The Komplex Solver Package
Reference Manual

Version 1.0

Written by Michael A. Heroux

March 2000

Contents

Chapter 1

Overview of the Komplex Solver Package

1.1 Introduction

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system $Ax = b$ by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find x_r and x_i, we can form the solution to the original system as $x = x_r + i * x_i$.

1.2 Installation

Installation and use of Komplex assumes a good working knowledge of Aztec 2.1. To install Komplex 1.0, you must:

1. Follow the Aztec 2.1 installation procedure to install Aztec.

2. Compile.

   (a) Edit the files 'komplex_lib/Makefile.template' and 'komplex_app/Makefile.template'. Set the Makefile variables MPI_INCLUDE_DIR and MPI_LIB to the appropriate directories and libraries (if using MPI). For example

       MPI_INCLUDE_DIR = -I/Net/local/mpi/include MPI_LIB = -L/Net/local/mpi -lmpich

   (b) TYPE ===> set_komplex_makefiles xxxx yyyy

       where xxxx specifies a machine (SOLARIS, SUN4, SGI, SGIM4, SGI10K, I860, DEC, HP, SUNMOS, NCUBE, SP2, T3E, LINUX, or TFLOP) and yyyy specifies a messaging system (MPI, SERIAL, I860, SUNMOS, or NCUBE).

       Example: set_komplex_makefiles SUN4 MPI

       This creates 'Makefile.xxxx.yyyy' which is linked to Makefile.

   (c) TYPE ===> cd komplex_lib; make; cd ../komplex_app; make

       'gmake' works on all machines except the Cray T3E where it appears necessary to uncomment Makefile lines refering to implicit compilation.

Other applications can be compiled by switching OBJ lines in app/Makefile.

NOTE: On some machines it is necessary to switch the linker 'LD_*' when the main program is Fortran.

When porting to other machines, the following issues are the most difficult:

1. Linking between Fortran and C differs between machines. 'lib/az_aztec.h' contains macros corresponding to CFORT in the Makefiles.

2. Timing routines are different between machines. With any luck, one of the following should work: md_timer_sun.c, md_timer_generic.c, md_timer_mpi.c.

1.3 Overview of Using Komplex

1.3.1 Possible Formulations

KOMPLEX accept user linear systems in three forms:

1. The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran). See AZK_create_linsys_c2k.

2. The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines. See AZK_create_linsys_ri2k.
3. The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form. See AZK_create_linsys_g2k.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform). Note that for Form 1 above, indices can be local or global but you cannot use AZ_transform if to create local indices since AZ_transform does not support complex matrices.

All input matrices are formed as AZ_MATRIX structs (see the Aztec 2.1 User Manual). Before using Komplex, you should become very familiar with Aztec.

1.3.2 Step-by-step use of Komplex

1. **Create input matrices, initial guess and right hand side.** Create the required input matrices in one of the three forms described in Section 1.3.1. See Section 1.3.4 for an example.
2. **Select solver options.** Set Aztec input options and parameters. All Aztec parameters and options are valid for Komplex.
3. **Create linear system.** Create the Komplex form of the linear system via a call to
   - AZK_create_linsys_c2k - Convert from complex to komplex.
   - AZK_create_linsys_ri2k - Convert from real/imaginary to komplex.
   - AZK_create_linsys_g2k - Convert from general to komplex.
4. **Compute initial residual (if desired).** Call AZK_residual_norm.
5. **Create preconditioner.** Create the Komplex preconditioner via a call to AZK_create_precon. Note that the Aztec options and parameters you set will determine the preconditioner.
6. **Solve problem.** Call AZ_iterate using the matrix, initial guess, RHS, and preconditioner created in the above steps. AZ_iterate is an Aztec 2.1 function.
7. **Verify final residual (if desired).** Call AZK_residual_norm.
8. **Extract the solution.** Call one of the following:
   - AZK_extract_solution_k2c - recovers solution in complex form.

- AZK_extract_solution_k2ri - recovers solution in real/imaginary form.
- AZK_extract_solution_k2g - recovers solution in real/imaginary form (same functionality as AZK_extract_solution_k2ri).

9. **Destroy preconditioner.** Destroy the preconditioner and all associated memory.

10. **Destroy the linear system.** Destroy the matrix, initial guess/solution vector and RHS vector allocated by AZK_create_linsys_xxx.

1.3.3 Calling Komplex to Solve Multiple Related Systems

Komplex has a very flexible create/destroy framework. Although the steps in Section 1.3.2 refer to creating an entire linear system at one time, it is possible to create and destroy the matrix, initial guess and right hand side in separate steps. The matrix is created (destroyed) by calling AZK_matrix_create_xxx (AZK_matrix_destroy_xxx) where xxx refers to the type of input problem as described in Section 1.3.1. Similarly, the initial guess and RHS can also be built separately. All Komplex objects are created from completely separate storage than what the user provides. As such they remain viable until destroyed.

