```
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* This file defines the structure and operations for the
 * 2d vector data structure implemented in vec2d.c */

#ifndef VEC2D_DEFINED__
#define VEC2D_DEFINED__

/* Inclusions */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>


/* general utility macros */
#define True  1
#define False 0


/* This defines a simple arithmetic 2d vector structure */
typedef struct _vec2d {
  long int length1; //m dim of m x n
  long int length2; //n dim of m x n
  double** data;
} vec2d;


/* General operations on vector structures */

/*   constructor (initializes values to 0.0) */
vec2d* vec2dNew(long int m, long int n);

/*   destructor */
void vec2dDestroy(vec2d* v);

/*   write a vec2d to stdout */
int vec2dWrite(vec2d* v);

/*   write a vec2d to a file */
int vec2dWriteFile(vec2d* v, const char* outfile);

/* arithmetic operations defined on a vec2d */
int vec2dLinearSum(vec2d* x, double a, vec2d* y,   /* x = a*y + b*z */
                   double b, vec2d* z);
int vec2dScale(vec2d* x, double a);                /* x = x*a */
int vec2dCopy(vec2d* x, vec2d* y);                 /* x = y */
int vec2dConstant(vec2d* x, double a);             /* x = a */

/* scalar quantities derived from vectors */
double vec2dMin(vec2d* x);
double vec2dMax(vec2d* x);
double vec2dDot(vec2d* x, vec2d* y);
double vec2dTwoNorm(vec2d* x);
double vec2dRmsNorm(vec2d* x);
double vec2dMaxNorm(vec2d* x);

/* extra constructors */
vec2d* vec2dLinspace(double a, double b, long int m, long int n);
vec2d* vec2dRandom(long int m, long int n);
```

```
#endif
```

```
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* Inclusions */
#include "vec2d.h"


/* This file implements the operations defined in vec2d.h.
 * The functions are the basis of mathematical operations
 * for a 2d vector class/structure. The operations are done
 * by iterating over the m "vectors" with n members, so that
 * the data structure has 2 indices.  */

/* constructor (initializes values to 0.0) */
vec2d* vec2dNew(long int m, long int n) {

  long int i;

  /* if len is illegal return NULL pointer */
  if (m < 1 || n < 1) {
    fprintf(stderr, "vec2dNew error, illegal vector length = %li %li\n", m, n);
    return NULL;
  }

  /* otherwise try to allocate new vec2d object (return NULL on any failure) */
  vec2d *x = (vec2d *) malloc(sizeof(vec2d));
  if (x == NULL) return NULL;
  x->length1 = m;
  x->length2 = n;
  x->data = (double **) malloc( m*sizeof(double*) );
  for (i=0; i<m; i++) //allocating space for m vectors n in length
    x->data[i] = (double *) calloc(n, sizeof(double));
  if (x->data == NULL) {
    free(x);
    return NULL;
  }
  return x;
};


/* destructor */
void vec2dDestroy(vec2d* v) {
  free(v->data);
  v->length1 = 0;
  v->length2 = 0;
}


/* write a vec2d to stdout */
int vec2dWrite(vec2d* v) {
  long int i;
  long int j;
  /* return with failure if data array isn’t allocated */
  if (v->data == NULL) {
    fprintf(stderr, "vec2dWrite error, empty data array\n");
    return 1;
  }

  /* print data to screen and return */
  /* Note from here onward, all iterations take place over m
   * vectors and their n members (i=0:m, j=0:n) */
  for (i=0; i<v->length1; i++){
          for (j=0; j<v->length2; j++){
                  printf("  %.16g\n",v->data[i][j]);
          }
  }
  printf("\n");
  return 0;
}


/* write a vec2d to a file */
int vec2dWriteFile(vec2d* v, const char* outfile) {
  long int i;
  long int j;
  FILE *fptr = NULL;

  /* return with failure if data array isn't allocated */
  if (v->data == NULL) {
    fprintf(stderr, "vec2dWriteFile error, empty data array\n");
    return 1;
  }

  /* return with failure if 'outfile' is empty */
  if (strlen(outfile) < 1) {
    fprintf(stderr, "vec2dWriteFile error, empty outfile\n");
    return 1;
  }

  /* open output file */
  fptr = fopen(outfile, "w");
  if (fptr == NULL) {
    fprintf(stderr, "vec2dWriteFile error, unable to open %s for writing\n", outfile);
    return 1;
  }

  /* print data to file */
  for (i=0; i<v->length1; i++){
          for (j=0; j<v->length2; j++){
                  fprintf(fptr, "  %.16g\n",v->data[i][j]);
          }
          fprintf(fptr, "\n");
