02635 Fall 2016 — Module 10

Homework

- Read chapter 13 in "Beginning C"
- Read sections 9.1-9.5 (pp. 118-130) in "Writing Scientific Software"

Exercises — Part I

Two weeks ago, we worked with a structure `struct sparse_triplet` representing a sparse matrix in triplet form. This week we are going to work on a library that implements basic linear algebra operations for vectors, matrices, and sparse matrices in triplet form.

1. Download `linalg.zip` from CampusNet and unzip it. The file contains four files:

    - a source file `linalg.c` and a header file `linalg.h` which provide a basic set of data structures and functions
    - a source file `test_basic.c` which is a short program with some test cases
    - a makefile which can be used to compile the test program.

    Look through the header file `linalg.h` to familiarize yourself with the data structures and functions that the library provides. Notice that there are three data structures:

    - `vector_t` represents a vector of length n and is defined as:

    ```
    typedef struct vector {
       unsigned long n;  /* length of vector                 */
       double * v;       /* pointer to array of length n     */
    } vector_t;
    ```

    - `matrix_t` represents a matrix of size $m \times n$ and is defined as:

```c
typedef struct matrix {
    unsigned long m;  /* number of rows                    */
    unsigned long n;  /* number of columns                 */
    double ** A;      /* pointer to two-dimensional array  */
} matrix_t;
```

- `sparse_triplet_t` represents a sparse matrix in triplet form of size $m \times n$ and is defined as:

```c
typedef struct sparse_triplet {
    unsigned long m;   /* number of rows                      */
    unsigned long n;   /* number of columns                   */
    unsigned long nnz; /* number of nonzeros                  */
    unsigned long * I; /* pointer to array with row indices    */
    unsigned long * J; /* pointer to array with column indices */
    double * V;        /* pointer to array with values         */
} sparse_triplet_t;
```

Notice also that for each of these data types, there are functions for the following operations:

- memory allocation/deallocation (e.g., `malloc_vector()` and `free_vector()`)
- file input/output (e.g., `read_vector()` and `write_vector()`)
- console output (e.g., `print_vector()`).

The functions that start with `malloc_` and `read_` allocate memory and return a pointer. This means that each call to one of these functions should be matched with a call to the corresponding function that starts with `free_`. For example, if your program contains a call to `read_matrix()`, there should also be a call to `free_matrix()` in order to deallocate the memory that was allocated by `read_matrix()`.

If you want to know more about the functions defined in `linalg.h`, open the source file `linalg.c` and take a look at the implementation of the different functions. The source file also includes basic documentation in the form of comments.

2. Open the `test_basic.c` source file and inspect the code. Compile the test program and run it:

```
$ make test
$ ./test_basic
```

The test program should print a vector, a matrix, a sparse matrix, and some error messages. The program should also create three files: `test_vector.txt`, `test_matrix.txt`, and `test_sparse_triplet.txt`. Open each of these files in a text editor and compare with the screen output.

3. Extend the library with a function that computes and returns the Euclidean norm of a vector x, i.e.,

$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}.$$

The function should have the following prototype

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

and the implementation should be documentated with comments in the source file. The Euclidean norm should be stored in the second argument `nrm`, and the functions should return one of the following values:

   - `LINALG_ILLEGAL_INPUT` if one of the inputs is `NULL`
   - `LINALG_DIMENSION_MISMATCH` if the length of the vector is 0
   - `LINALG_SUCCESS` if the function returns without any errors.

You may include the prototype in `linalg.h` and the implementation in `linalg.c`.

Write a short program (say, `test_norm2.c`) to test the Euclidean norm function. The program should test that the different error cases are handled correctly.

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

$$\|A\|_{\mathrm{F}} = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2\right)^{1/2}.$$

The function should have the following prototype

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

and the implementation should be documentated with comments in the source file. The Frobenius norm should be stored in the second argument `nrm`, and the functions should return one of the following values:

   - `LINALG_ILLEGAL_INPUT` if one of the inputs is `NULL`
   - `LINALG_DIMENSION_MISMATCH` if either m or n is zero

- `LINALG_SUCCESS` if the function returns without any errors.

Write a short program (say, `test_norm_fro.c`) to test the Frobenius norm function. The program should test that the different error cases are handled correctly.

5. Extend the library with a function that computes the Frobenius norm of a sparse matrix of size $m \times n$. The function should have the following prototype

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

and the implementation should be documentated with comments in the source file. The Frobenius norm should be stored in the second argument `nrm`, and the functions should return one of the following values:

- `LINALG_ILLEGAL_INPUT` if one of the inputs is `NULL`
- `LINALG_DIMENSION_MISMATCH` if either m or n is zero
- `LINALG_SUCCESS` if the function returns without any errors.

Write a short program (say, `test_norm_fro_sparse.c`) to test the function. The program should test that the different error cases are handled correctly.

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

$$x^T y = \sum_{i=1}^{n} x_i y_i.$$

You function should have the following prototype:

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

The inner product should be stored in the third argument `xy`, and the return value should be equal to

- `LINALG_ILLEGAL_INPUT` if one of the inputs is `NULL`
- `LINALG_DIMENSION_MISMATCH` if the input vectors are of different length or if both vectors have length 0
- `LINALG_SUCCESS` if the function returns without any errors.

Your implementation should include sufficient documentation in the form of comments.

Write a short program (say, `test_dot.c`) to test the inner product function. The program should test that the different error cases are handled correctly.

Optional exercise

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.