Numerical Linear Algebra

A.A. 2022/2023

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

Written test 27/10/2022

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 Gradient Method. Introduce the notation, the algorithm, the interpretation of the scheme as a minimization problem. Recall the main theoretical results.
3. Describe the Conjugate Gradient method. Introduce the notation, the algorithm, the interpretation of the scheme as a minimization problem. Recall the main theoretical results.
4. Comment on the main differences between the Gradient and Conjugate Gradient Methods.
5. Describe the preconditioned Gradient and the Conjugate Gradient Methods and state the main conditions the preconditioner P has to satisfy.
6. Download the matrix `diffreact.mtx` from the Exam folder in webeep and save it on the `/shared-folder/iter_sol++` directory. Load the matrix in a new file called `exer1.cpp` using the `unsupported/Eigen/SparseExtra` module and check if the matrix is symmetric. Report on the sheet $\|A\|$ and $\|A_{SS}\|$ where A_{SS} is the skew-symmetric part of A and $\|\cdot\|$ is the Euclidean norm.

```
Matrix size:256 X 256
Non zero entries:3976
Norm of A: 117.201
Norm of skew-symmetric part: 5.24291e-15
```

7. Define an `Eigen` vector $\boldsymbol{b} = A\boldsymbol{x}^*$, where $\boldsymbol{x}^* = (1, 1, \ldots, 1)^{\mathrm{T}}$. Report on the sheet $\|\boldsymbol{b}\|$.

```
norm of rhs: 0.300252
```

8. Solve the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ using both the Gradient Method and Conjugate Gradient method (implemented in the `grad.hpp` and `cg.hpp` templates, respectively). Fix a maximum number of iterations which is sufficient to reduce the (relative) residual below than 10^{-8}. 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: 55
tolerance achieved  : 8.59431e-09
Gradient method
iterations performed: 7690
tolerance achieved  : 9.99267e-09
```

9. Solve the same linear system using the Conjugate Gradient method of the LIS library. Explore the Additive Schwarz method in order to define a proper preconditioner. Report on the sheet the iteration counts and the relative residual at the last iteration.

```
mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis test1.c -o test1

mpirun -n 4 ./test1 diffreact.mtx 2 sol.mtx hist.txt -i cg -tol 1.e-8
--> CG: number of iterations = 67, relative residual    = 4.162011e-09

mpirun -n 4 ./test1 diffreact.mtx 2 sol.mtx hist.txt -i cg -p ilu
-ilu_fill 2 -tol 1.e-8
--> CG: number of iterations = 46, relative residual    = 4.784002e-09

mpirun -n 4 ./test1 diffreact.mtx 2 sol.mtx hist.txt -i cg -p ilu
-ilu_fill 2 -adds true -tol 1.e-8
--> CG: number of iterations = 31, relative residual    = 7.265872e-09

mpirun -n 4 ./test1 diffreact.mtx 2 sol.mtx hist.txt -i cg -p ilu
-ilu_fill 2 -adds true -adds_iter 3 -tol 1.e-8
--> CG: number of iterations = 22, relative residual    = 5.184540e-09
```

10. Compare and comment the results obtained at the two previous points.

Solution (c++ implementation):

```
#include <iostream>
#include <Eigen/SparseCore>
#include <Eigen/IterativeLinearSolvers>
#include <unsupported/Eigen/SparseExtra>
#include "cg.hpp"
#include "grad.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;
  Eigen::loadMarket(A, "diffreact.mtx");

  // Check matrix properties
  std::cout << "Matrix size:" << A.rows() << " X " << A.cols() << std::endl;
  std::cout << "Non zero entries:" << A.nonZeros() << std::endl;
  SpMat B = SpMat(A.transpose()) - A;
  std::cout << "Norm of A: " << A.norm() << std::endl;
  std::cout << "Norm of skew-symmetric part: " << B.norm() << std::endl;

  double tol = 1.e-8;              // Convergence tolerance
  int result, maxit = 10000;       // Maximum iterations

