```
/*
   An example showing how to call the easy_sdp() interface to CSDP.  In
   this example, we solve the problem

      max tr(C*X)
          tr(A1*X)=1
          tr(A2*X)=2
          X >= 0       (X is PSD)

   where

    C=[2 1
       1 2
           3 0 1
           0 2 0
           1 0 3
                 0
                   0]

   A1=[3 1
       1 3
           0 0 0
           0 0 0
           0 0 0
                 1
                   0]

   A2=[0 0
       0 0
           3 0 1
           0 4 0
           1 0 5
                 0
                   1]

  Notice that all of the matrices have block diagonal structure.  The first
  block is of size 2x2.  The second block is of size 3x3.  The third block
  is a diagonal block of size 2.

 */

/*
 * These standard declarations for the C library are needed.
 */

#include <stdlib.h>
#include <stdio.h>

/*
 * Include CSDP declarations so that we'll know the calling interfaces.
 */

#include "declarations.h"

/*
 * The main program.  Setup data structures with the problem data, write
 * the problem out in SDPA sparse format, and then solve the problem.
 */

int main()
{
```

```c
/*
 * The problem and solution data.
 */

struct blockmatrix C;
double *b;
struct constraintmatrix *constraints;

/*
 * Storage for the initial and final solutions.
 */

struct blockmatrix X,Z;
double *y;
double pobj,dobj;

/*
 * blockptr will be used to point to blocks in constraint matrices.
 */

struct sparseblock *blockptr;

/*
 * A return code for the call to easy_sdp().
 */

int ret;

/*
 * The first major task is to setup the C matrix and right hand side b.
 */

/*
 * First, allocate storage for the C matrix.  We have three blocks, but
 * because C starts arrays with index 0, we have to allocate space for
 * four blocks- we'll waste the 0th block.  Notice that we check to
 * make sure that the malloc succeeded.
 */

C.nblocks=3;
C.blocks=(struct blockrec *)malloc(4*sizeof(struct blockrec));
if (C.blocks == NULL)
  {
    printf("Couldn't allocate storage for C!\n");
    exit(1);
  };

/*
 * Setup the first block.
 */

C.blocks[1].blockcategory=MATRIX;
C.blocks[1].blocksize=2;
C.blocks[1].data.mat=(double *)malloc(2*2*sizeof(double));
if (C.blocks[1].data.mat == NULL)
  {
    printf("Couldn't allocate storage for C!\n");
    exit(1);
  };

/*
```

```
 * Put the entries into the first block.
 */

C.blocks[1].data.mat[ijtok(1,1,2)]=2.0;
C.blocks[1].data.mat[ijtok(1,2,2)]=1.0;
C.blocks[1].data.mat[ijtok(2,1,2)]=1.0;
C.blocks[1].data.mat[ijtok(2,2,2)]=2.0;

/*
 * Setup the second block.
 */

C.blocks[2].blockcategory=MATRIX;
C.blocks[2].blocksize=3;
C.blocks[2].data.mat=(double *)malloc(3*3*sizeof(double));
if (C.blocks[2].data.mat == NULL)
  {
    printf("Couldn't allocate storage for C!\n");
    exit(1);
  };

/*
 * Put the entries into the second block.
 */

C.blocks[2].data.mat[ijtok(1,1,3)]=3.0;
C.blocks[2].data.mat[ijtok(1,2,3)]=0.0;
C.blocks[2].data.mat[ijtok(1,3,3)]=1.0;
C.blocks[2].data.mat[ijtok(2,1,3)]=0.0;
C.blocks[2].data.mat[ijtok(2,2,3)]=2.0;
C.blocks[2].data.mat[ijtok(2,3,3)]=0.0;
C.blocks[2].data.mat[ijtok(3,1,3)]=1.0;
C.blocks[2].data.mat[ijtok(3,2,3)]=0.0;
C.blocks[2].data.mat[ijtok(3,3,3)]=3.0;

/*
 * Setup the third block.  Note that we have to allocate space for 3
 * entries because C starts array indexing with 0 rather than 1.
 */

C.blocks[3].blockcategory=DIAG;
C.blocks[3].blocksize=2;
C.blocks[3].data.vec=(double *)malloc((2+1)*sizeof(double));
if (C.blocks[3].data.vec == NULL)
  {
    printf("Couldn't allocate storage for C!\n");
    exit(1);
  };

/*
 * Put the entries into the third block.
 */

C.blocks[3].data.vec[1]=0.0;
C.blocks[3].data.vec[2]=0.0;


/*
 * Allocate storage for the right hand side, b.