  }

  /* close output file and return */
  fclose(fptr);
  return 0;
}


/******** Arithmetic operations defined on a given vec2d ********/

/* x = a*y + b*z */
int vec2dLinearSum(vec2d* x, double a, vec2d* y, double b, vec2d* z) {
  long int i;
  long int j;

  /* check that array sizes match */
  if ((y->length1 != x->length1) || (z->length1 != x->length1)
                        || (y->length2 != x->length2) || (z->length2 != x->length2)) {
    fprintf(stderr,"vec2dLinearSum error, vector sizes do not match\n");
    return 1;
  }

  /* check that data is not NULL */
  if (x->data == NULL || y->data == NULL || z->data == NULL) {
    fprintf(stderr, "vec2dLinearSum error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i][j] = a*y->data[i][j] + b*z->data[i][j];
          }
  }

  return 0;
}


/*   x = x*a  (scales x by a) */
int vec2dScale(vec2d* x, double a) {
  long int i;
  long int j;

  /* check that data is not NULL */
  if (x->data == NULL) {
    fprintf(stderr, "vec2dScale error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i][j] *= a;
          }
  }

  return 0;
}


/*   x = y  (copies y into x) */
int vec2dCopy(vec2d* x, vec2d* y) {
  long int i;
  long int j;

  /* check that array sizes match */
  if ((y->length1 != x->length1) || (y->length2 != x->length2)) {
    fprintf(stderr,"vec2dCopy error, vector sizes do not match\n");
    return 1;
  }

  /* check that data is not NULL */
  if (x->data == NULL || y->data == NULL) {
    fprintf(stderr, "vec2dCopy error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i][j] = y->data[i][j];
          }
  }

  return 0;
}
```

```
/*   x = a  (sets all entries of x to the scalar a) */
int vec2dConstant(vec2d* x, double a) {
  long int i;
  long int j;

  /* check that data is not NULL */
  if (x->data == NULL) {
    fprintf(stderr, "vec2dConst error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i][j] = a;
          }
  }
  return 0;
}






/******** scalar quantities derived from vectors ********/


/* min x_i */
double vec2dMin(vec2d* x) {
  double mn;
  long int i;
  long int j;
  mn = x->data[0][0];
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mn = (mn < x->data[i][j]) ? mn : x->data[i][j];
          }
  }
  return mn;
}


/* max x_i */
double vec2dMax(vec2d* x) {
  double mx;
  long int i;
  long int j;
  mx = x->data[0][0];
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mx = (mx > x->data[i][j]) ? mx : x->data[i][j];
          }
  }
  return mx;
}


/* dot-product of x and y */
double vec2dDot(vec2d* x, vec2d* y) {
  double sum;
  long int i;
  long int j;
  /* check that array sizes match */
  if ((y->length1 != x->length1) || (y->length2 != x->length2)){
    fprintf(stderr,"vec2dDot error, vector sizes do not match\n");
    return 0.0;
  }

  /* perform operation and return */
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i][j] * y->data[i][j]);
          }
  }
  return sum;
}


/* ||x||_2 */
double vec2dTwoNorm(vec2d* x) {
  double sum;
  long int i;
  long int j;
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i][j] * x->data[i][j]);
          }
  }
  return sqrt(sum);
}


/* ||x||_RMS */
double vec2dRmsNorm(vec2d* x) {
  double sum;
  long int i;
  long int j;
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i][j] * x->data[i][j]);
          }
  }
  return sqrt(sum/(x->length1 * x->length2));
}


/* ||x||_infty */
double vec2dMaxNorm(vec2d* x) {
  double mx;
  long int i;
  long int j;
  mx = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mx = (mx > fabs(x->data[i][j])) ? mx : fabs(x->data[i][j]);
          }
  }
  return mx;
}


/******** extra constructors ********/
```

```
/* create a vector of linearly spaced data */
vec2d* vec2dLinspace(double a, double b, long int m, long int n) {
  vec2d* x;
  long int i;
  long int j;

  /* create new vector of desired length */
  x = vec2dNew(m, n);
  if (x == NULL)  return NULL;

  /* fill in entries and return */
  /* works by filling each data set with the same linspace */
  for (i=0; i<m; i++){
          for (j=0; j<n; j++){
                  x->data[i][j] = a + ((b-a)/(n*m-1))*j + i*n;
          }
  }
  return x;
}


/* create a vector of uniformly-distributed random data */
