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;
}
```

Numerical Linear Algebra
A.A. 2022/2023

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

Exam simulation (Implementation part)

Exercise 1

Let A be an $n \times n$ matrix such that $A = B + C$, with

$$B = \begin{pmatrix} 2 & 1 & 0 & 0 & \dots & 0 \\ 1 & 4 & 2 & 0 & \dots & 0 \\ 0 & 2 & 6 & 3 & \dots & 0 \\ 0 & 0 & 3 & 8 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & n-1 \\ 0 & 0 & 0 & \dots & n-1 & 2n \end{pmatrix}; \qquad C = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & -1 \\ 0 & 0 & \dots & 0 & -2 & 0 \\ \vdots & \vdots & \iddots & -3 & 0 & 0 \\ 0 & 0 & \iddots & \iddots & \vdots & \vdots \\ 0 & -n+1 & 0 & \dots & 0 & 0 \\ -n & 0 & 0 & \dots & 0 & 0 \end{pmatrix}.$$

1. Let $n = 1000$. Define the matrix A using the `Eigen::SparseMatrix<double>` type.
2. Define an `Eigen` vector $\boldsymbol{b} = A\boldsymbol{x}^*$, where $\boldsymbol{x}^* = (1, 1, \dots, 1)^{\mathrm{T}}$.
3. Solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ using the Generalized Mininal Residual method (GMRES) implemented in the `gmres.hpp` template. Fix a maximum number of iterations equal to the linear system's size and assume a tolerance of 10^{-12} for the final residual. Use the diagonal preconditioner provided by the `Eigen::DiagonalPreconditioner<double>` function.
4. Compare the GMRES method with restart (`restart`$= 50$) and without restart. Comment the obtained results.

Solution:

```
#include <cstdlib>                        // System includes
#include <iostream>
#include <Eigen/SparseCore>
#include <Eigen/IterativeLinearSolvers>
#include "gmres.hpp"

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

  int n = 1000;
  SpMat A(n,n);                          // define matrix
  for (int i=0; i<n; i++) {
      A.coeffRef(i, i) = 2.0*(i+1);
      A.coeffRef(i,n-i-1) = -(i+1);
      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;

   // Create Rhs b
  SpVec e = SpVec::Ones(A.rows());
  SpVec b = A*e;
  SpVec x(A.rows());

  // Solve with GMRES method with restart
  double tol = 1.e-12;                 // Convergence tolerance
  int result, maxit = 1000;           // Maximum iterations
  int restart = 50;                        // Restart gmres
  Eigen::DiagonalPreconditioner<double> D(A); // Create diagonal preconditioner

  result = GMRES(A, x, b, D, restart, maxit, tol);
  std::cout << "GMRES with restart "    << std::endl;
  std::cout << "iterations performed: " << maxit << std::endl;
  std::cout << "tolerance achieved  : " << tol << std::endl;
  std::cout << "Error:                " << (x-e).norm()<< std::endl;

  // Solve with GMRES method without restart
  x=0*x; restart = 1000; maxit = 1000; tol = 1.e-12;
  result = GMRES(A, x, b, D, restart, maxit, tol);
  std::cout << "GMRES without restart " << 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

The aim of this exercise is to solve the eigenvalue problem $A\boldsymbol{x} = \lambda\boldsymbol{x}$ using the Library of Iterative Solvers for linear systems (LIS). Report the full list of bash commands required to perform the computations here below in a `.txt` file. Compile the LIS script using `mpi` and run the LIS executables using 4 processors.

1. Using `wget` and `gzip`, download and unzip the matrix `gr_30_30.mtx` from the matrix market website (https://math.nist.gov/MatrixMarket/).

2. Compute the largest (in absolute value) eigenvalue of the matrix that has been previously downloaded up to a tolerance of order 10^{-8}.

3. Compute the eight smallest (in absolute value) eigenvalues of the `gr_30_30.mtx` matrix and save the corresponding eigenvectors in a `.mtx` file. Explore different iterative methods and preconditioners (at least 3 alternative strategies) in order to achieve a precision smaller than 10^{-10}. Compare and comment the results.

