02635 Fall 2016 — Module 10 (solutions)

Exercises — Part I

1. Extend the library with a function that computes and returns the Euclidean norm of a vector:

```c
/* Add to linalg.h */

#define DIMENSION_ERR fprintf(stderr,"%s: dimension mismatch error\n",__func__)
int norm2(const vector_t * px, double * nrm);
```

```c
/* Add to linalg.c */

/* norm2
Purpose:
  Computes the Euclidean norm of a vector.

Arguments:
  px          a pointer to a vector_t
  nrm         a pointer to a double

Return value:
  An int with one of the following values:
     -  LINALG_SUCCESS if no error occured
     -  LINALG_ILLEGAL_INPUT if an input is NULL
     -  LINALG_DIMENSION_MISMATCH if the vector has length 0
*/
int norm2(const vector_t * px, double * nrm) {
    size_t i;
    if ( px == NULL || nrm == NULL ) {
        INPUT_ERR;
        return LINALG_ILLEGAL_INPUT;
    }
    if ( px->n == 0 ) {
        DIMENSION_ERR;
        return LINALG_DIMENSION_MISMATCH;
    }
    *nrm = 0;
    for (i=0;i<px->n;i++)
        *nrm += (px->v[i])*(px->v[i]);
    *nrm = sqrt(*nrm);
    return LINALG_SUCCESS;
}
```

Write a short program (say, `test_norm2.c`) to test the Euclidean norm function:

```c
#include <stdlib.h>
#include <assert.h>
#include "linalg.h"

int main(void) {

  double res;
  vector_t * pv=NULL;

  pv = malloc_vector(8);

  /* Test error handling */
  assert(norm2(NULL, &res) == LINALG_ILLEGAL_INPUT);
  assert(norm2(pv, NULL) == LINALG_ILLEGAL_INPUT);

  /* Check that norm of zero vector is zero */
  assert(norm2(pv, &res) == LINALG_SUCCESS);
  assert(res == 0.0);

  /* Compute norm of nonzero vector */
  pv->v[0] = 1.0;
  pv->v[7] = -1.0;
  assert(norm2(pv, &res) == LINALG_SUCCESS);
  assert(fabs(res - sqrt(2.0)) < 1e-14);

  /* Free vector and allocate vector of length 0 */
  free_vector(pv);
  pv = malloc_vector(0);
  assert(norm2(pv, &res) == LINALG_DIMENSION_MISMATCH);
  free_vector(pv);

  return EXIT_SUCCESS;
}
```

2. Extend the library with a function that computes the Frobenius norm of a matrix of size $m \times n$:

```
/* norm_fro
Purpose:
  Computes the Frobenius norm of a matrix.

Arguments:
  pA          a pointer to a matrix_t
  nrm         a pointer to a double

Return value:
  An int with one of the following values:
   -  LINALG_SUCCESS if no error occured
   -  LINALG_ILLEGAL_INPUT if an input is NULL
   -  LINALG_DIMENSION_MISMATCH if one of the matrix dimensions is 0
*/
int norm_fro(const matrix_t * pA, double * nrm) {
    size_t i,j;
    if ( pA == NULL || nrm == NULL ) {
        INPUT_ERR;
        return LINALG_ILLEGAL_INPUT;
    }
    if ( pA->m == 0 || pA->n == 0 ) {
        DIMENSION_ERR;
        return LINALG_DIMENSION_MISMATCH;
    }
    *nrm = 0;
    for (i=0;i<pA->m;i++) {
        for (j=0;j<pA->n;j++) {
            *nrm += (pA->A[i][j])*(pA->A[i][j]);
        }
    }
    *nrm = sqrt(*nrm);
    return LINALG_SUCCESS;
}
```

