MATH/CS 5466 · NUMERICAL ANALYSIS

Problem Set 6

Posted Friday 22 April 2016. Due Friday 29 April 2016 (5pm).
Please complete all 4 problems (total of 100 points).

1. [25 points]

   Consider the differential equation $x'(t) = \lambda x(t)$ with $x(0) = x_0$.

   (a) Show that when applied to this equation, Heun's method yields

   $$x_{k+1} = (1 + h\lambda + \tfrac{1}{2}h^2\lambda^2)x_k.$$

   (b) Develop an analogue of the formula for x_{k+1} in part (a), but now using the four-stage Runge–Kutta method.

   (c) Compare your answers from (a) and (b) to the Taylor series for $x(t_{k+1})$ (that is, the exact solution at t_{k+1}), expanded about the point $t = t_k$.

   (d) Use MATLAB to plot the set of all $h\lambda \in \mathbb{C}$ for which $|x_k| \to 0$ as $k \to \infty$ for Heun's method and the four stage Runge–Kutta method.

   [Adapted from Süli and Mayers]

2. [25 points]

   Consider 2-step linear multistep methods of the form

   $$x_{k+2} + Ax_{k+1} + Bx_k = hCf_{k+1}$$

   for the initial value problem $x'(t) = f(t, x(t))$, $x(t_0) = x_0$, where A, B, and C are constants.

   (a) Determine all choices of A, B, and C for which this method is consistent.

   (b) Determine a choice of A, B, and C that gives $O(h^2)$ truncation error.

   (c) Assess the zero-stability of the method found in part (b).

   (d) What does your answer to part (c) imply about the behavior of the linear multistep method as $h \to 0$ for such values of A, B, and C?

   (e) For the method found in part (b), calculate those values of λh for which $x_k \to 0$ as $k \to \infty$ when applied to the differential equation $x' = \lambda x$.

3. [25 points]

   Convection–diffusion equations play an important role in fluid dynamics. In one dimension, the simplest such equation takes the form

   $$-\varepsilon u''(x) + u'(x) = 0, \qquad u(0) = a, \quad u(1) = b.$$

   (The second derivative term, Δu in higher dimensions, gives diffusion; the first derivative term, $\mathbf{w}^T\nabla u$ in higher dimensions, gives convection in the direction of the 'wind', $\mathbf{w}$.)

   Note that this convection–diffusion equation is a *boundary value problem*, rather than an initial value problem. As stated, it is easy enough to solve by hand, but it will be useful to develop a numerical method that we could also apply to more difficult problems. The *shooting method* is one option:

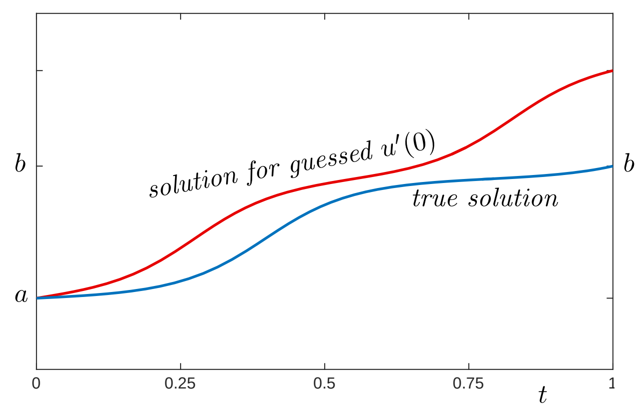
- Write this second-order ODE as a system of two first-order ODEs:

$$\begin{aligned} u_1'(x) &= u_2(x) \\ u_2'(x) &= \varepsilon^{-1} u_2(x). \end{aligned}$$

- Guess some value for $u'(0)$.
- Integrate this system (e.g., using a Runge–Kutta method) for $x \in [0, 1]$ with the initial values $u_1(0) = u(0) = a$ (given by the problem) and $u_2 = u'(0)$ (guessed).
- Unless you are lucky, the solution you obtain will not match the boundary condition $u(1) = b$, because the guessed value for $u'(0)$ is not correct. One can use a nonlinear root-finding algorithm (e.g., bisection, *regula falsi*, the secant method, or Newton's method) to adjust the guess $u'(0)$ until the integrated value at $x = 1$ agrees with the desired $u(1) = b$. That is, one seeks a zero of the objective function

$$f(\xi) = b - (u(1) \text{ computed with } u'(0) = \xi).$$

The following figure shows a schematic view of the shooting method (for a different differential equation). The solid line is the solution to the ODE with the correct value $u(0) = a$, but the incorrect $u'(0)$. Since this initial slope is incorrect, the corresponding value for $u(1)$ is also wrong. The dashed line shows the true solution, which satisfies $u(1) = b$. The challenge is to adjust the guessed value for $u'(0)$ so that the computed $u(1)$ satisfies the boundary condition $u(1) = b$.



Your task is to solve the convection-diffusion equation.