A few scenarios:

1. **Same matrix, new RHS**.
   - Solve 1:
     - AZK_create_linsys_xxx(matrix,x,rhs)
     - AZK_create_precon
     - AZ_iterate - solve problem
     - AZK_extract_solution_xxx
     - AZK_destroy_vector(rhs)
     - AZK_destroy_vector(solution)
   - Solve 2:
     - AZK_create_vector_xxx(new_rhs)
     - AZK_create_vector_xxx(new_x)
     - AZ_iterate - solve problem using same matrix and preconditioner
     - AZK_extract_solution_xxx
     - AZK_destroy_precon
     - AZK_destroy_linsys(matrix,x,rhs)
2. **Different matrices using the same preconditioner (where matrices may be related by being different time steps of the same problem).**
   - Solve 1:
     - AZK_create_linsys_xxx(matrix,x,rhs)
     - AZK_create_precon
     - AZ_iterate - solve problem
     - AZK_extract_solution_xxx
     - AZK_destroy_linsys(matrix,x,rhs)
   - Solve 2:

- AZK_create_linsys_xxx(new_matrix,new_x,new_rhs)
- AZ_iterate - solve problem using same matrix and preconditioner
- AZK_extract_solution_xxx
- AZK_destroy_precon
- AZK_destroy_linsys(matrix,x,rhs)

1.3.4 Example Codes

The following example code generates a simple diagonal matrix of dimension "n" where "n" is passed in as an argument. The diagonal entries are constructed in a way that exactly 20 unpreconditioned GMRES iteration should be required for convergence, independent of problem size and number of processors used.

The point of this example is to illustrate the flow of calls when using KOMPLEX. This example program can be found in the file komplex_app/simple.c.

A more elaborate sample test program can be found in the file komplex_-app/main.c. Note that it relies on service routines from SPARSKIT2. S-PARSKIT2 is available at

