LINMA2710 : Scientific computing

Project 1 : Sparse vectors and matrices

Mattias Van Eetvelt - 1660 18 00

1 Complexity analysis

1.1 Operator + between two sparse vectors

The implementation of this function iterates over the non-zero values of both sparse vector given in the input. Three cases are considered, each of them having a respective while loop in order to handle the different cases. The first case is when there is a non-zero value in each sparse vector for the same row index, in which case the sum is computed and stored. The other two cases correspond to the cases where there is only one non-zero value and only this value is stored, since the other corresponding value is zero. All the values and the corresponding indexes are stored into temporary arrays of size $nnz_1 + nnz_2$. Through out the iteration over the non-zero values, a counter is set in order to build the final correct-sized arrays and both the values and indexes are copied into them.

The time complexity of all three while loops is $\mathcal{O}(nnz_1 + nnz_2)$. The worst case scenario correspond to the scenario where both vectors actually are dense vectors. Thus the global time and space complexity is $\mathcal{O}(nnz_1 + nnz_2)$.

1.2 Operator * between a sparse matrix and a vector

The function iterates over the non-zero values of the sparse matrix. Two counters are set. The former keeps track of the column we are in, using the j^{th} and the $j + 1^{th}$ element of the `colptr` array. The latter allows to keep track of the row inside a given column. Using this information, it is possible to visit all non-zero values exactly one time and compute the matrix-vector product. Therefore, even if two `while` and `if` loops are nested, the complexity is in fact $\mathcal{O}(nnz)$.

2 Memory states during the = operator of sparse vector

This operator is used to overload the default assignment operator to copy the values of one sparse matrix to another.

First, the member variables `m, n, nnz` are set to the same value as the input object. Then, memory is allocated for the `coptr, roxidx, nzval` arrays. Next, the values of these array from the input object are copied into the newly declared arrays, i.e. the arrays of the output object. At the end of the function, a reference to the output object is returned using the `"*this"` notation. Therefore, at the output, the `coptr, rowidx, nzval` arrays of the output object point to newly allocated memory. At this address, the memory contains the same values as the values pointed to by the respective arrays in the input object. In other words, the arrays in the input and output objects have the same values but their memory address are different. The object being copied and the object being constructed have different memory addresses as well.