vec2d* vec2dRandom(long int m, long int n) {
  vec2d* x;
  long int i;
  long int j;

  /* create new vector of desired length */
  x = vec2dNew(m, n);
  if (x == NULL)  return NULL;

  /* fill in entries and return */
  for (i=0; i<m; i++){
          for (j=0; j<n; j++){
                  x->data[i][j] = random() / (pow(2.0,31.0) - 1.0);
          }
  }
  return x;
}
```

```
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* This file contains routines written to test
 * the vec2d 2-dimensional vector class. */

/* Inclusions */
#include <stdlib.h>
#include <stdio.h>
#include "vec2d.h"
#include "get_time.h"

/* prototype for Gram-Schmidt routine */
int GramSchmidt2d(vec2d** X, int numvectors);


/* Routine to test the vec2d "class" */
int main() {
  int i, j, ier;

  /* create some vecs of length 2x3, and set some entries */
  vec2d *a = vec2dNew(2,3);
  vec2d *b = vec2dNew(2,3);
  vec2d *c = vec2dNew(2,3);
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  b->data[i][j] = (j+1)*0.1 + i*3.0*0.1;
          }
  }
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  c->data[i][j] = (j+1) + i*3.0;
          }
  }

  /* output to screen */
  printf("writing array of zeros:\n");
  vec2dWrite(a);
  printf("writing array of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6:\n");
  vec2dWrite(b);
  printf("writing array of 1, 2, 3, 4, 5, 6:\n");
  vec2dWrite(c);

  /* verify that b has length 2x3 */
  if ((b->length1!=2) || (b->length2!=3))  printf("error: incorrect vector length\n");

  /* update a’s data and write to file */
  a->data[0][0] = 10.0;
  a->data[0][1] = 15.0;
  a->data[0][2] = 20.0;
  a->data[1][0] = 25.0;
  a->data[1][1] = 30.0;
  a->data[1][2] = 35.0;
  vec2dWriteFile(a, "a_data");
  printf("the new file ’a_data’ on disk should have entries 10, 15, 20, 25, 30, 35\n\n"
);

  /* access each entry of a and write to screen */
  printf("entries of a, one at a time: should give 10, 15, 20, 25, 30, 35\n");
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  printf("  %g\n", a->data[i][j]);
```

```
        }
}
printf("\n");

/* Test arithmetic operators */
printf("Testing vector constant, should all be -1\n");
ier = vec2dConstant(b, -1.0);
vec2dWrite(b);

printf("Testing vector copy, should be 1, 2, 3, 4, 5, 6\n");
ier = vec2dCopy(a, c);
vec2dWrite(a);

printf("Testing scalar multiply, should be 5, 10, 15, 20, 25, 30\n");
ier = vec2dScale(c, 5.0);
vec2dWrite(c);

/* create a few (5) vecs of length 2x2 */
vec2d* X[5];
for (i=0; i<5; i++){
        X[i] = vec2dNew(2,2);
}


/* fill in the vectors */
for (i=0; i<2; i++){
        for (j=0; j<2; j++){
                X[0]->data[i][j] = 1.0*i;
                X[1]->data[i][j] = 1.0*j;
                X[2]->data[i][j] = 1.0*i + 1.0*j;
                X[3]->data[i][j] = 1.0*i + 5.0*j;
                X[4]->data[i][j] = 5.0*i + 1.0*j;
        }
}


/* check the LinearSum routine */
ier = vec2dLinearSum(X[0], -2.0, X[1], 1.0, X[2]);
printf("Testing LinearSum, should be 0 -1 1 0:\n");
vec2dWrite(X[0]);

/* check the various scalar output routines */
printf("Testing TwoNorm, should be 2.4495\n");
printf("  %g\n\n", vec2dTwoNorm(b));

printf("Testing RmsNorm, should be 19.4722\n");
printf("  %g\n\n", vec2dRmsNorm(c));

printf("Testing MaxNorm, should be 1\n");
printf("  %g\n\n", vec2dMaxNorm(b));

printf("Testing Min, should be 1\n");
printf("  %g\n\n", vec2dMin(a));

printf("Testing Max, should be 30\n");
printf("  %g\n\n", vec2dMax(c));

printf("Testing Dot, should be 455\n");
printf("  %g\n\n", vec2dDot(a,c));

printf("Testing Linspace, should be 0 1 2 3 4 5 6 7 8 9 10 11\n");
vec2d *d = vec2dLinspace(0.0, 11.0, 4, 3);
vec2dWrite(d);
```

```
  printf("Testing Random\n");
  vec2d *f = vec2dRandom(5, 2);
  vec2dWrite(f);



  /*** performance/validity tests (Gram-Schmidt process) ***/

  printf("Running GramSchmidt2d process\n");

  vec2d** Y1 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y1[i] = vec2dRandom(10000, 1000);
  double stime = get_time();
  if (GramSchmidt2d(Y1, 5))
    printf("GramSchmidt2d returned error\n");
  double ftime = get_time();
  double rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y1[i],Y1[j]));
  printf("\n");
  printf("dimensions: (10000, 1000)\n");
