ПРИЛОЖЕНИЕ

Коды программ на C++

Ниже приведены листинги проверенных и работающих файлов, содержащих функции для решения СЛАУ прямыми и итерационными методами. Все функции, позволяющие решать СЛАУ прямыми методами, имеют два аргумента: первым является матрица СЛАУ, а вторым – вектор свободных членов.

Функция, позволяющая решать СЛАУ методом GMRES, имеет восемь аргументов. Первым аргументом является матрица СЛАУ, вторым – вектор свободных членов, а третьим – произвольное начальное условие. Итерационный процесс продолжается до тех пор, пока не будет получено решение с точностью, определяемой параметром *Tol* (четвертый параметр) или при достижении максимального числа итераций N_{it}^{max} (пятый параметр). Также необходимо, чтобы пользователь задал размерность *m* (шестой параметр). Седьмой параметр определяет, с каким типом предобусловливания использовать метод. Вспомогательный параметр для определения структуры разреженности матрицы предобусловливания (порог обнуления, допуск обнуления или число максимальных элементов в столбце) задается восьмым аргументом.

Первые пять параметров функций метода BiCG, BiCGStab, CG, CGS, QMR совпадают с аналогичными параметрами метода GMRES. Шестым параметром является тип предобусловливания. Седьмым задается вспомогательный параметр для определения структуры разреженности матрицы предобусловливания (порог обнуления, допуск обнуления или число максимальных элементов в столбце).

```
/***********************************************************
                          Math_Func.h
Description: вспомогательные функции.
***********************************************************/
// (c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __Math_Func_H__
#define __Math_Func_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
namespace TALGAT {
  template<class Type>
  class Math_Func{
    public:
      inline static void Mat_Vec(const mtl::matrix< Type >::type
      &A, const mtl::dense1D< Type > &B, mtl::dense1D< Type > &D){
        int_type i,j, p = B.size();
        for(i = 0; i < p; i++)
          for(D[i] = (0.0,0.0), j = 0; j < p; j++)
            D[i] += A(i,j) * B[j];
      }
      inline static void transMat_Vec(const mtl::matrix< Type
      >::type &A, const mtl::dense1D< Type > &B, mtl::dense1D<
      Type > &D){
        int_type i,j,p = B.size();
        for(i = 0; i < p; i++)
```

```
    for(D[i] = (0.0,0.0), j = 0; j < p; j++)
      D[i] += A(j,i) * B[j];
}
inline static void Initial_Guess(mtl::dense1D< Type > &B,
const fp_type x){
  for(int_type i = 0; i < B.size(); i++)
    B[i] = x;
}
inline static Type Vec_Vec(const mtl::dense1D< Type >
&A,const mtl::dense1D< Type > &B){
  int_type i; Type n = (0.0,0.0);
  for(i = 0; i < B.size(); i++)
    n += A[i] * B[i];
  return(n);
}
inline static fp_type norm(const mtl::dense1D< Type > &B){
  int_type i; fp_type n = 0.0;
  for(i = 0; i < B.size(); i++)
    n += abs(B[i]) * abs(B[i]);
  return(sqrt(n));
}
inline static void VecPlusScalarVec(const mtl::dense1D< Type
> &A, const Type alfa, const mtl::dense1D< Type > &B,
mtl::dense1D< Type > &D){
  for(int_type i = 0; i < D.size(); i++)
    D[i] = A[i] + alfa * B[i];
}
static void GivensRot_solve(const Type &a, const Type
&b,Type &sc, Type &cs){
  Type temp;
  temp = a / b;
  cs = 1.0 / sqrt(1.0 + temp * temp);
  sc = temp * cs;
}
inline static void GivensRot_apply( Type &a, Type &b, const
Type &cs,  const Type &sn){
  Type temp;
  temp = cs * a + sn * b;
  b =  - sn * a + cs * b ;
  a = temp;
}
inline static void VecDivScalar(const mtl::dense1D< Type >
&A,const Type alfa,
  mtl::dense1D< Type > &B){
    for(int_type i = 0; i < A.size(); i++)
      B[i] = A[i] / alfa;
}
static void OrthogonalArnoldi(const mtl::matrix< Type
>::type &A, mtl::matrix< Type >::type &H, mtl::matrix< Type
>::type &V, int_type &m_){
  int_type i,j,s,c;
```

```
        V(0,0) = (1.0,0.0);
        for(i = 1; i < A.nrows(); i++)
          V(i,0) = (0.0,0.0);
        mtl::dense1D< Type > W(A.nrows());
        for(j = 0; j < m_; j++){
          for(c = 0; c < A.nrows();c++)
            for(s = 0, W[c] = (0.0,0.0); s < A.nrows(); s++)
              W[c] += A(c,s) * V(s,j);
          for(i = 0; i <= j; i++){
            for(c = 0, H(i,j) = (0.0,0.0); c < A.nrows(); c++)
              H(i,j) += W[c] * V(c,i);
            for(c = 0; c < A.nrows(); c++)
              W[c] -= H(i,j) * V(c,i);
          }
          H(j+1,j) = norm(W);
          if(abs(H(j+1,j)) <= 1.e-15) {
            m_= j+1; H(j+1,j) = (0.0,0.0); return;}
          for(c = 0; c < A.nrows(); c++)
            V(c,j+1) = W[c] / H(j+1,j);
        }
      }
      static void Symmetrisation(const mtl::matrix< Type >::type
      &A, mtl::matrix< Type >::type &M){
        for(int_type i = 0; i < A.nrows(); i ++)
          for(int_type j = i; j < A.nrows(); j ++){
            if(i != j)
              M(i,j) = M(j,i) = (A(i,j) + A(j,i)) / 2.0;
            else
              M(i,j) = A(i,j);
          }
      }
  };
}
#endif // __Math_Func_H__
/*************************************************************
                        solvers_iterative.h
Description: итерационные методы решения СЛАУ.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __solvers_iterative_H__
#define __solvers_iterative_H__
// #define MATRIX_EXPORTS
// #define _DEBUG
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "DropRule.h"
```

```
namespace TALGAT {
enum PreconMethod{None_, LUnr_, LUnc_, LUnm_, LUdm_, LUdmr_,
LUkm_, LUld_, LUinr_, LUsunr_,LUslnr_, LUPnr_, LUPnc_, LUPnm_,
ILU0nr_, ILU0nc_, ILU0nm_,  ILU0dm_, ILU0km_, ILU0ld_,ILU0inr_,
ILU0dmr_, DILU0nm_, ILUpnr_, ILUpnc_, ILUpnm_, ILUTnr_, ILUTnc_,
ILUTnm_, LUCnr_, LUddm_, LUind_,LUdmd_, LUnd_};
template<class Type>
  class solvers_iterative{
    public:
//CG-symmetric (SPD)
      static void CG(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max, int_type type_pre, fp_type
    droptol);
//ABR1ORT-unsymmetric
      static void ABR1ORT(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_,mtl::dense1D< Type > &U_);
//BiCG-unsymmetric
      static void BiCG(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_,  mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max, int_type type_pre, fp_type
    droptol, int_type bl_num);
//GMRES_m-unsymmetric
      static void GMRES_m(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_,mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max, int_type m, int_type
    type_pre, fp_type droptol,int_type bl_num);
//QMR-unsymmetric
      static void QMR(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max,int_type type_pre, fp_type
    droptol, int_type bl_num);
//CGS-unsymmetric
      static void CGS(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max, int_type type_pre, fp_type
    droptol, int_type bl_num);
//BiCGStab-unsymmetric
      static void BiCGStab(mtl::matrix< Type >::type& A_,const
    mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_, fp_type x,
    fp_type tol, int_type niter_max, int_type type_pre, fp_type
    droptol, int_type bl_num);
  };//class solvers_iterative
}//namespace TALGAT
#endif //__solvers_iterative_H__

/************************************************************
                              DropRule.h
Description: способы предфильтрации.
************************************************************/
// (c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
```

```
#ifndef __DropRule_H__
#define __DropRule_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
namespace TALGAT {enum TypeDropRule{nr_, nc_, nm_, dm_, km_, ld_,
sunr_, inr_, dmr_};
template<class Type>
  class DropRule{
    public:
//km
    static int_type kMaxElem_in_column_(mtl::matrix< Type >::type
    &A, mtl::matrix< Type >::type &M, const int_type k){
      fp_type a_max,temp;
      int_type nnzM = 0;
      int_type i,j,n_,ind_,num_,p = A.nrows();
      mtl::dense1D< int_type > ind_max(k);
        for(j = 0; j < p; j ++){
          ind_max[0] = j;
          M(j,j) = A(j,j);
          a_max = abs(A(0,j));
          ind_ = 0;
          for(num_ = 1; num_ < k; num_ ++){
            for(i = 0; i < p; i ++){
              temp = abs(A(i,j));
              if(temp > a_max)
                if(M(i,j) == (0.0,0.0)){
                  a_max = temp;
                  ind_ = i;
                }
            }
            M(ind_,j) = A(ind_,j);
            if(num_ != k - 1){
              ind_max[num_] = ind_;
              ind_ = 0;
          here:
              for(n_ = 0; n_ <= num_; n_ ++){
                if(ind_max[n_] == ind_){
                  ind_ += 1;
                  goto here;
                }
              }
              a_max = abs(A(ind_,j));
            }
          }
        }
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++)
```

```
          if(M(i,j) != (0.0,0.0))
            nnzM += 1;
      return nnzM;
    }
//dm
    static int_type Drop_if_ElemDivMaxElem_Less_(mtl::matrix< Type
    >::type &A ,mtl::matrix< Type >::type &M, const fp_type eps){
      int_type nnzM = 0;
      fp_type a_max,temp;
      int_type i,j,p = A.nrows();
      a_max = abs(A(0,0));
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            temp = abs(A(i,j));
            if( temp > a_max)
              a_max = temp;
          }
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            temp = abs(A(i,j)) / a_max;
            if(temp >= eps || i == j){
              M(i,j) = A(i,j);
              nnzM += 1;
            }
          }
      return nnzM;
    }
//dmr
    static int_type Drop_if_ElemDivMaxElemInRow_Less_(mtl::matrix<
    Type >::type &A ,mtl::matrix< Type >::type &M, const fp_type
    droptol){
      int_type nnzM = 0;
      fp_type a_max,temp;
      int_type i,j,p = A.nrows();
        for(i = 0; i < p; i ++){
          a_max = abs(A(i,0));
          for(j = 1; j < p; j ++){
            temp = abs(A(i,j));
            if( temp > a_max)
              a_max = temp;
          }
          a_max = a_max * droptol;
          for(j = 0; j < p; j ++){
            if( i == j || abs(A(i,j)) > a_max ){
              M(i,j) = A(i,j);
              nnzM += 1;
            }
          }
        }
        return nnzM;
    }
```

```
//ld
    static int_type Drop_if_Elem_Less_(mtl::matrix< Type >::type
    &A, mtl::matrix< Type >::type &M, const fp_type eps){
      fp_type temp;
      int_type nnzM = 0;
      int_type i,j,p = A.nrows();
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            temp = abs(A(i,j));
            if(temp >= eps || i == j){
              M(i,j) = A(i,j);
              nnzM += 1;
            }
          }
        return nnzM;
    }
//nr
    static int_type BAND_BlDiag_NormRow_(mtl::matrix< Type >::type
    &A, mtl::matrix< Type >::type &M, int_type t, int_type d,const
    fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      fp_type A_norm;
      A_norm = 0.0;
      int_type nnzM = 0;
        for(nrow = t, y = t; nrow < p; nrow ++){
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++)
            A_norm += abs(A(nrow,ncol)) * abs(A(nrow,ncol));
          A_norm = sqrt(A_norm) * droptol;
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
            if(nrow != ncol && abs(A(nrow,ncol)) < A_norm)
              M(nrow,ncol) = (0.0,0.0);
            else{
              M(nrow,ncol) = A(nrow,ncol);
              nnzM += 1;
            }
          }
          if(nrow >= d) y += 1;
        }
        return nnzM;
    }
//nc
    static int_type BAND_BlDiag_NormColumn_(mtl::matrix< Type
    >::type &A ,mtl::matrix< Type >::type &M, int_type t, int_type
    d,const fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      fp_type A_norm;
      A_norm = 0.0;
      int_type nnzM = 0;
        for(nrow = t, y = t; nrow < p; nrow ++){
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++)
            A_norm += abs(A(ncol,nrow)) * abs(A(ncol,nrow));
```

```
        A_norm = sqrt(A_norm) * droptol;
        for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
          if(nrow != ncol && abs(A(ncol,nrow)) < A_norm)
            M(ncol,nrow) = (0.0,0.0);
          else{
            M(ncol,nrow) = A(ncol,nrow);
            nnzM += 1;
          }
        }
        if(nrow >= d) y += 1;
      }
      return nnzM;
    }
//nm
    static int_type BAND_BlDiag_NormMatrix_(mtl::matrix< Type
    >::type &A ,mtl::matrix< Type >::type &M, int_type t, int_type
    d,const fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      fp_type A_norm;
      A_norm = 0.0;
      int_type nnzM = 0;
        for(nrow = t, y = t; nrow < p; nrow ++)
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++)