(a) Implement the shooting method to solve the above convection-diffusion boundary value problem with $\varepsilon = 1/10$, $u(0) = 0$ and $u(1) = 1$. Please use MATLAB's built-in ODE integrator, `ode45`; you may use any root-finding algorithm you like, but please implement it yourself or use the codes on the class website. If you use the bisection or *regula falsi* algorithms, use $u'(0) = 0$ and $u'(0) = 1$ to obtain your initial bracket. If you use the secant method or Newton's method, try $u'(0) = 0$ as an initial guess.

Please present your code, a plot of $u(x)$ for $x \in [0, 1]$, and the value of $u'(0)$ that gives $u(1) = 1$.

(b) Repeat the same experiment for $\varepsilon = 1/50$. The exact solution demonstrates a *boundary layer* near $x = 1$.

(c) Derive the exact solution for this convection-diffusion problem. In particular, what are the exact values for $u'(0)$ in parts (a) and (b)? How do these values of $u'(0)$ compare to those you computed in (a) and (b)?

4. [25 points]

Method of Lines. Many physical models give rise to time-dependent partial differential equations. General techniques to solve such problems are beyond the scope of this course. However, many such problems can be attacked using standard ODE integrators via a technique known as the *method of lines*. In this problem, you will solve the *first-order wave equation*

$$u_t(t,x) = u_x(t,x).$$

Here $u(t,x)$ is a scalar function of two real variables; u_t denotes the time derivative, and u_x denotes the space derivative. The problem is posed on the temporal domain $t \geq 0$ and spatial domain $x \in (-\infty, \infty)$. The initial data will be

$$u(0,x) = \sin(2\pi x),$$

which gives the exact solution

$$u(t,x) = \sin(2\pi(x+t)).$$

The method of lines approximates the solution to a partial differential equation by first discretizing the domain in the x direction into points $x_j = j\Delta x$, where $\Delta x = 1/n$ for some fixed n. Since the initial data is periodic, we only need to discretize from $x_1 = \Delta x$ through $x_n = n\Delta x = 1$, and then assign $x_0 = x_n$ by periodicity.

Now approximate the spatial (x) derivative by the simple finite difference approximation

$$u_x(t,x_j) \approx \frac{u(t,x_{j+1}) - u(t,x_j)}{\Delta x};$$

we have previously observed that this approximation incurs an $O(\Delta x)$ error. The method of lines approximates the partial differential equation $u_t = u_x$ with an *ordinary differential equation* by replacing u_x with the finite difference approximation, giving

$$u_t(t,x_j) = \frac{u(t,x_{j+1}) - u(t,x_j)}{\Delta x}.$$

Exploiting periodicity (which implies that $u(t,x_n) = u(t,x_0)$), this reduces the partial differential equation to a system of n ordinary differential equations. Using the notation

$$\mathbf{u}(t) = \begin{bmatrix} u(t,x_1) \\ u(t,x_2) \\ \vdots \\ u(t,x_n) \end{bmatrix} \in \mathbb{R}^n,$$

this system of differential equations can be written as

$$\mathbf{u}_t(t) = \mathbf{A}\mathbf{u}(t).$$

(a) What is the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$?
(Be careful not to neglect the entry that arises because of periodicity.)

(b) Verify (by proving analytically, or by simply computing a numerical example for $n = 7$, whichever you prefer) that $\mathbf{A}$ has the n eigenvalues and associated eigenvectors

$$\lambda_j = (\mathrm{e}^{\mathrm{i}\theta_j} - 1)/\Delta x, \qquad \mathbf{v}_j = (\mathrm{e}^{\mathrm{i}\theta_j}, \mathrm{e}^{2\mathrm{i}\theta_j}, \ldots, \mathrm{e}^{n\mathrm{i}\theta_j})^T,$$

for $\theta_j = 2\pi j/n$ for $j = 1, \ldots, n$.

Finally, the method of lines solves $\mathbf{u}_t = \mathbf{A}\mathbf{u}$ using an ODE integrator. For simplicity, use the forward Euler method:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \Delta t\mathbf{A}\mathbf{u}_k.$$

(c) Consider the eigenvalues from part (b), together with the theory of absolute stability for the forward Euler method, to determine a sharp condition on Δt that ensures there are no exponentially growing solutions for a fixed value of Δx. (You have just derived the famous *CFL condition*, first noted in a seminal 1928 paper by Richard Courant, Kurt Otto Friedrichs, and Hans Lewy.)

(d) Implement your algorithm in MATLAB to confirm that your answer to (c) is correct. In particular, take $\Delta x = 1/50$ ($n = 50$) and give solutions when Δt is (1) twice the maximum and (2) equal to the maximum allowed by the stability requirement from (c).

You may show this data in several ways: You can plot the solution at time $t = 2$ in two dimensions ($u(2, x)$ versus x), or in three dimensions for $t \in [0, 2]$. For the latter, the following MATLAB commands may prove useful: `surf`, `mesh`, `waterfall`, `pcolor`, `shading interp`.