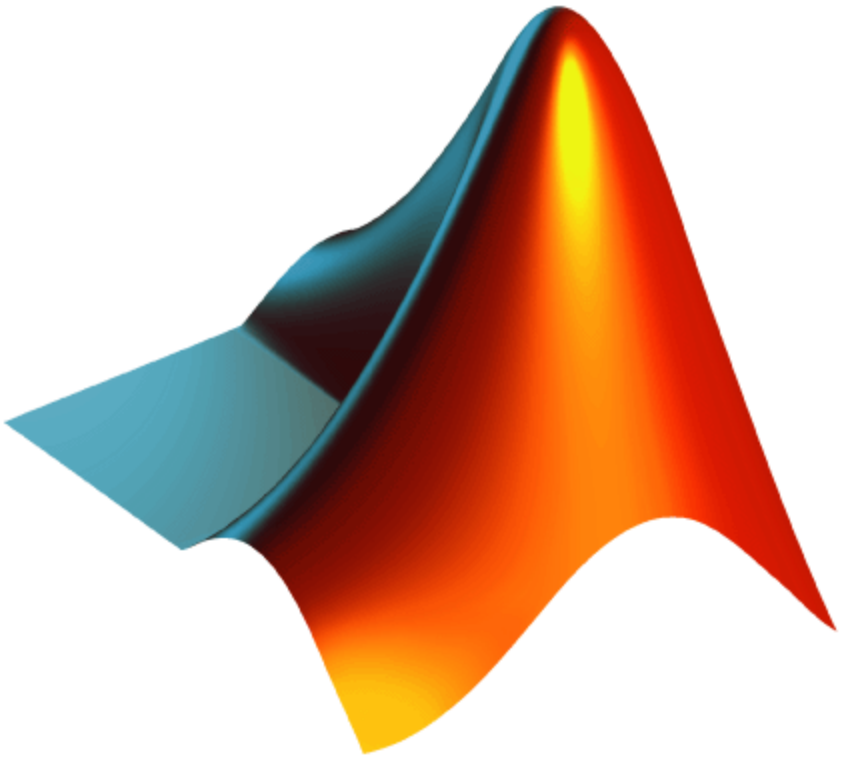

Matlab Logbook (example)

COMPUTER ALGEBRA AND TECHNICAL COMPUTING

Bart Vorselaars | MTH1006 | 2023-2024
University of Lincoln, School of Mathematics and Physics

Declaration

I confirm that this logbook is all my own work and that all references and quotations from both primary and secondary sources have been fully identified and properly acknowledged – Bart Vorselaars

Front cover: *picture taken from* http://uk.mathworks.com/company/newsletters/articles/the-mathworks-logo-is-an-eigenfunction-of-the-wave-equation.html

Table of Contents

Week 7

....

Week 8

15-11-2023

In week 8 we learned about the importance of efficient programming. In MATLAB timing a code can be carried out using the statements *tic* and *toc*.

One of the computationally intensive operations is to grow an array in a loop. Much quicker is to create the array before the loop and then fill in the values afterwards.

EXERCISES

Timing

1

A

The script is (copy-pasted from MATLAB to Word)

```
% The following script calculates the double sum:
%  S=sum_{n=1}^N sum_{m=1}^N delta_{nm}*n
% for N=10000. Here delta_{nm} is the Kronecker
% delta function, which is 1 for n=m and zero otherwise.
N=10000;
S=0;
for n=1:N
    for m=1:N
        if n==m
            d=1;
        else
            d=0;
        end
        S=S+d*n;
        disp(['n,m,d,S=',num2str([n,m,d,S])])
    end
end
S
```

The delta function is coded using the variable *d*. I've included a display statement (for debugging purposes) and it can be disabled by putting a percentage sign (%) in front of it. Each iteration within the two *for* loops the variable S is increased, so that at the end of the two *for* loops it should contain the requested quantity.

B

The script can be rewritten as a function with name *S1.m*:

```
function S=S1(N)
% S=S1(N)
%
% Returns the double sum:
%  S=sum_{n=1}^N sum_{m=1}^N delta_{nm}*n
% Here delta_{nm} is the Kronecker delta function,
% which is 1 for n=m and zero otherwise.
S=0;
for n=1:N
    for m=1:N
        if n==m
            d=1;
        else
            d=0;
        end
        S=S+d*n;
        %disp(['n,m,d,S=',num2str([n,m,d,S])])
    end
end
```

Here the *disp* function call has now been disabled, otherwise too much output is generated for large *N*. To call this function, the following script has been used:

```
% Test script for testing the function S1 with a known value.
N=1;
S=S1(N);
assert(S==1,'Unexpected value of the sum: for N=1 the sum S should be
1')
```

C

The function can be simplified using one for loop, with the function *S2.m*:

```
function S=S2(N)
% S=S2(N)
%
% Returns the sum
%  S=sum_{n=1}^N n
S=0;
for n=1:N
    S=S+n;
end
```

D

The function can be simplified further, since there is an analytical expression for the sum of elements ranging from 1 to N. This results in the function *S3.m*:

```
function S=S3(N)
% S=S3(N)
%
% Returns the sum
%  S=sum_{n=1}^N n
```

```
% which can be simplified to the analytical expression
%  S=0.5*(N^2+N)
S=0.5*(N^2+N);
```

E

We can call all three functions subsequently using the following script:

```
% Timing script for the three different functions.
% Each of them calculate the same quantity, but with
% a different computational complexity.
N=10000;
tic
S=S1(N);
toc
tic
S=S2(N);
toc
tic
S=S3(N);
toc
```

Running it gives the timing results for N=10000:

```
Elapsed time is 1.481639 seconds.
Elapsed time is 0.000130 seconds.
Elapsed time is 0.000024 seconds.
```

This shows that the S3 function is fastest. However, this only occurs when taking the minimum values after repeated execution of the script. Otherwise, because of other program running the timing could be very different:

```
Elapsed time is 6.185825 seconds.
Elapsed time is 0.004680 seconds.
Elapsed time is 0.013501 seconds.
```

Doubling N to 20000 results in:

```
Elapsed time is 6.391665 seconds.
Elapsed time is 0.000241 seconds.
Elapsed time is 0.000025 seconds.
```

This shows that function S1 is approximately 4 times slower, S2 approximately 2 times, and S3 approximately constant. For N=40000 one would expect that S1 runs by approximately 16 times slower, S2 approximately 4 times, and S3 would be independent on N. The reason is that in the function S1 the addition to the sum gets executed N*N=N^2 times, so that it needs N^2 operations. For S2 the addition to the sum gets executed N times, so it needs N operations. For S3 there is only 1 line to execute and it therefore only needs 1 operation. The time it takes for each operation is roughly constant, so that upon doubling N: the time for S1 quadruples, the time for S2 doubles and the time for S3 stays constant.

GENERAL REFLECTION ON MATERIAL

This week was not too difficult to understand and master, apart from implementing the double 'for' loop. However, after consulting more examples in the book by Stormy Attaway I managed that as well.

References

While solving some of the exercises, background information on the various commands is taken from:

- The Matlab documentation within the program itself, version R2023a, using *help* and *doc*
- Stormy Attaway, *Matlab, a practical introduction to programing and problem solving*, third edition, Elsevier, Amsterdam, 2013