Projekt 6: Introduction to the *fast multipole method*

Motivation: fully populated matrices can be realize in a "data-sparse" way.
We consider matrices $M \in \mathbb{R}^{N\times N}$ given by

$$M_{ij} = \begin{cases} K(x_i, x_j), & i \neq j \\ 0 & i = j \end{cases} \qquad \text{with} \qquad x_i = (i-1)h, \qquad h = \frac{1}{N-1}. \tag{1}$$

Here, the function K is given by

$$K(x, y) = \ln|x - y|. \tag{2}$$

M is a fully populated matrix with N^2 non-zero entries. This implies that $O(N^2)$ arithmetic operations are needed for the matrix-vector multiplication $x \mapsto Mx$. The goal of the project is to get acquainted with (approximate) representations of M that only require $O(N \log N)$ memory. The representations given here are a (grossly) simplified version of the so-called multipole method that is employed in many-particle problems (e.g., when Newton's law is employed to simulate the movement of $N = 10^6$ or $N = 10^8$ stars in a galaxy).

1. (Chebyshev interpolation) The classical Chebyshev points on $(-1, 1)$ are defined as

$$\xi_j^{Cheb,n} = \cos\frac{(j + 1/2)\pi}{n+1}, \qquad j = 0, \ldots, n$$

The Chebyshev interpolation operator is correspondingly defined as $I_n f(x) = \sum_{i=0}^n f(\xi_i^{Cheb,n})\ell_i(x)$ with

$$\ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - \xi_j^{Cheb,n}}{x_i^{Cheb,n} - \xi_j^{Cheb,n}}, \qquad i = 0, \ldots, n.$$

For functions $\widetilde{K} = \widetilde{K}(x, y)$ the *tensor product* Chebyshev interpolation operator is defined as

$$(I_n^{2D}\widetilde{K})(x, y) = \sum_{i,j=0}^{n} \widetilde{K}(\xi_i^{Cheb,n}, \xi_j^{Cheb,n})\ell_i(x)\ell_j(y).$$

Show that I_n^{2D} can be viewed as a concatination of two univariate Chebyshev interpolations acting on the first and the second variable, i.e.,

$$(I_n^{2D} f)(x, y) = (I_n^x \circ I_n^y f)(x, y), \quad \text{where } (I_n^x f)(x, y) = (I_n f(\cdot, y))(x), \qquad (I_n^y f)(x, y) = (I_n f(x, \cdot))(y).$$

2. Show: For points $x_i \in [-1, 1]$, $i = 0, \ldots, N$, and $y_j \in [-1, 1]$, $j = 0, \ldots, N'$, the matrix $\widetilde{M} \in \mathbb{R}^{(N+1)\times(N'+1)}$ with entries $\widetilde{M}_{ij} = (I_n^{2D}\widetilde{K})(x_i, y_j)$ can be represented in the form

$$\widetilde{M} = WSV^\top$$

with $W \in \mathbb{R}^{(N+1)\times(n+1)}$, with $V \in \mathbb{R}^{(N'+1)\times(n+1)}$, $S \in \mathbb{R}^{(n+1)\times(n+1)}$. What is memory requirement to store for W, V, S? What are, if n is small compared to N and N', the storage savings compared to the full matrix $\widetilde{M}$? What is the rank of $\widetilde{M}$? Note that, if $\widetilde{K} - I_n^{2D}\widetilde{K}$ is small, then the error $M - \widetilde{M}$ (measured, e.g., in the Frobenius norm) is small.

3. The Chebyshev interpolation can be formulated for general intervals $[a,b]$ by scaling. Formally: if $L:[-1,1]\to[a,b]$ is the affine bijection $L(\xi)=\frac{a+b}{2}+\frac{b-a}{2}\xi$, then $(I_{n,[a,b]}f)(x)=(I_n(f\circ L))(L^{-1}(x))$ is the Chebyshev interpolation on $[a,b]$. Formulate the 2D Chebyshev interpolation operator $I^{2D}_{n,[a_1,b_1]\times[a_2,b_2]}$ for Chebyshev interpolation on $[a_1,b_1]\times[a_2,b_2]$. Given points $x_i\in[a_1,b_1]$ and points $y_j\in[a_2,b_2]$ formulate the construction of the matrices W, S, V for $\widetilde{M}\in\mathbb{R}^{N+1,N'+1}$ with entries $\widetilde{M}_{ij}=(I^{2D}_{n,[a_1,b_1]\times[a_2,b_2]}\widetilde{K})(x_i,y_j)$