            A_norm += abs(A(nrow,ncol)) * abs(A(nrow,ncol));
          A_norm = sqrt(A_norm) * droptol;
        for(nrow = t, y = t; nrow < p; nrow ++){
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
            if(nrow != ncol && abs(A(nrow,ncol)) < A_norm)
              M(nrow,ncol) = (0.0,0.0);
            else {
              M(nrow,ncol) = A(nrow,ncol);
              nnzM += 1;
            }
          }
          if(nrow >= d) y += 1;
        }
        return nnzM;
    }
//inr
    static int_type BAND_BlDiag_NormInfRow_(mtl::matrix< Type
    >::type &A ,mtl::matrix< Type >::type &M, int_type t, int_type
    d,const fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      int_type nnzM = 0;
      fp_type Ai_norm;
      fp_type maxAi_norm;
      int_type maxi;
        for(nrow = t, y = t; nrow < p; nrow ++){
          Ai_norm = 0.0;
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++)
            Ai_norm += abs(A(nrow,ncol));
```

```
        if(nrow == t)
          maxAi_norm = Ai_norm;
        else{
          if(Ai_norm > maxAi_norm)
            maxAi_norm = Ai_norm;
        }
      }
      Ai_norm = maxAi_norm * droptol / p;
      for(nrow = t, y = t; nrow < p; nrow ++){
        for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
          if(nrow != ncol && abs(A(nrow,ncol)) < Ai_norm)
            M(nrow,ncol) = (0.0,0.0);
          else {
            M(nrow,ncol) = A(nrow,ncol);
            nnzM += 1;
          }
        }
        if(nrow >= d) y += 1;
      }
      return nnzM;
    }
//ddm
    static int_type
    Drop_if_ElemDivMaxElemInDiag_Less_(mtl::matrix< Type >::type
    &A, mtl::matrix< Type >::type &M, const fp_type eps){
      int_type nnzM = 0;
      fp_type a_max,temp;
      int_type i,j,p = A.nrows();
      a_max = abs(A(0,0));
        for(i = 1; i < p; i ++){
          temp = abs(A(i,i));
          if( temp > a_max)
            a_max = temp;
        }
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            temp = abs(A(i,j)) / a_max;
            if(temp >= eps || i == j){
              M(i,j) = A(i,j);
              nnzM += 1;
            }
          }
        return nnzM;
    }
//ind
    static int_type BAND_BlDiag_NormInfDiag_(mtl::matrix< Type
    >::type &A,mtl::matrix< Type >::type &M, int_type t, int_type
    d, const fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      int_type nnzM = 0;
      fp_type Ai_norm = 0.0;
```

```
        for(nrow = 0; nrow < p; nrow ++)
          Ai_norm += abs(A(nrow,nrow));
        Ai_norm = Ai_norm * droptol / p;
        for(nrow = t, y = t; nrow < p; nrow ++){
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
            if(nrow != ncol && abs(A(nrow,ncol)) < Ai_norm)
              M(nrow,ncol) = (0.0,0.0);
            else {
              M(nrow,ncol) = A(nrow,ncol);
              nnzM += 1;
            }
          }
          if(nrow >= d) y += 1;
        }
        return nnzM;
    }
//dmd
    static int_type
  Drop_if_ElemDivMaxElemOnDiagonal_Less_(mtl::matrix< Type >::type
  &A , mtl::matrix< Type >::type &M, const fp_type droptol){
      int_type nnzM = 0;
      fp_type a_max;
      int_type i,j,p = A.nrows();
        for(i = 0; i < p; i ++){
          a_max = abs(A(i,i)) * droptol;
          for(j = 0; j < p; j ++){
            if( i == j || abs(A(i,j)) > a_max ){
              M(i,j) = A(i,j);
              nnzM += 1;
            }
          }
        }
        return nnzM;
    }
//nd
    static int_type BAND_BlDiag_NormDiagonal_(mtl::matrix< Type
  >::type &A ,mtl::matrix< Type >::type &M, int_type t, int_type
  d, const fp_type droptol){
      int_type nrow,ncol,ntmp,y,p = A.nrows();
      fp_type A_norm;
      A_norm = 0.0;
      int_type nnzM = 0;
        for(nrow = 0; nrow < p; nrow ++)
          A_norm += abs(A(nrow,nrow)) * abs(A(nrow,nrow));
        A_norm = sqrt(A_norm) * droptol;
        for(nrow = t, y = t; nrow < p; nrow ++){
          for(ncol = y; ncol <= nrow + d && ncol < p; ncol ++){
            if(nrow != ncol && abs(A(nrow,ncol)) < A_norm)
              M(nrow,ncol) = (0.0,0.0);
            else{
              M(nrow,ncol) = A(nrow,ncol);
```

```
            nnzM += 1;
          }
        }
        if(nrow >= d) y += 1;
      }
      return nnzM;
    }
  };//class DropRule
}//namespace TALGAT
#endif//_DropRule_H_
/*************************************************************
                        Preconditioner.h
Description: способы формирования матрицы предобусловливания.
*************************************************************/
// (c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef  __Preconditioner_H__
#define  __Preconditioner_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include <time.h>
namespace TALGAT {
  template<class Type>
  class Preconditioner{
    public:
//LU
     static void Precond_form_LU_(mtl::matrix< Type >::type &A ,
     mtl::matrix< Type >::type &M, const fp_type droptol,
     int_type type_drop, int_type &nnzM_drp, int_type &nnzM_dec,
     fp_type &time_drp){
       fp_type timer;
       time_t timer0,timer1;
       int_type nrow,ncol,ntmp,p,d;
       p = A.nrows(); d = p - 1;
       timer0=clock();
       switch (type_drop){
         case LUnr_://nr
           nnzM_drp = DropRule< Type >::BAND_BlDiag_NormRow_(A, M,
           0, d, droptol); break;
         case 11://nc
           nnzM_drp = DropRule< Type >::BAND_BlDiag_NormColumn_(A,
           M, 0, d, droptol); break;
         case 12://nm
           nnzM_drp = DropRule< Type >::BAND_BlDiag_NormMatrix_(A,
           M, 0, d, droptol); break;
         case LUdm_://dm
           nnzM_drp=
```

```
    DropRule<Type>::Drop_if_ElemDivMaxElem_Less_(A, M,
    droptol);break;
  case 14://km
    int_type k;
    k = static_cast<int_type>(droptol);
    nnzM_drp = DropRule< Type
    >::kMaxElem_in_column_(A,M,k); break;
  case 15://ld
    nnzM_drp = DropRule< Type >::Drop_if_Elem_Less_(A, M,
    droptol); break;
  case 16://inr
    nnzM_drp = DropRule< Type >::BAND_BlDiag_NormInfRow_(A,
    M, 0, d, droptol); break;
  case 17://sunr
    nnzM_drp = DropRule< Type
    >::BAND_BlDiag_SymUpNormRow_(A, M, 0, d, droptol);
    break;
  case 18://slnr
    nnzM_drp = DropRule< Type
    >::BAND_BlDiag_SymLowNormRow_(A, M, 0, d, droptol);
    break;
  case 19://dmr
    nnzM_drp =
    DropRule<Type>::Drop_if_ElemDivMaxElemInRow_Less_(A, M,
    droptol); break;
  case LUddm_://ddm
    nnzM_drp =
    DropRule<Type>::Drop_if_ElemDivMaxElemInDiag_Less_(A,
    M, droptol); break;
  case LUind_://ind
    nnzM_drp = DropRule< Type
    >::BAND_BlDiag_NormInfDiag_(A, M, 0, d, droptol);
    break;
  case LUdmd_://dmd
    nnzM_drp = Dro-
    pRule<Type>::Drop_if_ElemDivMaxElemOnDiagonal_Less_(A,
    M, droptol); break;
  case LUnd_://nd
    nnzM_drp=DropRule<Type>::BAND_BlDiag_NormDiagonal_(A,
    M, 0, d, droptol); break;
}
timer1=clock();
time_drp=(difftime(timer1,timer0)) / CLOCKS_PER_SEC;
for(nrow = 1; nrow < p; nrow ++)
  for(ntmp = 0; ntmp <= nrow - 1; ntmp ++){
    if(M(nrow,ntmp) != (0.0,0.0) ){
      M(nrow,ntmp) /= M(ntmp,ntmp);
      for(ncol = ntmp + 1; ncol < p; ncol ++)
        if(M(ntmp,ncol) != (0.0,0.0))
          M(nrow,ncol) -= M(nrow,ntmp) * M(ntmp,ncol);
    }
```

```
        }
      nnzM_dec = Math_Func< Type >::NonZero(M);
    }
//ILU(0)
    static void Precond_form_ILU0_(mtl::matrix< Type >::type &A,
    mtl::matrix< Type >::type &M, const fp_type droptol,int_type
    type_drop, int_type &nnzM_drp, int_type &nnzM_dec, fp_type
    &time_drp){
      int_type nrow,ncol,ntmp,p,d;
      p = A.nrows(); d = p - 1;
      fp_type timer;
      time_t timer0,timer1;
      timer0=clock();
      switch (type_drop){
        case 20:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormRow_(A, M,
          0, d, droptol); break;
        case 21:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormColumn_(A,
          M, 0, d, droptol); break;
        case 22:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormMatrix_(A,
          M, 0, d, droptol); break;
        case 23:
          nnzM_drp = DropRule< Type
          >::Drop_if_ElemDivMaxElem_Less_(A, M, droptol);break;
        case 24:
          int_type k;
          k = static_cast<int_type>(droptol);
          nnzM_drp = DropRule< Type >::kMaxElem_in_column_(A, M,
          k); break;
        case 25:
          nnzM_drp = DropRule< Type >::Drop_if_Elem_Less_(A, M,
          droptol); break;
        case 26:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormInfRow_(A,
          M, 0, d, droptol); break;
        case 27:
          nnzM_drp=DropRule<Type>::Drop_if_ElemDivMaxElemInRow_Le
          ss_(A, M, droptol); break;
      }
      timer1=clock();
      time_drp=(difftime(timer1,timer0)) / CLOCKS_PER_SEC;
      for(nrow = 1; nrow < p; nrow ++)
        for(ntmp = 0; ntmp <= nrow-1; ntmp ++){
          if(M(nrow,ntmp) != (0.0,0.0)){
            M(nrow,ntmp) /= M(ntmp,ntmp);
            for(ncol = ntmp + 1; ncol < p; ncol ++)
              if(M(nrow,ncol) != (0.0,0.0))
                if(M(ntmp,ncol) != (0.0,0.0))
                  M(nrow,ncol) -= M(nrow,ntmp) * M(ntmp,ncol);
```

```
          }
        }
      nnzM_dec = Math_Func< Type >::NonZero(M);
    }
// LUP
    static void Precond_form_LUP(mtl::matrix< Type >::type &A ,
    mtl::matrix< Type >::type &M, const fp_type droptol,
    int_type type_drop, mtl::dense1D< int_type > &indj, int_type
    &nnzM_drp, int_type &nnzM_dec, fp_type &time_drp){
      int_type nrow,ncol,ntmp,p,d;
      fp_type timer;
      time_t timer0,timer1;
      timer0=clock();
      p = A.nrows(); d = p - 1;
      switch (type_drop){
        case 70:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormRow_(A, M,
          0, d, droptol); break;
        case 71:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormColumn_(A,
          M, 0, d, droptol); break;
        case 72:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormMatrix_(A,
          M, 0, d, droptol); break;
      }
      timer1=clock();
      time_drp=(difftime(timer1,timer0)) / CLOCKS_PER_SEC;
      int_type i,j,k;
      fp_type a_max,temp_;
      int_type ind,ind_;
      for(i = 0; i < p; i ++)
        indj[i] = i;
      for(i = 0; i < p; i ++){
        //lu decomposition
        if(i > 0){
          for(k = 0; k <= i - 1; k ++){
            if(M(i,indj[k]) != (0.0,0.0) ){
              M(i,indj[k]) /= M(k,indj[k]);
              for(j = k + 1; j < p; j ++)
                if(M(k,indj[j]) != (0.0,0.0))
                  M(i,indj[j]) -= M(i,indj[k]) * M(k,indj[j]);
            }
          }
        }//lu decomposition
        //pivoting
        a_max = abs(M(i,indj[i]));
        ind = indj[i];
        for(j = i + 1; j < p; j ++){
          temp_ = abs(M(i,indj[j]));
          if(temp_ > a_max){
            a_max = temp_;
```

```
            ind = j;
          }
        }
        if(ind != indj[i]){
          ind_ = indj[i];
          indj[i] = indj[ind];
          indj[ind] = ind_;
        }//pivoting
      }
      nnzM_dec = Math_Func< Type >::NonZero(M);
    }
//solve for LUP
    static void Precond_solve(const mtl::matrix< Type >::type
    &M,const mtl::dense1D< Type > &A, mtl::dense1D< Type > &B,
    const mtl::dense1D< int_type > &PERM){
      int_type nrow,ncol,p;
      p = B.size();
      mtl::dense1D< int_type > indj(p);
      for(nrow = 0; nrow < p; nrow ++)
        indj[nrow] = PERM[nrow];
      for(nrow = 0; nrow < p;nrow++)
        for(B[nrow] = A[nrow], ncol = 0; ncol < nrow; ncol++)
          if(M(nrow,indj[ncol]) != (0.0,0.0))
            B[nrow] -= M(nrow,indj[ncol]) * B[ncol];
      for(nrow = B.size() - 1; nrow >= 0; nrow--){
        for(ncol = nrow + 1; ncol < p; ncol++)
          if(M(nrow,indj[ncol]) != (0.0,0.0))
            B[nrow] -= M(nrow,indj[ncol]) * B[ncol];
          B[nrow] /= M(nrow,indj[nrow]);
      }//lu solve
      //back permutation
      Type temp;
      nrow = 0;
  here_:
      for(ncol = nrow; ncol < p; ncol ++)
        if(indj[ncol] == nrow){
          indj[ncol] = indj[nrow];
          temp = B[nrow];
          B[nrow] = B[ncol];
          B[ncol] = temp;
          nrow += 1;
          goto here_;
        }
    }
//solve
    static void Precond_solve_Band(const mtl::matrix< Type
    >::type &M,const mtl::dense1D< Type > &A, mtl::dense1D<Type>
    &B,int_type bl_num){
      int_type y,d,nrow,ncol;
      d = (bl_num - 1) / 2; int_type p = B.size();
      for(nrow = 0, y = 0; nrow < p; nrow ++){
```

```
    for(ncol = y, B[nrow] = A[nrow] ; ncol < nrow; ncol ++)
      if(M(nrow,ncol) != (0.0,0.0) )
        B[nrow] -=  M(nrow,ncol) * B[ncol];
    if(nrow >= d) y += 1;
  }
