4M053 : Grands systèmes linéaires
M.-S.Dupuy & X.Claeys

Project

This project must be conducted individualy. It must be uploaded on the moodle page no later than Sunday the 1rst of January 2024 at 22 :00.

Your work shall take the form of a .zip file containing the following items :
— the source code of your work,
— a readme file sumarizing the content of your work and the way to use your source code.

You are expected to devise a test protocole upon your own intiative so as to make sure that your work is reliable, bug-free and conforms with the subject below. During the examination, you will be asked to report on your test protocole. It is assumed that TP1, TP2 and TP3 have been completed.

Low rank perturbation

In this exercise, we wish to model square matrices $\mathrm{A} \in \mathbb{R}^{n\times n}$ defined as the sum of a sparse matrix and a low rank matrix i.e. taking the form

$$\mathrm{A} = \mathrm{D} + \sum_{j=0}^{r-1} \boldsymbol{u}_j \boldsymbol{v}_j^\top \tag{1}$$

where $\boldsymbol{u}_j, \boldsymbol{v}_j \in \mathbb{R}^n$ are vectors (hence modelled by `std::vector<double>`) and $\mathrm{D} \in \mathbb{R}^{n\times n}$ is a sparse matrix (modelled by a `MapMatrix<double>`). We are particularly interested in the situation where $r \ll n$ and even $r = \mathcal{O}(1)$. These matrices are finite dimensional counterparts of operators of Fredholm type. As a consequence, we shall call them *Fredholm matrices*.

Question 1 Implement a class named `FredholmMatrix` modelling Fredholm matrices as described above. We only focus on square matrices i.e. number of rows = number of columns. This class shall fit the requirements below regarding its data members, member functions and friend functions.

Data members :

- `int n` : the size of the matrix (= number of rows = number of columns)
- `MapMatrix<double> diag:` a sparse matrix modelling the contribution D in (1)
- `std::vector< std::vector<double> > lru:` representing the vectors $\{\boldsymbol{u}_j\}_{j=0}^{r-1}$ in (1)
- `std::vector< std::vector<double> > lrv:` representing the vectors $\{\boldsymbol{v}_j\}_{j=0}^{r-1}$ in (1)

Member functions :

- Constructor `FredholmMatrix(const int& n0 = 0)` initialising `n` to `n0` where The sparse and low rank parts shall be considered empty.
- Copy constructor and copy assignement operator `=`.
- Member function `void insert(const int& j, const int& k, const double& val)` that adds the value `v` to the coefficients of `diag` at position `(j,k)`. It shall have an effect equivalent to $\mathrm{D}_{j,k} = \mathrm{D}_{j,k} + \mathtt{val}$.
- Member function `void insert(const std::vector<double>& u,const std::vector<double>& v)` that appends `u` to `lru`, and appends `v` to `lrv`. A call to this member function will increment the size of `lru` and `lrv` by 1.

- Operator `std::vector<double> operator*(const std::vector<double>&)` that models the matrix-vector product. This operation should admit linear algorithmic complexity.
- Accessor operator `double operator()(const int& j, const int& k)` returning the read-only value of the coefficient $\mathrm{A}_{j,k}$.

Friend functions :

- Output stream operator `std::ostream& operator<<(std::ostream&, const FredholmMatrix&)` that prints a Fredholm matrix into the terminal.
- An iterative solver function `std::vector<double> MinResSolve(const FredholmMatrix& A, const std::vector<double>& b)` that returns the solution x to the linear system $\mathrm{A}x = b$ treated by means of the `MinRes` method described below.

The Minimal Residual (MinRes) method is an iterative strategy that approximates the solution to a linear system $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ (where $\mathrm{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$) by a sequence of iterates $\boldsymbol{x}^{(k)}$, starting from an initial guess $\boldsymbol{x}^{(0)}$ (typically $\boldsymbol{x}^{(0)} = 0$ is chosen), computed through the following recurrence relation

$$\begin{aligned}
\boldsymbol{r}^{(k)} &= \boldsymbol{b} - \mathrm{A}\boldsymbol{x}^{(k)},\\
\alpha_k &= (\boldsymbol{r}^{(k)})^\top \mathrm{A}\boldsymbol{r}^{(k)} / |\mathrm{A}\boldsymbol{r}^{(k)}|_2^2,\\
\boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{r}^{(k)}.
\end{aligned}$$

Low rank approximation

In the sequel, those matrix of the form (1) where $\mathrm{D} = 0$ shall be referred to as *low rank matrices*. We wish to investigate the cross-approximation algorithm that provides an approximation of a densely populated matrix by means of a low rank matrix.

