NUMERICAL LINEAR ALGEBRA

A.A. 2022/2023

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

WRITTEN TEST 27/01/2023

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

This exam has 2 questions with subparts, corresponding to 30 available points. You have **2 hours** to complete the exam. You cannot consult aids of any kind except for the labs' codes. Write answers legibly in the additional sheets and show all of your work. **Write your name** on the exam itself and on all the extra sheets. Concerning the Eigen implementations, upload only the `.cpp` main files. For the exercises requiring LIS, report the bash commands used to perform the computations in a unique `.txt` file. **Upload the files** following the instructions.

Exercise 1

Consider the 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.

1. Describe the LU factorization of A. Introduce the employed notation.
2. State the necessary and sufficient conditions for the existence and uniqueness of the LU factorization.
3. Describe how the LU factorization is used to compute the approximate solution of $A\boldsymbol{x} = \boldsymbol{b}$.
4. State the computational costs and prove the result.
5. Describe how the incomplete LU factorization con be used as preconditioner to accelerate the convergence of a linear iterative method. Introduce the employed notation.
6. Download the square matrices `A_1.mtx` and `A_2.mtx` from the Exam folder in webeep and save them on the `lis-2.0.34/test` folder. Using the BICGSTAB iterative solver available in the LIS library, compute the approximate solution of the linear systems $A_1\boldsymbol{y} = \boldsymbol{b}$ and $A_2\boldsymbol{x} = \boldsymbol{y}$ up to a tolerance of 10^{-10}, assuming that $\boldsymbol{y} = (1, 1, \ldots, 1)^{\mathrm{T}}$. Report on the sheet the iteration counts and the relative residuals at the last iteration.

```
mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis test1.c -o test1
mpirun -n 4 ./test1 A_1.mtx 2 sol.txt hist.txt -i bicgstab -tol 1e-10
BICGSTAB:  number of iterations = 90 , relative residual    = 6.691855e-11

mpirun -n 4 ./test1 A_2.mtx 1 sol.txt hist.txt -i bicgstab -tol 1e-10
BICGSTAB:  number of iterations = 20 , relative residual    = 9.670711e-11
```

7. Explore the *Incomplete Lower-Upper (ILU) factorization* method available in LIS in order to define proper preconditioners for the previous linear systems. Report the number of iterations required to reduce the residual below than 10^{-10} and compare with the results obtained by combining the ILU preconditioner with the Additive Schwarz method.

```
mpirun -n 4 ./test1 A_1.mtx 2 sol.txt hist.txt -i bicgstab -tol 1e-10 -p ilu
BICGSTAB:  number of iterations = 40 , relative residual    = 6.526092e-11

mpirun -n 4 ./test1 A_2.mtx 1 sol.txt hist.txt -i bicgstab -tol 1e-10
-p ilu -ilu_fill 2
BiCGSTAB: number of iterations = 8 , relative residual    = 1.919550e-11

mpirun -n 4 ./test1 A_1.mtx 2 sol.txt hist.txt -i bicgstab -tol 1e-10 -p ilu
-adds true -adds_iter 3
BICGSTAB:  number of iterations = 15 , relative residual   = 3.521106e-11

mpirun -n 4 ./test1 A_2.mtx 1 sol.txt hist.txt -i bicgstab -tol 1e-10
-p ilu -ilu_fill 2 -adds true
BiCGSTAB: number of iterations = 4 , relative residual    = 5.910363e-11
```

8. Using again the BICGSTAB solver implemented in LIS, solve the linear system $A_1 A_2 \boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$. Note that the previous problem can be equivalently written as: find $\boldsymbol{y}$ such that $A_1 \boldsymbol{y} = \boldsymbol{b}$ and than find $\boldsymbol{x}$ such that $A_2 \boldsymbol{x} = \boldsymbol{y}$.

```
mpirun -n 4 ./test1 A_1.mtx 1 y.mtx hist.txt -i bicgstab -tol 1e-10
BICGSTAB:  number of iterations = 91 , relative residual    = 7.096111e-11

mpirun -n 4 ./test1 A_2.mtx y.mtx x.mtx hist.txt -i bicgstab -tol 1e-10
BICGSTAB:  number of iterations = 24 , relative residual    = 4.717752e-11
```

9. Move the matrices `A_1.mtx` and `A_2.mtx` on the `/shared-folder/iter_sol++` directory. Load the matrices in the new `exer1.cpp` file using the `unsupported/Eigen/SparseExtra` module and compute $A = A_1 A_2$. Report on the sheet the matrix size, the number of non-zero entries, and the the Euclidean norm of A.

```
Matrix size:420 X 420
Non zero entries:21537
Norm of A: 381.32
```

10. Solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$, using both the `SparseLU` method available in `Eigen` and the BICGSTAB method implemented in the `bcgstab.hpp` template. For the BICGSTAB method fix a maximum number of iterations sufficient to reduce the residual below than 10^{-10} and use the diagonal preconditioner provided by `Eigen`. Report on the sheet $\|\overline{x}_{LU}\|$, $\|\overline{x}_{BCG}\|$, and $\|\overline{x}_{LU} - \overline{x}_{BCG}\|$, where $\overline{x}_{LU}$ and $\overline{x}_{BCG}$ are the approximate solution obtained with the LU and BICGSTAB methods, respectively.

```
Solution with Eigen LU:
Norm of approximate solution : 891.129

Solution with BiConjugate Gradient Stab:
Iterations performed: 242
Tolerance achieved  : 5.11773e-11
Norm of approximate solution: 891.129

Difference between LU and BCGSTAB: 8.29974e-09
```