http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html

```
/*******************************************************************************
 * Copyright 2000, Sandia Corporation.  The United States Government retains a *
 * nonexclusive license in this software as prescribed in AL 88-1 and AL 91-7. *
 * Export of this program may require a license from the United States         *
 * Government.                                                                 *
 ******************************************************************************/
```

```
int main(int argc, char *argv[])
{
  int    proc_config[AZ_PROC_SIZE];/* Processor information.                */
  int    options[AZ_OPTIONS_SIZE]; /* Array used to select solver options.  */
  double params[AZ_PARAMS_SIZE];   /* User selected solver paramters.       */
  double status[AZ_STATUS_SIZE];   /* Information returned from AZ_solve(). */

  int    *bindx_real;              /* index and values arrays for MSR matrices */
  double *val_real, *val_imag;

  int * update;                    /* List of global eqs owned by the processor */
  double *x_real, *b_real;         /* initial guess/solution, RHS  */
  double *x_imag, *b_imag;
```

```
int    N_local;                  /* Number of equations on this node */
double residual;                 /* Used for computing residual */

double *xx_real, *xx_imag, *xx; /* Known exact solution */
int myPID, nprocs;

AZ_MATRIX *Amat_real;             /* Real matrix structure */
AZ_MATRIX  *Amat;                 /* Komplex matrix to be solved. */
AZ_PRECOND *Prec;                 /* Komplex preconditioner */
double *x, *b;                    /* Komplex Initial guess and RHS */

int i;

/******************************/
/* First executable statement */
/******************************/

MPI_Init(&argc,&argv);

/* Get number of processors and the name of this processor */
AZ_set_proc_config(proc_config,MPI_COMM_WORLD);
nprocs = proc_config[AZ_N_procs];
myPID  = proc_config[AZ_node];

printf("proc %d of %d is alive\n",myPID, nprocs);

/* Define two real diagonal matrices. Will use as real and imaginary parts */

/* Get the number of local equations from the command line */
N_local = atoi(argv[1]);

/* Need N_local+1 elements for val/bindx arrays */
val_real = malloc((N_local+1)*sizeof(double));
val_imag = malloc((N_local+1)*sizeof(double));

/* bindx_imag is not needed since real/imag have same pattern  */
bindx_real = malloc((N_local+1)*sizeof(int));

update = malloc(N_local*sizeof(int)); /* Malloc equation update list */

b_real = malloc(N_local*sizeof(double)); /* Malloc x and b arrays */
b_imag = malloc(N_local*sizeof(double));
x_real = malloc(N_local*sizeof(double));
x_imag = malloc(N_local*sizeof(double));
xx_real = malloc(N_local*sizeof(double));
xx_imag = malloc(N_local*sizeof(double));

for (i=0; i<N_local; i++)
  {
    val_real[i] = 10 + i/(N_local/10); /* Some very fake diagonals */
```

```
      val_imag[i] = 10 - i/(N_local/10); /* Should take exactly 20 GMRES steps */

      x_real[i] = 0.0;         /* Zero initial guess */
      x_imag[i] = 0.0;

      xx_real[i] = 1.0;        /* Let exact solution = 1 */
      xx_imag[i] = 0.0;

      /* Generate RHS to match exact solution */
      b_real[i] = val_real[i]*xx_real[i] - val_imag[i]*xx_imag[i];
      b_imag[i] = val_imag[i]*xx_real[i] + val_real[i]*xx_imag[i];

      /* All bindx[i] have same value since no off-diag terms */
      bindx_real[i] = N_local + 1;

      /* each processor owns equations
         myPID*N_local through myPID*N_local + N_local - 1 */
      update[i] = myPID*N_local + i;

    }

  bindx_real[N_local] = N_local+1; /* Need this last index */

  /* Register Aztec Matrix for Real Part, only imaginary values are needed*/

  Amat_real = AZ_matrix_create(N_local);

  AZ_set_MSR(Amat_real, bindx_real, val_real, NULL, N_local, update, AZ_GLOBAL);

  /* initialize AZTEC options */

  AZ_defaults(options, params);
  options[AZ_solver]  = AZ_gmres; /* Use CG with no preconditioning */
  options[AZ_precond] = AZ_none;
  options[AZ_kspace] = 21;
  options[AZ_max_iter] = 21;
  params[AZ_tol] = 1.e-14;


    /**************************************************************/
    /* Construct linear system.  Form depends on input parameters */
    /**************************************************************/

       /**************************************************************/
       /* Method 1:  Construct A, x, and b in one call.             */
       /* Useful if using A,x,b only one time. Equivalent to Method 2*/
       /**************************************************************/

     AZK_create_linsys_ri2k (x_real,  x_imag,  b_real,  b_imag,
                  options,  params, proc_config,
```

```
             Amat_real, val_imag, &x, &b, &Amat);

  /*************************************************************/
  /* Method 2:  Construct A, x, and b in separate calls.       */
  /* Useful for having more control over the construction.     */
  /* Note that the matrix must be constructed first.           */
  /*************************************************************/

 /* Uncomment these three calls and comment out the above call

 AZK_create_matrix_ri2k (options,  params, proc_config,
                        Amat_real, val_imag, &Amat);

  AZK_create_vector_ri2k(options,  params, proc_config, Amat,
                        x_real, x_imag, &x);

  AZK_create_vector_ri2k(options,  params, proc_config, Amat,
                        b_real, b_imag, &b);
 */

 /*************************************************************/
 /* Build exact solution vector.                              */
 /* Check residual of init guess and exact solution           */
 /*************************************************************/

  AZK_create_vector_ri2k(options,  params, proc_config, Amat,
                        xx_real, xx_imag, &xx);

  residual = AZK_residual_norm(x, b, options, params, proc_config, Amat);
 if (proc_config[AZ_node]==0)
  printf("\n\n\nNorm of residual using initial guess = %12.4g\n",residual);

  residual = AZK_residual_norm(xx, b, options, params, proc_config, Amat);
 if (proc_config[AZ_node]==0)
  printf("\n\n\nNorm of residual using exact solution = %12.4g\n",residual);

 /*************************************************************/
 /* Create preconditioner                                     */
 /*************************************************************/

AZK_create_precon(options,  params, proc_config, x, b, Amat, &Prec);

 /*************************************************************/
 /* Solve linear system using Aztec.                          */
 /*************************************************************/

AZ_iterate(x, b, options, params, status, proc_config, Amat, Prec, NULL);

 /*************************************************************/
 /* Extract solution.                                         */
```

```
    /*************************************************************/

     AZK_extract_solution_k2ri(options, params, proc_config, Amat, Prec, x,
                              x_real,  x_imag);
    /*************************************************************/
    /* Destroy Preconditioner.                                   */
    /*************************************************************/

      AZK_destroy_precon (options,  params, proc_config, Amat, &Prec);

    /*************************************************************/
    /* Destroy linear system.                                    */
    /*************************************************************/

      AZK_destroy_linsys (options,  params, proc_config, &x, &b, &Amat);

  if (proc_config[AZ_node]==0)
    {
      printf("True residual norm squared   = %22.16g\n",status[AZ_r]);
      printf("True scaled res norm squared = %22.16g\n",status[AZ_scaled_r]);
      printf("Computed res norm squared    = %22.16g\n",status[AZ_rec_r]);
    }

  /* Print comparison between known exact and computed solution */
  {double sum = 0.0;

  for (i=0; i<N_local; i++) sum += fabs(x_real[i]-xx_real[i]);
  for (i=0; i<N_local; i++) sum += fabs(x_imag[i]-xx_imag[i]);
  printf(
  "Processor %d:  Difference between exact and computed solution = %12.4g\n",
         proc_config[AZ_node],sum);
  }
  /*  Free memory allocated */

  free((void *) val_real );
  free((void *) bindx_real );
  free((void *) val_imag );
  free((void *) update );
  free((void *) b_real );
  free((void *) b_imag );
  free((void *) x_real );
  free((void *) x_imag );
  free((void *) xx_real );
  free((void *) xx_imag );

  MPI_Finalize();

return 0 ;
}
```

