Week 10 solutions

Exercises

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

```
#include <math.h>
#include "matrix_io.h"

/* 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:
   - MATRIX_IO_SUCCESS if no error occured
   - MATRIX_IO_ILLEGAL_INPUT if an input is NULL
   - MATRIX_IO_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 MATRIX_IO_ILLEGAL_INPUT;
  }
  if (px->n == 0) {
    DIMENSION_ERR;
    return MATRIX_IO_DIMENSION_MISMATCH;
  }
  *nrm = 0;
  for (i = 0; i < px->n; i++)
    *nrm += (px->v[i]) * (px->v[i]);
  *nrm = sqrt(*nrm);
  return MATRIX_IO_SUCCESS;
}
```

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

```
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "matrix_io.h"
```

```c
int norm2(const vector_t *px, double *nrm);

int main(void) {

  double res;
  vector_t * pv=NULL;

  /* Allocate a vector_t */
  pv = malloc_vector(8);
  assert(norm2(NULL, &res) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm2(pv, NULL) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm2(pv, &res) == MATRIX_IO_SUCCESS);
  assert(res == 0.0);
  pv->v[0] = 1.0;
  pv->v[7] = -1.0;
  assert(norm2(pv, &res) == MATRIX_IO_SUCCESS);
  assert( fabs(res - sqrt(2.0)) < 1e-14 );
  free_vector(pv);

  pv = malloc_vector(0);
  assert(norm2(pv, &res) == MATRIX_IO_DIMENSION_MISMATCH);
  free_vector(pv);

  return EXIT_SUCCESS;
}
```

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

```c
#include <math.h>
#include "matrix_io.h"

/* 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:
   - MATRIX_IO_SUCCESS if no error occured
   - MATRIX_IO_ILLEGAL_INPUT if an input is NULL
   - MATRIX_IO_DIMENSION_MISMATCH if one of the matrix dim. is 0
*/
int norm_fro(const matrix_t * pA, double * nrm) {
        size_t i,j;
        if ( pA == NULL || nrm == NULL ) {
                INPUT_ERR;
                return MATRIX_IO_ILLEGAL_INPUT;
        }
        if ( pA->m == 0 || pA->n == 0 ) {
                DIMENSION_ERR;
                return MATRIX_IO_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 MATRIX_IO_SUCCESS;
}
```

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

```
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "matrix_io.h"

int norm_fro(const matrix_t * pA, double * nrm);

int main(void) {

  double res;
  matrix_t *pA=NULL;

  /* Allocate a matrix_t */
  pA = malloc_matrix(3,4);
  assert(norm_fro(NULL, &res) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm_fro(pA, NULL) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm_fro(pA, &res) == MATRIX_IO_SUCCESS);
  assert(res == 0.0);
  pA->A[0][0] = 1.0;
  pA->A[2][3] = -1.0;
  assert(norm_fro(pA, &res) == MATRIX_IO_SUCCESS);
  assert( fabs(res - sqrt(2.0)) < 1e-14 );
  free_matrix(pA);

  pA = malloc_matrix(4,0);
  assert(norm_fro(pA, &res) == MATRIX_IO_DIMENSION_MISMATCH);
  free_matrix(pA);
  return EXIT_SUCCESS;
}
```

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

```
#include <math.h>
#include "matrix_io.h"

/* 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:
   - MATRIX_IO_SUCCESS if no error occured
   - MATRIX_IO_ILLEGAL_INPUT if an input is NULL
   - MATRIX_IO_DIMENSION_MISMATCH if one of the matrix dim. is 0
*/
int norm_fro_sparse(const sparse_triplet_t * pA, double * nrm) {
  size_t i;
  if ( pA == NULL || nrm == NULL ) {
    INPUT_ERR;
    return MATRIX_IO_ILLEGAL_INPUT;
  }
  if ( pA->m == 0 || pA->n == 0 ) {
    DIMENSION_ERR;
    return MATRIX_IO_DIMENSION_MISMATCH;
  }
  *nrm = 0;
  for (i=0;i<pA->nnz;i++)
      *nrm += (pA->V[i])*(pA->V[i]);
  *nrm = sqrt(*nrm);
  return MATRIX_IO_SUCCESS;
}
```

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

```
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "matrix_io.h"
```

```c
int norm_fro_sparse(const sparse_triplet_t * pA, double * nrm);

int main(void) {

  double res;
  sparse_triplet_t *pA=NULL;

  /* Allocate a matrix_t */
  pA = malloc_sparse_triplet(3,4,8);
  assert(norm_fro_sparse(NULL, &res) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm_fro_sparse(pA, NULL) == MATRIX_IO_ILLEGAL_INPUT);
  assert(norm_fro_sparse(pA, &res) == MATRIX_IO_SUCCESS);
  assert(res == 0.0);
  pA->V[0] = 1.0;
  pA->V[7] = -1.0;
  assert(norm_fro_sparse(pA, &res) == MATRIX_IO_SUCCESS);
  assert( fabs(res - sqrt(2.0)) < 1e-14 );
  free_sparse_triplet(pA);

  pA = malloc_sparse_triplet(4,0,8);
  assert(norm_fro_sparse(pA, &res) == MATRIX_IO_DIMENSION_MISMATCH);
  free_sparse_triplet(pA);

  return EXIT_SUCCESS;
}
```

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

```c
#include "matrix_io.h"

/* 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:
   - MATRIX_IO_SUCCESS if no error occured
   - MATRIX_IO_ILLEGAL_INPUT if an input is NULL
   - MATRIX_IO_DIMENSION_MISMATCH if the vectors have diff. len.
*/
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 MATRIX_IO_ILLEGAL_INPUT;
  }
  if ( px->n != py->n || px->n == 0 ) {
    DIMENSION_ERR;
    return MATRIX_IO_DIMENSION_MISMATCH;
  }
  *xy = 0;
  for (i=0;i<px->n;i++)
    *xy += (px->v[i]) * (py->v[i]);
  return MATRIX_IO_SUCCESS;
}
```

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

```
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "matrix_io.h"

int dot(const vector_t * px, const vector_t * py, double * xy);

int main(void) {

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

  /* Allocate a vector_t */
  px = malloc_vector(8);
  py = malloc_vector(8);
  pz = malloc_vector(0);
  assert(dot(NULL, py, &res) == MATRIX_IO_ILLEGAL_INPUT);
  assert(dot(px, NULL, &res) == MATRIX_IO_ILLEGAL_INPUT);
  assert(dot(px, py, NULL) == MATRIX_IO_ILLEGAL_INPUT);
  assert(dot(px, py, &res) == MATRIX_IO_SUCCESS);
  assert(res == 0);
  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) == MATRIX_IO_SUCCESS);
  assert(fabs(res + 0.5) < 1e-14);
  assert(dot(px, pz, &res) == MATRIX_IO_DIMENSION_MISMATCH);
  assert(dot(pz, py, &res) == MATRIX_IO_DIMENSION_MISMATCH);
  free_vector(px);
  free_vector(py);
  free_vector(pz);

  return EXIT_SUCCESS;
}
```