MPCS 51087 Project 1, Milestone 1
Serial Advection

Sonia Sharapova

1 Introduction

This report focuses on a c implementation for numerically estimating discrete approximate solutions to the advection equation in two dimensions. The primary objective is to develop a serial implementation of the Lax method, a finite difference approach, to simulate the advection process over time.

2 Implementation

2.1 Parameters

The solution-space for this problem is a grid representing a discretized representation of the physical space in which the advection equation is being solved. Each grid point corresponds to a specific location in this space, and the spacing between these points affects the resolution of the simulation. A higher value of "N"means more grid points, leading to a finer grid with more detailed spatial resolution, which can provide more accurate simulations but requires more computational resources.

The parameters to the algorithm are as follow:

- N: Matrix Dimension (spacing between points on the grid)
- NT: Number of timesteps
- L: Physical Cartesian domain length
- T: Total simulation time
- u: X velocity Scalar
- v: Y velocity Scalar

These parameters are read-in through the command line upon running the c file.

2.2 Algorithm Implementation

The approach to this implementation involves the process of updating concentration levels at each grid point for every time step. This is done by converting derivatives into finite differences, where at any time-step, the spacial derivative is calculated from taking one point to the left and one point to the right over twice the distance between them.

This process follows the following strategy:

2.3 Initialization

During initialization, the computational grid representing the spatial domain of the simulation is established.

The initial concentration values at each grid point are then set by iterating over each point and assigning a value based the 2D Guassian pulse initial condition, reflecting the initial state of the system being modeled. The Guassian is centered at x0 and y0 with σ_x and σ_y representing the "spread". In this implementation, σ_x and σ_y are fixed at $\frac{L}{4}$.

2.4 Lax Scheme

Following initialization, the Lax scheme is used to numerically solve the advection equation with some constant non-zero velocity. This schema is repeated for every time-step.

2.4.1 Courant stability condition

To avoid numerical instability, the code asserts that the Courant stability condition is met prior to invoking the Lax Scheme.

2.4.2 Spacial derivative

The algorithm iterates over a discretized grid, with N representing the number of grid points in each dimension. For each grid point, indexed by i and j, the algorithm computes the concentration at the next time step, Cnext[i][j]. This computation is achieved by first determining the neighboring points—iprev, inext for the i dimension, and jprev, jnext for the j dimension.

2.4.3 Boundary Validation

To ensure that the boundary conditions are met, the algorithm implements a "wrap-around"method when determining the neighbours. In my code, this is handled but the prev() and next() functions which ensure that the grid behaves as if it is looped onto itself when at the edges, allowing for the simulation of a continuous space.

2.4.4 Calculating next point in time

The concentration at Cnext[i][j] is calculated using a weighted average of the current concentration and its neighbors. This averaging process is foundational to the Lax scheme, ensuring a balance between the propagation and dispersion of the concentration. The algorithm then adjusts for advection effects, incorporating the velocities u and v. This adjustment involves a differential term that subtracts the product of the time step (dt), the velocity components, and the gradient of concentration between neighboring points, all normalized by the spatial step (dx).

Through this approach, the Lax scheme effectively models the dynamics of the advection equation, maintaining stability and accuracy in the simulation.

2.5 Updating Grid

The grid at the next time-step is populated using the values calculated by the schema mentioned above.

2.6 Data Storage

To keep data from the populated grid for future use, the data for each time-step was captured and stored in a .txt file using ASCII format, where the matrix values were presented in scientific notation.

3 Simulation and Results

To ensure that the parameters meet the Courant stability conditions, the program was run on the given parameters:
- N = 400
- NT = 20000
- L = 1.0
- T = 1.0e6
- u = 5.0e-7
- v = 2.85e-7

The output of the simulation was stored in a file called 'output.txt' which could further be used for visualisation.

4 Data Visualization and Interpretation

To see the progression of the Gaussian over time, the progress of the simulation was captured over three time-steps: the initial time (t=0), the midpoint, and the last time step.

These states were plotted in Python using matplotlib.

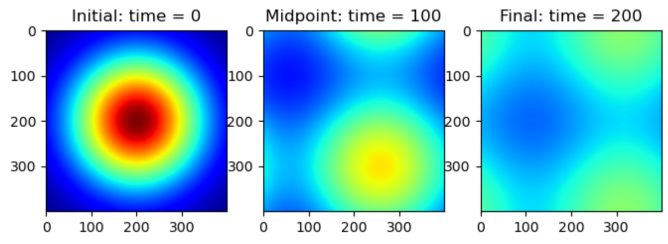

Figura 1: Plots.

To see how the simulation acted over time, an animation was also created depicting the continuous changes of the grid.

The plots showed a Gaussian Distribution illustrating the movement in the advection equation over time.

5 Conclusion

This report demonstrates the implementation of the Lax scheme for solving the advection equation. The periodic boundary conditions and data storage methods resulted in results which were able to be effectively visualized, contributing to a deeper understanding of the advection phenomena. This project not only underscores the importance of numerical methods in computational simulations but also showcases the potential of high-performance computing in solving complex physical problems.