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
/*******************************************************
    *****************/
/*                                 itersolv.c
                     */
/*******************************************************
    *****************/
/*
                     */
/* ITERative SOLVers for systems of linear 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 <stddef.h>
#include <math.h>

#include "laspack/itersolv.h"
#include "laspack/elcmp.h"
#include "laspack/errhandl.h"
#include "laspack/operats.h"
```

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

/* number of GMRES steps bevore restart */
static int GMRESSteps = 10;

Vector *JacobiIter(QMatrix *A, Vector *x, Vector *b, int
    MaxIter,
            PrecondProcType Dummy, double Omega)
{
    int Iter;
    double bNorm;
    size_t Dim;
    Vector r;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, Jacob
    iIterId)
            && Iter < MaxIter) {
            Iter++;
            /* x(i+1) = x(i) + Omega * D^(-1) * r */
            AddAsgn_VV(x, Mul_SV(Omega, MulInv_QV(Diag_Q(A)
    , &r)));
```

```
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            /* r = b - A * x(i) */
            if (Iter < MaxIter)
                Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        }
    }

    V_Destr(&r);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *SORForwIter(QMatrix *A, Vector *x, Vector *b, int
    MaxIter,
            PrecondProcType Dummy, double Omega)
{
    int Iter;
    double bNorm;
    size_t Dim;
    Vector r;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
```

```
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, SORFo
    rwIterId)
            && Iter < MaxIter) {
            Iter++;
            /* x(i+1) = x(i) + (D / Omega + L)^(-1) r */
            AddAsgn_VV(x, MulInv_QV(Add_QQ(Mul_SQ(1.0 / Om
    ega, Diag_Q(A)), Lower_Q(A)), &r));
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            /* r = b - A * x(i) */
            if (Iter < MaxIter)
                Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        }
    }

    V_Destr(&r);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *SORBackwIter(QMatrix *A, Vector *x, Vector *b, int
    MaxIter,
            PrecondProcType Dummy, double Omega)
{
    int Iter;
    double bNorm;
    size_t Dim;
    Vector r;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);
    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, SORBac
    kwIterId)
            && Iter < MaxIter) {
            Iter++;
            /* x(i+1) = x(i) + (D / Omega + U)^(-1) r */
            AddAsgn_VV(x, MulInv_QV(Add_QQ(Mul_SQ(1.0 / Om
    ega, Diag_Q(A)), Upper_Q(A)), &r));
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            /* r = b - A * x(i) */
            if (Iter < MaxIter)
                Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        }
    }

    V_Destr(&r);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *SSORIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType Dummy, double Omega)
{
    int Iter;
    double bNorm;
    size_t Dim;
    Vector r;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, SSOR
    IterId)
            && Iter < MaxIter) {
            Iter++;
            /* x(i+1) = x(i) + (2 - Omega) * (Diag(A) / Om
    ega + Upper(A))^(-1)
                   * Diag(A) / Omega * (Diag(A) / Omega +
    Lower(A))^(-1) r */
            AddAsgn_VV(x, Mul_SV((2.0 - Omega) / Omega,
                MulInv_QV(Add_QQ(Mul_SQ(1.0 / Omega, Diag_
    Q(A)), Upper_Q(A)),
                Mul_QV(Diag_Q(A),
                MulInv_QV(Add_QQ(Mul_SQ(1.0 / Omega, Diag_
    Q(A)), Lower_Q(A)), &r)))));
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            /* r = b - A * x(i) */
            if (Iter < MaxIter)
                Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        }
    }

    V_Destr(&r);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *ChebyshevIter(QMatrix *A, Vector *x, Vector *b,
    int MaxIter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     *  The original algorithm doesn't seem to work correc
    tly.