  printf("testing time: %g\n", rtime);

  vec2d** Y2 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y2[i] = vec2dRandom(1000, 10000);
  stime = get_time();
  if (GramSchmidt2d(Y2, 5))
    printf("GramSchmidt2d returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y2[i],Y2[j]));
  printf("\n");
  printf("dimensions: (1000, 10000)\n");
  printf("testing time: %g\n", rtime);

  vec2d** Y3 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y3[i] = vec2dRandom(100, 100000);
  stime = get_time();
  if (GramSchmidt2d(Y3, 5))
    printf("GramSchmidt2d returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y3[i],Y3[j]));
  printf("\n");
  printf("dimensions: (100, 100000)\n");
  printf("testing time: %g\n", rtime);

  vec2d** Y4 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y4[i] = vec2dRandom(10, 1000000);
  stime = get_time();
  if (GramSchmidt2d(Y4, 5))
    printf("GramSchmidt2d returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y4[i],Y4[j]));
  printf("\n");
  printf("dimensions: (10, 1000000)\n");
  printf("testing time: %g\n", rtime);

  vec2d** Y5 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y5[i] = vec2dRandom(100000, 100);
  stime = get_time();
  if (GramSchmidt2d(Y5, 5))
    printf("GramSchmidt2d returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y5[i],Y5[j]));
  printf("\n");
  printf("dimensions: (100000, 100)\n");
  printf("testing time: %g\n", rtime);

  vec2d** Y6 = (vec2d**) malloc(5 * sizeof(vec2d*));
  for (i=0; i<5; i++)
    Y6[i] = vec2dRandom(1000000, 10);
  stime = get_time();
  if (GramSchmidt2d(Y6, 5))
    printf("GramSchmidt2d returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2dDot(Y6[i],Y6[j]));
  printf("\n");
  printf("dimensions: (1000000, 10)\n");
  printf("testing time: %g\n", rtime);

  /* clean up */
  vec2dDestroy(a);
  vec2dDestroy(b);
  vec2dDestroy(c);
  vec2dDestroy(d);
  vec2dDestroy(f);
  for (i=0; i<5; i++)
    vec2dDestroy(X[i]);
  for (i=0; i<5; i++)
    vec2dDestroy(Y1[i]);
  free(Y1);
  for (i=0; i<5; i++)
    vec2dDestroy(Y2[i]);
  free(Y2);
  for (i=0; i<5; i++)
    vec2dDestroy(Y3[i]);
  free(Y3);
  for (i=0; i<5; i++)
    vec2dDestroy(Y4[i]);
  free(Y4);
  for (i=0; i<5; i++)
    vec2dDestroy(Y5[i]);
  free(Y5);
  for (i=0; i<5; i++)
    vec2dDestroy(Y6[i]);
  free(Y6);

  return 0;
} /* end main */
```

```
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* This file defines the structure and operations for the
 * 2d row major-ordered vector data structure implemented
 * in vec2d_b.c */

#ifndef VEC2D_B_DEFINED__
#define VEC2D_B_DEFINED__

/* Inclusions */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>


/* general utility macros */
#define True  1
#define False 0


/* This defines a simple arithmetic 2d, row major vector structure */
typedef struct _vec2d_b {
  long int length1; //m dim of m x n
  long int length2; //n dim of m x n
  double* data;
} vec2d_b;


/* General operations on vector structures */

/*   constructor (initializes values to 0.0) */
vec2d_b* vec2d_bNew(long int m, long int n);

/*   destructor */
void vec2d_bDestroy(vec2d_b* v);

/*   write a vec2d to stdout */
int vec2d_bWrite(vec2d_b* v);

/*   write a vec2d to a file */
int vec2d_bWriteFile(vec2d_b* v, const char* outfile);

/* arithmetic operations defined on a vec2d */
int vec2d_bLinearSum(vec2d_b* x, double a, vec2d_b* y,   /* x = a*y + b*z */
                   double b, vec2d_b* z);
int vec2d_bScale(vec2d_b* x, double a);                /* x = x*a */
int vec2d_bCopy(vec2d_b* x, vec2d_b* y);                 /* x = y */
int vec2d_bConstant(vec2d_b* x, double a);             /* x = a */

/* scalar quantities derived from vectors */
double vec2d_bMin(vec2d_b* x);
double vec2d_bMax(vec2d_b* x);
double vec2d_bDot(vec2d_b* x, vec2d_b* y);
double vec2d_bTwoNorm(vec2d_b* x);
double vec2d_bRmsNorm(vec2d_b* x);
double vec2d_bMaxNorm(vec2d_b* x);

/* extra constructors */
vec2d_b* vec2d_bLinspace(double a, double b, long int m, long int n);
vec2d_b* vec2d_bRandom(long int m, long int n);
```

```
#endif
```

```
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* Inclusions */
#include "vec2d_b.h"


/* This file implements the operations defined in vec2d_b.h.
