Numerical Linear Algebra
A.A. 2022/2023
Lecturer: Prof. P. F. Antonietti TA: Dott. M. Botti

Written test 12/06/2023

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, and show all of your work. 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 in a unique `.txt` file. **Upload the files** following the received instructions.

Exercise 1

1. Consider the following problem: find $\boldsymbol{x} \in \mathbb{R}^n$, $A\boldsymbol{x} = \boldsymbol{b}$, 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 Conjugate Gradient method. Introduce the notation, the algorithm, the interpretation of the scheme as a minimization problem. Recall the main theoretical results.
3. Describe the preconditioned Gradient Method (PCG) and state the main conditions the preconditioner P has to satisfy, and the main convergence result.
4. Describe the algebraic multigrid method as a preconditioning strategy to accelerate the convergence of PCG method.
5. Let A be a 800×800 symmetric, pentadiagonal matrix defined such that

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

   In a new file called `exer1.cpp`, define the matrix A in the sparse format. Report on the sheet $\|\boldsymbol{v}\|_A = \boldsymbol{v}^{\mathrm{T}} A \boldsymbol{v}$, where $\boldsymbol{v}$ is such that $v_i = 1$ for all $0 \le i < 800$.

```
norm_A(v) = 6
```

6. Define a 800×1 `Eigen` vector $\boldsymbol{b} = (1, 0, 1, 0 \ldots, 1, 0)^{\mathrm{T}}$. Report on the sheet $\|\boldsymbol{b}\|$.

```
norm of rhs: 20
```

7. Solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ using the Conjugate Gradient method implemented in the `cg.hpp` template. Fix a maximum number of iterations which is sufficient to reduce the (relative) residual below than 10^{-12}. Use the diagonal preconditioner provided by `Eigen`. Report on the sheet the iteration counts and the relative residual at the last iteration.

```
CG method
iterations performed: 458
tolerance achieved  : 6.61304e-13
```

8. Using the `unsupported/Eigen/SparseExtra` module, export matrix A and vector $\boldsymbol{b}$ in the matrix market format (save as `matA.mtx` and `vecb.mtx`) and move them to the folder `lis-2.0.34/test`. Solve the same linear system using the Conjugate Gradient method of the LIS library setting a tolerance of 10^{-12}. Report on the sheet the iteration counts and the relative residual at the last iteration.

```
mv matA.mtx vecB.mtx lis-2.0.34/test
mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis test1.c -o test1

mpirun -n 4 ./test1 matA.mtx vecB.mtx sol.mtx hist.txt -i cg -tol 1.e-12
--> CG: number of iterations = 458, relative residual    = 6.613703e-13
```

9. Explore the algebraic multigrid method available in the LIS library (`SA-AMG`) in order to define a proper preconditioner for the approximate solution of the previous linear system. Compare and comment the results.

```
Load the new LIS module: `module load lis/2.0.34`
gfortran test1.c -I${mkLisInc} -L${mkLisLib} -llis -o test1b

./test1b matA.mtx vecB.mtx sol.mtx hist.txt -i cg -p saamg
--> CG: number of iterations = 13, relative residual    = 5.278194e-14

Using the AMG preconditioning strategy the number of iterations required to
achieve the same tolerance is significantly reduced.
Since also the computational time is reduced, we can conclude that the SA-AMG
algorithm provides a good preconditioner for the considered linear system.
```

Solution (full c++ implementation):

```
#include <Eigen/SparseCore>
#include <iostream>
#include <fstream>
#include <unsupported/Eigen/SparseExtra>
#include <Eigen/IterativeLinearSolvers>
#include "cg.hpp"
```

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

  // Create matrix and vectors
  int n = 800;
  SpMat A(n,n);
  SpVec v = SpVec::Ones(A.rows());
  SpVec b = SpVec::Ones(A.rows());
  for (int i=0; i<n; i++) {
      A.coeffRef(i, i) = 4;
      if(i>0) A.coeffRef(i, i-1) = -1;
      if(i<n-1) A.coeffRef(i, i+1) = -1;
      if(i>1) A.coeffRef(i, i-2) = -1;
      if(i<n-2) A.coeffRef(i, i+2) = -1;
      if(i % 2 == 1) b(i) = 0;
   }

  // 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 A of v: " << v.dot(A*v) << std::endl;
  std::cout << "Alternative method: " << v.transpose()*(A*v) << std::endl;
  std::cout << "Norm of b: " << b.norm() << std::endl;

  // Parameters for CG solver
  double tol = 1.e-12;                  // Convergence tolerance
  int result, maxit = 1000;             // Maximum iterations