     *  The three term recursion was therefore in the cure
    nt version
     *  adopted after
     *
     *  W. Hackbusch:
     *  Iterative Solution of Large Sparse Systems of Equa
    tions,
     *  Springer-Verlag, Berlin, 1994
     *
     */

    int Iter;
    double MaxEigenval, MinEigenval;
    double c, d, Alpha, Beta;
    double T = 0.0, TOld = 0.0, TNew; /* values of Chebysh
    ev polynomials */
    double bNorm;
    size_t Dim;
    Vector r, p, z;


    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Constr(&z, "z", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        MaxEigenval = GetMaxEigenval(A, PrecondProc, Omeg
    aPrecond);
        MinEigenval = GetMinEigenval(A, PrecondProc, Omeg
    aPrecond);

        c = (MaxEigenval - MinEigenval) / 2.0;
        d = (MaxEigenval + MinEigenval) / 2.0;

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
    }
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned Chebyshev method */
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
      ChebyshevIterId)
                && Iter < MaxIter) {
                Iter++;
      if (PrecondProc != NULL)
                    (*PrecondProc)(A, &z, &r, OmegaPrecond)
      ;
      else
          Asgn_VV(&z, &r);
      if (Q_KerDefined(A))
          OrthoRightKer_VQ(&z, A);
                if (Iter == 1) {
                    TOld = 1.0;
              T = d / c;
                    Alpha = 1.0 / d;
                    Asgn_VV(&p, Mul_SV(Alpha, &z));
                } else {
          TNew = 2.0 * d / c * T - TOld;
          TOld = T;
          T = TNew;
                    Alpha = 2.0 / c * TOld / T;
                    Beta = 2.0 * d / c * TOld / T - 1.0;
                    Asgn_VV(&p, Add_VV(Mul_SV(Alpha, &z),
      Mul_SV(Beta, &p)));
                }
                AddAsgn_VV(x, &p);
                if (Iter < MaxIter)
                    Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
            }
        } else {
            /* plain Chebyshev method (z = r) */
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
      ChebyshevIterId)
                && Iter < MaxIter) {
                Iter++;
                if (Iter == 1) {
                    TOld = 1.0;
                T = d / c;
                    Alpha = 1.0 / d;
                    Asgn_VV(&p, Mul_SV(Alpha, &r));
                } else {
        TNew = 2.0 * d / c * T - TOld;
        TOld = T;
        T = TNew;
                    Alpha = 2.0 / c * TOld / T;
                    Beta = 2.0 * d / c * TOld / T - 1.0;
                    Asgn_VV(&p, Add_VV(Mul_SV(Alpha, &r),
    Mul_SV(Beta, &p)));
                }
                AddAsgn_VV(x, &p);
                if (Iter < MaxIter)
                    Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
            }
        }
        if (Q_KerDefined(A))
     OrthoRightKer_VQ(x, A);
    }

    V_Destr(&r);
    V_Destr(&p);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Destr(&z);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *CGIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    int Iter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector r, p, q, z;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Constr(&z, "z", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        if (PrecondProc != NULL || Q_KerDefined(A)) {
```

```
        /* preconditioned CG */
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
CGIterId)
            && Iter < MaxIter) {
            Iter++;
if (PrecondProc != NULL)
                (*PrecondProc)(A, &z, &r, OmegaPrecond)
;
else
    Asgn_VV(&z, &r);
if (Q_KerDefined(A))
   OrthoRightKer_VQ(&z, A);
            Rho = Mul_VV(&r, &z);
            if (Iter == 1) {
                Asgn_VV(&p, &z);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&z, Mul_SV(Beta, &p)
));
            }
            Asgn_VV(&q, Mul_QV(A, &p));
            Alpha = Rho / Mul_VV(&p, &q);
            AddAsgn_VV(x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r, Mul_SV(Alpha, &q));
            RhoOld = Rho;
        }
    } else {
        /* plain CG (z = r) */
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
CGIterId)
            && Iter < MaxIter) {
            Iter++;
            Rho = pow(l2Norm_V(&r), 2.0);
            if (Iter == 1) {
                Asgn_VV(&p, &r);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&r, Mul_SV(Beta, &p)
));
            }
            Asgn_VV(&q, Mul_QV(A, &p));
```

```