 * The functions are the basis of mathematical operations
 * for a 2d row major-ordered vector class/structure. The data
 * is stored as a 1d array of doubles, so that the data structure
 * has 1 index and operations are done by iterating over
 * the m "vectors" with n members, so that the (i,j) entry is
 * stored in index [i*n + j] */

/* constructor (initializes values to 0.0) */
vec2d_b* vec2d_bNew(long int m, long int n) {

  long int i;

  /* if len is illegal return NULL pointer */
  if (m < 1 || n < 1) {
    fprintf(stderr, "vec2d_bNew error, illegal vector length = %li %li\n", m, n);
    return NULL;
  }

  /* otherwise try to allocate new vec2d_b object (return NULL on any failure) */
  vec2d_b *x = (vec2d_b *) malloc(sizeof(vec2d_b));
  if (x == NULL) return NULL;
  x->length1 = m;
  x->length2 = n;
  x->data = (double *) calloc(m*n, sizeof(double));
  if (x->data == NULL) {
    free(x);
    return NULL;
  }
  return x;
};


/* destructor */
void vec2d_bDestroy(vec2d_b* v) {
  free(v->data);
  v->length1 = 0;
  v->length2 = 0;
}


/* write a vec2d_b to stdout */
int vec2d_bWrite(vec2d_b* v) {
  long int i;
  long int j;
  /* return with failure if data array isn’t allocated */
  if (v->data == NULL) {
    fprintf(stderr, "vec2d_bWrite error, empty data array\n");
    return 1;
  }

  /* print data to screen and return */
  /* Note from here onward, all iterations take place over m
   * vectors and their n members (i=0:m, j=0:n) */
  for (i=0; i<v->length1; i++){
          for (j=0; j<v->length2; j++){
                  printf("  %.16g\n",v->data[i*v->length2 + j]);
          }
  }
  printf("\n");
  return 0;
}


/* write a vec2d_b to a file */
int vec2d_bWriteFile(vec2d_b* v, const char* outfile) {
  long int i;
  long int j;
  FILE *fptr = NULL;

  /* return with failure if data array isn't allocated */
  if (v->data == NULL) {
    fprintf(stderr, "vec2d_bWriteFile error, empty data array\n");
    return 1;
  }

  /* return with failure if 'outfile' is empty */
  if (strlen(outfile) < 1) {
    fprintf(stderr, "vec2d_bWriteFile error, empty outfile\n");
    return 1;
  }

  /* open output file */
  fptr = fopen(outfile, "w");
  if (fptr == NULL) {
    fprintf(stderr, "vec2d_bWriteFile error, unable to open %s for writing\n", outfile)
;
    return 1;
  }

  /* print data to file */
  for (i=0; i<v->length1; i++){
          for (j=0; j<v->length2; j++){
                  fprintf(fptr, "  %.16g\n",v->data[i*v->length2 + j]);
          }
          fprintf(fptr, "\n");
  }

  /* close output file and return */
  fclose(fptr);
  return 0;
}


/******** Arithmetic operations defined on a given vec2d_b ********/

/* x = a*y + b*z */
int vec2d_bLinearSum(vec2d_b* x, double a, vec2d_b* y, double b, vec2d_b* z) {
  long int i;
  long int j;

  /* check that array sizes match */
  if ((y->length1 != x->length1) || (z->length1 != x->length1)
                        || (y->length2 != x->length2) || (z->length2 != x->length2)) {
    fprintf(stderr,"vec2d_bLinearSum error, vector sizes do not match\n");
    return 1;
  }

  /* check that data is not NULL */
  if (x->data == NULL || y->data == NULL || z->data == NULL) {
    fprintf(stderr, "vec2d_bLinearSum error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i*x->length2 + j] = a*y->data[i*y->length2 + j] + b*z->data[i
*x->length2 + j];
          }
  }

  return 0;
}


/*   x = x*a  (scales x by a) */
int vec2d_bScale(vec2d_b* x, double a) {
  long int i;
  long int j;

  /* check that data is not NULL */
  if (x->data == NULL) {
    fprintf(stderr, "vec2d_bScale error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i*x->length2 + j] *= a;
          }
  }

  return 0;
}


/*   x = y  (copies y into x) */
