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
/***********************************************************
    ******************/
/*                                  mlsolv.c
                     */
/***********************************************************
    ******************/
/*
                     */
/* Multi-Level SOLVers
                     */
/*
                     */
/* 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 <stdio.h>
#include <string.h>

#include "laspack/mlsolv.h"
#include "laspack/errhandl.h"
#include "laspack/operats.h"
```

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

Vector *MGStep(int NoLevels, QMatrix *A, Vector *x, Vector
    *b,
            Matrix *R, Matrix *P, int Level, int Gamma,
            IterProcType SmoothProc, int Nu1, int Nu2,
      PrecondProcType PrecondProc, double Omega,
            IterProcType SolvProc, int NuC,
      PrecondProcType PrecondProcC, double OmegaC)
/* one multigrid iteration */
{
    int CoarseMGIter; /* multi grid iteration counter for
    coarser grid */

    if (Level == 0) {
        /* solving of system of equations for the residual
    on the coarsest grid */
        (*SolvProc)(&A[Level], &x[Level], &b[Level], NuC,
    PrecondProcC, OmegaC);
    } else {
        /* pre-smoothing - Nu1 iterations */
        (*SmoothProc)(&A[Level], &x[Level], &b[Level], Nu1,
     PrecondProc, Omega);
        /* restiction of the residual to the coarser grid *
    /
        Asgn_VV(&b[Level - 1], Mul_MV(&R[Level - 1],
      Sub_VV(&b[Level], Mul_QV(&A[Level], &x[Level]))));
        /* initialisation of vector of unknowns on the coa
    rser grid */
        V_SetAllCmp(&x[Level - 1], 0.0);
        /* solving of system of equations for the residual
    on the coarser grid */
        for (CoarseMGIter = 1; CoarseMGIter <= Gamma; Coa
    rseMGIter++)
            MGStep(NoLevels, A, x, b, R, P, Level - 1, Gam
    ma,
       SmoothProc, Nu1, Nu2, PrecondProc, Omega,
                   SolvProc, NuC, PrecondProcC, OmegaC);
        /* interpolation of the solution from the coarser
    grid */
```

```
  if (P != NULL)
            AddAsgn_VV(&x[Level], Mul_MV(&P[Level], &x[Leve
    l - 1]));
  else
            AddAsgn_VV(&x[Level], Mul_MV(Transp_M(&R[Level
    - 1]), &x[Level - 1]));
        /* post-smoothing - Nu2 iterations */
        (*SmoothProc)(&A[Level], &x[Level], &b[Level], Nu2,
     PrecondProc, Omega);
    }

    return(&x[Level]);
}

Vector *MGIter(int NoLevels, QMatrix *A, Vector *x, Vector
    *b,
      Matrix *R, Matrix *P, int MaxIter, int Gamma,
            IterProcType SmoothProc, int Nu1, int Nu2,
      PrecondProcType PrecondProc, double Omega,
            IterProcType SolvProc, int NuC,
      PrecondProcType PrecondProcC, double OmegaC)
/* multigrid method with residual termination control */
{
    int Iter;
    double bNorm;
    size_t Dim;
    Vector r;

    Dim = Q_GetDim(&A[NoLevels - 1]);
    V_Constr(&r, "r", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(&b[NoLevels - 1]);

        Iter = 0;
        /* r = b - A x(i) at NoLevels - 1 */
        Asgn_VV(&r, Sub_VV(&b[NoLevels - 1], Mul_QV(&A[NoL
    evels - 1], &x[NoLevels - 1])));
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, MG
    IterId)
            && Iter < MaxIter) {
```

```
            Iter++;
            /* one multigrid step */
            MGStep(NoLevels, A, x, b, R, P, NoLevels - 1,
    Gamma,
       SmoothProc, Nu1, Nu2, PrecondProc, Omega,
                   SolvProc, NuC, PrecondProcC, OmegaC);
            /* r = b - A x(i) at NoLevels - 1 */