  for(nrow = p - 1, y = 0; nrow >= 0; nrow --){
    for(ncol = nrow + 1; ncol < p - y; ncol ++)
      if(M(nrow,ncol) != (0.0,0.0) )
        B[nrow] -=  M(nrow,ncol) * B[ncol];
    B[nrow] /= M(nrow,nrow);
    if(nrow < p - d) y += 1;
  }
}
static void Precond_solve_BlDiag(const mtl::matrix< Type
>::type &M,const mtl::dense1D< Type > &A,
  mtl::dense1D< Type > &B,int_type bl_num){
  int_type p = B.size();
  int_type nrow,ncol,m, bld = p / bl_num;
  for(m = 0; m < bl_num; m ++){
    for(nrow = m * bld; nrow < (m + 1) * bld && nrow < p;
    nrow ++)
      for(ncol = m * bld, B[nrow] = A[nrow]; ncol < nrow;
      ncol ++)
        B[nrow] -= M(nrow,ncol) * B[ncol];
      for(nrow = (m + 1) * bld - 1; nrow >= m * bld; nrow--){
        for(ncol = nrow + 1; ncol < (m + 1) * bld && ncol <
        p; ncol ++)
          B[nrow] -= M(nrow,ncol) * B[ncol];
        B[nrow] /= M(nrow,nrow);
      }
  }
  for(nrow = bl_num * bld; nrow < p; nrow ++)
    for(ncol = bl_num * bld, B[nrow] = A[nrow]; ncol < nrow;
    ncol ++)
      B[nrow] -= M(nrow,ncol) * B[ncol];
  for(nrow = p - 1; nrow >= bl_num * bld; nrow --){
    for(ncol = nrow + 1; ncol < p; ncol ++)
      B[nrow] -= M(nrow,ncol) * B[ncol];
    B[nrow] /= M(nrow,nrow);
  }
}
static void transPrecond_solve(const mtl::matrix< Type
>::type &M,const mtl::dense1D< Type > &A, mtl::dense1D< Type
> &B){
  int_type nrow,ncol,p;
  p = B.size();
  for(nrow = 0; nrow < p; nrow++){
    for(ncol = 0, B[nrow] = A[nrow]; ncol < nrow; ncol++)
      if(M(ncol,nrow) != (0.0,0.0))
        B[nrow] -= M(ncol,nrow) * B[ncol];
    B[nrow] /= M(nrow,nrow);
```

```
  }
  for(nrow = p - 1; nrow >= 0; nrow--)
    for(ncol = nrow + 1; ncol < p; ncol++)
      if(M(ncol,nrow) != (0.0,0.0))
        B[nrow] -= M(ncol,nrow) * B[ncol];
}
static void Precond_solve_M1(const mtl::matrix< Type >::type
&M,const mtl::dense1D< Type > &A, mtl::dense1D< Type > &B){
  int_type nrow,ncol;
  for(nrow = 0; nrow < B.size();nrow++)
    for(B[nrow] = A[nrow], ncol = 0; ncol < nrow; ncol++)
      if(M(nrow,ncol) != (0.0,0.0))
        B[nrow] -= M(nrow,ncol) * B[ncol];
}
static void transPrecond_solve_M1(const mtl::matrix< Type
>::type &M,const mtl::dense1D< Type > &A, mtl::dense1D< Type
> &B){
  int_type nrow,ncol,p;
  p = B.size();
  for(nrow = p - 1; nrow >= 0; nrow--)
    for(ncol = nrow + 1, B[nrow] = A[nrow]; ncol < p; ncol++)
      if(M(nrow,ncol) != (0.0,0.0))
        B[nrow] -= M(ncol,nrow) * B[ncol];
}
static void Precond_solve_M2(const mtl::matrix< Type >::type
&M, const mtl::dense1D< Type > &A, mtl::dense1D< Type > &B){
  int_type nrow,ncol,p;
  p = B.size();
  for(nrow = p - 1; nrow >= 0; nrow--){
    for(ncol = nrow + 1, B[nrow] = A[nrow]; ncol < p; ncol++)
      if(M(nrow,ncol) != (0.0,0.0))
        B[nrow] -= M(nrow,ncol) * B[ncol];
      B[nrow] /= M(nrow,nrow);
  }
}
static void transPrecond_solve_M2(const mtl::matrix< Type
>::type &M, const mtl::dense1D< Type > &A,mtl::dense1D< Type
> &B){
  int_type nrow,ncol;
  for(nrow = 0; nrow < B.size();nrow++){
    for(B[nrow] = A[nrow], ncol = 0; ncol < nrow; ncol++)
      if(M(nrow,ncol) != (0.0,0.0))
        B[nrow] -= M(ncol,nrow) * B[ncol];
    B[nrow] /= M(nrow,nrow);
  }
}
static void Precond_solve(const mtl::matrix< Type >::type
&M,const mtl::dense1D< Type > &A, mtl::dense1D< Type > &B){
  int_type nrow,ncol,p;
  p = B.size();
  for(nrow = 0; nrow < p;nrow++)
```

```
        for(B[nrow] = A[nrow], ncol = 0; ncol < nrow; ncol++)
          if(M(nrow,ncol) != (0.0,0.0))
            B[nrow] -= M(nrow,ncol) * B[ncol];
      for(nrow = B.size() - 1; nrow >= 0; nrow--){
        for(ncol = nrow + 1; ncol < p; ncol++)
          if(M(nrow,ncol) != (0.0,0.0))
            B[nrow] -= M(nrow,ncol) * B[ncol];
          B[nrow] /= M(nrow,nrow);
      }
    }
//IC(0)
    static void Precond_form_IC0_(mtl::matrix< Type >::type &A ,
    mtl::matrix< Type >::type &M, const fp_type droptol,int_type
    type_drop,int_type &nnzM_drp, int_type &nnzM_dec){
      int_type nrow,ncol,ntmp,p,d;
      p = A.nrows(); d = p - 1;
      switch (type_drop){
        case 10:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormRow_(A, M,
          0, d, droptol); break;
        case 11:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormColumn_(A,
          M, 0, d, droptol); break;
        case 12:
          nnzM_drp = DropRule< Type >::BAND_BlDiag_NormMatrix_(A,
          M, 0, d, droptol); break;
        case 13:
          nnzM_drp = DropRule< Type
          >::Drop_if_ElemDivMaxElem_Less_(A, M, droptol); break;
        case 14:
          int_type k;
          k = static_cast<int_type>(droptol);
          nnzM_drp = DropRule< Type >::kMaxElem_in_column_(A, M,
          k); break;
        case 15:
          nnzM_drp = DropRule< Type >::Drop_if_Elem_Less_(A,
          M,droptol); break;
      }
      for(ntmp = 1; ntmp < p; ntmp ++){
        for(ncol = 0; ncol <= ntmp - 1; ncol ++){
          for(nrow = 0; nrow <= ncol - 1; nrow ++)
            if(M(ntmp,ncol) != (0.0,0.0))
                M(ntmp,ncol) -= M(nrow,nrow) * M(ntmp,nrow) *
                M(ncol,nrow);
            M(ntmp,ncol) /= M(ncol,ncol);
        }
        for(nrow = 0; nrow <= ntmp - 1; nrow ++)
          if(M(ntmp,ntmp) != (0.0,0.0))
            M(ntmp,ntmp) -= M(ntmp,nrow) * M(ntmp,nrow) *
            M(nrow,nrow);
      }
```

```
        nnzM_dec = Math_Func< Type >::NonZero(M);
      }
      static void SPrecond_solve(const mtl::matrix< Type >::type
      &M,const mtl::dense1D< Type > &A, mtl::dense1D< Type > &B_){
        int_type nrow,ncol,p;
        p = B_.size();
        for(nrow = 0; nrow < p; nrow ++)
          for(ncol = 0; ncol < nrow; ncol ++)
            B_[nrow] -=  M(nrow,ncol) * B_[ncol];
        for(nrow = 0; nrow < p; nrow ++)
          B_[nrow] /= M(nrow,nrow);
        for(nrow = p - 2; nrow >= 0; nrow --)
          for(ncol = nrow + 1; ncol < p; ncol ++)
            B_[nrow] -= M(ncol,nrow) * B_[ncol];
      }
//ILUT
      static void Precond_form_ILUT_(mtl::matrix< Type >::type &A
      ,mtl::matrix< Type >::type &M, const fp_type drop-
      tol,int_type type_drop,int_type &nnzM_drp, int_type
      &nnzM_dec, fp_type &time_drp){
        fp_type timer;
        time_t timer0,timer1;
        int_type nrow,ncol,ntmp,p,d;
        p = A.nrows(); d = p - 1;
        mtl::dense1D< Type > W(p);
        timer0=clock();
        switch (type_drop){
          case 80:
            nnzM_drp = DropRule< Type
            >::BAND_BlDiag_NormRow_(A,M,0,d,droptol); break;
          case 81:
            nnzM_drp = DropRule< Type
            >::BAND_BlDiag_NormColumn_(A,M,0,d,droptol); break;
        }
        timer1=clock();
        time_drp=(difftime(timer1,timer0)) / CLOCKS_PER_SEC;
        fp_type normrow;
        for(nrow = 1; nrow < p; nrow ++){
          for(ncol = 0; ncol < p; ncol ++)
            W[ncol] = M(nrow,ncol);
          normrow = Math_Func< Type >::norm(W) * droptol;
          for(ntmp = 0; ntmp <= nrow - 1; ntmp ++){
            if(W[ntmp] != (0.0,0.0) ){
              W[ntmp] /= M(ntmp,ntmp);
              if(abs(W[ntmp]) < normrow)
                W[ntmp] = Type(0);
              else{
                for(ncol = ntmp + 1; ncol < p; ncol ++)
                  if(M(ntmp,ncol) != (0.0,0.0))
                    W[ncol] -= W[ntmp] * M(ntmp,ncol);
              }
```

```
          }
        }
        for(ncol = 0; ncol < p; ncol ++)
          if(abs(W[ncol]) < normrow)
            W[ncol] = Type(0);
        for(ncol = 0; ncol < p; ncol ++)
          M(nrow,ncol) = W[ncol];
      }
      nnzM_dec = Math_Func< Type >::NonZero(M);
    }
//LUC
    static void Precond_form_LUC_(mtl::matrix< Type >::type &A
    ,mtl::matrix< Type >::type &M, const fp_type droptol,
    int_type type_drop,int_type &nnzM_drp, int_type &nnzM_dec,
    fp_type &time_drp){
      fp_type timer;
      time_t timer0,timer1;
      int_type nrow,ncol,ntmp,p,d;
      p = A.nrows(); d = p - 1;
      timer0=clock();
      switch (type_drop){
        case 90:
          nnzM_drp = DropRule< Type
          >::BAND_BlDiag_NormRow_(A,M,0,d,droptol); break;
      }
      timer1=clock();
      time_drp = ((difftime(timer1,timer0)) / CLOCKS_PER_SEC);
      for(ntmp = 0; ntmp < p; ntmp ++){
        for(nrow = 0; nrow <= ntmp - 1; nrow ++)
          if(M(ntmp,nrow) != (0.0,0.0) )
            for(ncol = ntmp; ncol < p; ncol ++)
              M(ntmp,ncol) -= M(ntmp,nrow) * M(nrow,ncol);
        for(nrow = 0; nrow <= ntmp - 1; nrow ++)
          if(M(nrow,ntmp) != (0.0,0.0) )
            for(ncol = ntmp + 1; ncol < p; ncol ++)
              M(ncol,ntmp) -= M(nrow,ntmp) * M(ncol,nrow);
        for(ncol = ntmp + 1; ncol < p; ncol ++)
          M(ncol,ntmp) /= M(ntmp,ntmp);
      }
      nnzM_dec = Math_Func< Type >::NonZero(M);
    }
  };//class Preconditioner
}
#endif//_Preconditioner_H_
/*************************************************************
                            ABR1ORT.h
Description: метод Абрамова.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __ABR1ORT_H__
#define __ABR1ORT_H__
```

```
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//ABR1ORT
  template<class Type>
  void solvers_iterative< Type>::ABR1ORT(mtl::matrix< Type
  >::type& A_, mtl::dense1D< Type > &B_, mtl::dense1D< Type >
  &U_){
    int_type i,j,k;
    Type alfa;
    int_type p = B_.size();
    mtl::dense1D< Type > R_(p);
    Type a_norm;
    for(k = 0; k < p; k ++)
      R_[k] = B_[k];
    Math_Func< Type >::Initial_Guess(U_, 0.0);
    for(k = 0; k < p; k ++){
      for(j = 0, a_norm = 0.0; j < p; j ++)
        a_norm += A_(k,j) * A_(k,j);
      for(j = 0; j < p; j ++)
        U_[j] += (R_[k] / a_norm) * A_(k,j);
      for(i = k + 1; i < p; i ++){
        for(j = 0, alfa = 0.0; j < p; j ++)
          alfa += A_(k,j) * A_(i,j);
        alfa /= a_norm;
        for(j = 0; j < p; j ++)
          A_(i,j) -= alfa * A_(k,j);
        R_[i] = R_[i] - alfa * R_[k];
      }//i
    }//k
  }//ABR1ORT
  HandlerResult CallABR1ORT_r(hpRealMatrix a, hpRealMatrix b, In-
  terpreter*, Interpreter*){
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SRealMatrix::vec_type vec_b(a->m.nrows());
    SRealMatrix::vec_type result(a->m.nrows());
    SRealMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< fp_type >::ABR1ORT(a->m, vec_b, result);
    pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
    SRealMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
  HandlerResult CallABR1ORT(hpComplexMatrix a, hpComplexMatrix b,
  Interpreter*,Interpreter*){
```

```
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SComplexMatrix::vec_type vec_b(a->m.nrows());
    SComplexMatrix::vec_type result(a->m.nrows());
    SComplexMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< complex >::ABR1ORT(a->m, vec_b, result);
    pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
    SComplexMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
}//namespace TALGAT
#endif //__ABR1ORT_H__
/*************************************************************
                                BICG.h
Description: метод бисопряженных градиентов.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __BICG_H__
#define __BICG_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//BiCG
  template<class Type>
  void solvers_iterative< Type>::BiCG(mtl::matrix< Type >::type&
  A_, const mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_,
  fp_type x, fp_type tol, int_type niter_max,int_type type_pre,
  fp_type droptol, int_type bl_num){
    fp_type timer,timer_drp = 0.0;
    time_t timer0,timer1;
    TimeCounter timer_new;
    int_type p = B_.size();
    int_type nnzM = 0 , nnzM_ = 0;
    fp_type tol_;
    mtl::dense1D< Type >
    Rt(p),R(p),P(p),Pt(p),Z(p),Zt(p),Q(p),Qt(p);
    Type BETA,ALFA,RtZ;
    Math_Func< Type >::Initial_Guess(U_, x);
    Math_Func< Type >::Mat_Vec(A_,U_,R);
    Math_Func< Type >::VecPlusScalarVec(B_,-1.0,R,R);
    fp_type b_norm = Math_Func< Type >::norm(R);
    mtl::matrix< Type >::type M(p,p);
    timer0=clock();
    timer_new.Start();
    switch (type_pre){
```

```
  case LUnr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol,LUnr_, nnzM, nnzM_, timer_drp);break;
  case LUnc_:Preconditioner< Type>::Precond_form_LU_(A_, M,
  droptol, 11, nnzM, nnzM_,timer_drp);break;
  case LUnm_:Preconditioner< Type >::Precond_form_LU_(A_,M,
  droptol, 12, nnzM, nnzM_,timer_drp);break;
  case LUdm_:Preconditioner< Type >::Precond_form_LU_(A_,M,
  droptol, LUdm_, nnzM, nnzM_, timer_drp);break;
  case LUkm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 14, nnzM, nnzM_, timer_drp);break;
  case LUld_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 15, nnzM, nnzM_, timer_drp);break;
