AMS 562 FINAL PROJECT: SOLVING 1D POISSON ORDINARY DIFFERENTIAL EQUATION (ODE) WITH LAPACK/BLAS

DUE: MON, 12/18, 11:59 PM

Core components: BLAS/LAPACK, vector, classes.

Problem description:

In this project, we will solve a well-posted problem, the *Poisson* equation, numerically. 1D Poisson equation is a fundamental ODE in *boundary value problem* (BVP). The continuous formulation reads:

$$-u'' = f(x) \text{ for } x \in (a, b) \tag{0.1}$$

In order to solve this equation, one needs to know the boundary information, a.k.a. *boundary conditions*. In this project, we consider the most fundamental case, the fixed boundary conditions, a.k.a. *Dirichlet* boundary conditions:

$$u(a) = u_a \tag{0.2}$$

$$u(b) = u_b \tag{0.3}$$

Where u_a and u_b are constant values. The analytic solution is well-studied, but we will focus on solving it numerically. Assume we have a **uniformly** distributed grid $\boldsymbol{X} = x_0,\ x_1\ ..., x_n$ with $n+1$ grid points, where $x_0 = a$ and $x_n = b$. Denote $h = x_1 - x_0$ the interval size. Given an arbitrary grid position $k = 1,\ 2,\ ..., n-1$, we can approximate u''_k by:

$$u''_k = \frac{u'_{k+1} - u'_k}{h} \tag{0.4}$$

Similarly, we can have:

$$u'_{k+1} = \frac{u_{k+1} - u_k}{h} \tag{0.5}$$

$$u'_k = \frac{u_k - u_{k-1}}{h} \tag{0.6}$$

The above equations are based on the formulas we have learned in Calculus classes, i.e. for a continuous and smooth function $g(x)$ defined in (a, b), $g'(x) = \lim_{h\to 0} \frac{g(x+h)-g(x)}{h}$ or $g'(x) = \lim_{h\to 0} \frac{g(x)-g(x-h)}{h}$. Plug Eq's (0.5) and (0.6) into (0.4), we have:

$$u''_k = \frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} \tag{0.7}$$

Combining with Poisson equation, we have:

$$-\frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} = b_k \tag{0.8}$$

Where $b_k = f(x_k)$. The above numerical discretization method is called *finite difference method.*

Note that we are interested in the solutions on grid points $\boldsymbol{X}_{1:n-1}$, i.e. u_k for $k = 1,\ 2,\ ...,\ n-1$. Eq (0.8) gives us $n-1$ equations with exactly $n-1$ unknowns, we can solve this problem by just solving a *linear system.* Denote $\sigma = h^2$, we can have:

$$-u_{k-1} + 2u_k - u_{k+1} = \sigma b_k \tag{0.9}$$

When $k = 1$ and $k = n-1$, we have:

$$2u_1 - u_2 = \sigma b_1 + u_0 = \sigma b_1 + u_a \tag{0.10}$$

$$-u_{n-2} + 2u_{n-1} = \sigma b_{n-1} + u_n = \sigma b_{n-1} + u_b \tag{0.11}$$

Where u_a and u_b are boundary conditions defined in Eq's (0.2) and (0.3). The system can be derived in the following way:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & ... & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & ... & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & ... & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & ... & 0 \\ 0 & ... & ... & ... & ... & ... & ... & 0 \\ 0 & ... & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & ... & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ ... \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} \sigma b_1 + u_a \\ \sigma b_2 \\ \sigma b_3 \\ \sigma b_4 \\ \sigma b_5 \\ ... \\ \sigma b_{n-2} \\ \sigma b_{n-1} + u_b \end{bmatrix} \tag{0.12}$$

We denote the linear system of (0.12) by $\boldsymbol{Au} = \boldsymbol{b}$, where $\boldsymbol{u}$ is the vector of unknowns (solution of Poisson equation on grid $\boldsymbol{X}_{1:n-1}$) and $\boldsymbol{b}$ is the right-hand side vector.

An Example:

Here, I provide a tiny example that can be solved by hand. Let's consider the model problem $u(x) = x^2$ for $x \in (0, 1)$, so we have $u_a = u(0) = 0$, $u_b = u(1) = 1$ and $f = -u'' = -2$. Given a 5-point uniform grid, i.e. $h = \frac{1}{4}$, we have:

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{0.13}$$

$$\boldsymbol{b} = \begin{bmatrix} h^2 b_1 + u_0 \\ h^2 b_2 \\ h^2 b_3 + u_4 \end{bmatrix} = \begin{bmatrix} -\frac{1}{8} \\ -\frac{1}{8} \\ \frac{7}{8} \end{bmatrix} \tag{0.14}$$

Solve the system, we have $\boldsymbol{u} = [\frac{1}{16}\ \frac{1}{4}\ \frac{9}{16}]^T$, which is the exact solution on $\boldsymbol{X}_{1:3} = [\frac{1}{4}\ \frac{1}{2}\ \frac{3}{4}]$.