  // Create Rhs b
  SpVec e = SpVec::Ones(A.rows());
  SpVec b = A*e;
  std::cout << "norm of rhs: " << b.norm() << std::endl;
  SpVec x(A.rows());

  // Create preconditioner
  Eigen::DiagonalPreconditioner<double> D(A);

  // Conjugate Gradient method with diagonal precond
  result = CG(A, x, b, D, maxit, tol);
  std::cout << "CG method " << std::endl;
  std::cout << "iterations performed: " << maxit        << std::endl;
  std::cout << "tolerance achieved  : " << tol          << std::endl;
  std::cout << "Error norm: "           << (x-e).norm() << std::endl;

  // Gradient method with diagonal precond
  x = 0*x; tol = 1.e-8; maxit = 10000;
  result = GRAD(A, x, b, D, maxit, tol);
  std::cout << "Gradient method " << std::endl;
  std::cout << "iterations performed: " << maxit        << std::endl;
  std::cout << "tolerance achieved  : " << tol          << std::endl;
  std::cout << "Error norm: "           << (x-e).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, the algorithm, and the applicability conditions.

2. State the main theoretical result.

3. Discuss how the power method can be suitably modified in order to approximate the smallest in modulus eigenvalue of A and comment on the computational costs.

4. Let A be a $n \times n$ symmetric, tridiagonal matrix defined such that

$$A = \begin{pmatrix} a_1 & 0.5 & 0 & 0 & \dots & 0 \\ 0.5 & a_2 & 0.5 & 0 & \dots & 0 \\ 0 & 0.5 & \ddots & \ddots & \dots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0.5 \\ 0 & 0 & \dots & 0 & 0.5 & a_n \end{pmatrix} \qquad \text{with } a_i = \left| \frac{n+1}{2} - i \right| + 1, \quad \text{for all } 1 \le i \le n;$$

Let $n = 99$. In a new file called `exer2.cpp`, define the matrix A in the sparse format. Report on the sheet a_1 and a_n.

```
a_1 = 50, a_99 = 50
```

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.

```
0.60999 and 50.2254
```

6. 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 largest eigenvalue of A up to a tolerance of 10^{-10}. 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 pi -etol 1.0e-10

Computed eigenvalue 5.022544e+01 in 782 iterations of the Power method.
```

7. Can you use the Inverse method to perform the above operation? Why?

```
Yes, by using the Inverse method with a large enough shift.
mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.0e-10 -i cg -shift 50

Computed eigenvalue 5.022544e+01 in 17 iterations of the Inverse method with shift.
It can be observed that the total computational time is less (but comparable)
than in the previous case.
```

8. Compute the smallest eigenvalue of the `exer2.mtx` matrix. Explore different inner iterative methods and preconditioners (at least 2 alternative strategies). Report on the sheet the iteration counts and the residual at the last iteration. Comment the results.

```
mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.e-10 -i cg -p ssor
Inverse: eigenvalue           = 6.099899e-01
```

```
Inverse: number of iterations = 21
Inverse: elapsed time         = 8.823625e-03 sec.
Inverse: relative residual    = 3.444828e-11

mpirun -n 4 ./eigen1 exer2.mtx eigvec.mtx hist.txt -e ii -etol 1.e-10 -i gmres -p ilu
Inverse: eigenvalue           = 6.099899e-01
Inverse: number of iterations = 21
Inverse: elapsed time         = 1.294417e-02 sec.
Inverse: relative residual    = 3.444834e-11
```

Solution (c++ implementation):

```
#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 = 99;
  SparseMatrix<double> A(n,n);
  for (int i=0; i<n; i++) {
      A.coeffRef(i, i) = std::abs((n-1)/2 - i) + 1.0;
      if(i>0) A.coeffRef(i, i-1) = 0.5;
      if(i<n-1) A.coeffRef(i, i+1) = 0.5;
  }
  std::cout << "a1 = " << A.coeffRef(0, 0) << ", a99 = " << A.coeffRef(n-1, n-1) << 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;
}
```