  // Create Rhs b
  SpVec x(A.cols());
  Eigen::DiagonalPreconditioner<double> D(A);

  // Solution with Conjugate Gradient method
  x=0*x;
  result = CG(A, x, b, D, maxit, tol);        // Solve system
  std::cout << " Eigen CG " << std::endl;
  std::cout << "iterations performed: " << maxit << std::endl;
  std::cout << "tolerance achieved  : " << tol << std::endl;

  // Export matrix and vector
  std::string matrixFileOut("./matA.mtx");
  saveMarket(A, matrixFileOut);

  saveMarketVector(b, "./vecB.mtx");
  FILE* out = fopen("vecB.mtx","w");
  fprintf(out,"%%%%MatrixMarket vector coordinate real general\n");
  fprintf(out,"%d\n", n);
  for (int i=0; i<n; i++) {
      fprintf(out,"%d %f\n", i ,b(i));
  }
  fclose(out);
  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, the algorithm, and the applicability conditions.

2. State the main theoretical results.

3. Comment of the computational costs.

4. Download the square matrice `Aexer2.mtx` from the Exam folder in webeep and save it on the `/shared-folder/iter_sol`++ folder. Report on the sheet the matrix size, the number of non-zero entries, and the the Euclidean norm of A. Is the matrix A symmetric?

```
Matrix size:64 X 64
Non zero entries:189
Norm of A: 148.761
Norm of skew-symmetric part: 0.561249
Norm of A^T*A: 3880.26

The matrix is not symmetric
```

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

```
 lamda_min = 0.0685231
 lambda_max = 30.6515
```

6. Solve the eigenvalue problem $A^{\mathrm{T}}A\boldsymbol{x} = \overline{\lambda}\boldsymbol{x}$ using the proper solver provided by the `Eigen` library. Report on the sheet the smallest and largest computed eigenvalues of A. Which is the relation between the eigenvalues of A and the eigenvalues of $A^{\mathrm{T}}A$?

```
 lamda_min = 0.0046876
 lambda_max = 939.518

The eigenvalues of AT*A are the square of the eigenvalues of A
```

7. Move matrix `Aexer2.mtx` to the `lis-2.0.34/test` folder. Using the proper iterative solver available in the LIS library compute the largest eigenvalue of A up to a tolerance of 10^{-10}. Report the computed eigenvalue and the number of iterations required to achieve the prescribed tolerance.

```
mpirun -n 4 ./eigen1 Aexer2.mtx ev.mtx hist.txt -e pi -emaxiter 9000 -etol 1e-10

Power: eigenvalue           = 3.065148e+01
Power: number of iterations = 8388
Power: relative residual    = 9.986513e-11
```

8. Compute the three smallest eigenvalue of the `Aexer2.mtx` matrix up to a tolerance of 10^{-12}. 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 Aexer2.mtx eigvec.mtx hist.txt -e ii -i gmres -p ilu
Inverse: eigenvalue           = 6.852311e-02
Inverse: number of iterations = 42
Inverse: relative residual    = 6.804523e-13

mpirun -n 4 ./eigen1 Aexer2.mtx eigvec.mtx hist.txt -e ii -p ssor -shift 0.1
Inverse: eigenvalue           = 1.238031e-01
Inverse: number of iterations = 113
Inverse: relative residual    = 9.052636e-13

mpirun -n 4 ./eigen1 Aexer2.mtx eigvec.mtx hist.txt -e ii -i bicgstab -shift 0.2
Inverse: eigenvalue           = 1.500000e-01
Inverse: number of iterations = 57
Inverse: relative residual    = 8.039871e-15
```

Solution (full c++ implementation):

```
#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;
  // Load matrix
  SpMat A;
  Eigen::loadMarket(A, "Aexer2.mtx");
  SpMat A2 = SpMat(A.transpose())*A;

  // Matrix properties of A
  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;
  SpMat B = SpMat(A.transpose()) - A;
  std::cout << "Norm of skew-symmetric part: " << B.norm() << std::endl;

  // Compute eigenvalues of A
  MatrixXd A_f;
  A_f = MatrixXd(A);
  EigenSolver<MatrixXd> eigensolver(A_f);
  if (eigensolver.info() != Eigen::Success) abort();
  std::cout << "The eigenvalues of A are:\n" << eigensolver.eigenvalues() << std::endl;
  // Compute eigenvalues of A^T*A
  SelfAdjointEigenSolver<MatrixXd> eigensolver2(A2);
  if (eigensolver2.info() != Eigen::Success) abort();
  std::cout << "The eigenvalues of A^T*A are:\n" << eigensolver2.eigenvalues() << std::endl;
  return 0;
}
```