MAT352 Linear Boundary Value Problems Homework

Instructions

Use the format described in the Write Up Form document to present your work for this assignment. Your report must be written with LaTeX. Include your answers to specific questions posed in the assignment as part of your report, as well as the content listed in the write-up format description. You will need to write some computer code to solve the problems in this homework, but you are free to use any language of your choice.

1 Background

In this assignment, you will be solving the second-order boundary value problem:

$$y'' = y' + 2y + \cos(x)\,, \qquad 0 \le x \le \frac{\pi}{2}\,, \qquad y(0) = -0.1\,, \qquad y\left(\frac{\pi}{2}\right) = -0.1\,. \tag{1}$$

This problem has exact solution $y(x) = -\frac{1}{10}(\sin(x) + 3\cos(x))$. Your task is to approximate the solution of this linear boundary value problem using the **Shooting Method** and the **Finite Difference Method**. For each method, try $h = \frac{\pi}{8}$ and $h = \frac{\pi}{16}$ and characterize how your approximation compares to the exact solution.

1.1 The Shooting Method

Using the shooting method for this problem requires that you numerically solve two second-order IVPs, and then combine the solutions to these IVPs to obtain a solution to the BVP. In our example in class, we used the Forward Euler method to numerically solve these IVPs. In your assignment, you are free to use the Forward Euler method in your implementation of the Shooting Method, but you are permitted to use a different IVP method if you prefer.

Keep in mind that when using the Shooting Method, the error and convergence properties are dependent on the method you choose to solve the second-order IVPs. What should the order of convergence be for your implementation as $h \to 0$? Compare your approximations with the exact solution; do your observations agree with your hypothesis about the convergence for this method?

1.2 The Finite Difference Method

Use the finite difference scheme we developed in class to write an approximate solution to the BVP as the solution of a tridiagonal system of linear equations. In a real-world scenario, where the linear system is typically very large, we could gain a lot of computational performance by using a tridiagonal system solver. However, you do NOT need to do this in your assignment, in particular because the dimension of the linear system, n, should be very low in your implementation. Recall that the dimension, n, is equal to the number of unknowns in the model, which will be $n = 3$ for $h = \frac{\pi}{8}$ and $n = 7$ for $h = \frac{\pi}{16}$. If you use Python for your assignment, you may want to use the method `numpy.linalg.solve` to solve your linear system.

The finite difference scheme we developed in class uses second-order centered difference approximations for both y' and y''. Therefore, we expect that the convergence rate of the finite difference scheme for this problem should be $\mathcal{O}(h^2)$. Compare your finite difference approximations to the exact solution; do your results support this convergence hypothesis?

1.3 A Word About Error

The exact solution to the BVP is a continuous function defined over the interval $x \in [0, \frac{\pi}{2}]$. However, our approximations (using both methods) are not continuous functions! Instead, we have estimates of the value of the solution at discrete points, $\{x_1, x_2, \dots\}$. To measure the difference between our approximation and the exact solution, we typically consider only the difference in the values at the discrete nodes.

For example, suppose the we have a numerical solution with $n = 3$ unknowns. Then, our computational grid contains points $\{x_0, x_1, x_2, x_3, x_4\}$, and we know the value of $\tilde{y}_0 = y(x_0)$ and $\tilde{y}_4 = y(x_4)$ from the boundary conditions. Our numerical method produces estimates:

$$\tilde{y}_1 \approx y(x_1) \tag{2}$$

$$\tilde{y}_2 \approx y(x_2) \tag{3}$$

$$\tilde{y}_3 \approx y(x_3) \tag{4}$$

To compute the error of this method, we typically use the l_2 norm of the residual vector. That is, the error, ϵ, could be computed in this case as

$$\epsilon = \|\vec{r}\|_2 \tag{5}$$

where

$$\vec{r} = \begin{bmatrix} \tilde{y}_1 - y(x_1) \\ \tilde{y}_2 - y(x_2) \\ \tilde{y}_3 - y(x_3) \end{bmatrix} \tag{6}$$

so that

$$\epsilon = \sqrt{[\tilde{y}_1 - y(x_1)]^2 + [\tilde{y}_2 - y(x_2)]^2 + [\tilde{y}_3 - y(x_3)]^2} \tag{7}$$

In Python, if `y_exact`=$[y(x_1),\ y(x_2),\ y(x_3)]$ and `y_approx`=$[\tilde{y}_1,\ \tilde{y}_2,\ \tilde{y}_3]$, then we could compute: $\epsilon =$ `np.linalg.norm(y_exact - y_approx)`.