NUMERICAL LINEAR ALGEBRA
A.A. 2023/2024

Lecturer: Prof. P. F. Antonietti TA: Dott. M. Botti

WRITTEN TEST 23/01/2024

First Name: .... Last Name: ....

This exam has 2 questions, with subparts. You have **2 hours** to complete the exam. There are a total of 30 points available (sufficiency at 18 points). You cannot consult any notes, books or aids of any kind except for the codes implemented during the lab sessions. Write answers legibly in the additional sheets provided. Please **write your name** on the exam itself and on the extra sheets. Concerning the implementations using Eigen, upload only the `.cpp` main files. For the exercises requiring LIS, report the bash commands used to perform the computations and the obtained results in a unique `.txt` file. **Upload the files** following the instructions.

Exercise 1

1. Consider the following problem: find $\boldsymbol{x} \in \mathbb{R}^n$, such that $A\boldsymbol{x} = \boldsymbol{b}$, where $A \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are given. State under which conditions the mathematical problem is well posed.
2. Describe the LU factorization and its use to approximately solve the above linear system.
3. State the necessary and sufficient condition that guarantees existence and uniqueness of the LU factorization. For what classes of matrices the LU factorization exist and is unique?
4. Describe the main pivoting techniques and comment on the computational costs.
5. Download the sparse matrices `A.mtx`, `B.mtx`, and `C.mtx` from the Exam folder in webeep and save it on the `/shared-folder/iter_sol++` folder. Load the three matrices in a new file `exer1.cpp`. Define the block matrix $M = \begin{pmatrix} A & B^{\mathrm{T}} \\ B & C \end{pmatrix}$. Report on the `.txt` file the matrix size and the Euclidean norm of M. Is the matrix symmetric?

```
Matrix size M: 278X278
Norm of M: 459.79
Norm of M-M^t: 132.073
The matrix is not symmetric
```

6. Define an `Eigen` vector $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$ with size equal to the number of rows of M. Solve the linear system $M\boldsymbol{x} = \boldsymbol{b}$ using the LU decomposition method available in the `Eigen` library. Report on the `.txt` file the norm of the obtained absolute residual.

```
Solution with LU complete system:
absolute residual: 1.62069e-13
```

7. Using again the LU decomposition provided by `Eigen`, compute an approximation of the Schur complement of M with respect to the A block defined as the matrix $S = C - BA^{-1}B^{\mathrm{T}}$. Report on the `.txt` file the matrix size and the Euclidean norm of S.

```
Matrix size S: 36X36
Norm of S:9.7299
```

8. Solve the linear system

$$M\boldsymbol{x} = \begin{pmatrix} A & B^{\mathrm{T}} \\ B & C \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix} = \boldsymbol{b}$$

by exploiting the Schur complement S. First use the LU factorization method to approximate the solution of $S\boldsymbol{x}_2 = \boldsymbol{b}_2 - BA^{-1}\boldsymbol{b}_1$. Then compute the approximate solution of $A\boldsymbol{x}_1 = \boldsymbol{b}_1 - B^{\mathrm{T}}\boldsymbol{x}_2$. Report the norm of the absolute residual $\boldsymbol{r} = \boldsymbol{b} - M * [\overline{\boldsymbol{x}}_1, \overline{\boldsymbol{x}}_2]^{\mathrm{T}}$, where $\overline{\boldsymbol{x}}_1$ and $\overline{\boldsymbol{x}}_2$ are the computed approximations of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively.

```
Solution with Schur complement
absolute residual: 1.41566e-13
comparison solutions : 1.13901e-13
```

