Programming C++

Project **Dense Matrices Access**

Status: 11. Juli 2019, 15:25

Supervisor: Prof.Dr. G. Haase, gundolf.haase@uni-graz.at

Dense Matrices Access:

We will have a look at the effect of access patterns of dense matrices and of a special dense matrix structure on the run time of the matrix vector product.
We will use a 1D memory layout to store a $n \times n$ dense matrix. Start with code (intro_function_densematrix[1]).

a) Write a class for the dense matrix that contains:

- A constructor with n (#rows) and m (#columns) as input parameters.
  Initialize the matrix elements in this constructor via

$$M_{i,j} = f(x_i) \cdot f(x_j)$$

  with $x_k = \frac{10k}{nm-1} - 5 \quad \forall k = 0, \ldots, nm-1$ with $nm = \max(n, m)$ and the Sigmoid[2] function $f(x) := (1 + \exp(-x))^{-1}$.
- Implement a (const[3]!) Method `Mult` for multiplying this matrix with a vector passed as input parameter, returning the resulting vector to your main code.
  Use rowise access to the matrix elements.
- Write a (const) Method `MultT` that multiplies the transosed matrix with a vector. Do not transpose the matrix. You only have to change the rowise access from the matrix above to a columnwise access.

b) Use your class and functions in the main function and check the results. Your main function should look like (plus the output of vectors `f1` and `f2`):

```
#include "mylib.h"
#include <iostream>
#include <cassert>
#include <vector>
using namespace std;

int main()
{
    DenseMatrix const M(5,3);      // Dense matrix, also initialized

    vector<double> const u{{1,2,3}};
    vector<double> f1 = M.Mult(u);

    vector<double> const v{{-1,2,-3,4,-5}};
    vector<double> f2 = M.MultT(v);

    return 0;
}
```

[1] https://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/intro_vector_densematrix.tar
[2] https://en.wikipedia.org/wiki/Sigmoid_function
[3] https://isocpp.org/wiki/faq/const-correctness#const-member-fns

c) Contruct a dense square matrix with n rows/columns, choose $n \in [10^3, 10^4]$ depending on the amount of memory in your computer.
Measure the run time for `Mult` and for `MultT` with the same non-zero vector as input parameter. Explain the difference in run time!
Our dense square matrix is symmetric by construction (why?), therefor the two resulting vectors have to be equal. Check this!

```
//   code snippet
#include <ctime>
...
{
    ...
    int const NLOOPS=100;          // the overall code should run approx. 10 sec.
    ...

    double         t1 = clock();   // start timer
    vector<double> f1 = M.Mult(u);
    for (int k=1; k<NLOOPS; ++k)
    {
        f1 = M.Mult(u);
    }
    t1 = (clock()-t1)/CLOCKS_PER_SEC/NLOOPS;
    ...
}
```

d) Write another class for a dense matrix which is defined as $M = u^T * v$ with row vectors u and v.

- Implement the same functionality as above with methods `Mult` and `MultT` but taking advantage of the tensor product structure of the matrix.
- Initialize your matrix with $u = v = sigmoid(x_k)$ for $k = 0, \ldots, n$, i.e., the matrix will be symmetric.
- Perform the same run time tests and checks as above.

Hints: `#include <cmath>`, `exp`, `#include <vector>`, `vector`