For a densely populated matrix $\mathrm{B} \in \mathbb{R}^{n\times n}$ (represented by an object of type `DenseMatrix`), let us denote $\mathrm{B}(:,k) \in \mathbb{R}^{n\times 1}$ the k-th column of B, and $\mathrm{B}(j,:) \in \mathbb{R}^{1\times N}$ the j-th row of of B. For a sequence of positive numbers $r_{j,k}, j, k = 1 \dots n$, we shall denote $(p,q) = \mathrm{argmax}_{j,k=1\dots N}(r_{j,k})$ the pair (p,q) that reaches the maximum $r_{p,q} = \max_{j,k=1\dots n}(r_{j,k})$. The cross-approximation algorithm constructs an approximation of a dense matrix B under the low-rank form $\tilde{\mathrm{B}}_r = \sum_{j=0}^{r-1} \boldsymbol{u}_j \boldsymbol{v}_j^\top$ by iteratively defining $\boldsymbol{u}_j \boldsymbol{v}_j$ according to the procedure below.

Algorithm 1 Cross approximation algorithm

function CrossApprox(B)
  $\mathrm{R} = \mathrm{B}$
  $\tilde{\mathrm{B}} = 0$
  $j_\star = k_\star = 0$
  for $p = 0 \dots r-1$ **do**
    $(j_\star, k_\star) = \mathrm{argmax}_{j,k=1\dots N} |\mathrm{R}(j,k)|$
    $\boldsymbol{u}_p = \mathrm{R}(:,k_\star) / \mathrm{R}(j_\star, k_\star)$
    $\boldsymbol{v}_p = \mathrm{R}(j_\star, :)^\top$
    $\mathrm{R} = \mathrm{R} - \boldsymbol{u}_p \boldsymbol{v}_p^\top$
    $\tilde{\mathrm{B}} = \tilde{\mathrm{B}} + \boldsymbol{u}_p \boldsymbol{v}_p^\top$
  end for
return $\tilde{\mathrm{B}}$
end function

Question 2 Write a function `FredholmMatrix CrossApproximation(const DenseMatrix& B,const int& r)` that takes a fully populated matrix B and a rank r as input, and returns a low rank approximation $\tilde{\mathrm{B}}_r$ in `FredholmMatrix` format constructed according to a cross approximation algorithm truncated at iteration r.

Question 3 Write a function `double FrobeniusNorm(const DenseMatrix&)` that computes the Frobenius norm of a densely populated matrix. Likewise write a function `double FrobeniusNorm(const FredholmMatrix&)` that computes the Frobenius norm of a matrix admitting the form (1).

Question 4 Let us denote $\|\mathrm{A}\|_{\mathrm{F}}$ the Frobenius norm of a matrix A. For a given size $n > 0$ consider the matrix $\mathrm{B} = (\mathrm{B}_{j,k})$ where $\mathrm{B}_{j,k} = \exp(-(j-k)^2/n^2)$. Write a function `void PlotGraph(const std::size_t& n)` that takes a size n as input, and plots the function $\log(r) \mapsto \log(\|\mathrm{B} - \tilde{\mathrm{B}}_r\|_{\mathrm{F}})$ for $r = 1, \ldots, n$ producing an image file in `pdf` format named `graph.pdf`. For producing the plots, you may rely on intermediate scripts (written e.g. in `Python` and using `matplotlib`) called from your `C++` program.

Question 5 Write a function `DenseMatrix LoadDenseMatrix(const std::string& filename)` that reads a file named `filename` containing data describing a real valued densely populated matrix under the format described below, assembles the corresponding `DenseMatrix` object, and returns it.

Question 6 Write a function `void PlotGraph(const std::string& filename)` that reads the file named `filename` that describes a densely populated matrix B under the format described below, and plots the function $\log(r) \mapsto \log(\|\mathrm{B} - \tilde{\mathrm{B}}_r\|_{\mathrm{F}})$ for $r = 1, \ldots, n$ producing an image file in `pdf` format named `graph.pdf`. You may test this function on the file `matrix.txt` provided in the archive of this project.

Appendix : DenseMatrix file format

Below is the file format that we use to describe a densely populated matrix $\mathrm{A} = (a_{j,k})$ of size `nr`×`nc` taking the notation convention that $v_p = a_{j,k}$ for $p = j * \mathtt{nc} + k$.

```
#SIZE
nr nc
#DATA
v_0   v_1   v_2   ...  v_9
v_20  v_1   v_2   ...  v_19
⋮
...   ...   v_{nr×nr−1}
```

On this file format, we place at most 10 values per line which means that the last line of the file may contain less than 10 values. Below is an example. In the case where the matrix to be loaded is the following that admits 3 rows and 2 columns,

$$\mathrm{A} = \begin{bmatrix} 2.7 & 1.5 \\ 0.8 & 2.6 \\ 4.7 & 3.9 \end{bmatrix}$$

the corresponding matrix file writes :

```
#SIZE
3 2
#DATA
2.7 1.5 0.8 2.6 4.7 3.9
```