4. For the function K of (2) the approximation error $K-I^{2D}_{n,[0,1]\times[0,1]}K$ on $[0,1]^2$ is large. However, K can be approximated *piecewise* very well by its Chebyshev interpolant. In fact, one can show (without proof): If two intervals I_x, $I_y\subset[0,1]$ satisfy the *admissibility condition*

$$\max\{\mathrm{diam}I_x,\mathrm{diam}I_y\}\le\frac{1}{2}\mathrm{dist}(I_x,I_y) \tag{3}$$

then the interpolation error on $I_x\times I_y$ satisfies, for some $C>0$ and $q\in(0,1)$

$$\|K-I^{2D}_{n,I_x\times I_y}K\|_{\infty,I_x\times I_y}\le Cq^n.$$

Test this assertion for the choices $I_x=[0,2^{-i}]$ and $I_y=[2\cdot2^{-i},3\cdot2^{-i}]$ for $i=0,\dots,2$ and $n=0,\dots,5$ by plotting $\|K-I^{2D}_{n,I_x\times I_y}K\|_{\infty,I_x\times I_y}$ versus n. Which values of q (for each i) do you observe? You may approximate the $\|\cdot\|_{\infty,I_x\times I_y}$-norm with a discrete norm with sufficiently many points.

For a subset $\mathcal{I}=\{i\in\mathbb{N}\,|\,I_{lower}\le i\le I_{upper}\}$ of the indices we denote by $\mathtt{BB}(\mathcal{I})$ the "bounding box", i.e., the smallest interval that contains the points x_i, $i\in\mathcal{I}$, (see (1) for the definition of the points x_i). In this way, we can introduce the notion of admissibility to subsets of the index set: Two subsets $\mathcal{I}$, $\mathcal{J}\subset\{1,\dots,N\}$ are called *admissible*, if

$$\max\{\mathrm{diam}\mathtt{BB}(\mathcal{I}),\mathrm{diam}\mathtt{BB}(\mathcal{J})\}\le\frac{1}{2}\mathrm{dist}(\mathtt{BB}(\mathcal{I}),\mathtt{BB}(\mathcal{J})). \tag{4}$$

If two sets $\mathcal{I}$ and $\mathcal{J}$ are admissible, then the corresponding matrix block $M(\mathcal{I},\mathcal{J})$ can be approximated by a matrix of rank $n+1$ in the above fashion by replacing the function K in the definition of $M(\mathcal{I},\mathcal{J})$ with $I^{2D}_{n,\mathtt{BB}(\mathcal{I})\times\mathtt{BB}(\mathcal{J})}K$. As in Problem 2 this leads to an approximation of the form

$$M^{approx}(\mathcal{I},\mathcal{J}):=WSV^\top,\qquad\text{with } W\in\mathbb{R}^{|\mathcal{I}|\times(n+1)},\qquad V\in\mathbb{R}^{|\mathcal{J}|\times(n+1)},\quad S\in\mathbb{R}^{(n+1)\times(n+1)}$$

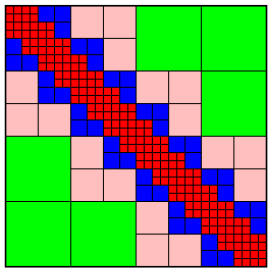
5. The goal, therefore, is to find a good block partitioning of the matrix M. Each matrix block is described by

```
struct block {
int I_lower, I_upper;// limits of the row indices of the block
int J_lower, J_upper;// limits of the column indices of the block
int rank;            // the rank of the block
double* V;           // V is an (I_upper - I_lower+1) * rank matrix
double* W;           // W is an  (J_upper - J_lower+1) * rank matrix
double* S;           // S is a rank * rank matrix
double* F;           // store the block as a full matrix F
                     // size :(I_upper-I_lower+1)*(J_upper-J_lower+1)
int typ;             // typ=EXACT: store the block as a full matrix F
                     // typ=APPROX: store the block as  W S V'
}
```