                Alpha = Rho / Mul_VV(&p, &q);
                AddAsgn_VV(x, Mul_SV(Alpha, &p));
                SubAsgn_VV(&r, Mul_SV(Alpha, &q));
                RhoOld = Rho;
            }
        }
        if (Q_KerDefined(A))
     OrthoRightKer_VQ(x, A);
    }

    V_Destr(&r);
    V_Destr(&p);
    V_Destr(&q);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Destr(&z);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *CGNIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    int Iter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector r, p, q, s, z;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    V_Constr(&z, "z", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Constr(&s, "s", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
}
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned CGN */
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
    CGNIterId)
                    && Iter < MaxIter) {
                    Iter++;
    if (PrecondProc != NULL) {
                        (*PrecondProc)(A, &z, &r, OmegaPrecond)
    ;
                        (*PrecondProc)(Transp_Q(A), &q, &z, Om
    egaPrecond);
                        Asgn_VV(&z, Mul_QV(Transp_Q(A), &q));
            } else {
        Asgn_VV(&z, &r);
    }
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&z, A);
                    Rho = pow(l2Norm_V(&z), 2.0);
                    if (Iter == 1) {
                        Asgn_VV(&p, &z);
                    } else {
                        Beta = Rho / RhoOld;
                        Asgn_VV(&p, Add_VV(&z, Mul_SV(Beta, &p)
    ));
```

```
            }
            Asgn_VV(&q, Mul_QV(A, &p));
if (PrecondProc != NULL)
                (*PrecondProc)(A, &s, &q, OmegaPrecond)
;
else
    Asgn_VV(&s, &q);
if (Q_KerDefined(A))
   OrthoRightKer_VQ(&s, A);
            Alpha = Rho / pow(l2Norm_V(&s), 2.0);
            AddAsgn_VV(x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r, Mul_SV(Alpha, &q));
            RhoOld = Rho;
        }
    } else {
        /* plain CGN */
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
CGNIterId)
            && Iter < MaxIter) {
            Iter++;
            Asgn_VV(&z, Mul_QV(Transp_Q(A), &r));
            Rho = pow(l2Norm_V(&z), 2.0);
            if (Iter == 1) {
                Asgn_VV(&p, &z);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&z, Mul_SV(Beta, &p)
));
            }
            Asgn_VV(&q, Mul_QV(A, &p));
            Alpha = Rho / pow(l2Norm_V(&q), 2.0);
            AddAsgn_VV(x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r, Mul_SV(Alpha, &q));
            RhoOld = Rho;
        }
    }
    if (Q_KerDefined(A))
 OrthoRightKer_VQ(x, A);
}

V_Destr(&r);
```

```
    V_Destr(&p);
    V_Destr(&q);
    V_Destr(&z);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Destr(&s);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *GMRESIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
                PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    char vName[10];
    int Iter, i, j, k;
    double h1, h2, r;
    double bNorm;
    double **h, *y, *s, *c1, *c2;
    size_t Dim;
    Boolean AllocOK;
    Vector *v;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    /* allocation of matrix H and vectors y, s, c1 and c2 *
    /
    AllocOK = True;
    h = (double **)malloc((GMRESSteps + 1) * sizeof(double
    *));
    if (h == NULL) {
        AllocOK = False;
    } else {
        for (i = 1; i <= GMRESSteps; i++) {
            h[i] = (double *)malloc((GMRESSteps + 2) * size
    of(double));
            if (h[i] == NULL)
                AllocOK = False;
  }
    }
    y = (double *)malloc((GMRESSteps + 1) * sizeof(double))
    ;
    if (y == NULL)
        AllocOK = False;
    s = (double *)malloc((GMRESSteps + 2) * sizeof(double))
    ;
    if (s == NULL)
        AllocOK = False;
    c1 = (double *)malloc((GMRESSteps + 1) * sizeof(double)
    );
    if (c1 == NULL)
        AllocOK = False;
    c2 = (double *)malloc((GMRESSteps + 1) * sizeof(double)
    );
    if (c2 == NULL)
        AllocOK = False;

    /* ... and vectors u */
    Dim = Q_GetDim(A);
    v = (Vector *)malloc((GMRESSteps + 2) * sizeof(Vector))
    ;
    if (v == NULL)
        AllocOK = False;
    else
        for (i = 1; i <= GMRESSteps + 1; i++) {
      sprintf(vName, "v[%d]", i % 1000);
            V_Constr(&v[i], vName, Dim, Normal, True);
  }

    if (!AllocOK)
        LASError(LASMemAllocErr, "GMRESIter", NULL, NULL,
