Problem Set 4

Problems to Computational Astrophysics, WS 2013/2014

Prof. Dr. Friedrich Röpke, Prof. Dr. Christian Klingenberg, Sebastian Ohlmann
Offices: Campus Hubland Nord, 31.01.017, 30.02.012, 31.01.003

Hand in until Monday, 25.11.2013, 12.00 pm
Tutorial on Tuesday, 26.11.2013, 10.15 am

Hint: Stability criterion — Given the assumption

$$||\mathcal{N}(P) - \mathcal{N}(Q)|| \leq (1 + \alpha\Delta t)||P - Q||$$

for a numerical scheme $\mathcal{N}$, the error after N steps is bounded as $\Delta t \to 0$ for a fixed time T, where $\mathcal{N}(u^n)$ denotes the application of the numerical scheme to the analytical solution u of the equation.

1. Accuracy and stability

a) **(P)** Consider the equation

$$u_t + cu_x = du \qquad \text{with} \quad u(x,0) = u_0(x) \quad \text{and} \quad c > 0, \;\; d \in \mathbb{R}.$$

Show that the method

$$Q_i^{n+1} = Q_i^n - \frac{c\Delta t}{\Delta x}\left(Q_i^n - Q_{i-1}^n\right) + \Delta t\, d\, Q_i^n$$

is first order accurate for this equation by computing the local truncation error. The local truncation error is given by

$$L^n = \frac{1}{\Delta t}\left[\mathcal{N}(u^n) - u^{n+1}\right],$$

where $\mathcal{N}(u^n)$ denotes the application of the numerical scheme to the analytical solution u of the equation.

Hint: Use Taylor expansions.

b) **(P)** Next show that the method is stable according to the criterion above in the 1-Norm, which is defined as

$$\|Q^n\|_1 = \Delta x \sum_i |Q_i^n|.$$

This leads a constraint on $|\frac{c\Delta t}{\Delta x}|$. What is it?

Hint: For $d > 0$ the solution grows exponentially in time but we want stability and convergence for a fixed time T.

c) **(H)** Program the scheme. Use as initial data

$$u(x,0) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x > 0. \end{cases}$$

Play with the above constraint and observe what happens. What happens if $c < 0$?

2. Godunov vs. Strang splitting (P)

In class, we considered the general problem $u_t = (\mathcal{D}_1 + \mathcal{D}_2)u$, with $\mathcal{D}_1$ and $\mathcal{D}_2$ being linear differential operators in x but not in t.

a) Godunov-type operator splitting solves the equations for both operators, i.e.

$$u_t = \mathcal{D}_1 u \qquad \text{and} \tag{1}$$
$$u_t = \mathcal{D}_2 u \tag{2}$$

separately over a time step Δt each. The result of the first step serves as initial value for the second step. It was stated in class that this introduces no error if the two differential operators commute. Show this using Taylor series expansion. Which order has the splitting error if the differential operators do not commute? Which order of accuracy does this imply for a numerical scheme based on this Godunov splitting?

b) Now, consider Strang splitting instead. Here, the first subproblem (1) is solved over half a time step, then the second subproblem (2) is solved over a full time step, and then the first subproblem is again solved over half a time step. As before, the results from the preceding subproblem are the initial values for each step. Using Taylor series expansion, show that this increases the order of the splitting error even if the two operators do not commute.

Hint: Note that $u_{tt} = (\mathcal{D}_1 + \mathcal{D}_2)u_t = (\mathcal{D}_1 + \mathcal{D}_2)^2 u$ and in general $\partial_t^j u = (\mathcal{D}_1 + \mathcal{D}_2)^j u$. This allows to formally write the Taylor series expansion as an exponential function.

Remark: Although it may seem from this analysis that Strang splitting is generally vastly superior to Godunov splitting this is not always the case in practice because of the respective coefficients in front of the terms of first and second order.

3. Time complexity of merge sort (P)

We analyzed the time complexity of *insertion sort* in class and found it to be only modest $\mathcal{O}(n)$. Insertion sort falls into the class of comparison sort algorithms. An alternative is *merge sort*, which is based on a divide-and-conquer approach. Here, we again start out from an unsorted array containing n elements. This array is split in two sub-arrays ("left" and "right") of about the same size ("divide step"). These sub-arrays are sorted each and then merged to give the sorted array of length n. The merger process is given as pseudocode in Algorithm1.
In the full algorithm, the divide step is performed recursively until arrays of size 1 are reached. Then the merger process is applied to successively longer pairs of arrays until a sorted array of length n results.

a) Write a pseudocode for the full algorithm (using calls to the merge function of Algorithm 1).

b) What is the time complexity of merge sort (give the result *and* a reasoning for it)?

Algorithm 1 function merge (left, right)

```
while length(left) > 0 or length(right) > 0 do
  if length(left) > 0 and length(right) > 0 then
    if first(left) ≤ first(right) then
      append first(left) to result
      left = remainder(left)
    else
      append first(right) to result
      right = remainder(right)
    end if
  else if length(left) > 0 then
    append first(left) to result
    left = remainder(left)
  else if length(right) > 0 then
    append first(right) to result
    left = remainder(right)
  end if
end while
```

Hint: For estimating the time complexity you may think of the divide step as constructing a binary tree. Start out with an array of, say, 2^k elements. The first division level gives you two arrays of length 2^{k-1}. How many levels do we have in the algorithm?

Exercises marked with (P) have to be presented in the exercise, those marked with (H) have to be handed in. Programs can be sent per e-mail to `sohlmann@astro.uni-wuerzburg.de`.