The following *greedy algorithm* creates a list blocklist of elements of type struct block such that each block is either admissible (and hence this block is represented in the form $WSV^\top$) or it is "small" (and is represented as a full matrix F).

```
build_blocks(I_lower,I_upper,J_lower,J_upper,blocklist){
if ( (I_upper==I_lower) || (J_upper==J_lower) ) {
 block newblock;
 newblock.I_lower = I_lower; newblock.I_upper = I_upper;
 newblock.J_lower = J_lower; newblock.J_upper = J_upper;
 newblock.typ = EXACT;  // block is stored as the matrix M
 append_to_blocklist(blocklist,newblock);
 }
else {
 if admissible(I_lower,I_upper,J_lower,J_upper) {
 block newblock;
 newblock.I_lower = I_lower; newblock.I_upper = I_upper;
 newblock.J_lower = J_lower; newblock.J_upper = J_upper;
 newblock.typ = APPROX; // block is stored as W S V'
 append_to_blocklist(blocklist,newblock);
 } else {
   I_mid = (I_lower+I_upper)/2;
   J_mid = (J_lower+J_upper)/2;
   build_blocks(I_lower,I_mid  ,J_lower,J_mid,blocklist);
   build_blocks(I_mid+1,I_upper,J_lower,J_mid,blocklist);
   build_blocks(I_lower,I_mid  ,J_mid+1,J_upper,blocklist);
   build_blocks(I_mid+1,I_upper,J_mid+1,J_upper,blocklist);
   }
}
}
```

The block structure that is created by this algorithm looks essentially like this:



Write a program to create this block list. For that, use the data structures that can be found at `http://www.math.tuwien.ac.at/~melenk/teach/computernumerik_WS1819`. Proceed in several steps:

1. Write the missing routines append_to_blocklist and admissible.
2. The blocklist is created in two steps. In the first step, the routine build_blocks(1,N,1,N,blocklist) is called to actually create the block list (without creating the matrices F, V, W, S).
3. Then, the matrices V, W, S or F for each block are created. To do so, write a program that loops through blocklist and creates the corresponding matrix for each block, i.e.,
   - if typ = EXACT, then the matrix F is filled with $M|_{\mathcal{I}\times\mathcal{J}}$;
   - if typ = APPROX, then the matrices W, S, V are created using Chebyshev interpolation.

6. The preceeding problem created a blockwise approximation of the exact matrix M. We now check the accuracy and the memory requirement for blocklist of the approximation.

a) Compute, for $n = 0, \ldots, 5$ and $N = 2^i$, $i = 2, \ldots, 15$, the memory requirement of blocklist by creating blocklist and then looping through it and summing the memory requirement for all blocks. Plot your result (memory versus N for the different n).

b) Determine, for $N = 2^i$, $i = 6, \ldots, 10$ and $n = 0, \ldots, 5$, the accuracy of your approximation in the Frobenius norm. Proceed as follows: Loop through blocklist , create for each block of type APPROX the matrix $M|_{\mathcal{I}\times\mathcal{J}}$ exactly and compute $\|M|_{\mathcal{I}\times\mathcal{J}} - WSV'\|_F^2$. Sum these errors over all blocks. Plot your result (accuracy versus N for different n).

Remark/extra problem: Given the block structure, the *best* approximation (in the Frobenius norm) would be to perform an SVD of the exact matrix blocks and truncate to a rank-$(n+1)$ matrix. To compare with the Chebyshev approximation proceed as follows: Loop through blocklist , create for each block of type APPROX the matrix $M|_{\mathcal{I}\times\mathcal{J}}$ exactly, compute its SVD $M|_{\mathcal{I}\times\mathcal{J}} = Q\Sigma O^\top$ and take the economy size SVD of rank $n+1$, i.e., $\widetilde{M} := Q(:, [1:n+1])\Sigma([1:n+1],[1+n+1])O(: \ : [1:n+1])^\top$. Compute the error $\|M|_{\mathcal{I}\times\mathcal{J}} - \widetilde{M}\|_F^2$ (This is simply the sum of the squares of the neglected sigular values.) Sum these errors over all blocks. Plot your result (accuracy versus N for different n). Compare with the approximation obtained by Chebyshev interpolation.