QV(Add_QQ(Diag_Q(ILUFactor(A)),
    Upper_Q(ILUFactor(A))),
            Mul_QV(Diag_Q(ILUFactor(A)),
            MulInv_QV(Add_QQ(Diag_Q(ILUFactor(A)), Lower_Q(
    ILUFactor(A))), c))));

    Q_Unlock(A);
    V_Unlock(y);
    V_Unlock(c);

    return(y);
}

#endif //PremiaCurrentVersion
```

References