    NULL);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        /* loop for 'MaxIter' GMRES cycles */
        Iter = 0;
        /* v[1] = r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&v[1], Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&v[1], b);
  }
        while (!RTCResult(Iter, l2Norm_V(&v[1]), bNorm, GMR
    ESIterId)
            && Iter < MaxIter) {
            if (PrecondProc != NULL)
          (*PrecondProc)(A, &v[1], &v[1], OmegaPrecond);
            s[1] = l2Norm_V(&v[1]);
            MulAsgn_VS(&v[1], 1.0 / s[1]);

            /* GMRES iteration */
            i = 0;
            while ((PrecondProc != NULL ? True : !RTCResul
    t(Iter, fabs(s[i+1]),
                bNorm, GMRESIterId)) && i < GMRESSteps &&
    Iter < MaxIter) {
           i++;
    Iter++;
                /* w = v[i+1] */
                if (PrecondProc != NULL)
        (*PrecondProc)(A, &v[i+1], Mul_QV(A, &v[i]), Omeg
    aPrecond);
                else
                    Asgn_VV(&v[i+1], Mul_QV(A, &v[i]));

                /* modified Gram-Schmidt orthogonalization
    */
                for (k = 1; k <= i; k++) {
                    h[i][k] = Mul_VV(&v[i+1], &v[k]);
                    SubAsgn_VV(&v[i+1], Mul_SV(h[i][k], &v[
    k]));
                }

                h[i][i+1] = l2Norm_V(&v[i+1]);
                MulAsgn_VS(&v[i+1], 1.0 / h[i][i+1]);

                /* Q-R algorithm */
                for (k = 1; k < i; k++) {
                    h1 = c1[k] * h[i][k] + c2[k] * h[i][k+1
    ];

                    h2 = - c2[k] * h[i][k] + c1[k] * h[i][
    k+1];
                    h[i][k] = h1;
                    h[i][k+1] = h2;
                }
                r = sqrt(h[i][i] * h[i][i] + h[i][i+1] * h[
    i][i+1]);
                c1[i] = h[i][i] / r;
                c2[i] = h[i][i+1] / r;
                h[i][i] = r;
                h[i][i+1] = 0.0;
                s[i+1] = - c2[i] * s[i];
                s[i] = c1[i] * s[i];
            }

            /* Solving of the system of equations : H y =
    s */
            for (j = i; j > 0; j--) {
                y[j] = s[j] / h[j][j];
                for (k = j - 1; k > 0; k--)
                    s[k] -= h[j][k] * y[j];
            }

            /* updating solution */
            for (j = i; j > 0; j--)
                AddAsgn_VV(x, Mul_SV(y[j], &v[j]));
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);

            /* computing new residual */
            Asgn_VV(&v[1], Sub_VV(b, Mul_QV(A, x)));
        }
    }

    /* release of vectors u, matrix H and vectors y, s, c1
    and c2 */
    if (v != NULL) {
        for (i = 1; i <= GMRESSteps + 1; i++)
            V_Destr(&v[i]);
        free(v);
    }

    if (h != NULL) {
        for (i = 1; i <= GMRESSteps; i++)
      if (h[i] != NULL)
                free(h[i]);
        free(h);
    }
    if (y != NULL)
        free(y);
    if (s != NULL)
        free(s);
    if (c1 != NULL)
        free(c1);
    if (c2 != NULL)
        free(c2);

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
```

```
}

Vector *BiCGIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    int Iter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector r, r_, p, p_, q, z, z_;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&r_, "r_", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&p_, "p_", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Constr(&z, "z", Dim, Normal, True);
        V_Constr(&z_, "z_", Dim, Normal, True);
    }
    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
}
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned BiCG */
            Asgn_VV(&r_, &r);
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm, Bi
    CGIterId)
                    && Iter < MaxIter) {
                    Iter++;
    if (PrecondProc != NULL)
                            (*PrecondProc)(A, &z, &r, OmegaPrecond)
    ;
    else
        Asgn_VV(&z, &r);
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&z, A);
    if (PrecondProc != NULL)
                            (*PrecondProc)(Transp_Q(A), &z_, &r_,
    OmegaPrecond);
    else
        Asgn_VV(&z_, &r_);
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&z_, Transp_Q(A));
                    Rho = Mul_VV(&z, &r_);
                    if (IsZero(Rho)){
                        LASError(LASBreakdownErr, "BiCGIter", "
    Rho", NULL, NULL);
                        break;
                    }
                    if (Iter == 1) {
                        Asgn_VV(&p, &z);
```