Chapter 2

Komplex File Index

2.1 Komplex File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Komplex File Documentation

3.1 azk_create_linsys.c File Reference

Creation routines for building Komplex systems.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.1.1 Functions

- void **AZK_create_linsys_c2k** (double *xc, double *bc, int *options, double *params, int *proc_config, AZ_MATRIX *Amat_complex, double **x, double **b, AZ_MATRIX **Amat_komplex)

  Create Komplex System from Complex System.

- void **AZK_create_linsys_g2k** (double *xr, double *xi, double *br, double *bi, int *options, double *params, int *proc_config, double c0r, double c0i, AZ_MATRIX *Amat_mat0, double c1r, double c1i, AZ_MATRIX *Amat_mat1, double **x, double **b, AZ_MATRIX **Amat_komplex)

  Create Komplex System from General System.

- void **AZK_create_linsys_ri2k** (double *xr, double *xi, double *br, double *bi, int *options, double *params, int *proc_config, AZ_MATRIX *Amat_real, double *val_imag, double **x, double **b, AZ_MATRIX **Amat_komplex)

  Create Komplex System from Real and Imaginary Parts.

3.1.2 Detailed Description

Creation routines for building Komplex systems.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.1.3 Function Documentation

3.1.3.1 void AZK_create_linsys_c2k (double * *xc*, double * *bc*, int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_complex*, double ** *x*, double ** *b*, AZ_MATRIX ** *Amat_komplex*)

Initial value:

Create Komplex System from Complex System.

Transforms a complex-valued system

Amat_complex * xc = bc

where double precision arrays hold the complex values of Amat_complex, xc and bc in Fortran complex format, i.e., if dimension of complex system is N then xc is of length 2*N and the first complex value is stored with the real part in xc[0] and the imaginary part in xc[1] and so on.

Parameters:

xc (In) Contains the complex initial guess/solution vector with the real/imag parts interleaved as in Fortran complex format.

bc (In) RHS in Fortran complex format.

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_complex (In) An AZ_MATRIX structure where Amat_complex->val contain the values of the complex matrix in Fortran complex format.

Parameters:

x (Out) Komplex version of initial guess and solution.

b (Out) Komplex version of RHS.

Amat_komplex (Out) Komplex version of matrix stored as an AZ_MATRIX structure.

3.1.3.2 void AZK_create_linsys_g2k (double * *xr*, double * *xi*, double * *br*, double * *bi*, int * *options*, double * *params*, int * *proc_config*, double *c0r*, double *c0i*, AZ_MATRIX * *Amat_mat0*, double *c1r*, double *c1i*, AZ_MATRIX * *Amat_mat1*, double ** *x*, double ** *b*, AZ_MATRIX ** *Amat_komplex*)

Create Komplex System from General System.

Transforms a complex-valued system

(c0r+i*c0i)*A0 +(c1r+i*c1i)*A1) * (xr+i*xi) = (br+i*bi)

to a Komplex system.

Parameters:

- ***xr*** (In) Real part of initial guess.
- ***xi*** (In) Imaginary part of initial guess.
- ***br*** (In) Real part of right hand side of linear system.
- ***bi*** (In) Imaginary part of right hand side of linear system.
- ***options*** (In) Determines specific solution method and other parameters.
- ***params*** (In) Drop tolerance and convergence tolerance info.
- ***proc_config*** (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.
- ***c0r*** (In) Real part of constant to be multiplied with first matrix.
- ***c0i*** (In) Imaginary part of constant to be multiplied with first matrix.
- ***c1r*** (In) Real part of constant to be multiplied with second matrix.
- ***c1i*** (In) Imaginary part of constant to be multiplied with second matrix.
- ***Amat_mat0*** (In) AZ_MATRIX object containing first real-valued matrix.
- ***Amat_mat1*** (In) AZ_MATRIX object containing second real-valued matrix.

Parameters:

- ***x*** (Out) Komplex version of initial guess and solution.
- ***b*** (Out) Komplex version of RHS.
- ***Amat_komplex*** (Out) Komplex version of matrix stored as an AZ_MATRIX structure.

