NumCSE exercise sheet 1
Cancellation errors, manipulating matrices

alexander.dabrowski@sam.math.ethz.ch
oliver.rietmann@sam.math.ethz.ch

September 28, 2018

Exercise 1.1. *Summing floating point numbers.*

Suppose you have a `vector<float> v` which contains some unknown numbers with very different magnitudes, and you want to compute its total sum.

(a) Suppose in this subpoint that `v` has size 10^7, and that it has a single element equal to `1` and the rest are all in the range `[1e-8, 2e-8]`. Consider the following simple code to sum all the numbers in `v`:

```
float sum = 0;
for (float x : v) sum += x;
```

What is the final value of `sum` if `1` is in the first position of `v`? What can you say if `1` is in the last position of `v`?

(b) Implement a function `sum` which returns the sum of `v` with good accuracy (if `v` is as in subpoint (a), the sum should have an error smaller than `1e-3` for any possible position of `1`).

Hint: add up first the numbers which have small magnitude.

(c) What if instead `v` has size 10^8, with all elements in the range `[1e-8, 2e-8]` except for a single element `1`?

(d) Implement a function `acc_sum` which returns the sum of `v` by adapting the simple code of Point (a) to keep track in an accumulator variable of cancellation errors which arise in each iteration of the loop.

Hint: There are many possible solutions to this problem. The following hint might be helpful to derive one of the "best" strategies. At each loop: add `x` to the accumulator, then try to add the accumulator to the running sum, then add to the accumulator the cancellation error.

Exercise 1.2. *Structured matrix–vector product.*

Let $n \in \mathbb{N}$ and A be a real $n \times n$ matrix defined as:

$$(A)_{i,j} = a_{i,j} = \min\{i, j\},\ i, j = 1, \ldots, n. \tag{1}$$

The matrix-vector product $y = Ax$ can be implemented as

```
23    VectorXd one = VectorXd::Ones(n);
24    VectorXd linsp = VectorXd::LinSpaced(n,1,n);
25    y = ( ( one * linsp.transpose() )
26          .cwiseMin( linsp * one.transpose()) ) * x;
```

Code 1: structured_matrix_vector.cpp

(a) What is the asymptotic complexity of Code 1 w.r.t. the problem size n?

(b) Write a `C++` function

```
void multAmin(const VectorXd &x, VectorXd &y);
```

which computes $\mathbf{y} = \mathbf{Ax}$ with complexity $O(n)$. You can test your implementation by comparing with the output from Code 1.

Hint: Make use of partial sums $\sum_{i=j}^{n} x_i$.

(c) Measure and compare the runtimes of your `multAmin` and Code 1 for $n = 2^4, \ldots, 2^{10}$. Report the minimum runtime in seconds with scientific notation using 3 decimal digits.

(d) Sketch the matrix B created by the following `C++` snippet:

```
190    MatrixXd B = MatrixXd::Zero(nn,nn);
191    for(unsigned int i = 0; i < nn; ++i) {
192        B(i,i) = 2;
193        if(i < nn-1) B(i+1,i) = -1;
194        if(i > 0) B(i-1,i) = -1;
195    }
196    B(nn-1,nn-1) = 1;
```

Code 2: structured_matrix_vector.cpp

(e) Write a short `Eigen`-based `C++` program, which computes ABe_j using A from Equation (1) and B from Code 2. Here e_j is the j-th unit vector in $\mathbb{R}^n$, $j = 1, \ldots, n$ and $n = 10$. Based on the result of the computation ABe_j, can you predict the relation between A and B?

Exercise 1.3. *Strassen algorithm*

Note: In this exercise focus on understanding the algorithms presented and calculating their complexity. The coding parts are optional, and their aim is only to allow you to check your understanding of the definitions and practice `C++`/`Eigen`. We recommend to not spend your time optimizing code in this exercise (for instance, don't worry about excessive copies of matrices in recursive calls).

Let $N = 2^n$ and let A, B be two matrices of size $N \times N$. Let $C = AB$. By definition of matrix product

$$c_{i,j} = \sum_k a_{i,k} b_{k,j}. \tag{2}$$

Equivalently, partitioning the matrices in equally sized sub-blocks

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right), C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right),$$

we have

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} \quad \text{for } i, j \in \{1, 2\}. \tag{3}$$

Strassen discovered[1] that defining the matrices

$$\begin{cases} M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 = (A_{21} + A_{22})B_{11} \\ M_3 = A_{11}(B_{12} - B_{22}) \\ M_4 = A_{22}(B_{21} - B_{11}) \\ M_5 = (A_{11} + A_{12})B_{22} \\ M_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \end{cases}$$

one can compute C as

$$\begin{cases} C_{11} = M_1 + M_4 - M_5 + M_7 \\ C_{12} = M_3 + M_5 \\ C_{21} = M_2 + M_4 \\ C_{22} = M_1 - M_2 + M_3 + M_6 \end{cases} \tag{4}$$

(a) Implement a function `mult` which computes the matrix product AB by applying directly the definition of (2).

(b) What is the asymptotic complexity of `mult`?

(c) Implement a function `mult_rec` which computes the matrix product recursively using (3) (the base case is when the blocks have size 1×1).

(d) What is the asymptotic complexity of `mult_rec`?

Hint: Indicate $f(n)$ as the number of elementary operations required to multiply two $2^n \times 2^n$ matrices, find a recurrence relation for f, and solve it.

(e) Implement a function `strassen` which computes the matrix product recursively using (4).

(f) What is the asymptotic complexity of `strassen`?

Hint: adapt the approach of Point (d).

[1] Volker Strassen, "Gaussian elimination is not optimal", Numerische Mathematik, vol. 13, 1969.