Numerical Methods for CSE
Final Exam - Summer 2019

Prof. Rima Alaifari

ETH Zürich, D-MATH

Exam Organisers:
Soumil Gurjar, Pratyuksh Bansal

August 26, 2019

Exercise 1. *Sparsity Preservation* (26 pts) [*Template: 1.cpp*]

Consider the following problem: Given $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$, we want to obtain the unique least squares solution of the linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.

(a) (2 pts) Implement a C++ function

```
VectorXd normalEquationSolve(const MatrixXd &A, const VectorXd &b);
```

that returns the least squares solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ by solving the normal equations.

(b) (2 pts) Now consider a matrix $\mathbf{A}$ which is sparse. While performing the computation of the solution by normal equations, the sparsity is not necessarily preserved. Construct an example of a sparse $n \times n$ matrix such that the system matrix of the normal equations has no zero entries.

(c) (6 pts) Through an augmented system $\mathbf{C}\mathbf{y} = \mathbf{d}$, sparsity can be maintained. Implement two C++ functions,

```
SparseMatrix<double> computeC(SparseMatrix<double> &A, const VectorXd &b);
```

and

```
VectorXd computeRHS(SparseMatrix<double> &A, const VectorXd &b);
```

that respectively compute the *sparse* matrix $\mathbf{C}$ and RHS-vector $\mathbf{d}$.

(d) (4 pts) Implement a C++ function

```
VectorXd computeSolution(SparseMatrix<double> &A, const VectorXd &b);
```

that solves the modified system using a standard sparse solver (SparseLU) and returns the required solution vector $\mathbf{x}$ corresponding to the original system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

(e) (6 pts) Now we consider a specific example of the 1D Poisson equation, $-\ddot{u} = f$ on $\Omega = [0, 1]$, with Dirichlet boundary conditions $u(0) = u(1) = 0$. We apply a finite difference method, which allows us to obtain the solution to this problem by solving a linear system of equations. This is done in the following way:

The independent variable, $x \in \Omega$, is discretized by choosing an equidistant grid $\Omega_N = \{x_j\}_{j=0}^{N+1} \in \Omega$, where the grid points are given by $x_j = j\Delta x$, and where the spacing between the grid points is referred to as the mesh width, $\Delta x = \frac{1}{N+1}$. This means that $x_0 = 0$ is the left endpoint, and $x_{N+1} = 1$ is the right endpoint. In between, there are N internal points in the interval [0, 1].

The solution will be approximated numerically on this grid, by a vector $\mathbf{u} = \{u_j\}_1^N$ of internal points, augmented by the boundary values, $u_0 = 0$ and $u_{N+1} = 0$. The vector $\mathbf{u}$ approximates the solution to the differential equation, according to $u_j \approx u(x_j)$. Now, the second order derivative $-\ddot{u}$ is approximated by a symmetric finite difference scheme,

$$-\ddot{u}(x_j) = -\left(\frac{u_{j-1} - 2u_j + u_{j+1}}{(\Delta x)^2}\right).$$

Applying the finite difference method for all the interior points, obtain a linear system of equations of the form

$$L_N \mathbf{u} = \mathbf{f},$$

where the vector $\mathbf{f} = (f(x_1), f(x_2), \ldots, f(x_N))^\top$, $\mathbf{u} = (u_1, u_2, \ldots, u_N)^\top$ and $L_N \in \mathbb{R}^{N,N}$. Explicitly write out the matrix L_N.

(f) (6 pts) For the 1D Poisson problem described above, if $f = \pi^2 \sin(\pi x)$ and $N = 25, 50, 100, ..., 1600$, implement two functions, `constructDenseA_Poisson` and `constructSparseA_Poisson`, that construct the matrix L_N in dense and sparse format respectively for each mesh-width. Solve the corresponding linear system $L_N \mathbf{u} = \mathbf{f}$, using the normal equations from sub-problem (a), as well as the modified system from sub-problem (d). Compare the two solutions to verify your implementation, and also compare their run-times for each value of N. Explain your observations.

Hint: A dense matrix `A_dense` can be converted into a sparse matrix `A_sparse` by using `A_sparse = A_dense.sparseView()`. Moreover, the code to compare the solutions and run-times is already implemented within the template.

Exercise 2. *Interpolation* (28 pts) [*Template: 2.cpp*]

Consider the function

$$f(x) = \begin{cases} f_1(x) := 2 + \frac{1}{1+25(6x-1)^2}, & x \in [0, \frac{1}{3}), \\ f_2(x) := 1 + \cos(\pi x), & x \in [\frac{1}{3}, 1]. \end{cases} \tag{1}$$

Let $I^p_{\mathcal{T}_N}[f]$ be a piecewise Lagrange interpolant of the function f, for a mesh $\mathcal{T}_N$ on the interval $[0, 1]$. The mesh $\mathcal{T}_N$ has N cells and the local polynomial degree $p \in \mathbb{N}_0$ of the interpolant is the same for each cell in the mesh.
Two functions are provided, see `2.cpp`, to generate equidistant and Chebyshev interpolation nodes in the reference interval $[0, 1]$. Let $t = \{t_0, t_1, \ldots, t_p\} \subset [0, 1]$ denote the set of these reference nodes.