3.1.3.3 void AZK_create_linsys_ri2k (double * *xr*, double * *xi*, double * *br*, double * *bi*, int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_real*, double * *val_imag*, double ** *x*, double ** *b*, AZ_MATRIX ** *Amat_komplex*)

Create Komplex System from Real and Imaginary Parts.

Transforms a complex-valued system

(Ar +i*Ai) * (xr + i*xi) = (br + i*bi)

where double precision arrays hold the real and imaginary parts separately. The pattern of the imaginary part matches the real part. Thus no structure for the imaginary part is passed in.

Parameters:

- ***xr*** (In) Real part of initial guess.
- ***xi*** (In) Imaginary part of initial guess.
- ***br*** (In) Real part of right hand side of linear system.
- ***bi*** (In) Imaginary part of right hand side of linear system.
- ***options*** (In) Determines specific solution method and other parameters.
- ***params*** (In) Drop tolerance and convergence tolerance info.
- ***proc_config*** (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.
- ***Amat_real*** (In) AZ_MATRIX object containing real matrix.
- ***val_imag*** (In) Double arrya containing the values ONLY for imaginary matrix.

Parameters:

- ***x*** (Out) Komplex version of initial guess and solution.
- ***b*** (Out) Komplex version of RHS.
- ***Amat_komplex*** (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

3.2 azk_create_matrix.c File Reference

Creation routines for building Komplex matrices.

`#include <stdlib.h>`

`#include <stdio.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.2.1 Functions

- void **AZK_create_matrix_c2k** (int options[ ], double params[ ], int proc_config[ ], AZ_MATRIX *Amat_complex, AZ_MATRIX **Amat_komplex)

    Create Komplex matrix from Complex matrix.

- void **AZK_create_matrix_c2k_fill_entry** (int nrow, int ncol, double *cur_complex, double *cur_komplex)
- void **AZK_create_matrix_g2k** (int options[ ], double params[ ], int proc_config[ ], double c0r, double c0i, AZ_MATRIX *Amat_mat0, double c1r, double c1i, AZ_MATRIX *Amat_mat1, AZ_MATRIX **Amat_komplex)

    Create Komplex Matrix from General Matrix.

- void **AZK_create_matrix_g2k_fill_entry** (int nrow, int ncol, double c0r, double c0i, double *mat0v, double c1r, double c1i, double *mat1v, double *komplex)
- void **AZK_create_matrix_ri2k** (int options[ ], double params[ ], int proc_config[ ], AZ_MATRIX *Amat_real, double *val_imag, AZ_MATRIX **Amat_komplex)

    Create Komplex Matrix from Real and Imaginary Parts.

- void **AZK_create_matrix_ri2k_fill_entry** (int nrow, int ncol, double *realv, double *imagv, double *komplex)

3.2.2 Detailed Description

Creation routines for building Komplex matrices.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.2.3 Function Documentation

3.2.3.1 void AZK_create_matrix_c2k (int *options*[ ], double *params*[ ], int *proc_config*[ ], AZ_MATRIX * *Amat_complex*, AZ_MATRIX ** *Amat_komplex*)

Create Komplex matrix from Complex matrix.

Transforms a complex-valued matrix where double precision array hold the complex values of Amat_complex in Fortran complex format.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_complex (In) An AZ_MATRIX structure where Amat_complex->val contain the values of the complex matrix in Fortran complex format.

Parameters:

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

3.2.3.2 void AZK_create_matrix_g2k (int *options*[ ], double *params*[ ], int *proc_config*[ ], double *c0r*, double *c0i*, AZ_MATRIX * *Amat_mat0*, double *c1r*, double *c1i*, AZ_MATRIX * *Amat_mat1*, AZ_MATRIX ** *Amat_komplex*)

Create Komplex Matrix from General Matrix.

Transforms a complex-valued Matrix

(c0r+i*c0i)*A0 +(c1r+i*c1i)*A1)

to a Komplex matrix.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

c0r (In) Real part of constant to be multiplied with first matrix.

c0i (In) Imaginary part of constant to be multiplied with first matrix.

Amat_mat0 (In) AZ_MATRIX object containing first real-valued matrix.

c1r (In) Real part of constant to be multiplied with second matrix.

c1i (In) Imaginary part of constant to be multiplied with second matrix.

Amat_mat1 (In) AZ_MATRIX object containing second real-valued matrix.

Parameters:

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

3.2.3.3 void AZK_create_matrix_ri2k (int *options*[ ], double *params*[ ], int *proc_config*[ ], AZ_MATRIX * *Amat_real*, double * *val_imag*, AZ_MATRIX ** *Amat_komplex*)

Create Komplex Matrix from Real and Imaginary Parts.

Transforms a complex-valued matrix

(Ar +i*Ai)

where double precision arrays hold the real and imaginary parts separately. The pattern of the imaginary part matches the real part. Thus no structure for the imaginary part is passed in.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_real (In) AZ_MATRIX object containing real matrix.

val_imag (In) Double arrya containing the values ONLY for imaginary matrix.