int vec2d_bCopy(vec2d_b* x, vec2d_b* y) {
  long int i;
  long int j;

  /* check that array sizes match */
  if ((y->length1 != x->length1) || (y->length2 != x->length2)) {
    fprintf(stderr,"vec2d_bCopy error, vector sizes do not match\n");
    return 1;
  }

  /* check that data is not NULL */
  if (x->data == NULL || y->data == NULL) {
    fprintf(stderr, "vec2d_bCopy error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i*x->length2 + j] = y->data[i*y->length2 + j];
          }
  }

  return 0;
```

```
}


/*   x = a  (sets all entries of x to the scalar a) */
int vec2d_bConstant(vec2d_b* x, double a) {
  long int i;
  long int j;

  /* check that data is not NULL */
  if (x->data == NULL) {
    fprintf(stderr, "vec2d_bConst error: empty data array\n");
    return 1;
  }

  /* perform operation and return */
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  x->data[i*x->length2 + j] = a;
          }
  }
  return 0;
}







/******** scalar quantities derived from vectors ********/


/* min x_i */
double vec2d_bMin(vec2d_b* x) {
  double mn;
  long int i;
  long int j;
  mn = x->data[0];
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mn = (mn < x->data[i*x->length2 + j]) ? mn : x->data[i*x->length2 + j
];
          }
  }
  return mn;
}


/* max x_i */
double vec2d_bMax(vec2d_b* x) {
  double mx;
  long int i;
  long int j;
  mx = x->data[0];
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mx = (mx > x->data[i*x->length2 + j]) ? mx : x->data[i*x->length2 + j
];
          }
  }
  return mx;
}


/* dot-product of x and y */
```

```
double vec2d_bDot(vec2d_b* x, vec2d_b* y) {
  double sum;
  long int i;
  long int j;

  /* check that array sizes match */
  if ((y->length1 != x->length1) || (y->length2 != x->length2)){
    fprintf(stderr,"vec2d_bDot error, vector sizes do not match\n");
    return 0.0;
  }

  /* perform operation and return */
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i*x->length2 + j] * y->data[i*y->length2 + j]);
          }
  }
  return sum;
}


/* ||x||_2 */
double vec2d_bTwoNorm(vec2d_b* x) {
  double sum;
  long int i;
  long int j;
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i*x->length2 + j] * x->data[i*x->length2 + j]);
          }
  }
  return sqrt(sum);
}


/* ||x||_RMS */
double vec2d_bRmsNorm(vec2d_b* x) {
  double sum;
  long int i;
  long int j;
  sum = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  sum += (x->data[i*x->length2 + j] * x->data[i*x->length2 + j]);
          }
  }
  return sqrt(sum/(x->length1 * x->length2));
}


/* ||x||_infty */
double vec2d_bMaxNorm(vec2d_b* x) {
  double mx;
  long int i;
  long int j;
  mx = 0.0;
  for (i=0; i<x->length1; i++){
          for (j=0; j<x->length2; j++){
                  mx = (mx > fabs(x->data[i*x->length2 + j])) ? mx : fabs(x->data[i*x->
length2 + j]);
          }
  }
  return mx;
}


/******** extra constructors ********/

/* create a vector of linearly spaced data */
vec2d_b* vec2d_bLinspace(double a, double b, long int m, long int n) {
  vec2d_b* x;
  long int i;
  long int j;

  /* create new vector of desired length */
  x = vec2d_bNew(m, n);
  if (x == NULL)  return NULL;

  /* fill in entries and return */
  /* works by filling each data set with the same linspace */
  for (i=0; i<m; i++){
          for (j=0; j<n; j++){
                  x->data[i*x->length2 + j] = a + ((b-a)/(n*m-1))*j + i*n;
          }
  }
  return x;
}


/* create a vector of uniformly-distributed random data */
vec2d_b* vec2d_bRandom(long int m, long int n) {
  vec2d_b* x;
  long int i;
  long int j;

  /* create new vector of desired length */
  x = vec2d_bNew(m, n);
  if (x == NULL)  return NULL;

  /* fill in entries and return */
