Week 8 solutions

CodeJudge exercises

BLAS level 1: dscal

```
#include <stdlib.h>

/* DSCAL (scale array)                               */
void dscal_(
    const int * n,          /* length of array      */
    const double * a,       /* scalar a             */
    double * x,             /* array x              */
    const int * incx        /* array x, stride      */
);

/* Scale the k'th row of a two-dimensional row-major array */
int scale_row(double alpha, double **A, int m, int n, int k) {
    int incx = 1;
    if ((A==NULL) || (m<0) || (n<0) || (k<0) || (k>=m)) return -1;
    dscal_(&n,&alpha,A[k],&incx);
    return 0;
}

/* Scale the k'th column of a two-dimensional row-major array */
int scale_column(double alpha, double **A, int m, int n, int k) {
    int incx = n;
    if ((A==NULL) || (m<0) || (n<0) || (k<0) || (k>=n)) return -1;
    dscal_(&m,&alpha,*A+k,&incx);
    return 0;
}

/* Scale the diagonal elements of a square two-dim. row-major array */
int scale_diag(double alpha, double **A, int m, int n) {
    int incx = n+1;
    if ((A==NULL) || (n!=m) || (m<0) || (n<0)) return -1;
    dscal_(&n,&alpha,*A,&incx);
    return 0;
}
```

BLAS level 1: daxpy

```
#include <stdlib.h>
#include <math.h>

/* DAXPY (double a x plus y)                            */
void daxpy_(
    const int * n,        /* length of arrays x and y */
    const double * a,     /* scalar a                 */
    const double * x,     /* array x                  */
    const int * incx,     /* array x, stride          */
    double * y,           /* array y                  */
    const int * incy      /* array y, stride          */
);

/** Adds alpha times row i to row j
 *
 * The input 'A' represents a two-dim. row-major array of size m-by-n
 */
int add_row(double alpha, double **A, int m, int n, int i, int j) {
    int inc=1;
    if (A==NULL || A[0]==NULL || m <= 0 || n <= 0) return -1;
    if (i == j || i < 0 || j < 0 || i >= m || j >= m) return -1;
    daxpy_(&n, &alpha, A[i], &inc, A[j], &inc);
    return 0;
}

/** Adds alpha times column i to column j
 *
 * The input 'A' represents a two-dim. row-major array of size m-by-n
 */
int add_column(double alpha, double **A, int m, int n, int i, int j) {
    int inc=n;
    if (A==NULL || A[0]==NULL || m <= 0 || n <= 0) return -1;
    if (i == j || i < 0 || j < 0 || i >= n || j >= n) return -1;
    daxpy_(&m, &alpha, A[0]+i, &inc, A[0]+j, &inc);
    return 0;
}
```

BLAS level 2: `dtrsv`

```
#include <stdlib.h>
#include <math.h>

/** DTRSV
 *  BLAS level 2 routine for forward/back substitution
 *  Documentation: http://www.netlib.org/blas/#_level_2
 */
void dtrsv_(
  const char * uplo,  /* upper 'U' or lower 'L'               */
  const char * trans, /* not trans. 'N' or trans. 'T'         */
  const char * diag,  /* not unit diag. 'N' or unit diag. 'U' */
  const int * n,      /* dimension                            */
  const double * A,   /* column-major matrix of order n       */
  const int * lda,    /* leading dimension of A               */
  double * x,           /* right-hand side                          */
  const int * incx      /* stride for array x                       */
);


/** Solves system of equations L*U*x = b where
 *  L is unit lower triangular and U is upper triangular.
 *  The matrices L and U must be stored in a single array M
 *  of size n-by-n. On exit, the array b is overwritten by
 *  the solution x.
 *
 *  If successful, the function returns zero, and in case
 *  of an error, the return value is -1.
 *
 *  Inputs:
 *    n   the size of the array M
 *    M   dynamically allocated two-dimensional array of size n-by-n
 *    b   one-dimensional array of length n
 */
int lu_solve(int n, double ** M, double * b) {

  int incx=1;
  char uplo, trans, diag;

  /* Check inputs */
  if ((M==NULL)||(M[0]==NULL)||(b==NULL)||(n<=0)) return -1;

  /* Check for singularity */
  for (int i=0;i<n;i++) {
      // Minimal check; room for improvements
      if (!isnormal(M[i][i])) return -1;
  }

  /* Solve L*z = b:
  Account for row-major storage:
  If we interpret M as column-major storage of M',
  L' is stored in the upper triangular part of M'.
  */
  uplo = 'U'; trans = 'T'; diag = 'U';
  dtrsv_(&uplo,&trans,&diag,&n,*M,&n,b,&incx);

  /* Solve U*x = z
  Account for row-major storage:
  If we interpret M as column-major storage of M',
  U' is stored in the lower triangular part of M'.
  */
  uplo = 'L'; trans = 'T'; diag = 'N';
  dtrsv_(&uplo,&trans,&diag,&n,*M,&n,b,&incx);

  return 0;
}
```

CBLAS level 1: dscal

```
#include <stdlib.h>

#if defined(__MACH__) && defined(__APPLE__)
#include <Accelerate/Accelerate.h>
#else
#include <cblas.h>
#endif

/* Scale the k'th row of a two-dimensional row-major array */
int scale_row(double alpha, double **A, int m, int n, int k) {
    if ((A==NULL) || (m<0) || (n<0) || (k<0) || (k>=m)) return -1;
    cblas_dscal(n,alpha,A[k],1);
    return 0;
}

/* Scale the k'th column of a two-dimensional row-major array */
int scale_column(double alpha, double **A, int m, int n, int k) {
    if ((A==NULL) || (m<0) || (n<0) || (k<0) || (k>=n)) return -1;
    cblas_dscal(m,alpha,*A+k,n);
    return 0;
}

/* Scale the diagonal elements of a square two-dim. row-major array */
int scale_diag(double alpha, double **A, int m, int n) {
    if ((A==NULL) || (n!=m) || (m<0) || (n<0)) return -1;
    cblas_dscal(n,alpha,*A,n+1);
    return 0;
}
```

CBLAS level 1: daxpy

```
#include <stdlib.h>

#if defined(__MACH__) && defined(__APPLE__)
#include <Accelerate/Accelerate.h>
#else
#include <cblas.h>
#endif

/** Adds alpha times row i to row j
 *
 * The input 'A' represents a two-dim. row-major array of size m-by-n
 */
int add_row(double alpha, double **A, int m, int n, int i, int j) {
    int inc=1;
    if (A==NULL || A[0]==NULL || m <= 0 || n <= 0) return -1;
    if (i == j || i < 0 || j < 0 || i >= m || j >= m) return -1;
    cblas_daxpy(n, alpha, A[i], inc, A[j], inc);
    return 0;
}

/** Adds alpha times column i to column j
 *
 * The input 'A' represents a two-dim. row-major array of size m-by-n
 */
int add_column(double alpha, double **A, int m, int n, int i, int j) {
    int inc=n;
    if (A==NULL || A[0]==NULL || m <= 0 || n <= 0) return -1;
    if (i == j || i < 0 || j < 0 || i >= n || j >= n) return -1;
    cblas_daxpy(m, alpha, A[0]+i, inc, A[0]+j, inc);
    return 0;
}
```