Programming (FEB22012[X])

4. Exercise

Deadline for submission: 2014-10-05 23:59 CET

Instructions

Testing your code in an automated manner is very important for developing working programs. Most programs are not written for one-time use, but they develop iteratively, and new features are added with time. However, adding new features can break existing functionality. *Unit tests* can help to give you confidence in your program code by testing functionality at the granularity of a single method call. For example, if you have a function that computes the input + 2:

```
function a = addTwo(b)
        a = b+2;
end
```

you can make a unit test for testing the function with some input values:

```
function testAddTwo()
        assert(addTwo(2) == 4);
        assert(addTwo(-2) == 0);
        "addTwo OK"
        end
end
```

Now although you cannot be sure about whether the addTwo works properly with all possible inputs (as you in general can almost never be sure, remember the halting problem?), the unit test gives you some assurance that the addTwo-method works.

In the third lecture you were shown different algorithms for matrix multiplication. In this week's first exercise you will implement them and compare the running times similarly as you did in the 2nd exercise for different sorting algorithms. Comparing the algorithms should provide you also insights on the implications of Matlab not containing procedures (but only functions).

Exercise

Implement as Matlab functions the following two matrix multiplication algorithms, both of them taking as parameters matrices A and B and returning $C = AB$.

1. Naive "schoolbook" algorithm
2. Strassen's algorithm

Document and check pre-conditions in the implemented functions, and make unit tests for assuring that the methods work.

Make a script file for assessing the running times of the two multiplication algorithms in multiplying matrices of sizes $n \times n$, where $n \in \{2, 3, \ldots, 100\}$. In each iteration of the computational test, construct the matrices (A and B) to be multiplied to contain random integers from the interval (1, 10). Plot *differences* of the running times of the two algorithms for each size of the matrix so, that the x-axis contains the is n and the y-axis the differences $T_{naive}(n) - T_{strassen}(n)$. Remember to include axis titles and legend in the graph. How large does n have to be for the Strassen's algorithm to be faster? Why? (Answer this just for yourself, you don't need to include an explanation in the exercise to be handed in).