  case LUinr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 16, nnzM, nnzM_, timer_drp);break;
  case LUdmr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 19, nnzM, nnzM_, timer_drp); break;
  case LUddm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUddm_, nnzM, nnzM_, timer_drp);break;
  case LUind_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUind_, nnzM, nnzM_, timer_drp); break;
  case LUdmd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUdmd_, nnzM, nnzM_, timer_drp);break;
  case LUnd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUnd_, nnzM, nnzM_, timer_drp);break;
  case ILU0nr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 20, nnzM, nnzM_, timer_drp);break;
  case ILU0nc_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 21, nnzM, nnzM_, timer_drp);break;
  case ILU0nm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 22, nnzM, nnzM_, timer_drp);break;
  case ILU0dm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 23, nnzM, nnzM_, timer_drp);break;
  case ILU0km_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 24, nnzM, nnzM_, timer_drp);break;
  case ILU0ld_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 25, nnzM, nnzM_, timer_drp);break;
  case ILU0inr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 26, nnzM, nnzM_, timer_drp);break;
  case ILU0dmr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 27, nnzM, nnzM_, timer_drp);break;
  case ILUpnr_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 60, bl_num, nnzM, nnzM_, timer_drp);break;
  case ILUpnc_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 61, bl_num, nnzM, nnzM_, timer_drp);break;
  case ILUpnm_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 62, bl_num, nnzM, nnzM_, timer_drp);break;
}
timer1=clock();
timer_new.Stop();
  for(int_type i = 0; i < p; i++)
    Rt[i] =  R[i];
```

```
    for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
      if(type_pre == None_)
        for(int_type i = 0; i < p; i++){
          Z[i] = R[i];
          Zt[i] = Rt[i];
        }
      else {
        Preconditioner< Type >::Precond_solve(M,R,Z);
        Preconditioner< Type >::transPrecond_solve(M,Rt,Zt);
      }
      RtZ = Math_Func< Type >::Vec_Vec(Z,Rt);
      if(NumIter == 1){
        for(int_type i = 0; i < p; i++){
          P[i] = Z[i];
          Pt[i] = Zt[i];
        }
      }
      else{
        BETA = RtZ / BETA;
        Math_Func< Type >::VecPlusScalarVec(Z,BETA,P,P);
        Math_Func< Type >::VecPlusScalarVec(Zt,BETA,Pt,Pt);
      }
      Math_Func< Type >::Mat_Vec(A_,P,Q);
      Math_Func< Type >::transMat_Vec(A_,Pt,Qt);
      BETA = Math_Func< Type >::Vec_Vec(Pt,Q);
      ALFA = RtZ / BETA;
      BETA = RtZ;
      Math_Func< Type >::VecPlusScalarVec(U_,ALFA,P,U_);
      Math_Func< Type >::VecPlusScalarVec(R,-ALFA,Q,R);
      Math_Func< Type >::VecPlusScalarVec(Rt,-ALFA,Qt,Rt);
      tol_ = Math_Func< Type >::norm(R) / b_norm;
      if ( tol_ <= tol  || NumIter == niter_max) break;
    }//NumIter
}//BiCG
HandlerResult CallBICG_r(hpRealMatrix a, hpRealMatrix b, Inter-
preter* ir, fp_type x, fp_type tol, int_type niter_max, int_type
type_pre, fp_type droptol){
  int_type bl_num = a->m.nrows();
  HandlerResult hr = ir->InterpretForResult();
  if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::vec_type result(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_iterative< fp_type >::BiCG(a->m, vec_b, result, x,
  tol, niter_max, type_pre, droptol,bl_num);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(result, ret->m);
  return HandlerResult(pMatrix(ret));
}
```

```
  HandlerResult CallBICG(hpComplexMatrix a, hpComplexMatrix b, In-
  terpreter* ir, fp_type x, fp_type tol, int_type niter_max,
  int_type type_pre, fp_type droptol){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SComplexMatrix::vec_type vec_b(a->m.nrows());
    SComplexMatrix::vec_type result(a->m.nrows());
    SComplexMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< complex >::BiCG(a->m, vec_b, result, x,
    tol, niter_max, type_pre, droptol,bl_num);
    pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
    SComplexMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
}//namespace TALGAT
#endif //__BICG_H__
/*************************************************************
                            BICGSTAB.h
Description: стабилизированный метод бисопряженных градиентов.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __BICGSTAB_H__
#define __BICGSTAB_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//BiCGStab
  template<class Type>
  void solvers_iterative< Type>::BiCGStab(mtl::matrix< Type
  >::type& A_,const mtl::dense1D< Type > &B_, mtl::dense1D< Type >
  &U_, fp_type x, fp_type tol, int_type niter_max,int_type
  type_pre, fp_type droptol, int_type bl_num){
    int_type nnzM = 0 , nnzM_ = 0;
    fp_type tol_;
    int_type p = B_.size();
    mtl::dense1D< Type >
    Rt(p),R(p),P(p),V(p),Pt(p),S(p),St(p),T(p);
    Type BETA,ALFA,RtR_,omega,RtR;
    Math_Func< Type >::Initial_Guess(U_, x);
    Math_Func< Type >::Mat_Vec(A_,U_,R);
    Math_Func< Type >::VecPlusScalarVec(B_,-1.0,R,R);
    fp_type b_norm = Math_Func< Type >::norm(R);
```

```
mtl::matrix< Type >::type M(p,p);
mtl::dense1D< int_type > PERM(p);
fp_type timer,timer_drp = 0.0;
time_t timer0,timer1;
TimeCounter timer_new;
timer0=clock();
timer_new.Start();
if(type_pre == DLUnr_ || type_pre == DLUnc_ || type_pre ==
DLUnm_ || type_pre == DILU0nr_ || type_pre == DILU0nc_ ||
type_pre == DILU0nm_){
  if(bl_num >= 2 * p)
    bl_num = 2*p - 1;
  if(bl_num % 2 == 0)
    bl_num += 1;
}
switch (type_pre){
  case LUnr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUnr_, nnzM, nnzM_, timer_drp); break;
  case LUnc_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 11, nnzM, nnzM_, timer_drp); break;
  case LUnm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 12, nnzM, nnzM_, timer_drp); break;
  case LUdm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUdm_, nnzM, nnzM_, timer_drp); break;
  case LUkm_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,14,nnzM,nnzM_,timer_drp);
  break;
  case LUld_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 15, nnzM, nnzM_, timer_drp);break;
  case LUinr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 16, nnzM, nnzM_, timer_drp);break;
  case LUdmr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 19, nnzM, nnzM_, timer_drp);break;
  case LUddm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUddm_, nnzM, nnzM_, timer_drp);break;
  case LUind_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUind_, nnzM, nnzM_, timer_drp);break;
  case LUdmd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUdmd_, nnzM, nnzM_, timer_drp);break;
  case LUnd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUnd_, nnzM, nnzM_, timer_drp);break;
  case ILU0nr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 20, nnzM, nnzM_, timer_drp);break;
  case ILU0nc_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 21, nnzM, nnzM_, timer_drp);break;
  case ILU0nm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 22, nnzM, nnzM_, timer_drp);break;
  case ILU0dm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 23, nnzM, nnzM_, timer_drp);break;
  case ILU0km_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 24, nnzM, nnzM_, timer_drp);break;
```

```
case ILU0ld_:Preconditioner< Type >::Precond_form_ILU0_(A_,
M, droptol, 25, nnzM, nnzM_, timer_drp);break;
case ILU0inr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
M, droptol, 26, nnzM, nnzM_, timer_drp);break;
case ILU0dmr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
M, droptol, 27, nnzM, nnzM_, timer_drp);break;
case DLUnr_:Preconditioner< Type
>::Precond_form_BAND_LU_(A_, M, droptol, 30, bl_num, nnzM,
nnzM_); break;
case DLUnc_:Preconditioner< Type
>::Precond_form_BAND_LU_(A_, M, droptol, 31, bl_num, nnzM,
nnzM_);break;
case DLUnm_:Preconditioner< Type
>::Precond_form_BAND_LU_(A_, M, droptol, 32, bl_num, nnzM,
nnzM_);break;
case DILU0nr_:Preconditioner< Type
>::Precond_form_BAND_ILU0_(A_, M, droptol, 40, bl_num, nnzM,
nnzM_); break;
case DILU0nc_:Preconditioner< Type
>::Precond_form_BAND_ILU0_(A_, M, droptol, 41, bl_num, nnzM,
nnzM_); break;
case DILU0nm_:Preconditioner< Type
>::Precond_form_BAND_ILU0_(A_, M, droptol, 42, bl_num, nnzM,
nnzM_); break;
case 50:Preconditioner< Type >::Precond_form_BlDiag_LU_(A_,
M, droptol, 50, bl_num, nnzM, nnzM_); break;
case 51:Preconditioner< Type >::Precond_form_BlDiag_LU_(A_,
M, droptol, 51, bl_num, nnzM, nnzM_); break;
case 52:Preconditioner< Type
>::Precond_form_BlDiag_LU_(A_,M,droptol,52,bl_num,nnzM,nnzM_
);break;
case ILUpnr_:Preconditioner< Type >::Precond_form_ILUp_(A_,
M, droptol, 60, bl_num, nnzM, nnzM_, timer_drp); break;
case ILUpnc_:Preconditioner< Type >::Precond_form_ILUp_(A_,
M, droptol, 61, bl_num, nnzM, nnzM_, timer_drp); break;
case ILUpnm_:Preconditioner< Type >::Precond_form_ILUp_(A_,
M, droptol, 62, bl_num, nnzM, nnzM_, timer_drp); break;
case LUPnr_:Preconditioner< Type >::Precond_form_LUP(A_, M,
droptol, 70, PERM, nnzM, nnzM_, timer_drp);break;
case LUPnc_:Preconditioner< Type >::Precond_form_LUP(A_, M,
droptol, 71, PERM, nnzM, nnzM_, timer_drp); break;
case LUPnm_:Preconditioner< Type >::Precond_form_LUP(A_, M,
droptol, 72, PERM, nnzM, nnzM_, timer_drp); break;
case ILUTnr_:Preconditioner< Type >::Precond_form_ILUT_(A_,
M, droptol, 80, nnzM, nnzM_, timer_drp); break;
case ILUTnc_:Preconditioner< Type >::Precond_form_ILUT_(A_,
M, droptol, 81, nnzM, nnzM_, timer_drp); break;
case ILUTnm_:Preconditioner< Type >::Precond_form_ILUT_(A_,
M, droptol, 82, nnzM, nnzM_, timer_drp); break;
case LUCnr_:Preconditioner< Type >::Precond_form_LUC_(A_, M,
droptol, 90, nnzM, nnzM_, timer_drp); break;
```

```
}
timer1=clock();
timer_new.Stop();
  for(int_type i = 0; i < p; i++)
    Rt[i] = P[i] = R[i];
  for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
    RtR = Math_Func< Type >::Vec_Vec(Rt,R);
    if(NumIter > 1){
      BETA = (RtR / RtR_) * (ALFA / omega);
      Math_Func< Type >::VecPlusScalarVec(P,-omega,V,P);
      Math_Func< Type >::VecPlusScalarVec(R,BETA,P,P);
    }
    if(type_pre == None_)
      for(int_type i = 0; i < p; i++)
        Pt[i] = P[i];
    else if(type_pre == LUnr_ || type_pre == LUnc_ || type_pre
    == LUnm_ || type_pre == LUdm_ || type_pre == LUkm_ ||
    type_pre == LUld_ || type_pre == LUinr_ || type_pre ==
    LUsunr_ || type_pre == LUslnr_ || type_pre == LUdmr_ ||
    type_pre == ILU0nr_ || type_pre == ILU0nc_ || type_pre ==
    ILU0nm_ || type_pre == ILU0dm_ || type_pre == ILU0km_ ||
    type_pre == ILU0dmr_ || type_pre == ILU0inr_ ||type_pre ==
    ILU0ld_ || type_pre == ILUpnr_ || type_pre == ILUpnc_ ||
    type_pre == ILUpnm_ || type_pre == ILUTnr_ || type_pre ==
    ILUTnc_ || type_pre == ILUTnm_ || type_pre == LUCnr_ ||
    type_pre == LUddm_ || type_pre == LUind_ || type_pre ==
    LUdmd_ || type_pre == LUnd_)
        Preconditioner< Type >::Precond_solve(M,P,Pt);
    else if(type_pre == DLUnr_ || type_pre == DLUnc_ ||
    type_pre == DLUnm_ || type_pre == DILU0nr_ || type_pre ==
    DILU0nc_ || type_pre == DILU0nm_)
        Preconditioner< Type >::Precond_solve_Band(M, P, Pt,
        bl_num);
    else if(type_pre == 50 || type_pre == 51 || type_pre == 52)
      Preconditioner< Type >::Precond_solve_BlDiag(M, P, Pt,
      bl_num);
    else if(type_pre == LUPnr_ || type_pre == LUPnc_ ||
    type_pre == LUPnm_)
      Preconditioner< Type >::Precond_solve(M,P,Pt,PERM);
    Math_Func< Type >::Mat_Vec(A_,Pt,V);
    RtR_ = Math_Func< Type >::Vec_Vec(Rt,V);
    ALFA = RtR/RtR_;
    RtR_ = RtR;
    Math_Func< Type >::VecPlusScalarVec(R,-ALFA,V,S);
    tol_ = abs(Math_Func< Type >::norm(S));
    if (  tol_ <= tol ){
      Math_Func< Type >::VecPlusScalarVec(U_,ALFA,Pt,U_);
      goto END;
    }
    if(type_pre == None_)
      for(int_type i = 0; i < p; i++)
```

```
          St[i] = S[i];
        else if(type_pre == LUnr_ || type_pre == LUnc_ || type_pre
        == LUnm_ || type_pre == LUdm_ || type_pre == LUkm_ ||
        type_pre == LUld_ || type_pre == LUinr_|| type_pre ==
        LUsunr_ || type_pre == LUslnr_ || type_pre == LUdmr_ ||
        type_pre == ILU0nr_ || type_pre == ILU0nc_|| type_pre ==
        ILU0nm_ || type_pre == ILU0dm_ || type_pre == ILU0km_ ||
        type_pre == ILU0dmr_ || type_pre == ILU0inr_ || type_pre ==
        ILU0ld_ ||type_pre == ILUpnr_ || type_pre == ILUpnc_ ||
        type_pre == ILUpnm_ || type_pre == ILUTnr_ || type_pre ==
        ILUTnc_ || type_pre == ILUTnm_ || type_pre == LUCnr_ ||