```
                Asgn_VV(&p_, &z_);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&z, Mul_SV(Beta, &p)
));
                Asgn_VV(&p_, Add_VV(&z_, Mul_SV(Beta, &
p_)));
            }
            Asgn_VV(&q, Mul_QV(A, &p));
            Alpha = Rho / Mul_VV(&p_, &q);
            AddAsgn_VV(x, Mul_SV(Alpha, &p));
            SubAsgn_VV(&r, Mul_SV(Alpha, &q));
            SubAsgn_VV(&r_, Mul_SV(Alpha, Mul_QV(Transp
_Q(A), &p_)));
            RhoOld = Rho;
        }
    } else {
        /* plain BiCG (z = r, z_ = r_) */
        Asgn_VV(&r_, &r);
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm, Bi
CGIterId)
            && Iter < MaxIter) {
            Iter++;
            Rho = Mul_VV(&r, &r_);
            if (IsZero(Rho)) {
                LASError(LASBreakdownErr, "BiCGIter", "
Rho", NULL, NULL);
                break;
            }
            if (Iter == 1) {
                Asgn_VV(&p, &r);
                Asgn_VV(&p_, &r_);
            } else {
                Beta = Rho / RhoOld;
                Asgn_VV(&p, Add_VV(&r, Mul_SV(Beta, &p)
));
                Asgn_VV(&p_, Add_VV(&r_, Mul_SV(Beta, &
p_)));
            }
            Asgn_VV(&q, Mul_QV(A, &p));
            Alpha = Rho / Mul_VV(&p_, &q);
```

```
                AddAsgn_VV(x, Mul_SV(Alpha, &p));
                SubAsgn_VV(&r, Mul_SV(Alpha, &q));
                SubAsgn_VV(&r_, Mul_SV(Alpha, Mul_QV(Transp
    _Q(A), &p_)));
                RhoOld = Rho;
            }
        }
        if (Q_KerDefined(A))
     OrthoRightKer_VQ(x, A);
    }

    V_Destr(&r);
    V_Destr(&r_);
    V_Destr(&p);
    V_Destr(&p_);
    V_Destr(&q);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Destr(&z);
        V_Destr(&z_);
    }

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *QMRIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
```

```
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    int Iter;
    double Rho, RhoNew, Xi, Gamma, GammaOld, Eta, Delta,
    Beta, Epsilon = 0.0,
        Theta, ThetaOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector r, p, p_, q, y, y_, v, v_, w, w_, z, z_, s, d;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&p_, "p_", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    V_Constr(&v, "v", Dim, Normal, True);
    V_Constr(&v_, "v_", Dim, Normal, True);
    V_Constr(&w, "w", Dim, Normal, True);
    V_Constr(&w_, "w_", Dim, Normal, True);
    V_Constr(&s, "s", Dim, Normal, True);
    V_Constr(&d, "d", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Constr(&y, "y", Dim, Normal, True);
        V_Constr(&y_, "y_", Dim, Normal, True);
        V_Constr(&z, "z", Dim, Normal, True);
        V_Constr(&z_, "z_", Dim, Normal, True);
    }

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned QMR (M1 ~ A, M2 = I) */
            Asgn_VV(&v_, &r);
      if (PrecondProc != NULL)
    (*PrecondProc)(A, &y, &v_, OmegaPrecond);
      else
    Asgn_VV(&y, &v_);
      if (Q_KerDefined(A))
          OrthoRightKer_VQ(&y, A);
            RhoNew = l2Norm_V(&y);
            Asgn_VV(&w_, &r);
            Asgn_VV(&z, &w_);
            Xi = l2Norm_V(&z);
            Gamma = 1.0;
            Eta = - 1.0;
            GammaOld = Gamma;
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
    QMRIterId)
                && Iter < MaxIter) {
                Iter++;
                Rho = RhoNew;
                if (IsZero(Rho) || IsZero(Xi)) {
                    LASError(LASBreakdownErr, "QMRIter", "
    Rho", "Xi", NULL);
                    break;
                }
                Asgn_VV(&v, Mul_SV(1.0 / Rho, &v_));
                MulAsgn_VS(&y, 1.0 / Rho);
                Asgn_VV(&w, Mul_SV(1.0 / Xi, &w_));
                MulAsgn_VS(&z, 1.0 / Xi);
                Delta = Mul_VV(&z, &y);
                if (IsZero(Delta)) {
                    LASError(LASBreakdownErr, "QMRIter", "
    Delta", NULL, NULL);
                    break;
```

```
            }
            Asgn_VV(&y_, &y);
      if (PrecondProc != NULL)
                (*PrecondProc)(Transp_Q(A), &z_, &z, Om
egaPrecond);
      else
    Asgn_VV(&z_, &z);
if (Q_KerDefined(A))
   OrthoRightKer_VQ(&z_, Transp_Q(A));
            if (Iter == 1) {
                Asgn_VV(&p, &y_);
                Asgn_VV(&q, &z_);
            } else {
                Asgn_VV(&p, Sub_VV(&y_, Mul_SV(Xi * De
lta / Epsilon, &p)));
                Asgn_VV(&q, Sub_VV(&z_, Mul_SV(Rho * De
lta / Epsilon, &q)));
            }
            Asgn_VV(&p_, Mul_QV(A, &p));
