Uppsala university, Division of Scientific computing

Project – Nonlinear convection-diffusion equations

Scientific Computing III 1TD056 (5.0 hp)

September 29, 2022

Introduction

In this assignment, we are interested in solving a time-dependent nonlinear convection-diffusion (NCD) equation in one space dimension. Let I be an open bounded interval. Our aim is to solve the following initial value problem: find u such that

$$\begin{aligned} \partial_t u + \partial_x \beta(u) - \partial_x(\varepsilon \partial_x u) &= f, \qquad (x,t) \in I \times (0,T], \\ u(x,0) &= u_0, \qquad\qquad\quad x \in I, \end{aligned} \tag{1}$$

where $f := f(x,t)$, $u_0 := u_0(x)$ are given functions, $T > 0$ is the final time and the boundary conditions are to be defined.

The NCD equation (1) is a simplified version of the so-called Navier-Stokes equations. The Navier-Stokes equations model fluid flow and widely used in scientific and industrial applications. In general, the NCD equation cannot be solved analytically, due to the nonlinear term. Nevertheless, the numerical approximation of the NCD equation is widely used to understand the underlying physical processes modeled by it.

The above equation (1) is sometimes referred as advection-diffusion equation. The term of *convection* is usually used for motion of a fluid flow due to change of density of temperature, whereas *advection* is usually used to describe the motion of some material by the velocity of the fluid.

The project consists of two parts: Part A is on a finite element method, and Part B is on numerical linear algebra.

The aim

The goal of the project is 1. to get familiar with partial differential equations (PDEs); 2. to learn some basic discretization techniques to numerically approximating these PDEs; to learn some basic iterative and direct methods to solve the resulting linear system of equations. After completing this project you should be able to discretize and solve numerically many scalar linear or nonlinear PDEs.

Part A

Finite Element Methods

Consider (1) with $\beta(u) = \frac{1}{2}u^2$, $I = (a, b)$ and the following boundary data:

$$\begin{aligned} \partial_t u + \partial_x\Big(\frac{u^2}{2}\Big) - \partial_x(\varepsilon \partial_x u) &= 0, & (x,t) &\in I \times (0,T], \\ u(a,t) &= g_a(t), & t &\in (0,T], \\ u(b,t) &= g_b(t), & t &\in (0,T], \\ u(x,0) &= u_0(x), & x &\in I, \end{aligned} \tag{2}$$

where the values of u_0, a, b and g_a, g_b are yet to be defined. Equation (2) is a well-known viscous Burger's equation which is widely used fluid dynamics. This is a nonlinear equation and is a simplified version of the Navier-Stokes system. Since the system of Navier-Stokes equations is difficult to solve in general, the viscous Burger's equation is considered to be a good model to understand many interesting complex flow phenomenons.

Assume that the domain is split into N equal sub-intervals. Start by writing a weak and a continuous piecewise linear Galerkin finite element approximation of the Burger's equation. Once the equation is discretized in space, we obtain a system of N Ordinary Differential Equations of the form

$$\mathbf{M}\frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}t} = -\mathbf{A}\beta(\boldsymbol{U}) - \varepsilon\mathbf{S}\boldsymbol{U}, \tag{3}$$

where $\boldsymbol{U}(t)$ is a vector consisting of the nodal values of the finite element solution $u_h(x,t) = \sum_{i=0}^{N} U_j(t)\varphi_j(x)$, $\mathbf{M}, \mathbf{A}, \mathbf{S}$ are the mass, advection and diffusion matrices correspondingly:

$$\mathbf{M}_{ij} := \int_I \varphi_j\varphi_i \,\mathrm{d}x, \quad \mathbf{A}_{ij} := \int_I \partial_x\varphi_j\varphi_i \,\mathrm{d}x, \quad \mathbf{S}_{ij} := \int_I \partial_x\varphi_j\partial_x\varphi_i \,\mathrm{d}x.$$

Note, that we approximate the nonlinear flux with its finite element interpolant, i.e., $\beta(u) \approx \sum_{i=1}^{N-1} \beta(U_j)\varphi_j$, where $\beta(U_j) = \frac{1}{2}U_j^2$, $j = 1, \ldots, N-1$.

Approximate the resulting system using the standard 4th order explicit Runge-Kutta method in time. Note that at every Runge-Kutta stages you will have to solve a linear system involving the mass matrix.

The inhomogeneous Dirichlet boundary condition can be applied in a number of ways:

1. injection, i.e., you assign boundary data to the solution after each time step (or Runge-Kutta stage);
2. a strong form, i.e., after assembling the linear system $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$ replace $\mathbf{M}(1,1) = \mathbf{M}(N,N) = 10^6$, $\boldsymbol{b}(1) = 10^6 g_a(t)$, $\boldsymbol{b}(N) = 10^6 g_b(t)$;
3. a strong form, i.e., after assembling the linear system $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$, replace the first row with $\mathbf{M}(1,:) = (1, 0, \cdots, 0)$, the last row with $\mathbf{M}(N,:) = (0, 0, \cdots, 1)$, and set $\boldsymbol{b}(1) = g_a(t)$, $\boldsymbol{b}(N) = g_b(t)$.