Solution (full c++ implementation):

```
#include <Eigen/SparseCore>
#include <Eigen/SparseLU>
#include <iostream>
#include <string>
#include <unsupported/Eigen/SparseExtra>

int main(int argc, char** argv){
  //using namespace Eigen;
  using SpMat = Eigen::SparseMatrix<double>;
  using SpVec = Eigen::VectorXd;

  // Read matrices
  SpMat A, B, C;
  Eigen::loadMarket(A, "A.mtx");
  Eigen::loadMarket(B, "B.mtx");
  Eigen::loadMarket(C, "C.mtx");
  // Create matrix M from blocks
  Eigen::MatrixXd MM = Eigen::MatrixXd::Zero(A.rows()+C.rows(),A.cols()+C.cols());
  MM.topLeftCorner(A.rows(),A.cols()) = A;
  MM.bottomLeftCorner(C.rows(),A.cols()) = B;
  MM.topRightCorner(A.rows(),C.cols()) = B.transpose();
  MM.bottomRightCorner(C.rows(),C.cols()) = C;
  SpMat M = MM.sparseView();
  std::cout<<"Matrix size M: "<<M.rows()<<"X"<<M.cols()<<std::endl;
  std::cout<<"Norm of M: "<<M.norm()<<std::endl;
  SpMat SK = SpMat(M.transpose()) - M;  // Check symmetry
  std::cout << "Norm of M-M^t: " << SK.norm() << std::endl;

  // Create Rhs b
```

```
  SpVec b = SpVec::Ones(M.rows());
  SpVec x(M.rows());
  // Solve monolithically with SparseLU
  Eigen::SparseLU<Eigen::SparseMatrix<double> > solvelu;
  solvelu.compute(M);
  if(solvelu.info()!=Eigen::Success) {
      std::cout << "cannot factorize the matrix" << std::endl;
      return 0;
  }
  x = solvelu.solve(b);
  std::cout << "Solution with LU complete system:" << std::endl;
  std::cout << "absolute residual: "<<(b-M*x).norm()<<std::endl<<std::endl;

  // Compute Schur complement
  Eigen::SparseLU<Eigen::SparseMatrix<double> > solveA;
  solveA.compute(A);
  if(solveA.info()!=Eigen::Success) {
      std::cout << "cannot factorize the matrix" << std::endl;
      return 0;
  }
  SpMat S = C - B*(solveA.solve(SpMat(B.transpose())));
  std::cout << "Matrix size S: " <<S.rows()<<"X"<<S.cols()<<std::endl;
  std::cout << "Norm of S:" <<S.norm()<<std::endl;

    // Solve in two step exploiting the Schur complement
  Eigen::SparseLU<Eigen::SparseMatrix<double>> solveS;
  solveS.compute(S);
  if(solveS.info()!=Eigen::Success) {
      std::cout << "cannot factorize the matrix" << std::endl;
      return 0;
  }
  SpVec x2(S.rows()), x1(A.rows());
  x2 =solveS.solve(b.tail(S.rows()) - B*(solveA.solve(b.head(A.rows()))));
  x1 = solveA.solve(b.head(A.rows()) - B.transpose()*x2);
  SpVec xS(M.rows());
  xS.head(A.rows()) = x1;
  xS.tail(C.rows()) = x2;
  std::cout << "Solution with Schur complement"<<std::endl;
  std::cout << "absolute residual: "<<(b-M*xS).norm()<<std::endl;
  std::cout << "comparison solutions : " << (x-xS).norm() << std::endl;
  return 0;
}
```

Exercise 2

1. Consider the following eigenvalue problem: $A\boldsymbol{x} = \lambda\boldsymbol{x}$, where $A \in \mathbb{R}^{n\times n}$ is given. Describe the inverse power method for the numerical approximation of the smallest in modulus eigenvalue of A. Introduce the notation and discuss the applicability conditions.