 */
```

```
  b=(double *)malloc((2+1)*sizeof(double));
  if (b==NULL)
    {
      printf("Failed to allocate storage for a!\n");
      exit(1);
    };

  /*
   * Fill in the entries in b.
   */

  b[1]=1.0;
  b[2]=2.0;

  /*
   * The next major step is to setup the two constraint matrices A1 and A2.
   * Again, because C indexing starts with 0, we have to allocate space for
   * one more constraint.  constraints[0] is not used.
   */

  constraints=(struct constraintmatrix *)malloc
(                                              (2+1)*sizeof(struct constraintmatrix));
  if (constraints==NULL)
    {
      printf("Failed to allocate storage for constraints!\n");
      exit(1);
    };

  /*
   * Setup the A1 matrix.  Note that we start with block 3 of A1 and then
   * do block 1 of A1.  We do this in this order because the blocks will
   * be inserted into the linked list of A1 blocks in reverse order.
   */

  /*
   * Terminate the linked list with a NULL pointer.
   */

  constraints[1].blocks=NULL;

  /*
   * Now, we handle block 3 of A1.
   */

  /*
   * Allocate space for block 3 of A1.
   */

  blockptr=(struct sparseblock *)malloc(sizeof(struct sparseblock));
  if (blockptr==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };

  /*
   * Initialize block 3.
   */

  blockptr->blocknum=3;
  blockptr->blocksize=2;
```

```
  blockptr->constraintnum=1;
  blockptr->next=NULL;
  blockptr->nextbyblock=NULL;
  blockptr->entries=(double *) malloc((1+1)*sizeof(double));
  if (blockptr->entries==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };
  blockptr->iindices=(int *) malloc((1+1)*sizeof(int));
  if (blockptr->iindices==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };
  blockptr->jindices=(int *) malloc((1+1)*sizeof(int));
  if (blockptr->jindices==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };

  /*
   * We have 1 nonzero entry in the upper triangle of block 3 of A1.
   */

  blockptr->numentries=1;

  /*
   * The entry in the 1,1 position of block 3 of A1 is 1.0
   */

  blockptr->iindices[1]=1;
  blockptr->jindices[1]=1;
  blockptr->entries[1]=1.0;

  /*
   * Note that the entry in the 2,2 position of block 3 of A1 is 0,
   * So we don't store anything for it.
   */

  /*
   * Insert block 3 into the linked list of A1 blocks.
   */

  blockptr->next=constraints[1].blocks;
  constraints[1].blocks=blockptr;

  /*
   * Now, we handle block 1.
   */

  /*
   * Allocate space for block 1.
   */

  blockptr=(struct sparseblock *)malloc(sizeof(struct sparseblock));
  if (blockptr==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
```

```
  };

/*
 * Initialize block 1.
 */

blockptr->blocknum=1;
blockptr->blocksize=2;
blockptr->constraintnum=1;
blockptr->next=NULL;
blockptr->nextbyblock=NULL;
blockptr->entries=(double *) malloc((3+1)*sizeof(double));
if (blockptr->entries==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };
blockptr->iindices=(int *) malloc((3+1)*sizeof(int));
if (blockptr->iindices==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };
blockptr->jindices=(int *) malloc((3+1)*sizeof(int));
if (blockptr->jindices==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };

/*
 * We have 3 nonzero entries in the upper triangle of block 1 of A1.
 */

blockptr->numentries=3;

/*
 * The entry in the 1,1 position of block 1 of A1 is 3.0
 */

blockptr->iindices[1]=1;
blockptr->jindices[1]=1;
blockptr->entries[1]=3.0;

/*
 * The entry in the 1,2 position of block 1 of A1 is 1.0
 */

blockptr->iindices[2]=1;
blockptr->jindices[2]=2;
blockptr->entries[2]=1.0;

/*
 * The entry in the 2,2 position of block 1 of A1 is 3.0
 */

blockptr->iindices[3]=2;
blockptr->jindices[3]=2;
blockptr->entries[3]=3.0;

/*
```

```
 * Note that we don't have to store the 2,1 entry- this is assumed to be
 * equal to the 1,2 entry.
 */

/*
 * Insert block 1 into the linked list of A1 blocks.
 */

blockptr->next=constraints[1].blocks;
constraints[1].blocks=blockptr;

/*
 * Note that the second block of A1 is 0, so we didn't store anything for it.
 */

/*
 * Setup the A2 matrix.  This time, there are nonzero entries in block 3
 * and block 2.  We start with block 3 of A2 and then do block 1 of A2.