Solution:

```
wget https://math.nist.gov/pub/MatrixMarket2/Harwell-Boeing/laplace/gr_30_30.mtx.gz

gzip -dk gr_30_30.mtx.gz
```

```
mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis etest1.c -o eigen1

mpirun -n 4 ./eigen1 gr_30_30.mtx eigvec.txt hist.txt -e pi -emaxiter 5000 -etol 1.0e-8

mpicc -DUSE_MPI -I${mkLisInc} -L${mkLisLib} -llis etest5.c -o eigen2

mpirun -n 4 ./eigen2 gr_30_30.mtx evals.mtx eigvecs.mtx res.txt iters.txt
-ss 8 -e si -p jacobi -etol 1.0e-10 -emaxiter 2000

mpirun -n 4 ./eigen2 gr_30_30.mtx evals.mtx eigvecs.mtx res.txt iters.txt
-e si -ie ii -ss 8 -i cg -p ilu ilu_fill 3 -etol 1.0e-10

mpirun -n 4 ./eigen2 gr_30_30.mtx evals.mtx eigvecs.mtx res.txt iters.txt
-e si -ie ii -ss 8 -i bicgstab -p ssor -etol 1.0e-10
```

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 $A\boldsymbol{x} = \boldsymbol{b}$ 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

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

   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;
}
```

NUMERICAL LINEAR ALGEBRA
A.A. 2022/2023

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

WRITTEN TEST 13/01/2023

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 general form of a linear iterative method for the approximate solution of $A\boldsymbol{x} = \boldsymbol{b}$ and describe the stopping criteria.

3. State the necessary and sufficient condition for convergence.

4. State and prove the sufficient condition for convergence.

5. Describe the Generalized Minimal Residual Method (GMRES). Recall the interpretation of the scheme as a Krylov subspace method and the main theoretical results.

6. Download the matrix `A_stokes.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 $\|A\|$ where $\|\cdot\|$ denotes the Euclidean norm.

7. Define the `Eigen` vector $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$ and solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ using the GMRES method (implemented in the `gmres.hpp` template) without *restart*. Fix a maximum number of iterations which is sufficient to reduce the (relative) residual below than 10^{-9}. Use the diagonal preconditioner provided by `Eigen`. Report on the sheet the iteration counts and $\overline{x}_1$ (the first component of the obtained approximated solution $\overline{\boldsymbol{x}}$).

8. Let n be the size of the matrix loaded in the previous point and let $\tilde{A}$ be a $n \times n$ matrix

defined such that

$$\tilde{A} = A + \|A\| \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ 1 & 2 & -1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \ddots & \dots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & -1 \\ 0 & 0 & \dots & 0 & 1 & 2 \end{pmatrix}.$$

Report on the sheet $\|\tilde{A}_S\|$, where $\tilde{A}_S$ is the symmetric part of $\tilde{A}$, namely $2\tilde{A}_S = \tilde{A} + \tilde{A}^T$.

9. Export the matrix $\tilde{A}$ in the matrix market format (save it as `A_tilde.mtx`) and move it to the `lis-2.0.34/test` folder. Using the GMRES iterative solver available in the LIS library compute the approximate solution of the linear system $\tilde{A}\boldsymbol{x} = \boldsymbol{b}$ up to a tolerance of 10^{-9}. Explore at least two different preconditioning strategies that yield a decrease in the number of required iterations with respect to the GMRES method without preconditioning. Report on the sheet the iteration counts and the relative residual at the last iteration.

Exercise 2

1. Consider the rectangular linear system $A\boldsymbol{x} = \boldsymbol{b}$, where A is an $m \times n$ matrix, $m \geq n$. Provide the definition of the solution in the least-square sense and state under which condition the problem is well posed.

2. Describe the QR factorization of the rectangular matrix A.

3. Discuss how the QR factorization can be employed to solve the above linear system in the least-square sense.

4. What can be done if A does not have full rank?

5. Download the matrix `A_rect.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 `exer2.cpp` using the `unsupported/Eigen/SparseExtra` module. Report on the sheet $\|A^T A\|$.

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

7. Use the `SparseQR` solver available in the `Eigen` library to compute the approximate solution of the least-square problem associated to $A\boldsymbol{x} = \boldsymbol{b}$. Report on the sheet the Euclidean norm of the error $\|\boldsymbol{x}_{SQR} - \boldsymbol{x}^*\|$, where $\boldsymbol{x}_{SQR}$ is the obtained approximate solution.

8. Use the `LeastSquaresConjugateGradient` solver available in the `Eigen` library to compute the approximate solution of the previous least-square problem up to a tolerance of 10^{-10}. Report on the sheet the iteration counts and the error norm $\|\boldsymbol{x}_{LSCQ} - \boldsymbol{x}^*\|$, where $\boldsymbol{x}_{LSCQ}$ is the obtained approximate solution.

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

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

```
#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;
}
```

NUMERICAL LINEAR ALGEBRA
A.A. 2022/2023

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

WRITTEN TEST 04/09/2023

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 general form of a linear iterative method for the approximate solution of $A\boldsymbol{x} = \boldsymbol{b}$ and describe the stopping criteria.

3. State the necessary and sufficient condition for convergence.

4. Describe the Generalized Minimal Residual Method (GMRES). Recall the interpretation of the scheme as a Krylov subspace method and the main theoretical results.

5. Download the matrix `Aex1.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 $\|A\|$ where $\|\cdot\|$ denotes the Euclidean norm.

