Department of Computer Science, University College London

COMP0043 Numerical Methods for Finance

Prof. Guido Germano

Exercises for Section 1 Introduction

1. Generate two $n \times n$ random matrices $\mathbf{A}$ and $\mathbf{B}$ using the function `rand()`; start with $n = 3$.

   (a) Compute the matrix product $\mathbf{C} = \mathbf{AB}$ using MATLAB's built-in matrix product operator `*` (or the function `mtimes`).

   (b) Compute $\mathbf{C} = \mathbf{AB}$ from the definition

   $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

   using three `for` loops.

   (c) Repeat (b) swapping the two outer loops over i and j.

   (d) Repeat (b) with two `for` loops over i and j and `sum()` in place of the `for` loop over k.

   (e) Repeat (d) transposing $\mathbf{A}$ to increase the data locality and thus the cache hits with `sum()`.

   (f) Repeat (e) transposing $\mathbf{B}$ to reduce the data locality and thus the cache hits with `sum()`.

   (g) Repeat (a–f) for $n = 10, 100, 1000$ suppressing the output and monitoring the CPU time with `tic` and `toc`. Be patient with `for` loops when $n = 1000$.

   (h) Find how the CPU time of matrix multiplication scales with the matrix size n: produce a double logarithmic plot for Matlab's built-in product operator (use $n = 10,\ 20,\ 50,\ 100,\ 200,\ 500,\ 1000,\ 2000,\ 5000$) and for the triple loop (use $n = 10,\ 20,\ 50,\ 100,\ 200,\ 500,\ 1000$); fit the data with `fit`.

   For further reading, see Section 2.11 of Numerical Recipes, which briefly introduces Strassen's algorithm for matrix multiplication.

2. The machine precision or machine epsilon ε gives an upper bound to the relative rounding error in floating point arithmetics with a finite number of digits: it is the smallest number which, added to 1, gives a result different from 1.

   By default, Matlab outputs the result in exponential notation; `format long` provides only 15 decimal digits. Print ε in fixed-point notation with 22 decimal digits using `fprintf`.

   (a) Compute the machine precision with the commands `for, if, break`: initialise $\varepsilon = 1$, divide it by 2 within a for loop from $i = 1$ to 100, and when $1+\varepsilon = 1$ interrupt the loop with a break instruction. Output also the final value of i.

   (b) The machine precision is a simple function $f(i)$ of the final value of the for loop counter i; find the general expression of $f(i)$ and print $\varepsilon = f(i)$ inserting the value of i from (a).

   (c) More elegantly, compute the machine precision with a `while` loop. This does not require a loop counter and thus an upper limit for the latter, and results in less code.

   (d) Check the result ε by printing the sum $1 + \varepsilon$ and the sum $1 + \varepsilon/2$: only the latter should be equal to 1.