            Epsilon = Mul_VV(&q, &p_);
            if (IsZero(Epsilon)) {
                LASError(LASBreakdownErr, "QMRIter", "
Epsilon", NULL, NULL);
                break;
            }
            Beta = Epsilon / Delta;
            if (IsZero(Beta)) {
                LASError(LASBreakdownErr, "QMRIter", "
Beta", NULL, NULL);
                break;
            }
            Asgn_VV(&v_, Sub_VV(&p_, Mul_SV(Beta, &v)))
;
      if (PrecondProc != NULL)
                (*PrecondProc)(A, &y, &v_, OmegaPrecon
d);
      else
    Asgn_VV(&y, &v_);
if (Q_KerDefined(A))
   OrthoRightKer_VQ(&y, A);
            RhoNew = l2Norm_V(&y);
            Asgn_VV(&w_, Sub_VV(Mul_QV(Transp_Q(A), &q)
, Mul_SV(Beta, &w)));
            Asgn_VV(&z, &w_);
            Xi = l2Norm_V(&z);
            Theta = RhoNew / (GammaOld * fabs(Beta));
            Gamma = 1.0 / sqrt(1.0 + Theta * Theta);
            if (IsZero(Gamma)) {
                LASError(LASBreakdownErr, "QMRIter", "
Gamma", NULL, NULL);
                break;
            }
            Eta = - Eta * Rho * Gamma * Gamma / (Beta *
 GammaOld * GammaOld);
            if (Iter == 1) {
                Asgn_VV(&d, Mul_SV(Eta, &p));
                Asgn_VV(&s, Mul_SV(Eta, &p_));
            } else {
                Asgn_VV(&d, Add_VV(Mul_SV(Eta, &p),
                    Mul_SV(ThetaOld * ThetaOld * Gamma
* Gamma, &d)));
                Asgn_VV(&s, Add_VV(Mul_SV(Eta, &p_),
                    Mul_SV(ThetaOld * ThetaOld * Gamma
* Gamma, &s)));
            }
            AddAsgn_VV(x, &d);
            SubAsgn_VV(&r, &s);
            GammaOld = Gamma;
            ThetaOld = Theta;
        }
    } else {
        /* plain QMR (M1 ~ A, M2 = I, y = y_ = v_, z =
z_ = w_) */
        Asgn_VV(&v_, &r);
        RhoNew = l2Norm_V(&v_);
        Asgn_VV(&w_, &r);
        Xi = l2Norm_V(&w_);
        Gamma = 1.0;
        Eta = - 1.0;
        GammaOld = Gamma;
        while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
QMRIterId)
```

```
            && Iter < MaxIter) {
            Iter++;
            Rho = RhoNew;
            if (IsZero(Rho) || IsZero(Xi)) {
                LASError(LASBreakdownErr, "QMRIter", "
Rho", "Xi", NULL);
                break;
            }
            Asgn_VV(&v, Mul_SV(1.0 / Rho, &v_));
            MulAsgn_VS(&v_, 1.0 / Rho);
            Asgn_VV(&w, Mul_SV(1.0 / Xi, &w_));
            MulAsgn_VS(&w_, 1.0 / Xi);
            Delta = Mul_VV(&w_, &v_);
            if (IsZero(Delta)) {
                LASError(LASBreakdownErr, "QMRIter", "
Rho", "Xi", NULL);
                break;
            }
            if (Iter == 1) {
                Asgn_VV(&p, &v_);
                Asgn_VV(&q, &w_);
            } else {
                Asgn_VV(&p, Sub_VV(&v_, Mul_SV(Xi * De
lta / Epsilon, &p)));
                Asgn_VV(&q, Sub_VV(&w_, Mul_SV(Rho * De
lta / Epsilon, &q)));
            }
            Asgn_VV(&p_, Mul_QV(A, &p));
            Epsilon = Mul_VV(&q, &p_);
            if (IsZero(Epsilon)) {
                LASError(LASBreakdownErr, "QMRIter", "
Epsilon", NULL, NULL);
                break;
            }
            Beta = Epsilon / Delta;
            if (IsZero(Beta)) {
                LASError(LASBreakdownErr, "QMRIter", "
Beta", NULL, NULL);
                break;
            }
            Asgn_VV(&v_, Sub_VV(&p_, Mul_SV(Beta, &v)))
```

```
;
            RhoNew = l2Norm_V(&v);
            Asgn_VV(&w_, Sub_VV(Mul_QV(Transp_Q(A), &q)
, Mul_SV(Beta, &w)));
            Xi = l2Norm_V(&w_);
            Theta = RhoNew / (GammaOld * fabs(Beta));
            Gamma = 1.0 / sqrt(1.0 + Theta * Theta);
            if (IsZero(Gamma)) {
                LASError(LASBreakdownErr, "QMRIter", "
Gamma", NULL, NULL);
                break;
            }
            Eta = - Eta * Rho * Gamma * Gamma / (Beta *
 GammaOld * GammaOld);
            if (Iter == 1) {
                Asgn_VV(&d, Mul_SV(Eta, &p));
                Asgn_VV(&s, Mul_SV(Eta, &p_));
            } else {
                Asgn_VV(&d, Add_VV(Mul_SV(Eta, &p),
                    Mul_SV(ThetaOld * ThetaOld * Gamma
* Gamma, &d)));
                Asgn_VV(&s, Add_VV(Mul_SV(Eta, &p_),
                    Mul_SV(ThetaOld * ThetaOld * Gamma
* Gamma, &s)));
            }
            AddAsgn_VV(x, &d);
            SubAsgn_VV(&r, &s);
GammaOld = Gamma;
            ThetaOld = Theta;
        }
    }
    if (Q_KerDefined(A))
 OrthoRightKer_VQ(x, A);
}

V_Destr(&r);
V_Destr(&p);
V_Destr(&p_);
V_Destr(&q);
V_Destr(&v);
V_Destr(&v_);
```

```