Write a short program (say, `test_norm_fro.c`) to test the Frobenius norm function.

```c
#include <stdlib.h>
#include <assert.h>
#include "linalg.h"

int main(void) {

  double res;
  matrix_t *pA=NULL;
  pA = malloc_matrix(3,4);

  /* Test error handling */
  assert(norm_fro(NULL, &res) == LINALG_ILLEGAL_INPUT);
  assert(norm_fro(pA, NULL) == LINALG_ILLEGAL_INPUT);

  /* Check that norm of zero matrix is zero */
  assert(norm_fro(pA, &res) == LINALG_SUCCESS);
  assert(res == 0.0);

  /* Compute norm of nonzero matrix */
  pA->A[0][0] = 1.0;
  pA->A[2][3] = -1.0;
  assert(norm_fro(pA, &res) == LINALG_SUCCESS);
  assert( fabs(res - sqrt(2.0)) < 1e-14 );

  /* Free matrix and allocate matrix with 0 columns */
  free_matrix(pA);
  pA = malloc_matrix(4,0);
  assert(norm_fro(pA, &res) == LINALG_DIMENSION_MISMATCH);
  free_matrix(pA);

  return EXIT_SUCCESS;
}
```

3. Extend the library with a function that computes the Frobenius norm of a sparse matrix of size $m \times n$:

```
/* norm_fro_sparse
Purpose:
  Computes the Frobenius norm of a sparse matrix in triplet form.

Arguments:
  pA          a pointer to a matrix_t
  nrm         a pointer to a double

Return value:
  An int with one of the following values:
   -  LINALG_SUCCESS if no error occured
   -  LINALG_ILLEGAL_INPUT if an input is NULL
   -  LINALG_DIMENSION_MISMATCH if one of the matrix dimensions is 0
*/
int norm_fro_sparse(const sparse_triplet_t * pA, double * nrm) {
    size_t i;
    if ( pA == NULL || nrm == NULL ) {
        INPUT_ERR;
        return LINALG_ILLEGAL_INPUT;
    }
    if ( pA->m == 0 || pA->n == 0 ) {
        DIMENSION_ERR;
        return LINALG_DIMENSION_MISMATCH;
    }
    *nrm = 0;
    for (i=0;i<pA->nnz;i++)
            *nrm += (pA->V[i])*(pA->V[i]);
    *nrm = sqrt(*nrm);
    return LINALG_SUCCESS;
}
```

Write a short program (say, `test_norm_fro_sparse.c`) to test the function:

```c
#include <stdlib.h>
#include <assert.h>
#include "linalg.h"

int main(void) {

  double res;
  sparse_triplet_t *pA=NULL;
  pA = malloc_sparse_triplet(3,4,8);

  /* Test error handling */
  assert(norm_fro_sparse(NULL, &res) == LINALG_ILLEGAL_INPUT);
  assert(norm_fro_sparse(pA, NULL) == LINALG_ILLEGAL_INPUT);

  /* Check that norm of zero matrix is zero */
  assert(norm_fro_sparse(pA, &res) == LINALG_SUCCESS);
  assert(res == 0.0);

  /* Compute norm of nonzero matrix */
  pA->V[0] = 1.0;
  pA->V[7] = -1.0;
  assert(norm_fro_sparse(pA, &res) == LINALG_SUCCESS);
  assert( fabs(res - sqrt(2.0)) < 1e-14 );

  /* Free matrix and allocate matrix with 0 columns */
  free_sparse_triplet(pA);
  pA = malloc_sparse_triplet(4,0,8);
  assert(norm_fro_sparse(pA, &res) == LINALG_DIMENSION_MISMATCH);
  free_sparse_triplet(pA);

  return EXIT_SUCCESS;
}
```