You tasks:

- (10%) In `solvers/BaseSolvers.h`, implement `assign_bcs`, `get_solution` (inline). Demonstrations of the `protected` variables:
  - `xl_`: left most grid point, i.e. a;
  - `xr_`: right most grid point, i.e. b;
  - `h_`: uniform grid size, i.e. h;
  - `N_`: linear system size, i.e. $n - 1$;
  - `lb_`: left boundary value, i.e. u_a;
  - `rb_`: right boundary value, i.e. u_b;
  - `u_`: solution vector $\boldsymbol{u}$;
  - `A_`: matrix $\boldsymbol{A}$.
- (25%) In `solvers/LuSolver.cpp`:
  - implement the overrided function `assemble` for assembling the complete dense matrix, i.e. using `N_*N_` storage.
  - implement the overrided function `solve` to solve the problem given a vector `b_`.
    * Call `cblas_dcopy` and `cblas_dscal` to first copy ans scale `b_` to `this->u_`.
    * Call `LAPACKE_dgesv` to solve the system and store the results to `this->u_`.
- (25%) In `solvers/PpSolver.cpp`:
  - implement the overrided function `assemble` for assembling the **packed** dense matrix of upper half, i.e. using `N_*(N_+1)/2` storage to only store the **upper** or **lower** half of `A_`.
  - implement the overrided function `solve` to solve the problem given a vector `b_`.
    * Call `cblas_dcopy` and `cblas_dscal` to first copy ans scale `b_` to `this->u_`.
    * Call `LAPACKE_dppsv` to solve the system and store the results to `this->u_`.
- (25%) In `solvers/TriSolver.cpp`:
  - implement the overrided function `assemble` for assembling the **tridiagonal** matrix of upper half, i.e. using `2*N_-1` storage to only store the diagonal and off-diagonal of `A_`.
  - implement the overrided function `solve` to solve the problem given a vector `b_`.
    * Call `cblas_dcopy` and `cblas_dscal` to first copy ans scale `b_` to `this->u_`.
    * Call `LAPACKE_dptsv` to solve the system and store the results to `this->u_`.
- (15%) Write a main program that:
  - solves the model problem $u(x) = 2x^2$ in $(0, 1)$, with number of grid points 202, i.e. `N_` is 200.

- solves the model problem $u(x) = \sin(\frac{\pi}{2}x)$ in $(0, 1)$, with number of grid points 202, i.e. `N_` is 200.
- for each of the model problems:
  - Call `cblas_dnrm2` to compute the 2 norm error.
  - Call `cblas_idamax` to compute the inf norm error.
- You should also provide a `README` file for your program.

Hints:

- Note that `A_` is given as a 1D vector!
- You can compute the error vectors by using `u_` minus the exact solution that can be obtained by explicitly evaluating the grid points in function $u(x)$. (For the vector subtraction, you can call `cblas_daxpy` or just write a simple loop.)
- **Don't forget to apply the boundary conditions when you implement `solve`!!**
- To get the raw pointer (`double *`) of the first position in `std::vector<double>`, you can call `std::vector<double>::data`, e.g. `A_.data()`, `u_.data()` and `b.data()`.
- MKL reference pages:
  - `cblas_dnrm2`: 50
  - `cblas_idamax`: 56
  - `cblas_dcopy`: 47
  - `LAPACKE_dgesv`: 568
  - `LAPACKE_dppsv`: 617
  - `LAPACKE_dptsv`: 627
- The general routines for solving a problem should be:
  - `ams562::XXSolver solver(0.0, 1.0, 202);`
  - `solver.assign_bc(lb, rb); //you should have lb, rb evaluated`
  - `solver.assemble();`
  - `solver.solve(b); //you should compute b before call this function`
  - `auto sol = solver.get_solution();`
  - `// Now doing the post-processing, i.e. analyze errors`
- For the `Makefile`:
  - You need to modify it by:
    - add libraries in `LIBS`, i.e. linking to LAPACK/BLAS
    - add your main program file in `MAIN`
  - Then `make`: build; `make test ARGS=...`: build main and run main executable, `ARGS` is optional; `make clean`: clean up object and executable files.