

On efficient finite element assembly in MATLAB

Alejandro A. Ortiz

alortiz@ucdavis.edu

Department of Civil and Environmental Engineering
University of California, Davis, CA 95616, U.S.A.

February 7, 2010

Posted at: <http://www.imechanica.org/node/7552>

As we all know, finite element matrices are sparse and many memory can be saved if their sparsity is exploited. Due to the latter, we are often provoked to declare the $n \times n$ stiffness matrix as a sparse one just before the element loop. Be careful of this method since the larger the matrix the slower the assembly operations. There is a better and more efficient way to code the assembly procedure in MATLAB. I will show how to accomplish this. For the sake of simplicity, the stiffness matrix of the two-dimensional heat conduction problem on the domain $(0,1) \times (0,1)$ is considered. A uniform mesh of $m \times m$ triangles is used, where m is the number of subdivisions per side. Three assembly methods are tested. The computations are done with increasing number of subdivisions. The complete MATLAB code that I used for the tests can be downloaded from [my webpage](#).

Method 1 (assembly1.m)

$$K(i,j) = K(i,j) + \text{number},$$

with K a sparse matrix declared before the loop over finite elements. This method is the one that many of us would normally use to perform the assembly in a finite element code. However, it turns out that this method is the slowest, but it can save significant amount of memory.

Method 2 (assembly2.m)

$$K(i,j) = K(i,j) + \text{number},$$

with K a full matrix declared before the loop over finite elements. This scheme is faster than method 1. However, it will rapidly run out of memory. Because of the latter, its applicability is limited to matrices of modest size.

Method 3 (assembly3.m)

On using three vector arrays I , J , X such that $K(I(a), J(a)) = X(a)$, the sparse matrix K of size $n \times n$ is built as

$$K = \text{sparse}(I, J, X, n, n).$$

The X values of repeated pairs in I, J are added each other in MATLAB sparse function, which represents the assembly procedure. This method can save as much memory as method 1 but it is significantly faster. The allocation (with MATLAB `zeros` function) of I , J and X is done before the finite element loop. Note that the size of these arrays is known a priori. For instance, consider the case of the heat conduction problem. Each triangle contributes with 9 entries to the stiffness matrix. Hence, the size of I , J and X is 9 times the number of elements. If 100×100 subdivisions are specified, that is $100 \times 100 \times 2$ triangle elements, the size of I , J or X is 180000 numbers. The total size that one needs to allocate for is then 540000 numbers, which represents about 1/192 of the amount needed to allocate K in method 2. After the construction of K with `sparse(I,J,X,n,n)`, the arrays I , J and X can be deleted as they are not further needed, which results in additional memory saving.

The tests

In the tests, the following machines are used:

Machine 1: Intel Core2 Duo CPU T7500 @ 2.20 GHz, 789 MHz, Windows XP (32-bit).

Machine 2: Intel Xeon CPU X3363 @ 2.83 GHz, 2833.342 MHz, Red Hat (64-bit)

Test 1: Square domain $(0,1) \times (0,1)$ with 100×100 subdivisions. The results are the following:

Machine 1:

Method 1 = 13.7357 s

Method 2 = 8.3654 s

Method 3 = 3.3775 s

Machine 2:

Method 1 = 2.6798 s

Method 2 = 1.9539 s

Method 3 = 0.8609 s

Test 2: Square domain $(0,1) \times (0,1)$ with 200×200 subdivisions. The results are the following:

Machine 1:

Method 1 = 145.4049 s

Method 2 = out of memory

Method 3 = 13.1721 s

Machine 2:

Method 1 = 22.8242 s

Method 2 = 14.9338 s

Method 3 = 3.4414 s

Test 3: Square domain $(0,1) \times (0,1)$ with 500×500 subdivisions. The results are the following:

Machine 1:

Method 1 = not tested

Method 2 = not tested

Method 3 = not tested

Machine 2:

Method 1 = 603.6082 s

Method 2 = out of memory

Method 3 = 21.4391 s

Test 4: Square domain (0,1)x(0,1) with 800x800 subdivisions. The results are the following: Ma-

chine 1:

Method 1 = not tested

Method 2 = not tested

Method 3 = not tested

Machine 2:

Method 1 = 3805.9 s Meth-

od 2 = out of memory

Method 3 = 54.6115 s

Test 5: Square domain (0,1)x(0,1) with 1500x1500 subdivisions. The results are the following: Ma-

chine 1:

Method 1 = not tested

Method 2 = not tested

Method 3 = not tested

Machine 2:

Method 1 = 186260 s

Method 2 = out of memory

Method 3 = 190.6444 s

Conclusions

The assembly computations can significantly be improved if MATLAB sparse function is used appropriately. Method 1 is not suitable since in the finite element loop there are as many insertion operations as the number of $K(i,j)$ evaluations. In other words, if a new non-zero entry is to be introduced in the sparse structure, then MATLAB has to make room for it by inserting the value into the structure. An insertion operation into an array is proportional to the number of data in the array, in this case, the number of non-zero (nnz) entries in K . Therefore, the loop over $K(i,j)$ takes time proportional to $\text{square}(nnz)$. On the other hand, method 2 is preferable over method 1, but it has limited applicability since the system will run out of memory sooner than later. Finally, as we have seen from the tests, method 3 is by far the more efficient way to perform the assembly of finite element matrices with faster computations. I highly recommend using method 3 and avoid method 1.