Full implementation of Exercise 1

```cpp
#include <iostream>
#include <Eigen/SparseCore>
#include <Eigen/SparseLU>
#include <unsupported/Eigen/SparseExtra>
#include <Eigen/IterativeLinearSolvers>
#include "bcgstab.hpp"

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

  // Load matrix
  SpMat A_1;
  SpMat A_2;
  Eigen::loadMarket(A_1, "A_1.mtx");
  Eigen::loadMarket(A_2, "A_2.mtx");
  SpMat A = A_1*A_2;

  // Matrix properties
  std::cout << "Matrix size:" << A.rows() << " X " << A.cols() << std::endl;
  std::cout << "Non zero entries:" << A.nonZeros() << std::endl;
  std::cout << "Norm of A: " << A.norm() << std::endl;
  std::cout << "Norm of A_1 " << A_1.norm() << " and A_2 " << A_2.norm() << std::endl;

  // Create Rhs b
  SpVec b = SpVec::Ones(A.rows());
  SpVec x1(A.cols());
  Eigen::DiagonalPreconditioner<double> D(A);

  // First with Eigen SparseLU solver
  Eigen::SparseLU<Eigen::SparseMatrix<double> > solvelu;       // LU factorization
  solvelu.compute(A);
  if(solvelu.info()!=Eigen::Success) {                          // sanity check
      std::cout << "cannot factorize the matrix" << std::endl;
      return 0;
  }
  x1 = solvelu.solve(b);                                        // solving
  std::cout << "Solution with Eigen LU:" << std::endl;
  std::cout << "Norm of approximate solution : "<< x1.norm() << std::endl;

  // Now with BICGSTAB
  double tol = 1.e-10;                 // Convergence tolerance
  int result, maxit = 1000;            // Maximum iterations
  SpVec x2 = 0*x1;
  result = BiCGSTAB(A, x2, b, D, maxit, tol);           // Call BCG function
  std::cout << "Solution with BiConjugate Gradient Stab: " << std::endl;
  std::cout << "Iterations performed: " << maxit << std::endl;
  std::cout << "Tolerance achieved  : " << tol << std::endl;
  std::cout << "Norm of approximate solution: "<< x2.norm() << std::endl;
  std::cout << "Difference between LU and BCGSTAB: " << (x1-x2).norm() << std::endl;
  return result;
}
```

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 power method for the numerical approximation of the largest in modulus eigenvalue of A. Introduce the notation, describe the algorithm, and state the applicability conditions.

2. Prove that, under the condition stated above, the method converges.

3. Describe the inverse power method for the numerical approximation of the smallest in modulus eigenvalue of A. Introduce the notation, describe the algorithm, and state the applicability conditions.

4. Discuss the computational costs of both the power and inverse power methods.

5. Describe the deflation technique. Present the algorithm and the applicability conditions.

6. Let A be a $n \times n$ symmetric matrix defined such that

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

Let $n = 200$. In a new file called `exer2.cpp`, define the matrix A in the sparse format. Report on the sheet $\|A\|$ where $\|\cdot\|$ denotes the Euclidean norm.

```
Non zero entries:796
Norm of A: 4005
```

7. Solve the eigenvalue problem $A\boldsymbol{x} = \lambda \boldsymbol{x}$ using the proper solver provided by the `Eigen` library. Report on the sheet the two smallest computed eigenvalues of A.

```
The two smallest eigenvalues of A are:
0.0953697
0.360085
```

8. Export matrix A in the matrix market format using the `unsupported/Eigen/SparseExtra` module (save it as `exer2.mtx`) and move it to the `lis-2.0.34/test` folder. Using the proper iterative solver available in the LIS library compute the smallest eigenvalue of A up to a tolerance of 10^{-12}. Report the computed eigenvalue and motivate the choice made.

```
mv exer2.mtx lis-2.0.34/test

mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis etest1.c -o eigen1
mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.0e-12

Computed eigenvalue 9.537019e-02 in 23 iterations of the Inverse Power method.
```

9. Approximate the second smallest eigenvalue of the **exer2.mtx** matrix up to a tolerance of 10^{-9}. Explore different inner iterative methods and preconditioners (at least 3 alternative strategies). Report on the sheet the iteration counts and the residual at the last iteration.

```
mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.e-9
-i bicgstab -shift 0.4
Inverse: eigenvalue           = 3.600936e-01
Inverse: number of iterations = 23
Inverse: relative residual    = 2.203956e-10

mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.e-9
-i cg -p ssor -shift 0.4
Inverse: eigenvalue           = 3.600936e-01
Inverse: number of iterations = 12
Inverse: relative residual    = 2.647718e-10

mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.e-9
-i gmres -p ilu -shift 0.4
Inverse: eigenvalue           = 3.600936e-01
Inverse: number of iterations = 12
Inverse: relative residual    = 2.648045e-10
```

Full implementation of Exercise 2

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

int main(int argc, char** argv)
{
  // Create matrix
  int n = 200;
  SparseMatrix<double> A(n,n);
  for (int i=0; i<n; i++) {
      A.coeffRef(i, i) = 2.0*(i+1);
      A.coeffRef(i,n-i-1) += 1.0;
      if(i>0) A.coeffRef(i, i-1) += -i;
      if(i<n-1) A.coeffRef(i, i+1) += -(i+1);
  }
  std::cout << "Matrix size:" << A.rows() << " X " << A.cols() << std::endl;
  std::cout << "Non zero entries:" << A.nonZeros() << std::endl;
  std::cout << "Norm of A: " << A.norm() << std::endl;

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

  // Export matrix
  std::string matrixFileOut("./exer2.mtx");
  saveMarket(A, matrixFileOut);
  return 0;
}
```