Parameters:

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

3.3 azk_create_precon.c File Reference

Creation routines for constructing a preconditioner for a Komplex matrix.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "az_aztec.h"
#include "azk_komplex.h"
```

3.3.1 Functions

- void **AZK_create_precon** (int *options, double *params, int *proc_config,double *x, double *b, AZ_MATRIX *Amat, AZ_PRECOND **Prec)

  Create a Preconditioner for a Komplex matrix.

3.3.2 Detailed Description

Creation routines for constructing a preconditioner for a Komplex matrix.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems. As such, all Aztec preconditioners are available. To learn how to set preconditioner options, please see the Aztec 2.1 User Guide.

3.3.3 Function Documentation

3.3.3.1 void AZK_create_precon (int * *options*, double * *params*, int * *proc_config*, double * *x*, double * *b*, AZ_MATRIX * *Amat*, AZ_PRECOND ** *Prec*)

Create a Preconditioner for a Komplex matrix.

Constructs a preconditioner for a Komplex matrix Amat. All preconditioning options available in Aztec are supported.

Parameters:

options (In) Determines specific preconditioner method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Parameters:

x (In/Out) Komplex version of initial guess and solution. May be modified depending on preconditioner options.

b (In/Out) Komplex version of RHS. May be modified depending on preconditioner options.

Parameters:

Amat (In) Komplex version of matrix stored as an AZ_MATRIX structure.

Parameters:

Prec (Out) Preconditioner for Amat stored as an AZ_PRECOND structure.

3.4 azk_create_vector.c File Reference

Creation routines for building Komplex vectors.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.4.1 Functions

- void **AZK_create_vector_c2k** (int *options, double *params, int *proc_config, AZ_MATRIX *Amat_komplex, double *vc, double **vk)

  Create Komplex vector from Complex vector.

- void **AZK_create_vector_g2k** (int *options, double *params, int *proc_config, AZ_MATRIX *Amat_komplex, double *vr, double *vi, double **vk)

  Create Komplex vector from Real and Imaginary Parts.

- void **AZK_create_vector_ri2k** (int *options, double *params, int *proc_config, AZ_MATRIX *Amat_komplex, double *vr, double *vi, double **vk)

  Create Komplex vector from Real and Imaginary Parts.

3.4.2 Detailed Description

Creation routines for building Komplex vectors.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.4.3 Function Documentation

3.4.3.1 void AZK_create_vector_c2k (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, double * *vc*, double ** *vk*)

Create Komplex vector from Complex vector.

Transforms a complex-valued vector vc to a real vector where vc in Fortran complex format, i.e., if dimension of complex system is N then vc is of length 2*N and the first complex value is stored with the real part in vc[0] and the imaginary part in vc[1] and so on.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (In) Komplex version of matrix stored as an AZ_-MATRIX structure.

vc (In) Contains a complex vector with the real/imag parts interleaved as in Fortran complex format.

Parameters:

vk (Out) Komplex version of vc.

3.4.3.2 void AZK_create_vector_g2k (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, double * *vr*, double * *vi*, double ** *vk*)

Create Komplex vector from Real and Imaginary Parts.

Transforms a complex vector where double precision arrays hold the real and imaginary parts separately.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

vr (In) Real part of input vector.

vi (In) Imaginary part of input vector.

Parameters:

vk (Out) Komplex version of input vector.

3.4.3.3 void AZK_create_vector_ri2k (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, double * *vr*, double * *vi*, double ** *vk*)

Create Komplex vector from Real and Imaginary Parts.

Transforms a complex vector where double precision arrays hold the real and imaginary parts separately.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure.

vr (In) Real part of input vector.

vi (In) Imaginary part of input vector.

Parameters:

vk (Out) Komplex version of input vector.

3.5 azk_destroy_linsys.c File Reference

Destruction routine for deleting Komplex systems.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "az_aztec.h"
#include "azk_komplex.h"
```

3.5.1 Functions

- void **AZK_destroy_linsys** (int *options, double *params, int *proc_config, double **x, double **b, AZ_MATRIX **Amat_komplex)

  Destroy a Komplex System.

3.5.2 Detailed Description

Destruction routine for deleting Komplex systems.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex

values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.5.3 Function Documentation

3.5.3.1 void AZK_destroy_linsys (int * *options*, double * *params*, int * *proc_config*, double ** *x*, double ** *b*, AZ_MATRIX ** *Amat_komplex*)

Destroy a Komplex System.

Destroys a komplex system created by any of the AZK_create_linsys functions. Deletes any memory allocated by creation routine.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Parameters:

x (Out) Deleted komplex version of solution. Remember to call AZK_-extract_solution_[k2c,g2k,ri2k] before calling this routine.

b (Out) Deleted komplex version of RHS.