(a) (2 pts) Is $f \in C^0([0, 1])$? Explain.

(b) (5 pts) For $N = 2$, design the interpolant:

  i) specify the mesh and

  ii) specify the strategy to generate local interpolation nodes,

such that the interpolation error $\|I^p_{\mathcal{T}_N}[f] - f\|_{L^\infty([0,1])}$ is small for an arbitrarily large p. Support your answer.

(c) (2 pts) Consider a *uniform* mesh $\mathcal{T}_M$ on the interval $[a, b]$. Implement a function

```
Eigen::MatrixXd genNodes(const double a,
                         const double b,
                         const int M,
                         const Eigen::VectorXd &t);

```

that uses a vector of reference nodes $\mathtt{t} \in \mathbb{R}^n$ and generates a matrix $\mathbb{R}^{n \times M}$ of interpolation nodes in the mesh $\mathcal{T}_M$.

Hint: Map the reference nodes to a given cell.

(d) (8 pts) Given the declaration of a `class` **Newton**, implement the member functions:

  i) `Interpolate` : Computes the coefficients of a piecewise Lagrange interpolant in the Newton basis.

  ii) `Evaluate` : Uses the *Horner scheme* to evaluate a piecewise Lagrange interpolant in the Newton basis.

(e) (8 pts) Use the components from sub-problems (c) and (d), and write a `C++` function to compute a piecewise Lagrange interpolant of the given function $f(x)$ from (1) on a uniform mesh $\mathcal{T}_N$. This function should also evaluate and return the interpolation error.

Hint: Use the given member function `EvalError`, of `class` **Newton**, to compute the interpolation error.

(f) (3 pts) Implement a `C++` function to study the convergence of your interpolant $I^p_{\mathcal{T}_N}$ with respect to p, for $p = 1, 2, 3, \ldots, 20$. Explain the results from your convergence test.

Exercise 3. *Singly Diagonally Implicit Runge-Kutta Methods* (26 pts) [*Template: 3.cpp*]

For general implicit Runge-Kutta methods, the k_i's cannot be evaluated successively since they are coupled in the system of implicit equations that is given for their determination. One way to decouple a non-linear system is to use diagonally implicit Runge-Kutta methods. A special family of those methods are Singly Diagonally Implicit Runge-Kutta methods (SDIRK), in which the matrix of the method is lower triangular and all the diagonal entries are equal. Continuing in this direction, we define the following one parameter family of SDIRK-methods:

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-\gamma & 1-2\gamma & \gamma \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Table I: Singly Diagonally Implicit Runge-Kutta

(a) (3 pts) Determine the value(s) of γ for which the corresponding Runge-Kutta method is of at least (consistency) order 3.

Hint: In addition to conditions necessary for a consistency order 2, the following conditions are necessary for a consistency order 3:

$$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3} \qquad \text{and} \qquad \sum_{i=1}^{s}\sum_{j=1}^{s} b_i a_{ij} c_j = \frac{1}{6},$$

where $\{a_{ij}\}_{i,j=1}^{s}$, $\{b_i\}_{i=1}^{s}$, $\{c_j\}_{j=1}^{s}$ are entries of a standard Butcher tableau.

(b) (4 pts) Write down the iterations of the Runge-Kutta method defined in Table I, when applied to the ODE

$$\dot{y} = \lambda y, \qquad y(0) = 1, \qquad \lambda \in \mathbb{C}$$

HINT: Use the stability function and the fact that the ODE is linear.

(c) (3 pts) For which value(s) of γ obtained from sub-problem (a) (the value(s) with which the method is of order 3), can we conjecture the corresponding Runge-Kutta method to be A-stable? Explain.

Hint: You may use the plotting script *'stabilityRegion_plot.cpp'* to visualize the region of stability for a chosen value of γ.

(d) (2 pts) We consider a scalar linear initial value problem of second order

$$\ddot{y} + \dot{y} + y = 0, \qquad y(0) = 1, \quad \dot{y}(0) = 0 \tag{2}$$

that should be solved numerically using the SDIRK-method described in Table I.

Formulate (2) as an initial value problem (IVP) for a linear first order system.

(e) (6 pts) Implement a `C++/Eigen` function

```
template <class StateType>
StateType sdirkStep(const StateType & z0, double h, double gamma);
```

that realizes the numerical evolution of one step of the SDIRK-method for the IVP obtained in sub-problem (d), starting from the initial value `z0` and returning the value after a time step of size h.

Hint: For an SDIRK-method, compute stage k_1 first and then use it to compute stage k_2.

(f) (8 pts) Conduct a numerical experiment to deduce the convergence order of the method by computing the global error of the *A-stable* SDIRK method for the first order IVP from subproblem (d) at the end time. Choose `T=10` as end time and `N=20,40,80,...,10240` as steps. Report the observed order of convergence.