  for (i=0; i<m; i++){
          for (j=0; j<n; j++){
                  x->data[i*x->length2 + j] = random() / (pow(2.0,31.0) - 1.0);
          }
  }
  return x;
}
```

```c
/* Nicole Deyerl
   SMU Mathematics
   Math 4370/6370
   3 March 2017 */

/* This file contains routines written to test
 * the vec2d_b "2-dimensional", row major-ordered
 * vector class. */

/* Inclusions */
#include <stdlib.h>
#include <stdio.h>
#include "vec2d_b.h"
#include "get_time.h"

/* prototype for Gram-Schmidt routine */
int GramSchmidt2d_b(vec2d_b** X, int numvectors);


/* Routine to test the vec2d_b "class" */
int main() {
  int i, j, ier;

  /* create some vecs of length 2x3, and set some entries */
  vec2d_b *a = vec2d_bNew(2,3);
  vec2d_b *b = vec2d_bNew(2,3);
  vec2d_b *c = vec2d_bNew(2,3);
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  b->data[i*b->length2 + j] = (j+1)*0.1 + i*3.0*0.1;
          }
  }
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  c->data[i*c->length2 + j] = (j+1) + i*3.0;
          }
  }

  /* output to screen */
  printf("writing array of zeros:\n");
  vec2d_bWrite(a);
  printf("writing array of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6:\n");
  vec2d_bWrite(b);
  printf("writing array of 1, 2, 3, 4, 5, 6:\n");
  vec2d_bWrite(c);

  /* verify that b has length 2x3 */
  if ((b->length1!=2) || (b->length2!=3))  printf("error: incorrect vector length\n");

  /* update a’s data and write to file */
  a->data[0] = 10.0;
  a->data[1] = 15.0;
  a->data[2] = 20.0;
  a->data[3] = 25.0;
  a->data[4] = 30.0;
  a->data[5] = 35.0;
  vec2d_bWriteFile(a, "a_data");
  printf("the new file ’a_data’ on disk should have entries 10, 15, 20, 25, 30, 35\n\n"
);

  /* access each entry of a and write to screen */
  printf("entries of a, one at a time: should give 10, 15, 20, 25, 30, 35\n");
  for (i=0; i<2; i++){
          for (j=0; j<3; j++){
                  printf("  %g\n", a->data[i*a->length2 + j]);
          }
  }
  printf("\n");

  /* Test arithmetic operators */
  printf("Testing vector constant, should all be -1\n");
  ier = vec2d_bConstant(b, -1.0);
  vec2d_bWrite(b);

  printf("Testing vector copy, should be 1, 2, 3, 4, 5, 6\n");
  ier = vec2d_bCopy(a, c);
  vec2d_bWrite(a);

  printf("Testing scalar multiply, should be 5, 10, 15, 20, 25, 30\n");
  ier = vec2d_bScale(c, 5.0);
  vec2d_bWrite(c);

  /* create a few (5) vecs of length 2x2 */
  vec2d_b* X[5];
  for (i=0; i<5; i++){
          X[i] = vec2d_bNew(2,2);
  }


  /* fill in the vectors */
  for (i=0; i<2; i++){
          for (j=0; j<2; j++){
                  X[0]->data[i*X[0]->length2 + j] = 1.0*i;
                  X[1]->data[i*X[1]->length2 + j] = 1.0*j;
                  X[2]->data[i*X[2]->length2 + j] = 1.0*i + 1.0*j;
                  X[3]->data[i*X[3]->length2 + j] = 1.0*i + 5.0*j;
                  X[4]->data[i*X[4]->length2 + j] = 5.0*i + 1.0*j;
          }
  }


  /* check the LinearSum routine */
  ier = vec2d_bLinearSum(X[0], -2.0, X[1], 1.0, X[2]);
  printf("Testing LinearSum, should be 0 -1 1 0:\n");
  vec2d_bWrite(X[0]);

  /* check the various scalar output routines */
  printf("Testing TwoNorm, should be 2.4495\n");
  printf("  %g\n\n", vec2d_bTwoNorm(b));

  printf("Testing RmsNorm, should be 19.4722\n");
  printf("  %g\n\n", vec2d_bRmsNorm(c));

  printf("Testing MaxNorm, should be 1\n");
  printf("  %g\n\n", vec2d_bMaxNorm(b));

  printf("Testing Min, should be 1\n");
  printf("  %g\n\n", vec2d_bMin(a));

  printf("Testing Max, should be 30\n");
  printf("  %g\n\n", vec2d_bMax(c));

  printf("Testing Dot, should be 455\n");
  printf("  %g\n\n", vec2d_bDot(a,c));

  printf("Testing Linspace, should be 0 1 2 3 4 5 6 7 8 9 10 11\n");