 */

/*
 * Terminate the linked list with a NULL pointer.
 */

constraints[2].blocks=NULL;

/*
 * First, we handle block 3 of A2.
 */

/*
 * Allocate space for block 3 of A2.
 */

blockptr=(struct sparseblock *)malloc(sizeof(struct sparseblock));
if (blockptr==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };

/*
 * Initialize block 3.
 */

blockptr->blocknum=3;
blockptr->blocksize=2;
blockptr->constraintnum=2;
blockptr->next=NULL;
blockptr->nextbyblock=NULL;
blockptr->entries=(double *) malloc((1+1)*sizeof(double));
if (blockptr->entries==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };
blockptr->iindices=(int *) malloc((1+1)*sizeof(int));
if (blockptr->iindices==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
```

```
  };
blockptr->jindices=(int *) malloc((1+1)*sizeof(int));
if (blockptr->jindices==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };

/*
 * We have 1 nonzero entry in the upper triangle of block 3 of A2.
 */

blockptr->numentries=1;


/*
 * The entry in the 2,2 position of block 3 of A2 is 1.0
 */

blockptr->iindices[1]=2;
blockptr->jindices[1]=2;
blockptr->entries[1]=1.0;

/*
 * Insert block 3 into the linked list of A2 blocks.
 */

blockptr->next=constraints[2].blocks;
constraints[2].blocks=blockptr;

/*
 * Now, we handle block 2.
 */

/*
 * Allocate space for block 2.
 */

blockptr=(struct sparseblock *)malloc(sizeof(struct sparseblock));
if (blockptr==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };

/*
 * Initialize block 2.
 */

blockptr->blocknum=2;
blockptr->blocksize=3;
blockptr->constraintnum=2;
blockptr->next=NULL;
blockptr->nextbyblock=NULL;
blockptr->entries=(double *) malloc((4+1)*sizeof(double));
if (blockptr->entries==NULL)
  {
    printf("Allocation of constraint block failed!\n");
    exit(1);
  };
blockptr->iindices=(int *) malloc((4+1)*sizeof(int));
```

```
  if (blockptr->iindices==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };
  blockptr->jindices=(int *) malloc((4+1)*sizeof(int));
  if (blockptr->jindices==NULL)
    {
      printf("Allocation of constraint block failed!\n");
      exit(1);
    };

  /*
   * We have 4 nonzero entries in the upper triangle of block 2 of A2.
   */

  blockptr->numentries=4;


  /*
   * The entry in the 1,1 position of block 2 of A2 is 3.0
   */

  blockptr->iindices[1]=1;
  blockptr->jindices[1]=1;
  blockptr->entries[1]=3.0;

  /*
   * The entry in the 2,2 position of block 2 of A2 is 4.0
   */

  blockptr->iindices[2]=2;
  blockptr->jindices[2]=2;
  blockptr->entries[2]=4.0;

  /*
   * The entry in the 3,3 position of block 2 of A2 is 5.0
   */

  blockptr->iindices[3]=3;
  blockptr->jindices[3]=3;
  blockptr->entries[3]=5.0;

  /*
   * The entry in the 1,3 position of block 2 of A2 is 1.0
   */

  blockptr->iindices[4]=1;
  blockptr->jindices[4]=3;
  blockptr->entries[4]=1.0;

  /*
   * Note that we don't store the 0 entries and entries below the diagonal!
   */

  /*
   * Insert block 2 into the linked list of A2 blocks.
   */

  blockptr->next=constraints[2].blocks;
  constraints[2].blocks=blockptr;
```

```
  /*
   * At this point, we have all of the problem data setup.
   */

  /*
   * Write the problem out in SDPA sparse format.
   */

  write_prob("prob.dat-s",7,2,C,b,constraints);

  /*
   * Create an initial solution.  This allocates space for X, y, and Z,
   * and sets initial values.
   */

  initsoln(7,2,C,b,constraints,&X,&y,&Z);

  /*
   * Solve the problem.
   */

  ret=easy_sdp(7,2,C,b,constraints,0.0,&X,&y,&Z,&pobj,&dobj);

  if (ret == 0)
    printf("The objective value is %.7e \n",(dobj+pobj)/2);
  else
    printf("SDP failed.\n");

  /*
   * Write out the problem solution.
   */

  write_sol("prob.sol",7,2,X,y,Z);

  /*
   * Free storage allocated for the problem and return.
   */

  free_prob(7,2,C,b,constraints,X,y,Z);
  exit(0);

}
```