Amat_komplex (Out) Deleted komplex version of matrix stored as an AZ_MATRIX structure.

3.6 azk_destroy_matrix.c File Reference

Destruction routine for deleting Komplex matrices.

`#include <stdlib.h>`

`#include <stdio.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.6.1 Functions

- void **AZK_destroy_matrix** (int options[ ], double params[ ], int proc_-config[ ], AZ_MATRIX **Amat_komplex)

  Destroy a Komplex Matrix.

3.6.2 Detailed Description

Destruction routine for deleting Komplex matrices.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.6.3 Function Documentation

3.6.3.1 void AZK_destroy_matrix (int *options*[ ], double *params*[ ], int *proc_config*[ ], AZ_MATRIX ** *Amat_komplex*)

Destroy a Komplex Matrix.

Destroys a komplex matrix created by any of the AZK_create_matrix functions. Deletes any memory allocated by creation routine.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Parameters:

Amat_komplex (Out) Deleted komplex version of matrix stored as an AZ_MATRIX structure.

3.7 azk_destroy_precon.c File Reference

Destruction routine for deleting a preconditioner for a Komplex matrix.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "az_aztec.h"
#include "azk_komplex.h"
```

3.7.1 Functions

- void **AZK_destroy_precon** (int *options, double *params, int *proc_config, AZ_MATRIX *Amat, AZ_PRECOND **Prec)

  Destroy a Komplex preconditioner.

3.7.2 Detailed Description

Destruction routine for deleting a preconditioner for a Komplex matrix.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems. As such, all Aztec preconditioners are available. To learn how to set preconditioner options, please see the Aztec 2.1 User Guide.

3.7.3 Function Documentation

3.7.3.1 void AZK_destroy_precon (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat*, AZ_PRECOND ** *Prec*)

Destroy a Komplex preconditioner.

Destroys a komplex preconditioner created by the AZK_create_preconditioner function. Deletes any memory allocated by creation routine.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat (In) Komplex version of matrix stored as an AZ_MATRIX structure.

Prec (Out) Deleted komplex version of preconditioner stored as an AZ_PRECOND structure.

3.8 azk_destroy_vector.c File Reference

Destruction routine for deleting Komplex vectors.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.8.1 Functions

- void **AZK_destroy_vector** (int *options, double *params, int *proc_config, AZ_MATRIX *Amat_komplex, double **vk)

  Destroy a Komplex vector.

3.8.2 Detailed Description

Destruction routine for deleting Komplex vectors.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex

values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.8.3 Function Documentation

3.8.3.1 void AZK_destroy_vector (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, double ** *vk*)

Destroy a Komplex vector.

Destroys a komplex vector created by any of the AZK_create_vector functions. Deletes any memory allocated by creation routine.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (In) Komplex version of matrix stored as an AZ_-MATRIX structure.

Parameters:

vk (Out) Deleted komplex version of a vector. Remember to call AZK_-extract_solution_[k2c,g2k,ri2k] before calling this routine.

3.9 azk_extract_solution.c File Reference

Extraction routine for getting the solution of a Komplex system.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.9.1 Functions

- void **AZK_extract_solution_k2c** (int ∗options, double ∗params, int ∗proc_config, AZ_MATRIX ∗Amat_komplex, AZ_PRECOND ∗Prec, double ∗vk, double ∗vc)

  Extract a Complex vector from a Komplex vector.

- void **AZK_extract_solution_k2g** (int ∗options, double ∗params, int ∗proc_config, AZ_MATRIX ∗Amat_komplex, AZ_PRECOND ∗Prec, double ∗vk, double ∗vr, double ∗vi)

  Extract real/imaginary parts of a complex vector from a Komplex vector.

- void **AZK_extract_solution_k2ri** (int ∗options, double ∗params, int ∗proc_config, AZ_MATRIX ∗Amat_komplex, AZ_PRECOND ∗Prec, double ∗vk, double ∗vr, double ∗vi)

  Extract real/imaginary parts of a complex vector from a Komplex vector.

3.9.2 Detailed Description

Extraction routine for getting the solution of a Komplex system.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.9.3 Function Documentation

3.9.3.1 void AZK_extract_solution_k2c (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, AZ_PRECOND * *Prec*, double * *vk*, double * *vc*)

Extract a Complex vector from a Komplex vector.

Transforms a komplex vector to a complex vector.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (In) Komplex version of matrix stored as an AZ_MATRIX structure.

Prec (In) Preconditioner for Amat stored as an AZ_PRECOND structure.

vk (In) Komplex version of vector.

Parameters:

vc (Out) Contains a complex vector with the real/imag parts interleaved as in Fortran complex format. Note that the user must allocate sufficient storage for results.