            Asgn_VV(&r, Sub_VV(&b[NoLevels - 1], Mul_QV(&A[
    NoLevels - 1], &x[NoLevels - 1])));
        }
    }

    V_Destr(&r);

    return(&x[NoLevels - 1]);
}

Vector *NestedMGIter(int NoLevels, QMatrix *A, Vector *x,
    Vector *b,
      Matrix *R, Matrix *P, int Gamma,
            IterProcType SmoothProc, int Nu1, int Nu2,
      PrecondProcType PrecondProc, double Omega,
            IterProcType SolvProc, int NuC,
      PrecondProcType PrecondProcC, double OmegaC)
/* nested multigrid method */
{
    int Level;

    /* solution of system of equations on coarsest grid */
    V_SetAllCmp(&x[0], 0.0);
    MGStep(NoLevels, A, x, b, R, P, 0, Gamma,
           SmoothProc, Nu1, Nu2, PrecondProc, Omega,
           SolvProc, NuC, PrecondProcC, OmegaC);

    for (Level = 1; Level < NoLevels; Level++) {
        /* prolongation of solution to finer grid */
        if (P != NULL)
            Asgn_VV(&x[Level], Mul_MV(&P[Level], &x[Level -
     1]));
  else
            Asgn_VV(&x[Level], Mul_MV(Transp_M(&R[Level - 1
    ]), &x[Level - 1]));
        /* solution of system of equations on finer grid
    with
           multigrid method */
        MGStep(NoLevels, A, x, b, R, P, Level, Gamma,
               SmoothProc, Nu1, Nu2, PrecondProc, Omega,
               SolvProc, NuC, PrecondProcC, OmegaC);
    }

    /* submission of reached accuracy to RTC */
    RTCResult(1, l2Norm_V(Sub_VV(&b[NoLevels - 1],
              Mul_QV(&A[NoLevels - 1], &x[NoLevels - 1]))),
              l2Norm_V(&b[NoLevels - 1]), NestedMGIterId);

    return(&x[NoLevels - 1]);
}

Vector *MGPCGIter(int NoLevels, QMatrix *A, Vector *z, Vec
    tor *r,
       Matrix *R, Matrix *P, int MaxIter, int NoMGIter,
    int Gamma,
                   IterProcType SmoothProc, int Nu1, int
    Nu2,
       PrecondProcType PrecondProc, double Omega,
                   IterProcType SolvProc, int NuC,
       PrecondProcType PrecondProcC, double OmegaC)/* mu
    ltigrid preconditioned CG method */
{
    int Iter, MGIter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector x, p, q, b;

    Dim = Q_GetDim(&A[NoLevels - 1]);
    V_Constr(&x, "x", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    V_Constr(&b, "b", Dim, Normal, True);

    if (LASResult() == LASOK) {
        /* copy solution and right hand side stored in para
    meters z and r */
        Asgn_VV(&x, &z[NoLevels - 1]);
        Asgn_VV(&b, &r[NoLevels - 1]);

        bNorm = l2Norm_V(&b);

        Iter = 0;
        Asgn_VV(&r[NoLevels - 1], Sub_VV(&b, Mul_QV(&A[NoL
    evels - 1], &x)));
        while (!RTCResult(Iter, l2Norm_V(&r[NoLevels - 1]),
     bNorm, MGPCGIterId)
            && Iter < MaxIter) {
            Iter++;
            /* multigrid preconditioner */
            V_SetAllCmp(&z[NoLevels - 1], 0.0);
            for (MGIter = 1; MGIter <= NoMGIter; MGIter++)
                MGStep(NoLevels, A, z, r, R, P, NoLevels -
    1, Gamma,
                   SmoothProc, Nu1, Nu2, PrecondProc, Omeg
    a,
                   SolvProc, NuC, PrecondProcC, OmegaC);
            Rho = Mul_VV(&r[NoLevels - 1], &z[NoLevels - 1]
    );
            if (Iter == 1) {
                Asgn_VV(&p, &z[NoLevels - 1]);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&z[NoLevels - 1], Mul_
    SV(Beta, &p)));
            }
            Asgn_VV(&q, Mul_QV(&A[NoLevels - 1], &p));
            Alpha = Rho / Mul_VV(&p, &q);
            AddAsgn_VV(&x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r[NoLevels - 1], Mul_SV(Alpha, &q))
    ;
            RhoOld = Rho;
        }

  /* put solution and right hand side vectors back */