        type_pre == LUddm_ || type_pre == LUind_  || type_pre ==
        LUdmd_ || type_pre == LUnd_)
          Preconditioner< Type >::Precond_solve(M,S,St);
        else if(type_pre == DLUnr_ || type_pre == DLUnc_ ||
        type_pre == DLUnm_ || type_pre == DILU0nr_ || type_pre ==
        DILU0nc_ || type_pre == DILU0nm_)
          Preconditioner< Type >::Precond_solve_Band(M, S, St,
          bl_num);
        else if(type_pre == 50 || type_pre == 51 || type_pre == 52)
          Preconditioner< Type >::Precond_solve_BlDiag(M, S, St,
          bl_num);
        else if(type_pre == LUPnr_ || type_pre == LUPnc_ ||
        type_pre == LUPnm_)
          Preconditioner< Type >::Precond_solve(M,S,St,PERM);
        Math_Func< Type >::Mat_Vec(A_,St,T);
        omega= Math_Func< Type >::Vec_Vec(T,S);
        RtR = Math_Func< Type >::Vec_Vec(T,T);
        omega /= RtR;
        Math_Func< Type >::VecPlusScalarVec(U_,ALFA,Pt,U_);
        Math_Func< Type >::VecPlusScalarVec(U_,omega,St,U_);
        Math_Func< Type >::VecPlusScalarVec(S,-omega,T,R);
        tol_ = Math_Func< Type >::norm(R) / b_norm;
        if ( tol_ <= tol  || NumIter == niter_max) break;
      }//NumIter
END:;
  }//BiCGStab
  HandlerResult CallBICGSTAB_r(hpRealMatrix a, Interpreter* ir,
  hpRealMatrix b, fp_type x, fp_type tol, int_type niter_max,
  int_type type_pre, fp_type droptol){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SRealMatrix::vec_type vec_b(a->m.nrows());
    SRealMatrix::vec_type result(a->m.nrows());
    SRealMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< fp_type >::BiCGStab(a->m, vec_b, result, x,
    tol, niter_max, type_pre, droptol, bl_num);
    pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
```

```
    SRealMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
  HandlerResult CallBICGSTAB(hpComplexMatrix a, Interpreter* ir,
  hpComplexMatrix b, fp_type x, fp_type tol, int_type niter_max,
  int_type type_pre, fp_type droptol){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SComplexMatrix::vec_type vec_b(a->m.nrows());
    SComplexMatrix::vec_type result(a->m.nrows());
    SComplexMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< complex >::BiCGStab(a->m, vec_b, result, x,
    tol, niter_max, type_pre, droptol, bl_num);
    pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
    SComplexMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
}//namespace TALGAT
#endif //__BICGSTAB_H__
/*************************************************************
                                CG.h
Description: метод сопряженных градиентов.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __CG_H__
#define __CG_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//CG
  template<class Type>
  void solvers_iterative< Type>::CG(mtl::matrix< Type >::type&
  A_,const mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_,
  fp_type x, fp_type tol, int_type niter_max, int_type type_pre,
  fp_type droptol){
    fp_type timer,timer_drp = 0.0;
    int_type p = B_.size();
    int_type i;
    int_type nnzM = 0 , nnzM_ = 0;
    fp_type tol_;
    Type rho,rho_,BETA,ALFA;
    mtl::dense1D< Type > R(p),Z(p),P(p),Q(p);
```

```
Math_Func< Type >::Initial_Guess(U_, x);
Math_Func< Type >::Mat_Vec(A_,U_,R);
Math_Func< Type >::VecPlusScalarVec(B_,-1.0,R,R);
fp_type b_norm = Math_Func< Type >::norm(R);
mtl::matrix< Type >::type M(p,p);
switch (type_pre){
  case LUnr_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,LUnr_,nnzM,nnzM_,timer_drp)
  ; break;
  case LUnc_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,11,nnzM,nnzM_,timer_drp);
  break;
  case LUnm_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,12,nnzM,nnzM_,timer_drp);
  break;
  case LUdm_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,LUdm_,nnzM,nnzM_,timer_drp)
  ; break;
  case LUkm_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,14,nnzM,nnzM_,timer_drp);
  break;
  case LUld_:Preconditioner< Type
  >::Precond_form_LU_(A_,M,droptol,15,nnzM,nnzM_,timer_drp);
  break;
  case ILU0nr_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,20,nnzM,nnzM_,timer_drp);
  break;
  case ILU0nc_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,21,nnzM,nnzM_,timer_drp);
  break;
  case ILU0nm_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,22,nnzM,nnzM_,timer_drp);
  break;
  case ILU0dm_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,23,nnzM,nnzM_,timer_drp);
  break;
  case ILU0km_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,24,nnzM,nnzM_,timer_drp);
  break;
  case ILU0ld_:Preconditioner< Type
  >::Precond_form_ILU0_(A_,M,droptol,25,nnzM,nnzM_,timer_drp);
  break;
}
for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
  if(type_pre == None_)
    for(i = 0; i < p; i++)
      Z[i] = R[i];
  else
    Preconditioner< Type >::Precond_solve(M,R,Z);
  rho = Math_Func< Type >::Vec_Vec(R,Z);
  if(NumIter == 1)
```

```
      for(i = 0; i < p; i++)
        P[i] = Z[i];
    else{
      BETA = rho / rho_;
      Math_Func< Type >::VecPlusScalarVec(Z,BETA,P,P);
    }
    Math_Func< Type >::Mat_Vec(A_,P,Q);
    ALFA = Math_Func< Type >::Vec_Vec(P,Q);
    ALFA = rho / ALFA;
    rho_ = rho;
    Math_Func< Type >::VecPlusScalarVec(U_,ALFA,P,U_);
    Math_Func< Type >::VecPlusScalarVec(R,-ALFA,Q,R);
    tol_ = Math_Func< Type >::norm(R) / b_norm;
    if ( tol_ <= tol  || NumIter == niter_max) break;
  }//NumIter
}//CG
HandlerResult CallCG_r(hpRealMatrix a, hpRealMatrix b, fp_type
x, Interpreter*, fp_type tol, int_type niter_max, int_type
type_pre, fp_type droptol){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::vec_type result(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_iterative< fp_type >::CG(a->m, vec_b, result, x, tol,
  niter_max, type_pre, droptol);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(result, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallCG(hpComplexMatrix a, hpComplexMatrix b,
fp_type x, Interpreter*, fp_type tol, int_type niter_max,
int_type type_pre, fp_type droptol){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::vec_type result(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_iterative< complex >::CG(a->m, vec_b, result, x, tol,
  niter_max, type_pre, droptol);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(result, ret->m);
  return HandlerResult(pMatrix(ret));
}
}//namespace TALGAT
#endif //__CG_H__
/*************************************************************
                              CGS.h
Description: квадратичный метод сопряженных градиентов.
*************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
```

```
#ifndef __CGS_H__
#define __CGS_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//CGS
  template<class Type>
  void solvers_iterative< Type>::CGS(mtl::matrix< Type >::type&
  A_,const mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_,
  fp_type x, fp_type tol, int_type niter_max, int_type type_pre,
  fp_type droptol, int_type bl_num){
    fp_type timer,timer_drp = 0.0;
    time_t timer0,timer1;
    int_type p = B_.size();
    mtl::dense1D< Type > Rt(p), R(p), Ut(p), P(p), Q(p), V(p),
    Pt(p), U(p), Qt(p);
    Type BETA,ALFA,RtR;
    fp_type tol_;
    int_type nnzM = 0 , nnzM_ = 0;
    Math_Func< Type >::Initial_Guess(U_, x);
    Math_Func< Type >::Mat_Vec(A_,U_,R);
    Math_Func< Type >::VecPlusScalarVec(B_,-1.0,R,R);
    fp_type b_norm = Math_Func< Type >::norm(R);
    mtl::matrix< Type >::type M(p,p);
    timer0=clock();
    TimeCounter timer_new;
    timer0=clock();
    timer_new.Start();
    switch (type_pre){
      case LUnr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, LUnr_, nnzM, nnzM_, timer_drp); break;
      case LUnc_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 11, nnzM, nnzM_, timer_drp); break;
      case LUnm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 12, nnzM, nnzM_, timer_drp); break;
      case LUdm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, LUdm_, nnzM, nnzM_, timer_drp); break;
      case LUkm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 14, nnzM, nnzM_, timer_drp); break;
      case LUld_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 15, nnzM, nnzM_, timer_drp); break;
      case LUinr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 16, nnzM, nnzM_, timer_drp); break;
      case LUdmr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 19, nnzM, nnzM_, timer_drp); break;
```

```
  case LUddm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUddm_, nnzM, nnzM_, timer_drp); break;
  case LUind_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUind_, nnzM, nnzM_, timer_drp); break;
  case LUdmd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUdmd_, nnzM, nnzM_, timer_drp); break;
  case LUnd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUnd_, nnzM, nnzM_, timer_drp); break;
  case ILU0nr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 20, nnzM, nnzM_, timer_drp); break;
  case ILU0nc_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 21, nnzM, nnzM_, timer_drp); break;
  case ILU0nm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 22, nnzM, nnzM_, timer_drp); break;
  case ILU0dm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 23, nnzM, nnzM_, timer_drp); break;
  case ILU0km_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 24, nnzM, nnzM_, timer_drp); break;
  case ILU0ld_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 25, nnzM, nnzM_, timer_drp); break;
  case ILU0inr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 26, nnzM, nnzM_, timer_drp); break;
  case ILU0dmr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 27, nnzM, nnzM_, timer_drp); break;
  case ILUpnr_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 60, bl_num, nnzM, nnzM_, timer_drp); break;
  case ILUpnc_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 61, bl_num, nnzM,nnzM_, timer_drp); break;
  case ILUpnm_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 62, bl_num, nnzM, nnzM_, timer_drp); break;
}
timer1=clock();
timer_new.Stop();
  for(int_type i = 0; i < p; i++)
    Rt[i] = P[i] = U[i] = R[i];
  for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
    RtR = Math_Func< Type >::Vec_Vec(Rt,R);
    if(NumIter > 1){
      BETA=RtR/BETA;
      Math_Func< Type >::VecPlusScalarVec(R,BETA,Q,U);
      Math_Func< Type >::VecPlusScalarVec(Q,BETA,P,P);
      Math_Func< Type >::VecPlusScalarVec(U,BETA,P,P);
    }
    if(type_pre == None_)
      for(int_type i = 0; i < p; i++)
        Pt[i] = P[i];
    else
      Preconditioner< Type >::Precond_solve(M,P,Pt);
    Math_Func< Type >::Mat_Vec(A_,Pt,V);
    BETA = Math_Func< Type >::Vec_Vec(V,Rt);
    ALFA = RtR/BETA;
```

```
      BETA = RtR;
      Math_Func< Type >::VecPlusScalarVec(U,-ALFA,V,Q);
      Math_Func< Type >::VecPlusScalarVec(U,1.0,Q,Qt);
      if(type_pre == None_)
        for(int_type i = 0; i < p; i++)
          Ut[i] = Qt[i];
      else
        Preconditioner< Type >::Precond_solve(M,Qt,Ut);
      Math_Func< Type >::VecPlusScalarVec(U_,ALFA,Ut,U_);
      Math_Func< Type >::Mat_Vec(A_,Ut,Qt);
      Math_Func< Type >::VecPlusScalarVec(R,-ALFA,Qt,R);
      tol_ = Math_Func< Type >::norm(R) / b_norm;
      if ( tol_ <= tol  || NumIter == niter_max) break;
    }//NumIter
  fp_type prec_time_form=(difftime(timer1,timer0)) /
  CLOCKS_PER_SEC;
}//CGS
HandlerResult CallCGS_r(Interpreter* ir, hpRealMatrix a, hpReal-
Matrix b, fp_type x, fp_type tol, int_type niter_max, int_type
type_pre, fp_type droptol){
  int_type bl_num = a->m.nrows();
  HandlerResult hr = ir->InterpretForResult();
  if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::vec_type result(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_iterative< fp_type >::CGS(a->m, vec_b, result, x, tol,
  niter_max, type_pre, droptol,bl_num);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(result, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallCGS(Interpreter* ir, hpComplexMatrix a, hpCom-
plexMatrix b, fp_type x, fp_type tol, int_type niter_max,
int_type type_pre, fp_type droptol){
  int_type bl_num = a->m.nrows();
  HandlerResult hr = ir->InterpretForResult();
  if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::vec_type result(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_iterative< complex >::CGS(a->m, vec_b, result, x, tol,
  niter_max, type_pre, droptol,bl_num);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(result, ret->m);
  return HandlerResult(pMatrix(ret));
}
```

```
}//namespace TALGAT
#endif //__CGS_H__

/************************************************************
                              GMRES.h
Description: обобщенный метод минимальных невязок.