    V_Destr(&w);
    V_Destr(&w_);
    V_Destr(&s);
    V_Destr(&d);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Destr(&y);
        V_Destr(&y_);
        V_Destr(&z);
        V_Destr(&z_);
    }

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *CGSIter(QMatrix *A, Vector *x, Vector *b, int Max
    Iter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    int Iter;
    double Alpha, Beta, Rho, RhoOld = 0.0;
    double bNorm;
    size_t Dim;
    Vector r, r_, p, p_, q, u, u_, v_;
    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&r_, "r_", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&q, "q", Dim, Normal, True);
    V_Constr(&u, "u", Dim, Normal, True);
    V_Constr(&u_, "u_", Dim, Normal, True);
    V_Constr(&v_, "v_", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Constr(&p_, "p_", Dim, Normal, True);

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
  }
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned CGS */
            Asgn_VV(&r_, &r);
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
    CGSIterId)
                && Iter < MaxIter) {
                Iter++;
                Rho = Mul_VV(&r_, &r);
                if (IsZero(Rho)) {
                    LASError(LASBreakdownErr, "CGSIter", "
    Rho", NULL, NULL);
                    break;
                }
                if (Iter == 1) {
                    Asgn_VV(&u, &r);
                    Asgn_VV(&p, &u);
                } else {
                    Beta = Rho / RhoOld;
                    Asgn_VV(&u, Add_VV(&r, Mul_SV(Beta, &q)
    ));
                    Asgn_VV(&p, Add_VV(&u, Mul_SV(Beta, Ad
    d_VV(&q, Mul_SV(Beta, &p)))));
                }
          if (PrecondProc != NULL)
                    (*PrecondProc)(A, &p_, &p, OmegaPrecon
    d);
          else
        Asgn_VV(&p_, &p);
    if (Q_KerDefined(A))
        OrthoRightKer_VQ(&p_, A);
                Asgn_VV(&v_, Mul_QV(A, &p_));
                Alpha = Rho / Mul_VV(&r_, &v_);
                Asgn_VV(&q, Sub_VV(&u, Mul_SV(Alpha, &v_)))
    ;
          if (PrecondProc != NULL)
                    (*PrecondProc)(A, &u_, Add_VV(&u, &q),
    OmegaPrecond);
          else
        Asgn_VV(&u_, Add_VV(&u, &q));
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&u_, A);
                AddAsgn_VV(x, Mul_SV(Alpha, &u_));
                SubAsgn_VV(&r, Mul_SV(Alpha, Mul_QV(A, &u_)
    ));
                RhoOld = Rho;
            }
        } else {
            /* plain CGS (p_ = p) */
            Asgn_VV(&r_, &r);
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm,
    CGSIterId)
                && Iter < MaxIter) {
                Iter++;
                Rho = Mul_VV(&r_, &r);
                if (IsZero(Rho)) {
                    LASError(LASBreakdownErr, "CGSIter", "
    Rho", NULL, NULL);
                    break;
                }
                if (Iter == 1) {
                    Asgn_VV(&u, &r);
                    Asgn_VV(&p, &u);
                } else {
                    Beta = Rho / RhoOld;
                    Asgn_VV(&u, Add_VV(&r, Mul_SV(Beta, &q)
    ));
                    Asgn_VV(&p, Add_VV(&u, Mul_SV(Beta, Ad
    d_VV(&q, Mul_SV(Beta, &p)))));
                }
                Asgn_VV(&v_, Mul_QV(A, &p));
                Alpha = Rho / Mul_VV(&r_, &v_);
                Asgn_VV(&q, Sub_VV(&u, Mul_SV(Alpha, &v_)))
    ;
                Asgn_VV(&u_, Add_VV(&u, &q));
                AddAsgn_VV(x, Mul_SV(Alpha, &u_));
                SubAsgn_VV(&r, Mul_SV(Alpha, Mul_QV(A, &u_)
    ));
                RhoOld = Rho;
            }
        }
        if (Q_KerDefined(A))
     OrthoRightKer_VQ(x, A);
    }

    V_Destr(&r);
    V_Destr(&r_);
    V_Destr(&p);
    V_Destr(&q);
    V_Destr(&u);
    V_Destr(&u_);
    V_Destr(&v_);
    if (PrecondProc != NULL || Q_KerDefined(A))
        V_Destr(&p_);

    Q_Unlock(A);
```

```
    V_Unlock(x);
    V_Unlock(b);

    return(x);
}

Vector *BiCGSTABIter(QMatrix *A, Vector *x, Vector *b, int
    MaxIter,
            PrecondProcType PrecondProc, double OmegaPrecon
    d)
{
    /*
     *  for details to the algorithm see
     *
     *  R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Don
    ato, J. Dongarra,
     *  V. Eijkhout, R. Pozo, Ch. Romine, H. van der Vorst:
     *  Templates for the Solution of Linear Systems: Buil
    ding Blocks
     *  for Iterative Solvers;
     *  SIAM, Philadelphia, 1994
     *
     */

    int Iter;
    double Alpha = 0.0, Beta, Rho, RhoOld = 0.0, Omega = 0.