4. Extend the library with a function that computes the inner product of two vectors x and y of length n:

```c
/* dot
Purpose:
  Computes the inner product of two vectors.

Arguments:
  px          a pointer to a vector_t
  py          a pointer to a vector_t
  xy          a pointer to a double

Return value:
  An int with one of the following values:
   -  LINALG_SUCCESS if no error occured
   -  LINALG_ILLEGAL_INPUT if an input is NULL
   -  LINALG_DIMENSION_MISMATCH if the vectors have different lengths
*/
int dot(const vector_t * px, const vector_t * py, double * xy) {
    size_t i;
    if ( px == NULL || py == NULL || xy == NULL ) {
        INPUT_ERR;
        return LINALG_ILLEGAL_INPUT;
    }
    if ( px->n != py->n || px->n == 0 ) {
        DIMENSION_ERR;
        return LINALG_DIMENSION_MISMATCH;
    }
    *xy = 0;
    for (i=0;i<px->n;i++)
        *xy += (px->v[i]) * (py->v[i]);
    return LINALG_SUCCESS;
}
```

Write a short program (say, `test_dot.c`) to test the inner product function:

```c
#include <stdlib.h>
#include <assert.h>
#include "linalg.h"

int main(void) {

  double res;
  vector_t *px=NULL, *py=NULL, *pz=NULL;

  /* Allocate vectors */
  px = malloc_vector(8);
  py = malloc_vector(8);
  pz = malloc_vector(0);

  /* Check error handling */
  assert(dot(NULL, py, &res) == LINALG_ILLEGAL_INPUT);
  assert(dot(px, NULL, &res) == LINALG_ILLEGAL_INPUT);
  assert(dot(px, py, NULL) == LINALG_ILLEGAL_INPUT);

  /* Compute inner product of two zero vectors */
  assert(dot(px, py, &res) == LINALG_SUCCESS);
  assert(res == 0);

  /* Compute inner product of two nonzero vectors */
  px->v[0] = 1.0;
  px->v[7] = 0.5;
  py->v[0] = -1.0;
  py->v[7] = 1.0;
  assert(dot(px, py, &res) == LINALG_SUCCESS);
  assert(fabs(res + 0.5) < 1e-14);

  /* Check error handling, dimension mismatch */
  assert(dot(px, pz, &res) == LINALG_DIMENSION_MISMATCH);
  assert(dot(pz, py, &res) == LINALG_DIMENSION_MISMATCH);

  /* Deallocate vectors */
  free_vector(px);
  free_vector(py);
  free_vector(pz);

  return EXIT_SUCCESS;
}
```

Optional exercise

1. Write a program that measures the CPU time required to compute the Frobenius norm of a matrix of size $m \times n$ where m and n are user inputs. If the CPU time is less than 1 second, create a loop that repeats the computation a number of times in order to obtain better timing accuracy.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "linalg.h"

int main(int argc, const char * argv[]) {

  unsigned long m,n;
  matrix_t *pm;
  double res, T=0;
  clock_t t1,t2;
  size_t i,k=0;

  if (argc != 3) {
    printf("Usage: %s m n\n", argv[0]);
    return 0;
  }
  m = atoi(argv[1]);
  n = atoi(argv[2]);
  pm = malloc_matrix(m,n);

  /* Approach 1:
    - repeat measurement until total time > 1 second
  */
  do {
    t1 = clock();
    norm_fro(pm, &res);
    t2 = clock();
    k++;
    T += (double) (t2-t1)/CLOCKS_PER_SEC;
  } while (T < 1.0);
  T /= k;

  printf("Repetitions: %zu\n", k);
  printf("CPU time: %.6g ms\n",T*1e3);

  /* Approach 2:
```

```
    - estimate CPU time
    - compute number of repetitions k
    - use for loop to time k repetitions
  */
  t1 = clock();
  norm_fro(pm, &res);
  t2 = clock();
  T = (double) (t2-t1)/CLOCKS_PER_SEC;
  k = ( T > 1 ? 1 : (size_t) 1.0/T );

  t1 = clock();
  for (i=0;i<k;i++) {
    norm_fro(pm, &res);
  }
  t2 = clock();
  T = (double) (t2-t1)/CLOCKS_PER_SEC/k;

  printf("Repetitions: %zu\n", k);
  printf("CPU time: %.6g ms\n",T*1e3);

  /* Clean up and exit */
  free_matrix(pm);
  return 0;
}
```