2. Write the (pseudo) algorithm.

3. State the main theoretical result.

4. Let A be a 100×100 tetradiagonal matrix defined such that

$$A = \begin{pmatrix} 8 & -4 & -1 & 0 & 0 & \dots & 0 \\ -2 & 8 & -4 & -1 & 0 & \dots & 0 \\ 0 & -2 & 8 & -4 & -1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & -1 \\ \vdots & \vdots & \vdots & 0 & -2 & 8 & -4 \\ 0 & 0 & \dots & 0 & 0 & -2 & 8 \end{pmatrix}.$$

   In a new file called `exer2.cpp`, define the matrix A in the sparse format. Report $\|A\|$ on the `.txt` file, where $\|\cdot\|$ denotes the Euclidean norm.

```
Norm of A: 92.0761
```

5. Solve the eigenvalue problem $A\boldsymbol{x} = \lambda\boldsymbol{x}$ using the proper solver provided by `Eigen`. Report on the `.txt` file the computed smallest and largest (in modulus) eigenvalues of A.

```
 lamda_min = 1.91332
 lambda_max = 12.999
```

6. Using the `unsupported/Eigen/SparseExtra` module, export matrix A in the matrix market format (save as `Aex2.mtx`) and move it to the folder `lis-2.0.34/test`. Using the proper iterative solver available in the LIS library compute the largest eigenvalue λ_{max} of A up to a tolerance of 10^{-7}. Report on the `.txt` file the computed eigenvalue and the number of iterations required to achieve the prescribed tolerance.

```
mv Aex2.mtx  ../lis-2.0.34/test
mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis etest1.c -o eigen1

mpirun -n 4 ./eigen1 Aex2.mtx eigvec.mtx hist.txt -e pi -emaxiter 50000 -etol 1.e-7
Power: eigenvalue           = 1.299901e+01
Power: number of iterations = 39800
Power: relative residual    = 9.997934e-08
```

7. Find a `shift` $\mu \in \mathbb{R}$ yielding an acceleration of the previous eigensolver. Report μ and the number of iterations required to achieve a tolerance of 10^{-7}.

```
mpirun -n 4 ./eigen1 Aex2.mtx eigvec.mtx hist.txt -e pi -emaxiter 20000
 -etol 1.e-7 -shift 7.0

Computed eigenvalue 1.299901e+01 in 19920 iterations of the Power method.
```

8. Using the proper iterative solver available in the LIS library compute the smallest eigenvalue λ_{min} of A up to a tolerance of 10^{-7}. Report the computed eigenvalue and the number of iterations required to achieve the prescribed tolerance.

```
mpirun -n 4 ./eigen1 Aex2.mtx eigvec.mtx hist.txt -e ii -emaxiter 10000 -etol 1e-7
Inverse: eigenvalue           = 1.913297e+00
Inverse: number of iterations = 1348
Inverse: relative residual    = 9.961482e-08
```

Solution (full c++ implementation):

```cpp
#include <iostream>
#include <Eigen/Sparse>
#include <Eigen/Dense>
#include <unsupported/Eigen/SparseExtra>
using namespace Eigen;

int main(int argc, char** argv)
{
  using SpMat=Eigen::SparseMatrix<double>;
  using SpVec=Eigen::VectorXd;

  // Create matrix and vectors
  int n = 100;
  SpMat A(n,n);
  for (int i=0; i<n; i++) {
      A.coeffRef(i, i) = 8;
      if(i>0) A.coeffRef(i, i-1) = -2;
      if(i<n-1) A.coeffRef(i, i+1) = -4;
      if(i<n-2) A.coeffRef(i, i+2) = -1;
   }
  // Matrix properties
  std::cout << "Norm of A: " << A.norm() << std::endl;

  // Compute eigenvalues of A
  MatrixXd Af;
  Af = MatrixXd(A);
  EigenSolver<MatrixXd> eigensolver(Af);
  if (eigensolver.info() != Eigen::Success) abort();
  std::cout << "The eigenvalues of A are:\n" << eigensolver.eigenvalues() << std::endl;

  // Export matrix
  std::string matrixFileOut("./Aex2.mtx");
  saveMarket(A, matrixFileOut);
  return 0;
}
```