    0;
    double bNorm;
    size_t Dim;
    Vector r, r_, p, p_, v, s, s_, t;

    Q_Lock(A);
    V_Lock(x);
    V_Lock(b);

    Dim = Q_GetDim(A);
    V_Constr(&r, "r", Dim, Normal, True);
    V_Constr(&r_, "r_", Dim, Normal, True);
    V_Constr(&p, "p", Dim, Normal, True);
    V_Constr(&v, "v", Dim, Normal, True);
    V_Constr(&s, "s", Dim, Normal, True);
    V_Constr(&t, "t", Dim, Normal, True);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Constr(&p_, "p_", Dim, Normal, True);
        V_Constr(&s_, "s_", Dim, Normal, True);
    }

    if (LASResult() == LASOK) {
        bNorm = l2Norm_V(b);

        Iter = 0;
        /* r = b - A * x(i) */
        if (!IsZero(l1Norm_V(x) / Dim)) {
            if (Q_KerDefined(A))
         OrthoRightKer_VQ(x, A);
            Asgn_VV(&r, Sub_VV(b, Mul_QV(A, x)));
        } else {
            Asgn_VV(&r, b);
}
        if (PrecondProc != NULL || Q_KerDefined(A)) {
            /* preconditioned BiCGStab */
            Asgn_VV(&r_, &r);
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm, Bi
    CGSTABIterId)
                    && Iter < MaxIter) {
                    Iter++;
                    Rho = Mul_VV(&r_, &r);
                    if (IsZero(Rho)) {
                        LASError(LASBreakdownErr, "BiCGSTABIter
    ", "Rho", NULL, NULL);
                        break;
                    }
                    if (Iter == 1) {
                        Asgn_VV(&p, &r);
                    } else {
                        Beta = (Rho / RhoOld) * (Alpha / Omega)
    ;
                        Asgn_VV(&p, Add_VV(&r, Mul_SV(Beta, Su
    b_VV(&p, Mul_SV(Omega, &v)))));
                    }
          if (PrecondProc != NULL)
                        (*PrecondProc)(A, &p_, &p, OmegaPrecon
    d);
          else
        Asgn_VV(&p_, &p);
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&p_, A);
                    Asgn_VV(&v, Mul_QV(A, &p_));
                    Alpha = Rho / Mul_VV(&r_, &v);
                    Asgn_VV(&s, Sub_VV(&r, Mul_SV(Alpha, &v)));
          if (PrecondProc != NULL)
                        (*PrecondProc)(A, &s_, &s, OmegaPrecon
    d);
          else
        Asgn_VV(&s_, &s);
    if (Q_KerDefined(A))
       OrthoRightKer_VQ(&s_, A);
                    Asgn_VV(&t, Mul_QV(A, &s_));
                    Omega = Mul_VV(&t, &s) / pow(l2Norm_V(&t),
    2.0);
                    AddAsgn_VV(x, Add_VV(Mul_SV(Alpha, &p_), Mu
    l_SV(Omega, &s_)));
                    Asgn_VV(&r, Sub_VV(&s, Mul_SV(Omega, &t)));
                    RhoOld = Rho;
                    if (IsZero(Omega))
                        break;
            }
        } else {
            /* plain BiCGStab (p_ = p) */
            Asgn_VV(&r_, &r);
            while (!RTCResult(Iter, l2Norm_V(&r), bNorm, Bi
    CGSTABIterId)
                    && Iter < MaxIter) {
                    Iter++;
                    Rho = Mul_VV(&r_, &r);
                    if (IsZero(Rho)) {
                        LASError(LASBreakdownErr, "BiCGSTABIter
    ", "Rho", NULL, NULL);
                        break;
                    }
                    if (Iter == 1) {
                        Asgn_VV(&p, &r);
                    } else {
                        Beta = (Rho / RhoOld) * (Alpha / Omega)
    ;
                        Asgn_VV(&p, Add_VV(&r, Mul_SV(Beta, Su
    b_VV(&p, Mul_SV(Omega, &v)))));
                    }
                    Asgn_VV(&v, Mul_QV(A, &p));
                    Alpha = Rho / Mul_VV(&r_, &v);
                    Asgn_VV(&s, Sub_VV(&r, Mul_SV(Alpha, &v)));
                    Asgn_VV(&t, Mul_QV(A, &s));
                    Omega = Mul_VV(&t, &s) / pow(l2Norm_V(&t),
    2.0);
                    AddAsgn_VV(x, Add_VV(Mul_SV(Alpha, &p), Mu
    l_SV(Omega, &s)));
                    Asgn_VV(&r, Sub_VV(&s, Mul_SV(Omega, &t)));
                    RhoOld = Rho;
                    if (IsZero(Omega))
                        break;
            }
        }
        if (Q_KerDefined(A))
     OrthoRightKer_VQ(x, A);
    }

    V_Destr(&r);
    V_Destr(&r_);
    V_Destr(&p);
    V_Destr(&v);
    V_Destr(&s);
    V_Destr(&t);
    if (PrecondProc != NULL || Q_KerDefined(A)) {
        V_Destr(&p_);
        V_Destr(&s_);
    }

    Q_Unlock(A);
    V_Unlock(x);
    V_Unlock(b);

    return(x);
    }
```

```
void SetGMRESRestart(int Steps)
{
    GMRESSteps = Steps;
}

#endif //PremiaCurrentVersion
```

References