  vec2d_b *d = vec2d_bLinspace(0.0, 11.0, 4, 3);
  vec2d_bWrite(d);
```

```
printf("Testing Random\n");
vec2d_b *f = vec2d_bRandom(5, 2);
vec2d_bWrite(f);



/*** performance/validity tests (Gram-Schmidt process) ***/

printf("Running GramSchmidt2d_b process\n");

vec2d_b** Y1 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
for (i=0; i<5; i++)
  Y1[i] = vec2d_bRandom(10000, 1000);
double stime = get_time();
if (GramSchmidt2d_b(Y1, 5))
  printf("GramSchmidt2d_b returned error\n");
double ftime = get_time();
double rtime = ftime-stime;
printf("Resulting vectors should be orthonormal:\n");
for (i=0; i<5; i++)
  for (j=i; j<5; j++)
    printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y1[i],Y1[j]));
printf("\n");
printf("dimensions: (10000, 1000)\n");
printf("testing time: %g\n", rtime);

vec2d_b** Y2 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
for (i=0; i<5; i++)
  Y2[i] = vec2d_bRandom(1000, 10000);
stime = get_time();
if (GramSchmidt2d_b(Y2, 5))
  printf("GramSchmidt2d_b returned error\n");
ftime = get_time();
rtime = ftime-stime;
printf("Resulting vectors should be orthonormal:\n");
for (i=0; i<5; i++)
  for (j=i; j<5; j++)
    printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y2[i],Y2[j]));
printf("\n");
printf("dimensions: (1000, 10000)\n");
printf("testing time: %g\n", rtime);

vec2d_b** Y3 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
for (i=0; i<5; i++)
  Y3[i] = vec2d_bRandom(100, 100000);
stime = get_time();
if (GramSchmidt2d_b(Y3, 5))
  printf("GramSchmidt2d_b returned error\n");
ftime = get_time();
rtime = ftime-stime;
printf("Resulting vectors should be orthonormal:\n");
for (i=0; i<5; i++)
  for (j=i; j<5; j++)
    printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y3[i],Y3[j]));
printf("\n");
printf("dimensions: (100, 100000)\n");
printf("testing time: %g\n", rtime);

vec2d_b** Y4 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
for (i=0; i<5; i++)
  Y4[i] = vec2d_bRandom(10, 1000000);
stime = get_time();
if (GramSchmidt2d_b(Y4, 5))
```

```
    printf("GramSchmidt2d_b returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y4[i],Y4[j]));
  printf("\n");
  printf("dimensions: (10, 1000000)\n");
  printf("testing time: %g\n", rtime);

  vec2d_b** Y5 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
  for (i=0; i<5; i++)
    Y5[i] = vec2d_bRandom(100000, 100);
  stime = get_time();
  if (GramSchmidt2d_b(Y5, 5))
    printf("GramSchmidt2d_b returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y5[i],Y5[j]));
  printf("\n");
  printf("dimensions: (100000, 100)\n");
  printf("testing time: %g\n", rtime);

  vec2d_b** Y6 = (vec2d_b**) malloc(5 * sizeof(vec2d_b*));
  for (i=0; i<5; i++)
    Y6[i] = vec2d_bRandom(1000000, 10);
  stime = get_time();
  if (GramSchmidt2d_b(Y6, 5))
    printf("GramSchmidt2d_b returned error\n");
  ftime = get_time();
  rtime = ftime-stime;
  printf("Resulting vectors should be orthonormal:\n");
  for (i=0; i<5; i++)
    for (j=i; j<5; j++)
      printf("  <Y1[%i],Y1[%i]> = %g\n", i, j, vec2d_bDot(Y6[i],Y6[j]));
  printf("\n");
  printf("dimensions: (1000000, 10)\n");
  printf("testing time: %g\n", rtime);

  /* clean up */
  vec2d_bDestroy(a);
  vec2d_bDestroy(b);
  vec2d_bDestroy(c);
  vec2d_bDestroy(d);
  vec2d_bDestroy(f);
  for (i=0; i<5; i++)
    vec2d_bDestroy(X[i]);
  for (i=0; i<5; i++)
    vec2d_bDestroy(Y1[i]);
  free(Y1);
  for (i=0; i<5; i++)
    vec2d_bDestroy(Y2[i]);
  free(Y2);
  for (i=0; i<5; i++)
    vec2d_bDestroy(Y3[i]);
  free(Y3);
  for (i=0; i<5; i++)
    vec2d_bDestroy(Y4[i]);
  free(Y4);
  for (i=0; i<5; i++)
```

```
    vec2d_bDestroy(Y5[i]);
  free(Y5);
  for (i=0; i<5; i++)
    vec2d_bDestroy(Y6[i]);
  free(Y6);

  return 0;
} /* end main */
```