```
Matrix size:500 X 500
Non zero entries:7708
Norm of A: 105.193
Norm of skew-symmetric part: 47.9613
The matrix is non-symmetric since the norm of the skew-symmetric part of A is
positive with magnitude comparable to the one of the norm of A.
```

6. Define the `Eigen` vector $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$ and find the approximate solution $\boldsymbol{x}_j$ of $A\boldsymbol{x} = \boldsymbol{b}$ using the Jacobi method (implemented in the `jacobi.hpp` template). Fix a maximum number of iterations which is sufficient to reduce the (relative) residual below than 10^{-5} and take $\boldsymbol{x}_0 = \boldsymbol{b}$ as initial guess. Report on the sheet the iteration counts and $\|\boldsymbol{x}_j\|$.

```
Jacobi method
iterations performed 14
tolerance achieved 9.343e-06
norm of computed sol 7.07627
```

7. Compute the approximate solution $\boldsymbol{x}_g$ of $A\boldsymbol{x} = \boldsymbol{b}$ obtained using the GMRES method without *restart*. Fix a maximum number of iterations which is sufficient to reduce the residual below than 10^{-10} considering $\boldsymbol{x}_j$ (computed in the previous point) as initial guess. Use the `Eigen` diagonal preconditioner. Report the iteration counts and $\|\boldsymbol{x}_j - \boldsymbol{x}_g\|$.

```
GMRES method
iterations performed: 14
tolerance achieved  : 3.52916e-11
distance btwn 2 sols: 3.90184e-05
```

8. Repeat the previous points using the iterative solvers available in the LIS library. First, compute the approximate solution obtained with the Jacobi method prescribing a tolerance of 10^{-5}. Then, using the GMRES solver compute the approximate solution up to a tolerance of 10^{-10}. Find a preconditioning strategy that yield a decrease in the number of required iterations with respect to the GMRES method without preconditioning. 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 Aex1.mtx 1 sol.mtx hist.txt -i jacobi -tol 1.e-5
--> Jacobi: number of iterations = 14, relative residual    = 9.603771e-06

mpirun -n 4 ./test1 Aex1.mtx 1 sol.mtx hist.txt -i gmres -tol 1.e-10
--> GMRES: number of iterations = 17, relative residual    = 2.509352e-11

mpirun -n 4 ./test1 Aex1.mtx 1 sol.mtx hist.txt -i gmres -tol 1.e-10
-p ssor -ssor_omega 1.4
--> GMRES: number of iterations = 12, relative residual    = 2.682046e-11
```

Full implementation of Exercise 1

```
#include <iostream>
#include <Eigen/SparseCore>
#include <Eigen/IterativeLinearSolvers>
#include <unsupported/Eigen/SparseExtra>
#include "gmres.hpp"
#include "jacobi.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, "Aex1.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;
  double Anorm = A.norm();
  std::cout << "Norm of A: " << Anorm << std::endl;
  std::cout << "Norm of skew-symmetric part: " << B.norm() << std::endl;

  double tol = 1.e-5;              // Convergence tolerance
  int result, maxit = 1000;        // Maximum iterations
  SpVec b = SpVec::Ones(A.rows());
  std::cout << "norm of rhs: " << b.norm() << std::endl;
  SpVec x(A.rows());
  Eigen::DiagonalPreconditioner<double> D(A);

  // Jacobi method
  result = Jacobi(A, x, b, D, maxit, tol);
  std::cout << "Jacobi method " << std::endl;
  std::cout << "iterations performed " << maxit << std::endl;
  std::cout << "tolerance achieved " << tol << std::endl;
  std::cout << "norm of computed sol " << x.norm() << std::endl;

  // GMRES method with diagonal precond
  SpVec xg(A.rows());
  xg = x; tol = 1.e-10; maxit = 500;
  int restart = maxit;  // set restart = maxit to avoid restart
  result = GMRES(A, x, b, D, restart, maxit, tol);
  std::cout << "GMRES method " << std::endl;
  std::cout << "iterations performed: " << maxit        << std::endl;
  std::cout << "tolerance achieved  : " << tol          << std::endl;
  std::cout << "distance btwn 2 sols: " << (x - xg).norm() << std::endl;

  return result;
}
```

Exercise 2

1. Consider the rectangular linear system $A\boldsymbol{x} = \boldsymbol{b}$, where A is an $m \times n$ matrix, $m \geq n$. Provide the definition of the solution in the least-square sense and state under which condition the problem is well posed.