3.9.3.2 void AZK_extract_solution_k2g (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, AZ_PRECOND * *Prec*, double * *vk*, double * *vr*, double * *vi*)

Extract real/imaginary parts of a complex vector from a Komplex vector.

Transforms a komplex vector to real and imaginary parts.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (In) Komplex version of matrix stored as an AZ_MATRIX structure.

Prec (In) Preconditioner for Amat stored as an AZ_PRECOND structure.

vk (In) Komplex version of vector.

Parameters:

vc (Out) Contains a complex vector with the real/imag parts interleaved as in Fortran complex format. Note that the user must allocate sufficient storage for results.

3.9.3.3 void AZK_extract_solution_k2ri (int * *options*, double * *params*, int * *proc_config*, AZ_MATRIX * *Amat_komplex*, AZ_PRECOND * *Prec*, double * *vk*, double * *vr*, double * *vi*)

Extract real/imaginary parts of a complex vector from a Komplex vector.

Transforms a komplex vector to real and imaginary parts.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Amat_komplex (In) Komplex version of matrix stored as an AZ_MATRIX structure.

Prec (In) Preconditioner for Amat stored as an AZ_PRECOND structure.

vk (In) Komplex version of vector.

Parameters:

vc (Out) Contains a complex vector with the real/imag parts interleaved as in Fortran complex format. Note that the user must allocate sufficient storage for results.

3.10 azk_permute_ri.c File Reference

Permutation routine that checks real and imaginary parts and swaps if needed for better numerical stability.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <math.h>`

`#include "az_aztec.h"`

`#include "azk_komplex.h"`

3.10.1 Functions

- void **AZK_permute_ri** (int *options, double *params, int *proc_config, double *b, AZ_MATRIX *Amat_komplex)

    Permute a Komplex system for better numerical stability.

3.10.2 Detailed Description

Permutation routine that checks real and imaginary parts and swaps if needed for better numerical stability.

KOMPLEX is an add-on module to AZTEC that allows users to solve complex-valued linear systems.

KOMPLEX solves a complex-valued linear system Ax = b by solving an equivalent real-valued system of twice the dimension. Specifically, writing in terms of real and imaginary parts, we have

$$(A_r + i * A_i) * (x_r + i * x_i) = (b_r + i * b_i)$$

or by separating into real and imaginary equations we have

$$\begin{pmatrix} A_r & -A_i \\ A_i & A_r \end{pmatrix} \begin{pmatrix} x_r \\ x_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

which is a real-valued system of twice the size. If we find xr and xi, we can form the solution to the original system as x = xr +i*xi.

KOMPLEX accept user linear systems in three forms with either global or local index values.

1) The first form is true complex. The user passes in an MSR or VBR format matrix where the values are stored like Fortran complex numbers. Thus, the values array is of type double that is twice as long as the number of complex values. Each complex entry is stored with real part followed by imaginary part (as in Fortran).

2) The second form stores real and imaginary parts separately, but the pattern for each is identical. Thus only the values of the imaginary part are passed to the creation routines.

3) The third form accepts two real-valued matrices with no assumption about the structure of the matrices. Each matrix is multiplied by a user-supplied complex constant. This is the most general form.

Each of the above forms supports a global or local index set. By this we mean that the index values (stored in bindx) refer to the global problem indices, or the local indices (for example after calling AZ_transform).

3.10.3 Function Documentation

3.10.3.1 void AZK_permute_ri (int ∗ *options*, double ∗ *params*, int ∗ *proc_config*, double ∗ *b*, AZ_MATRIX ∗ *Amat_komplex*)

Permute a Komplex system for better numerical stability.

An alternative to the standard Komplex formulation is to permute the block rows so that the imaginary part is on the main diagonal. For example:

$$\left(\begin{array}{cc} A_i & A_r \\ A_r & -A_i \end{array}\right)\left(\begin{array}{c} x_r \\ x_i \end{array}\right)=\left(\begin{array}{c} b_i \\ b_r \end{array}\right)$$

This action may be desirable, or necessary in situations where the real part has small or zero diagonal entries. This routine looks at each real/imaginary pair and, based on a heuristic may swap the real and imaginary parts. This action does not affect the sparsity pattern, but only the mapping from the complex (or real/imaginary) mapping to the komplex mapping, and back.

Parameters:

options (In) Determines specific solution method and other parameters.

params (In) Drop tolerance and convergence tolerance info.

proc_config (In) Machine configuration. proc_config[AZ_node] is the node number. proc_config[AZ_N_procs] is the number of processors.

Parameters:

b (Out) Komplex version of RHS, possibly permuted.

Amat_komplex (Out) Komplex version of matrix stored as an AZ_-MATRIX structure, possibly permuted.

Index