        Asgn_VV(&z[NoLevels - 1], &x);
```

```
        Asgn_VV(&r[NoLevels - 1], &b);
    }

    V_Destr(&x);
    V_Destr(&p);
    V_Destr(&q);
    V_Destr(&b);

    return(&z[NoLevels - 1]);
}

Vector *BPXPrecond(int NoLevels, QMatrix *A, Vector *y, Vec
    tor *c,
            Matrix *R, Matrix *P, int Level,
            IterProcType SmoothProc, int Nu,
            PrecondProcType PrecondProc, double Omega,
            IterProcType SmoothProcC, int NuC,
      PrecondProcType PrecondProcC, double OmegaC)
/* BPX preconditioner (recursively defined) */
{
    if (Level == 0) {
        /* smoothing on the coarsest grid - NuC iterations
    */
        V_SetAllCmp(&y[Level], 0.0);
        (*SmoothProcC)(&A[Level], &y[Level], &c[Level], NuC
    , PrecondProcC, OmegaC);
    } else {
        /* smoothing - Nu iterations */
        V_SetAllCmp(&y[Level], 0.0);
        (*SmoothProc)(&A[Level], &y[Level], &c[Level], Nu,
    PrecondProc, Omega);
        /* restiction of the residual to the coarser grid *
    /
        Asgn_VV(&c[Level - 1], Mul_MV(&R[Level - 1], &c[
    Level]));
        /* smoothing on the coarser grid */
        BPXPrecond(NoLevels, A, y, c, R, P, Level - 1,
      SmoothProc, Nu, PrecondProc, Omega, SmoothProcC, NuC
    , PrecondProcC, OmegaC);
        /* interpolation of the solution from coarser grid
    */
```

```
  if (P != NULL)
            AddAsgn_VV(&y[Level], Mul_MV(&P[Level], &y[Leve
    l - 1]));
  else
            AddAsgn_VV(&y[Level], Mul_MV(Transp_M(&R[Level
    - 1]), &y[Level - 1]));
    }

    return(&y[Level]);
}

Vector *BPXPCGIter(int NoLevels, QMatrix *A, Vector *z, Vec
    tor *r,
       Matrix *R, Matrix *P, int MaxIter,
                   IterProcType SmoothProc, int Nu,
       PrecondProcType PrecondProc, double Omega,
                   IterProcType SmoothProcC, int NuC,
       PrecondProcType PrecondProcC, double OmegaC)
/* BPX preconditioned CG method */
{
    int Iter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector x, p, q, b;

    Dim = Q_GetDim(&A[NoLevels - 1]);
    V_Constr(&x, "x", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    V_Constr(&b, "b", Dim, Normal, True);

    if (LASResult() == LASOK) {
        /* copy solution and right hand side stored in para
    meters z and r */
        Asgn_VV(&x, &z[NoLevels - 1]);
        Asgn_VV(&b, &r[NoLevels - 1]);

        bNorm = l2Norm_V(&b);

        Iter = 0;
        Asgn_VV(&r[NoLevels - 1], Sub_VV(&b, Mul_QV(&A[NoL
    evels - 1], &x)));
        while (!RTCResult(Iter, l2Norm_V(&r[NoLevels - 1]),
     bNorm, BPXPCGIterId)
            && Iter < MaxIter) {
            Iter++;
            /* BPX preconditioner */
            BPXPrecond(NoLevels, A, z, r, R, P, NoLevels -
    1,
    SmoothProc, Nu, PrecondProc, Omega, SmoothProcC, NuC,
     PrecondProcC, OmegaC);
            Rho = Mul_VV(&r[NoLevels - 1], &z[NoLevels - 1]
    );
            if (Iter == 1) {
                Asgn_VV(&p, &z[NoLevels - 1]);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&z[NoLevels - 1], Mul_
    SV(Beta, &p)));
            }
            Asgn_VV(&q, Mul_QV(&A[NoLevels - 1], &p));
            Alpha = Rho / Mul_VV(&p, &q);
            AddAsgn_VV(&x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r[NoLevels - 1], Mul_SV(Alpha, &q))
    ;
            RhoOld = Rho;
        }

  /* put solution and right hand side vectors back */
        Asgn_VV(&z[NoLevels - 1], &x);
        Asgn_VV(&r[NoLevels - 1], &b);
    }

    V_Destr(&x);
    V_Destr(&p);
    V_Destr(&q);
    V_Destr(&b);

    return(&z[NoLevels - 1]);
}
```

```
#endif //PremiaCurrentVersion
```

References