************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __GMRES_H__
#define __GMRES_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//GMRES_m
  template<class Type>
  void solvers_iterative< Type>::GMRES_m(mtl::matrix< Type
  >::type& A_,const mtl::dense1D< Type > &B_, mtl::dense1D< Type >
  &U_, fp_type x, fp_type tol, int_type niter_max, int_type m,
  int_type type_pre, fp_type droptol,int_type bl_num) {
    fp_type timer,timer_drp = 0.0;
    int_type i,j,k;
    fp_type tol_;
    int_type nnzM = 0 , nnzM_ = 0;
    int_type p = B_.size();
    mtl::dense1D< Type > R(p),Z(p),W(p),Q(m+1),CS(m+1),SN(m+1);
    mtl::matrix< Type >::type M(p,p),H(m+1,m),V(p,m+1);
    Math_Func< Type >::Initial_Guess(U_, x);
    Math_Func< Type >::Mat_Vec(A_,U_,Z);
    Math_Func< Type >::VecPlusScalarVec(B_,-1.0,Z,Z);
    fp_type b_norm = Math_Func< Type >::norm(Z);
    switch (type_pre){
      case LUnr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, LUnr_, nnzM, nnzM_, timer_drp); break;
      case LUnc_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 11, nnzM, nnzM_, timer_drp); break;
      case LUnm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 12, nnzM, nnzM_, timer_drp); break;
      case LUdm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, LUdm_, nnzM, nnzM_, timer_drp); break;
      case LUkm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 14, nnzM, nnzM_, timer_drp); break;
      case LUld_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 15, nnzM, nnzM_, timer_drp); break;
      case LUinr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
      droptol, 16, nnzM, nnzM_, timer_drp); break;
```

```
  case LUdmr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, 19, nnzM, nnzM_, timer_drp); break;
  case LUddm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUddm_, nnzM, nnzM_, timer_drp); break;
  case LUind_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUind_, nnzM, nnzM_, timer_drp); break;
  case LUdmd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUdmd_, nnzM, nnzM_, timer_drp); break;
  case LUnd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
  droptol, LUnd_, nnzM, nnzM_, timer_drp); break;
  case ILU0nr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 20, nnzM, nnzM_, timer_drp); break;
  case ILU0nc_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 21, nnzM, nnzM_, timer_drp); break;
  case ILU0nm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 22, nnzM, nnzM_, timer_drp); break;
  case ILU0dm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 23, nnzM, nnzM_, timer_drp); break;
  case ILU0km_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 24, nnzM, nnzM_, timer_drp); break;
  case ILU0ld_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 25, nnzM, nnzM_, timer_drp); break;
  case ILU0inr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 26, nnzM, nnzM_, timer_drp); break;
  case ILU0dmr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 27, nnzM, nnzM_, timer_drp); break;
  case ILUpnr_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 60, bl_num, nnzM, nnzM_, timer_drp); break;
  case ILUpnc_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 61, bl_num, nnzM, nnzM_, timer_drp); break;
  case ILUpnm_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 62, bl_num, nnzM, nnzM_, timer_drp); break;
}
  if(type_pre == None_)
    for(i = 0; i < p; i++)
      R[i] = Z[i];
  else
    Preconditioner< Type >::Precond_solve(M,Z,R);
  fp_type BETA = Math_Func < Type >::norm(R);
  for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
    Q[0] = BETA;
    for(i = 1; i < m + 1; i ++)
      Q[i] = Type(0);
    for(i = 0; i < p; i++)
      V(i,0) = R[i] / BETA;
    for(int_type ncol = 0; ncol < m; ncol ++){//цикл по m
      for(i = 0; i < p; i ++)
        for(Z[i] = Type(0), j = 0; j < p; j ++)
          Z[i] += A_(i,j) * V(j,ncol);
      if(type_pre == None_)
        for(i = 0; i < p; i++)
```

```
            W[i] = Z[i];
          else
            Preconditioner< Type >::Precond_solve(M,Z,W);
          for(k = 0; k <= ncol; k++){
            for(i = 0, H(k,ncol) = Type(0); i < p; i++)
              H(k,ncol) += conj(W[i]) * V(i,k);
            for(i = 0; i < p; i++)
              W[i] -= H(k,ncol) * V(i,k);
          }
          H(ncol+1,ncol) = Math_Func< Type >::norm(W);
          for(i = 0; i < p; i++)
            V(i,ncol+1) = W[i] / H(ncol+1,ncol);
          for(k = 0; k < ncol; k ++)
            Math_Func< Type >::GivensRot_apply(H(k,ncol),
            H(k+1,ncol), CS[k], SN[k]);
          Math_Func< Type >::GivensRot_solve(H(ncol,ncol),
          H(ncol+1,ncol), CS[ncol], SN[ncol]);
          Math_Func< Type >::GivensRot_apply(H(ncol,ncol),
          H(ncol+1,ncol), CS[ncol], SN[ncol]);
          Math_Func< Type >::GivensRot_apply(Q[ncol],Q[ncol+1],
          CS[ncol], SN[ncol]);
          if(abs(Q[ncol + 1]) / b_norm <= tol)
            goto here;
        }//ncol
        ncol = m - 1;
here:
        for(i = ncol; i >= 0; i--){
          for(j = i + 1; j <= ncol; j++)
            Q[i] -= H(i,j) * Q[j];
          Q[i] /= H(i,i);
        }
        for(i = 0; i <= ncol; i++)
          for(j = 0; j < p; j++)
            U_[j] += Q[i] * V(j,i);
        Math_Func< Type >::Mat_Vec(A_,U_,Z);
        Math_Func< Type >::VecPlusScalarVec(B_,-1.0,Z,Z);
        if(type_pre == None_)
          for(i = 0; i < p; i++)
            R[i] = Z[i];
        else
          Preconditioner< Type >::Precond_solve(M,Z,R);

        BETA = Math_Func < Type >::norm(R);
        tol_ = BETA / b_norm;
        if ( tol_ <= tol  || NumIter == niter_max) break;
      }//NumIter
    }//GMRES_m
    HandlerResult CallGMRES_r(hpRealMatrix a, hpRealMatrix b,
    fp_type x, fp_type tol, int_type niter_max,int_type m, int_type
    type_pre, fp_type droptol, Interpreter* ir){
      int_type bl_num = a->m.nrows();
```

```
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SRealMatrix::vec_type vec_b(a->m.nrows());
    SRealMatrix::vec_type result(a->m.nrows());
    SRealMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< fp_type >::GMRES_m(a->m, vec_b, result, x,
    tol, niter_max, m, type_pre, droptol, bl_num);
    pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
    SRealMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
  HandlerResult CallGMRES(hpComplexMatrix a, hpComplexMatrix b,
  fp_type x, fp_type tol, int_type niter_max,int_type m, int_type
  type_pre, fp_type droptol, Interpreter* ir){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SComplexMatrix::vec_type vec_b(a->m.nrows());
    SComplexMatrix::vec_type result(a->m.nrows());
    SComplexMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< complex >::GMRES_m(a->m, vec_b, result, x,
    tol, niter_max, m, type_pre, droptol, bl_num);
    pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
    SComplexMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
}//namespace TALGAT
#endif //__GMRES_H__

/*****************************************************************
                                QMR.h
Description: метод квази-минимальных невязок.
*****************************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
#ifndef __QMR_H__
#define __QMR_H__
#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif // _MSC_VER && _MSC_VER >= 1020
#include "../CORE/Matrix.h"
#include "../CORE/Core.h"
#include "Math_Func.h"
#include "Preconditioner.h"
#include "solvers_iterative.h"
namespace TALGAT {
//QMR
  template<class Type>
```

```
void solvers_iterative< Type>::QMR(mtl::matrix< Type >::type&
A_,const mtl::dense1D< Type > &B_, mtl::dense1D< Type > &U_,
fp_type x, fp_type tol, int_type niter_max, int_type type_pre,
fp_type droptol, int_type bl_num) {
  fp_type timer,timer_drp = 0.0;
  int_type p = B_.size();
  fp_type tol_;
  int_type result = 0;
  int_type nnzM = 0 , nnzM_ = 0;
  mtl::dense1D< Type > R(p), Vt(p), Y(p), Wt(p), Z(p), V(p),
  W(p), Yt(p), Zt(p), P(p), Q(p), Pt(p), D(p), S(p), AtQ(p);
  Type
  rho,rho_,xi,gamma,gamma_,theta,theta_,eta,delta,ep,beta,a,b;
  Math_Func< Type >::Initial_Guess(U_, x);
  Math_Func< Type >::Mat_Vec(A_,U_,R);
  Math_Func< Type >::VecPlusScalarVec(B_,-1.0,R,R);
  fp_type b_norm = Math_Func< Type >::norm(R);
  mtl::matrix< Type >::type M(p,p);
  switch (type_pre){
    case LUnr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUnr_, nnzM, nnzM_, timer_drp); break;
    case LUnc_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 11, nnzM, nnzM_, timer_drp); break;
    case LUnm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 12, nnzM, nnzM_, timer_drp); break;
    case LUdm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUdm_, nnzM, nnzM_, timer_drp); break;
    case LUkm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 14, nnzM, nnzM_, timer_drp); break;
    case LUld_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 15, nnzM, nnzM_, timer_drp); break;
    case LUinr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 16, nnzM, nnzM_, timer_drp); break;
    case LUdmr_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, 19, nnzM, nnzM_, timer_drp); break;
    case LUddm_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUddm_, nnzM, nnzM_, timer_drp); break;
    case LUind_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUind_, nnzM, nnzM_, timer_drp); break;
    case LUdmd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUdmd_, nnzM, nnzM_, timer_drp); break;
    case LUnd_:Preconditioner< Type >::Precond_form_LU_(A_, M,
    droptol, LUnd_, nnzM, nnzM_, timer_drp); break;
    case ILU0nr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
    M, droptol, 20, nnzM, nnzM_, timer_drp); break;
    case ILU0nc_:Preconditioner< Type >::Precond_form_ILU0_(A_,
    M, droptol, 21, nnzM, nnzM_, timer_drp); break;
    case ILU0nm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
    M, droptol, 22, nnzM, nnzM_, timer_drp); break;
    case ILU0dm_:Preconditioner< Type >::Precond_form_ILU0_(A_,
    M, droptol, 23, nnzM, nnzM_, timer_drp); break;
```

```
  case ILU0km_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 24, nnzM, nnzM_, timer_drp); break;
  case ILU0ld_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 25, nnzM, nnzM_, timer_drp); break;
  case ILU0inr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 26, nnzM, nnzM_, timer_drp); break;
  case ILU0dmr_:Preconditioner< Type >::Precond_form_ILU0_(A_,
  M, droptol, 27, nnzM, nnzM_, timer_drp); break;
  case ILUpnr_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 60, bl_num, nnzM, nnzM_, timer_drp); break;
  case ILUpnc_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 61, bl_num, nnzM, nnzM_, timer_drp); break;
  case ILUpnm_:Preconditioner< Type >::Precond_form_ILUp_(A_,
  M, droptol, 62, bl_num, nnzM, nnzM_, timer_drp); break;
}
  for(int_type i = 0; i < p; i++)
    Wt[i] = Vt[i] = R[i];
  if(type_pre == None_)
    for(int_type i = 0; i < p; i++)
      Y[i] = Vt[i];
  else
    Preconditioner< Type >::Precond_solve_M1(M,Vt,Y);
  rho = Math_Func< Type >::norm(Y);
  if(type_pre == None_)
    for(int_type i = 0; i < p; i++)
      Z[i] = Wt[i];
  else
    Preconditioner< Type >::transPrecond_solve_M2(M,Wt,Z);
  xi = Math_Func< Type >::norm(Z);
  gamma = 1.0; eta = -1.0;
  theta = 0.0;
  for(int_type NumIter = 1; NumIter <= niter_max; NumIter ++){
    if(rho == 0.0){ result = 1; goto END;}
    if(xi == 0.0){result = 2; goto END;}
    Math_Func< Type >::VecDivScalar(Vt,rho,V);
    Math_Func< Type >::VecDivScalar(Y,rho,Y);
    Math_Func< Type >::VecDivScalar(Wt,xi,W);
    Math_Func< Type >::VecDivScalar(Z,xi,Z);
    delta = Math_Func< Type >::Vec_Vec(Z,Y);
    if(delta == 0.0){result = 3; goto END;}
    if(type_pre == None_)
      for(int_type i = 0; i < p; i++)