2. Describe the QR factorization of the rectangular matrix A.

3. Discuss how the QR factorization can be employed to solve the above linear system in the least-square sense.

4. Download the matrix `Aex2.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 `exer2.cpp` using the `unsupported/Eigen/SparseExtra` module. Report on the sheet $\|A^T A\|$.

```
Matrix size:800 X 400, Non zero entries:8431
Norm of A: 70.8894, Norm of A^T*A: 335.01
```

5. 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: 31.7075
```

6. Use the `SparseQR` solver available in the `Eigen` library to compute the approximate solution of the least-square problem associated to $A\boldsymbol{x} = \boldsymbol{b}$. Report on the sheet the Euclidean norm of the error $\|\boldsymbol{x}_{SQR} - \boldsymbol{x}^*\|$, where $\boldsymbol{x}_{SQR}$ is the obtained approximate solution.

```
Solution with Eigen QR:
effective error: 4.20802e-14
```

7. Use the `LeastSquaresConjugateGradient` solver available in the `Eigen` library to compute the approximate solution of the previous least-square problem up to a tolerance of 10^{-10}. Report on the sheet the iteration counts and the error norm $\|\boldsymbol{x}_{LSCQ} - \boldsymbol{x}^*\|$, where $\boldsymbol{x}_{LSCQ}$ is the obtained approximate solution.

```
Solution with Eigen LSCG:
#iterations:     97
relative residual: 9.95923e-11
effective error: 5.74656e-09
```

8. Export the matrix A^TA in the matrix market format (save it as `AtA.mtx`) and move it to the `lis-2.0.34/test` folder. Using the Conjugate Gradient iterative solver available in LIS compute the approximate solution of the linear system $A^TA\boldsymbol{x} = A^T\boldsymbol{b}$ up to a tolerance of 10^{-10}. Report the iteration counts and the relative residual at the last iteration.

```
mpirun -n 4 ./test1 AtA.mtx 2 sol.mtx hist.txt -i cg -tol 1.e-10
--> CG: number of iterations = 114, relative residual  = 9.222078e-11
```

Full implementation of Exercise 2

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

int main(int argc, char** argv){
  using namespace Eigen;

  // Some useful alias
  using SpMat = SparseMatrix<double>;
  using SpVec = VectorXd;

  // Load matrix
  SpMat A;
  Eigen::loadMarket(A, "Aex2.mtx");
```

```
  // Check 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;
  SpMat A_square = A.transpose()*A;
  std::cout << "Norm of A^T*A: " << A_square.norm() << std::endl;

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

  // solve with Eigen QR factorization
  Eigen::SparseQR<Eigen::SparseMatrix<double>, COLAMDOrdering<int>> solver;
  solver.compute(A);
  if(solver.info()!=Eigen::Success) {
      std::cout << "cannot factorize the matrix" << std::endl;
      return 0;
  }
  x = solver.solve(b);
  std::cout << "Solution with Eigen QR:" << std::endl;
  std::cout << "effective error: "<<(x-e).norm()<< std::endl;

  // solve with Eigen LeastSquareConjugateGradient solver
  double tol = 1.e-10;           // Convergence tolerance
  int maxit = 1000;              // Maximum iterations

  LeastSquaresConjugateGradient<SparseMatrix<double> > lscg;
  lscg.setMaxIterations(maxit);
  lscg.setTolerance(tol);
  lscg.compute(A);
  x = lscg.solve(b);
  std::cout << "Solution with Eigen LSCG:" << std::endl;
  std::cout << "#iterations:     " << lscg.iterations() << std::endl;
  std::cout << "relative residual: " << lscg.error()    << std::endl;
  std::cout << "effective error: " << (x-e).norm()      << std::endl;

  // Export matrix
  std::string matrixFileOut("./AtA.mtx");
  saveMarket(A_square, matrixFileOut);
  return 1;
}
```

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):

```
#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;
}
```