Problem A.1. Implement a finite element solver using continuous piecewise linear approximations to solve the Burger's equation in (2). Start with initial condition

$$u_0(x) = c - \tanh\left(\frac{x + \frac{1}{2}}{2\,\varepsilon}\right),$$

where c is an arbitrary constant. An analytic solution to the Burger's equation using the above initial condition is given by,

$$u_{exact}(x,t) = c - \tanh\left(\frac{x + \frac{1}{2} - c\,t}{2\,\varepsilon}\right), \tag{4}$$

Set $a = -1$, $b = 1$, $g_a(t) = u_{exact}(a,t)$, $g_b(t) = u_{exact}(b,t)$, $c = 2$, $\varepsilon = 0.1$, and the final time $T = 0.4$. Investigate a convergence rate of the finite element approximation in a sequence of successively refined intervals: $N = 41, 81, 161, 321, 641$ and plot the error as a function of mesh-size $h = 1/(N{-}1)$. In one figure, plot your solution at the final time from all different meshes as well as the exact solution.

Problem A.2. Now, solve the Burger's equation using the following initial condition

$$u_0(x) = \sin(x),$$

over the interval $I = (0, 2\pi)$, (i.e., set $a = 0$, $b = 2\pi$). The boundary condition is $g_a(t) = g_b(t) = 0$. Run the problem until the final time $T = 2$ over the mesh consisting 201 points with $\varepsilon = 1, 0.1, 0.001, 0$. Plot your solution on one figure and discuss your result.

Problem A.3. Consider the same problem setting as in Part C.2. Now, set $\varepsilon = \frac{1}{2}h$ and run your program with a sequence of meshes $N = 41, 81, 161, 321, 641$. Plot your solution at the final time $T = 2$. What do you observe now?

Part B

Linear Systems

Assume that once the partial differential equation (1) is discretized using some approximation technique, the following linear system of equations is obtained

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \tag{5}$$

where $\mathbf{A}$ is a large, (usually) sparse matrix, $\boldsymbol{b}$ is known load vector and $\boldsymbol{x}$ is unknown solution vector.

Table 1: Comparison between different methods

Method	**Iteration**	**Time**
Jacobi		
Gauss-Seidel		
CG		
myownLU		
Matlab LU		
Matlab $\mathbf{A}\backslash\boldsymbol{b}$		

This part aims to implement three iterative methods, namely Jacobi, Gauss-Seidel and Conjugate Gradient as well as a direct method LU decomposition. Then, we compare our implementation against Matlab standard backslash and LU functions. You will be asked to fill Table 1. The number iterations are reported only for iterative methods. You are welcome to use and modify the Matlab code from the first laboration.

We are going to study performance of several methods for solving linear system of equations. For this purpose, one can use the Matlab stopwatch timer functions `tic, toc`. For instance, your code could have a `main.m` file which could contain the following lines:

```
...
tic;
x1 = A\b;
fprintf('Backslash␣took␣%g␣sec␣\n', toc);

tic;
x2 = jacobi(A,b, TOL);
fprintf('Jacobi␣took␣%g␣sec␣\n', toc);

tic;
x3 = gs(A,b, TOL);
fprintf('GS␣took␣%g␣sec␣\n', toc);

tic;
x4 = cg(A,b, TOL);
fprintf('CG␣took␣%g␣sec␣\n', toc);

tic;
x5 = myownLU(A,b);
fprintf('myownLU␣took␣%g␣sec␣\n', toc);

tic;
[L,U] = lu(A);
y = L\b;
x6=U\y;
fprintf('Matlab␣LU␣took␣%g␣sec␣\n', toc);
```

Problem B.1. Implement three separate Matlab functions: `cg.m`, `jacobi.m`, `gauss_seidel.m`, `myownLU.m`. All these four functions should have the same input arguments: the matrix $\mathbf{A}$, the right hand side vector $\boldsymbol{b}$, and a constant $\mathrm{TOL} = 0.001$. The output is the solution vector $\boldsymbol{x}$ and the number of iteration in the iterative solvers. Then, test your code using the following data:

$$\mathbf{A} = \begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 6 \\ 25 \\ -11 \\ 15 \end{pmatrix}.$$

Fill out Table 1. How long the Matlab backslash function takes to compute the solution? Discuss your result.

Problem B.2. Now, let us use the following randomly generated linear system of equation:

```
w = ..;           % the diagonal weight
N = ..;           % the size of the linear system
TOL = 0.0001;     % torelance
A = rand(N) + diag(w*ones(N,1));
b = rand(N,1);
```

where `w` denotes the weight value of the diagonal. Repeat the task of Problem A.1 for this linear system for TOL=10^{-5} and `N=100`, `N=500`, `N=1000`, and `w=1`, `w=5`, `w=10`, `w=100`. Modify Table 1 and report your result. What do you observe? Motivate your result with some theoretical explanation.

Problem B.3. In this problem we are going to use our solver developed in the above tasks to solve a large sparse linear system (5) using the following $n \times n$ matrix:

$$\mathbf{A} = \begin{pmatrix} 2+\alpha & -1 & 0 & \dots & \dots & 0 \\ -1 & 2+\alpha & -1 & \ddots & & \vdots \\ 0 & -1 & 2+\alpha & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -1 & 2+\alpha & -1 \\ 0 & \dots & \dots & 0 & -1 & 2+\alpha \end{pmatrix},$$

and the right hand side is a randomly generated $n \times 1$ vector. Set $n = 10000$ and solve the linear system using $\alpha = \{1, 0.1, 0.001, 0.00001\}$ for TOL=10^{-5}. Report your result in Table 1 and discuss your solution.

Good luck!
Murtazo
Uppsala, September 2022