        Yt[i] = Y[i];
    else
      Preconditioner< Type >::Precond_solve_M2(M,Y,Yt);
    if(type_pre == None_)
      for(int_type i = 0; i < p; i++)
        Zt[i] = Z[i];
    else
      Preconditioner< Type >::transPrecond_solve_M1(M,Z,Zt);
    if(NumIter > 1){
```

```
  b = delta / ep;
  a = xi * b;
  b *= rho;
  Math_Func< Type >::VecPlusScalarVec(Yt,-a,P,P);
  Math_Func< Type >::VecPlusScalarVec(Zt,-b,Q,Q);
}
else
  for(int_type i = 0; i < p; i++){
    P[i] = Yt[i];
    Q[i] = Zt[i];
  }
Math_Func< Type >::Mat_Vec(A_,P,Pt);
ep = Math_Func< Type >::Vec_Vec(Q,Pt);
if(ep == 0.0){result = 4; goto END;}
beta = ep / delta;
if(beta == 0.0){result = 5; goto END;}
Math_Func< Type >::VecPlusScalarVec(Pt,-beta,V,Vt);
if(type_pre == None_)
  for(int_type i = 0; i < p; i++)
    Y[i] = Vt[i];
else
  Preconditioner< Type >::Precond_solve_M1(M,Vt,Y);
rho_ = rho;
rho = Math_Func< Type >::norm(Y);
Math_Func< Type >::transMat_Vec(A_,Q,AtQ);
Math_Func< Type >::VecPlusScalarVec(AtQ,-beta,W,Wt);
if(type_pre == None_)
  for(int_type i = 0; i < p; i++)
    Z[i] = Wt[i];
else
  Preconditioner< Type >::transPrecond_solve_M2(M,Wt,Z);
xi = Math_Func< Type >::norm(Z);
gamma_ = gamma;
theta_ = theta;
theta = rho / (gamma_ * beta);
gamma = 1.0 / sqrt(1.0 + theta * theta);
if(gamma == 0.0){
  result = 6;
  goto END;
}
eta = -eta * rho_ * gamma * gamma / (beta * gamma_ *
gamma_);
if(NumIter > 1){
  a = theta_ * theta_ * gamma * gamma;
  for(int_type i = 0; i < p; i++){
    D[i] = eta * P[i] + a * D[i];
    S[i] = eta * Pt[i] + a * S[i];
  }
}
else
  for(int_type i = 0; i < p; i++){
```

```
          D[i] = eta * P[i];
          S[i] = eta * Pt[i];
        }
      Math_Func< Type >::VecPlusScalarVec(U_,1.0,D,U_);
      Math_Func< Type >::VecPlusScalarVec(R,-1.0,S,R);
      tol_ = Math_Func< Type >::norm(R) / b_norm;
      if (tol_ <= tol  || NumIter == niter_max) break;
    }//NumIter
    END:;
  }//QMR
  HandlerResult CallQMR_r(hpRealMatrix a, hpRealMatrix b, fp_type
  x, fp_type tol, Interpreter* ir,  int_type niter_max, int_type
  type_pre, fp_type droptol){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SRealMatrix::vec_type vec_b(a->m.nrows());
    SRealMatrix::vec_type result(a->m.nrows());
    SRealMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< fp_type >::QMR(a->m, vec_b, result, x, tol,
    niter_max, type_pre, droptol,bl_num);
    pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
    SRealMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
  HandlerResult CallQMR(hpComplexMatrix a, hpComplexMatrix b,
  fp_type x, fp_type tol, Interpreter* ir, int_type niter_max,
  int_type type_pre, fp_type droptol){
    int_type bl_num = a->m.nrows();
    HandlerResult hr = ir->InterpretForResult();
    if(hr.GetType() == SData::DataInt) bl_num = int_type(hr);
    SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a->m.ncols()
    == b->m.ncols());
    SComplexMatrix::vec_type vec_b(a->m.nrows());
    SComplexMatrix::vec_type result(a->m.nrows());
    SComplexMatrix::m_to_vec(b->m, vec_b);
    solvers_iterative< complex >::QMR(a->m, vec_b, result, x, tol,
    niter_max, type_pre, droptol,bl_num);
    pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
    SComplexMatrix::vec_to_m(result, ret->m);
    return HandlerResult(pMatrix(ret));
  }
}//namespace TALGAT
#endif //__QMR_H__
/*********************************************************
                        solvers_direct.h
Description: прямые методы решения СЛАУ.
*********************************************************/
//(c) S. Kuksenko, 2006 - KSergP@mail.ru - Tomsk->Siberia->Russia
```

```
#ifndef __solvers_direct_H__
#define __solvers_direct_H__
#include "../CORE/Matrix.h"
#include "../mtl/mtl.h"
namespace TALGAT {
  using std::abs;
  using std::arg;
  using std::sqrt;
  using std::endl;
  using std::conj;
  using std::cout;
    template<class Type>
    class solvers_direct{
      public:
//GE (метод исключения Гаусса)
       static void GE(mtl::matrix< Type >::type& A_,mtl::dense1D<
       Type > &B_) ;
//GJE (метод Гаусса-Жордана)
       static void GJE(mtl::matrix< Type >::type& A_,mtl::dense1D<
       Type > &B_) ;
//PGJE (метод Гаусса-Жордана с частичным упорядочиванием по столб-
цам)
       static void PGJE(mtl::matrix< Type >::type&
       A_,mtl::dense1D< Type > &B_) ;
//PGE (метод Гаусса с частичным упорядочиванием по столбцам)
       static void PGE(mtl::matrix< Type >::type& A_,mtl::dense1D<
       Type > &B_) ;
//LU_Fact (LU-разложение матрицы)
       static void LU_Fact(mtl::matrix< Type >::type& A_);
//Chol (разложение Холецкого)
       static void Chol_Fact(mtl::matrix< Type >::type& A_);
//LU_Solve (решение СЛАУ с помощью LU-разложения)
       static void LU_Solve(mtl::matrix< Type >::type&
       A_,mtl::matrix< Type >::type& B_,bool lu_inverse =false);
//Chol_Solve (решение СЛАУ с помощью разложения Холецкого)
       static void Chol_Solve(mtl::matrix< Type >::type&
       A_,mtl::matrix< Type >::type& B_,bool lu_inverse =false);
//PLU (решение СЛАУ с помощью LU-разложения с частичным упорядочи-
ванием по столбцам)
       static void PLU(mtl::matrix< Type >::type& A_,mtl::dense1D<
       Type > &B_);
//LUP (решение СЛАУ с помощью LU-разложения с частичным упорядочи-
ванием по строкам)
       static void LUP(mtl::matrix< Type >::type& A_,mtl::dense1D<
       Type > &B_);
//LU_Inv (обращение матрицы с помощью LU-разложения)
       static void LU_Inv(mtl::matrix< Type >::type& A_);
//QR (QR-разложение матрицы)
       static void QR_mGS(mtl::matrix< Type >::type&
       A_,mtl::dense1D< Type > &B_);
   };//class solvers_direct
```

```
//GE
    template<class Type>
    void solvers_direct< Type >::GE(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_) {
      int_type i,j,k, p = B_.size();
      Type temp;
      mtl::dense1D< Type > AX(p);
      for(k = 0; k < p - 1; k ++)
        for(i = k + 1; i < p; i ++){
          if(abs(A_(k,k)) < 1e-20)
            SSH_THROW(Math,"[GE]: diagonal element of the matrix
            less then 1e-20");
          temp = A_(i,k) / A_(k,k);
          B_[i] -= temp * B_[k];
          for(j = k + 1; j < p; j ++)
            A_(i,j) -= temp * A_(k,j);
        }
        B_[p-1] /= A_(p-1,p-1);
        for(k = p - 2; k >= 0; k --){
          for(j = k + 1, AX[k] = (0.0,0.0); j < p; j ++)
            AX[k] += A_(k,j) * B_[j];
          B_[k] = (B_[k] - AX[k]) / A_(k,k);
        }
    }//GE
//GJE-Gaussian-Jordan elumination
    template<class Type>
    void solvers_direct< Type >::GJE(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_) {
      int_type i,j,k, p = B_.size();
      for(k = 0; k < p; k ++){
        for(j = k + 1; j < p; j ++){
          if(abs(A_(k,k)) < 1e-20)
            SSH_THROW(Math,"[GJE]: diagonal element of the matrix
            less then 1e-20");
          A_(k,j) /= A_(k,k);
        }
        B_[k] /= A_(k,k);
        for(j = k + 1; j < p; j ++)
          for(i = 0; i < p; i ++)
            if(i != k)
              A_(i,j) -= A_(i,k) * A_(k,j);
        for(i = 0; i < p; i ++)
          if(i != k)
            B_[i] -= A_(i,k) * B_[k];
      }
    }//GJE
//PGJE
    template<class Type>
    void solvers_direct< Type >::PGJE(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_) {
      int_type i,j,k, p = B_.size(),ind;
```

```
      fp_type a_max,temp_;
      mtl::dense1D< Type > AX(p);
      for(k = 0; k < p; k ++){
        //pivoting
        a_max = abs(A_(k,k)); ind = k;
        for(i = k +1; i < p; i ++){
          temp_ = abs(A_(i,k));
          if(temp_ > a_max){
            a_max = temp_;
            ind = i;
          }
        }
        if(ind != k){
          for(j = 0; j < p; j ++){
            AX[j] = A_(k,j);
            A_(k,j) = A_(ind,j);
            A_(ind,j) = AX[j];
          }
          AX[0] = B_[k];
          B_[k] = B_[ind];
          B_[ind] = AX[0];
        }//pivoting
        for(j = k + 1; j < p; j ++)
          A_(k,j) /= A_(k,k);
        B_[k] /= A_(k,k);
        for(j = k + 1; j < p; j ++)
          for(i = 0; i < p; i ++)
            if(i != k)
              A_(i,j) -= A_(i,k) * A_(k,j);
        for(i = 0; i < p; i ++)
          if(i != k)
            B_[i] -= A_(i,k) * B_[k];
      }
    }//PGJE
//PGE
    template<class Type>
    void solvers_direct< Type >::PGE(mtl::matrix< Type >::type&
    A_, mtl::dense1D< Type > &B_) {
      int_type i,j,k, p = B_.size(),ind;
      Type temp;
      fp_type a_max,temp_;
      mtl::dense1D< Type > AX(p);
      for(k = 0; k < p - 1; k ++){
        //pivoting
        a_max = abs(A_(k,k)); ind = k;
        for(i = k +1; i < p; i ++){
          temp_ = abs(A_(i,k));
          if(temp_ > a_max){
            a_max = temp_;
            ind = i;
          }
```

```
        }
        if(ind != k){
          for(j = 0; j < p; j ++){
            AX[j] = A_(k,j);
            A_(k,j) = A_(ind,j);
            A_(ind,j) = AX[j];
          }
          AX[0] = B_[k];
          B_[k] = B_[ind];
          B_[ind] = AX[0];
        }//pivoting
        for(i = k + 1; i < p; i ++){
          temp = A_(i,k) / A_(k,k);
          B_[i] -= temp * B_[k];
          for(j = k + 1; j < p; j ++)
            A_(i,j) -= temp * A_(k,j);
        }
      }
      B_[p-1] /= A_(p-1,p-1);
      for(k = p - 2; k >= 0; k --){
        for(j = k + 1, AX[k] = (0.0,0.0); j < p; j ++)
          AX[k] += A_(k,j) * B_[j];
        B_[k] = (B_[k] - AX[k]) / A_(k,k);
      }
    }//PGE
//LU_Fact
    template<class Type>
    void solvers_direct< Type >::LU_Fact(mtl::matrix< Type
    >::type& A_){
      int_type i,j,k, p = A_.nrows();
      int_type ntmp,nrow,ncol;
      for(i = 1; i < p; i ++)
        for(k = 0; k <= i - 1; k ++){
          if(abs(A_(k,k)) < 1e-20)
            SSH_THROW(Math,"[LU_FACT]: diagonal element of the ma-
            trix less then 1e-20");
          A_(i,k) /= A_(k,k);
          for(j = k + 1; j < p; j ++)
            A_(i,j) -= A_(i,k) * A_(k,j);
        }

    }//LU_Fact
//QR_MGS
    template<class Type>
    void solvers_direct< Type >::QR_mGS(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_){
      int_type i,j,k, p = A_.nrows();
      mtl::matrix< Type >::type R(p, p);
      mtl::dense1D< Type > Qt(p);
      R(0,0) = (0.0,0.0);
      for(i = 0; i < p; i ++)
```

```
      R(0,0) += A_(i,0) * A_(i,0);
    R(0,0) = sqrt(R(0,0));
    for(i = 0; i < p; i ++)
      A_(i,0) = A_(i,0) / R(0,0);
    for(j = 1; j < p; j ++){
      for(i = 0; i < p; i ++)
        Qt[i] = A_(i,j);
      for(i = 0; i <= j - 1; i ++){
        R(i,j) = (0.0,0.0);
        for(k = 0; k < p; k ++)
          R(i,j) += Qt[k] * A_(k,i);
        for(k = 0; k < p; k ++)
          Qt[k] -= R(i,j) * A_(k,i);
      }
      R(j,j) = (0.0,0.0);
      for(i = 0; i < p; i ++)
        R(j,j) += Qt[i] * Qt[i];
      R(j,j) = sqrt(R(j,j));
      for(i = 0; i < p; i ++)
        A_(i,j) = Qt[i] / R(j,j);
    }
    Math_Func< Type >::transMat_Vec(A_,B_,Qt);
    for(i = p - 1; i >= 0; i --){
      for(j = i + 1, B_[i] = Qt[i]; j < p; j ++)
        B_[i] -= R(i,j) * B_[j];
      B_[i] /= R(i,i);
    }
  }//QR_MGS_Fact
//Chol_Fact
  template<class Type>
  void solvers_direct< Type >::Chol_Fact(mtl::matrix< Type
  >::type& A_){
    int_type i,j,k, p = A_.nrows();
    for(k = 1; k < p; k ++){
      for(j = 0; j <= k - 1; j ++){
        for(i = 0; i <= j - 1; i ++)
          A_(k,j) -= A_(i,i) * A_(k,i) * A_(j,i);
        if(abs(A_(j,j)) < 1e-20)
          SSH_THROW(Math,"[Chol_Fact]: diagonal element of the
          matrix less then 1e-20");
        A_(k,j) /= A_(j,j);
      }
      for(i = 0; i <= k - 1; i ++)
        A_(k,k) -= A_(k,i) * A_(k,i) * A_(i,i);
    }
  }//Chol_Fact
//PLU
  template<class Type>
  void solvers_direct< Type >::PLU(mtl::matrix< Type >::type&
  A_,mtl::dense1D< Type > &B_){
    int_type i,j,k, p = A_.nrows(),ind;
```

```
      fp_type a_max,temp;
      mtl::dense1D< Type > Temp(p);
      for(k = 0; k < p - 1; k ++){
        //pivoting
        a_max = abs(A_(k,k)); ind = k;
        for(i = k +1; i < p; i ++){
          temp = abs(A_(i,k));
          if(temp > a_max){
            a_max = temp;
            ind = i;
          }
        }
        if(ind != k){
          for(j = 0; j < p; j ++){
            Temp[j] = A_(k,j);
            A_(k,j) = A_(ind,j);
            A_(ind,j) = Temp[j];
          }
          Temp[0] = B_[k];
          B_[k] = B_[ind];
          B_[ind] = Temp[0];
        }//pivoting
        //lu decomposition
        for(i = k + 1; i < p; i ++){
          A_(i,k) /= A_(k,k);
          for(j = k + 1; j < p; j ++)
            A_(i,j) -= A_(i,k) * A_(k,j);
        }//lu decomposition
      }
      //lu solve
      mtl::matrix< Type >::type B(1, p);
      for(i = 0; i < p; i ++)
        B(0,i) = B_[i];
      solvers_direct< Type >::LU_Solve(A_, B, false);
      for(i = 0; i < p; i ++)
        B_[i] = B(0,i);
      //lu solve
    }//PLU
//LUP
    template<class Type>
    void solvers_direct< Type >::LUP(mtl::matrix< Type >::type&
    A_,mtl::dense1D< Type > &B_){
      int_type i,j,k, p = A_.nrows(),ind,ind_;
      Type temp;
      fp_type a_max,temp_;
      mtl::dense1D< int_type > indj(p);
      for(i = 0; i < p; i ++)
        indj[i] = i;
      for(i = 0; i < p; i ++){
        //lu decomposition
        if(i > 0){
```

```
        for(k = 0; k <= i - 1; k ++){
          A_(i,indj[k]) /= A_(k,indj[k]);
          for(j = k + 1; j < p; j ++)
            A_(i,indj[j]) -= A_(i,indj[k]) * A_(k,indj[j]);
        }
      }//lu decomposition
      //pivoting
      a_max = abs(A_(i,indj[i]));
      ind = indj[i];
      for(j = i + 1; j < p; j ++){
        temp_ = abs(A_(i,indj[j]));
        if(temp_ > a_max){
          a_max = temp_;
          ind = j;
        }
      }
      if(ind != indj[i]){
        ind_ = indj[i];
        indj[i] = indj[ind];
        indj[ind] = ind_;
      }//pivoting
    }
    //lu solve
      for(i = 0; i < p; i ++)
        for(j = 0; j < i; j ++)
          B_[i] -=  A_(i,indj[j]) * B_[j];
      for(i = p - 1; i >= 0; i --) {
        for(j = i + 1; j < p; j ++)
          B_[i] -= A_(i,indj[j]) * B_[j];
        B_[i] /= A_(i,indj[i]);
      }//lu solve
    //back permutation
    i = 0;
here_:
    for(j = i; j < p; j ++)
      if(indj[j] == i){
        indj[j] = indj[i];
        temp = B_[i];
        B_[i] = B_[j];
        B_[j] = temp;
        i += 1;
        goto here_;
      }
  }//LUP
//LU_Solve
  template<class Type>
  void solvers_direct< Type >::LU_Solve(mtl::matrix< Type
  >::type& A_,mtl::matrix< Type >::type& B_,bool lu_inverse
  =false) {
    int_type i,j,k,lim, p = B_.ncols();
    if(!lu_inverse){ lim = 1; }
```

```
      else{
        lim = p;
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            if(i == j) B_(i,j) = 1.0;
            else B_(i,j) = 0.0;
          }
      }
      for(k = 0; k < lim; k++){
        for(i = 0; i < p; i ++)
          for(j = 0; j < i; j ++)
            B_(k,i) -=  A_(i,j) * B_(k,j);
        for(i = p - 1; i >= 0; i --) {
          for(j = i + 1; j < p; j ++)
            B_(k,i) -= A_(i,j) * B_(k,j);
          B_(k,i) /= A_(i,i);
        }
      }
    }//LU_Solve
//Chol_Solve
    template<class Type>
    void solvers_direct< Type >::Chol_Solve(mtl::matrix< Type
    >::type& A_,mtl::matrix< Type >::type& B_,bool lu_inverse
    =false) {
      int_type i,j,k,lim, p = B_.ncols();
      if(!lu_inverse){ lim = 1; }
      else{
        lim = p;
        for(i = 0; i < p; i ++)
          for(j = 0; j < p; j ++){
            if(i == j) B_(i,j) = 1.0;
            else B_(i,j) = 0.0;
          }
      }
      for(k = 0; k < lim; k++){
        for(i = 0; i < p; i ++)
          for(j = 0; j < i; j ++)
            B_(k,i) -=  A_(i,j) * B_(k,j);
        for(i = 0; i < p; i ++)
          B_(k,i) /= A_(i,i);
        for(i = p - 2; i >= 0; i --)
          for(j = i + 1; j < p; j ++)
            B_(k,i) -= A_(j,i) * B_(k,j);
      }
    }//Chol_Solve
//LU_Inv
    template<class Type>
    void solvers_direct< Type >::LU_Inv(mtl::matrix< Type >::type&
    A_){
      int_type p = A_.nrows();
      solvers_direct< Type >::LU_Fact(A_);
```

```
  mtl::matrix< Type >::type B_(p, p);
  solvers_direct< Type >::LU_Solve(A_, B_, true);
  for(int_type i = 0; i < p; i++)
    for(int_type j = 0; j < p; j ++)
      A_(i,j) = B_(j,i);
}//LU_Inv
HandlerResult CallLU_Inv(Interpreter*,hpComplexMatrix A) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pComplexMatrix ret( new SComplexMatrix(A->m) );
  solvers_direct< complex >::LU_Inv(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallLU_Inv_r(Interpreter*,hpRealMatrix A) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pRealMatrix ret( new SRealMatrix(A->m) );
  solvers_direct< fp_type >::LU_Inv(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallGE_r(Interpreter*, hpRealMatrix a, hpRealMa-
trix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< fp_type >::GE(a->m, vec_b);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallGE(Interpreter*, hpComplexMatrix a, hpCom-
plexMatrix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::GE(a->m, vec_b);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallPGE_r(hpRealMatrix a, Interpreter*, hpReal-
Matrix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< fp_type >::PGE(a->m, vec_b);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
```

```
HandlerResult CallPGE(hpComplexMatrix a,Interpreter*,  hpCom-
plexMatrix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::PGE(a->m, vec_b);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallPLU_r(hpRealMatrix a, hpRealMatrix
b,Interpreter*){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< fp_type >::PLU(a->m, vec_b);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallPLU(hpComplexMatrix a, hpComplexMatrix
b,Interpreter*){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::PLU(a->m, vec_b);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallLUP_r(hpRealMatrix
a,Interpreter*,Interpreter*, hpRealMatrix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< fp_type >::LUP(a->m, vec_b);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallLUP(hpComplexMatrix
a,Interpreter*,Interpreter*, hpComplexMatrix b){
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::LUP(a->m, vec_b);
```

```
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallLU_Fact(hpComplexMatrix A) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pComplexMatrix ret( new SComplexMatrix(A->m) );
  solvers_direct< complex >::LU_Fact(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallLU_Fact_r(hpRealMatrix A) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pRealMatrix ret( new SRealMatrix(A->m) );
  solvers_direct< fp_type >::LU_Fact(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallChol_Fact(hpComplexMatrix A,Interpreter*) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pComplexMatrix ret( new SComplexMatrix(A->m) );
  solvers_direct< complex >::Chol_Fact(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallChol_Fact_r(hpRealMatrix A,Interpreter*) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  pRealMatrix ret( new SRealMatrix(A->m) );
  solvers_direct< fp_type >::Chol_Fact(ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallLU_Solve(hpComplexMatrix A, hpComplexMatrix
B) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  SSH_ASSERT_ALWAYS(A->m.nrows()==B->m.ncols() && A->m.ncols()
  == B->m.ncols());
  pComplexMatrix ret( new SComplexMatrix(B->m) );
  solvers_direct< complex >::LU_Solve(A->m, ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallLU_Solve_r(hpRealMatrix A, hpRealMatrix B) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  SSH_ASSERT_ALWAYS(A->m.nrows() == B->m.ncols() && A-
  >m.ncols() == B->m.ncols());
  pRealMatrix ret( new SRealMatrix(B->m) );
  solvers_direct< fp_type >::LU_Solve(A->m, ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallChol_Solve(Interpreter*,hpComplexMatrix
A,hpComplexMatrix B,Interpreter*) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  SSH_ASSERT_ALWAYS(A->m.nrows() == B->m.ncols() && A-
  >m.ncols() == B->m.ncols());
  pComplexMatrix ret( new SComplexMatrix(B->m) );
```

```
  solvers_direct< complex >::Chol_Solve(A->m, ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallChol_Solve_r(Interpreter*,hpRealMatrix
A,hpRealMatrix B,Interpreter*) {
  SSH_ASSERT_ALWAYS(A->m.nrows() == A->m.ncols());
  SSH_ASSERT_ALWAYS(A->m.nrows()==B->m.ncols() && A->m.ncols()
  == B->m.ncols());
  pRealMatrix ret( new SRealMatrix(B->m) );
  solvers_direct< fp_type >::Chol_Solve(A->m, ret->m);
  return HandlerResult(pMatrix( ret ));
}
HandlerResult CallGJE(Interpreter*,hpComplexMatrix
a,Interpreter*,hpComplexMatrix b) {
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::GJE(a->m, vec_b);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallGJE_r(Interpreter*,hpRealMatrix
a,Interpreter*,hpRealMatrix b) {
  SSH_ASSERT_ALWAYS(a->m.nrows()==b->m.ncols() && a->m.ncols()
  == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< fp_type >::GJE(a->m, vec_b);
  pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
  SRealMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallPGJE(Interpreter*, Interpreter*, hpComplex-
Matrix a, hpComplexMatrix b) {
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SComplexMatrix::vec_type vec_b(a->m.nrows());
  SComplexMatrix::m_to_vec(b->m, vec_b);
  solvers_direct< complex >::PGJE(a->m, vec_b);
  pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
  SComplexMatrix::vec_to_m(vec_b, ret->m);
  return HandlerResult(pMatrix(ret));
}
HandlerResult CallPGJE_r(Interpreter*, Interpreter*, hpRealMa-
trix a, hpRealMatrix b) {
  SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
  >m.ncols() == b->m.ncols());
  SRealMatrix::vec_type vec_b(a->m.nrows());
  SRealMatrix::m_to_vec(b->m, vec_b);
```

```
      solvers_direct< fp_type >::PGJE(a->m, vec_b);
      pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
      SRealMatrix::vec_to_m(vec_b, ret->m);
      return HandlerResult(pMatrix(ret));
    }
    HandlerResult CallQR_MGS_r(hpRealMatrix a, Interpreter*,
    hpRealMatrix b, Interpreter*) {
      SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
      >m.ncols() == b->m.ncols());
      SRealMatrix::vec_type vec_b(a->m.nrows());
      SRealMatrix::m_to_vec(b->m, vec_b);
      solvers_direct< fp_type >::QR_mGS(a->m, vec_b);
      pRealMatrix ret( new SRealMatrix(1, a->m.nrows()) );
      SRealMatrix::vec_to_m(vec_b, ret->m);
      return HandlerResult(pMatrix(ret));
    }
    HandlerResult CallQR_MGS(hpComplexMatrix a, Interpreter*,
    hpComplexMatrix b,Interpreter*) {
      SSH_ASSERT_ALWAYS(a->m.nrows() == b->m.ncols() && a-
      >m.ncols() == b->m.ncols());
      SComplexMatrix::vec_type vec_b(a->m.nrows());
      SComplexMatrix::m_to_vec(b->m, vec_b);
      solvers_direct< complex >::QR_mGS(a->m, vec_b);
      pComplexMatrix ret( new SComplexMatrix(1, a->m.nrows()) );
      SComplexMatrix::vec_to_m(vec_b, ret->m);
      return HandlerResult(pMatrix(ret));
    }
} // namespace TALGAT
#endif //__solvers_direct_H__
```

Научное издание

Сергей Петрович Куксенко
Тальгат Рашитович Газизов

Итерационные методы решения системы линейных алгебраических уравнений с плотной матрицей

Издание подготовлено в авторской редакции.
Оригинал макет – Т.Р. Газизов.
Подписано к печати 06.09.2007 г.
Формат 60×84/16. Бумага офсетная. Гарнитура Times.
Усл. печ. л. 12,09. Тираж 250 экз. Заказ № 181.
Отпечатано на оборудовании
Редакционно-издательского отдела
Томского государственного университета
634050, г. Томск, пр. Ленина, 36.
Тел. 8+(382-2)–52-98-49

ISBN 5-94621-226-5