Applied Numerical Methods - Lab 4

Goyens Florentin & Weicker David

6^{th} November 2015

Stationary heat conduction in 1-D

In a one dimensional pipe we are interested in the temperature evolution along the z-axis. We will study the behaviour of the numerical solution based on finite differences.

a) Reformulation of the problem

Bla bla bla

b) Discretize to a system of equation

Bla bla bla

c) Resolution

Bla bla bla

d) Comparison of methods

In this section, we will compare two methods for solving the system of differential equations (the one obtained in part 2) : the explicit method *ode*23 and the implicit method *ode*23*s*. As already said, an implicit method is often more suitable for stiff problems.

The two methods will be compared for differents constant stepsizes for the spatial discretization ($N \in [10, 20, 40]$) and for a determined time interval ($\tau \in [0; 2]$). We will based our analysis our three different factors : the number of steps needed to reach $\tau = 2$, the cpu-time needed to do the computations and the maximal value of the stepsize for the time discretization. The default tolerances are used for both methods.

The Matlab code used to do this analysis can be found at the end of this section. As we will reuse this code for part e, this is how the function was called :

$$[timeStep, cpuTime, hMax] = tempOde('Gaussian')$$

The returned variables are arrays containing the data. The first column is when *ode*23 is used and the second is for *ode*23*s*. The lines correspond to the different stepsizes for the spatial discretization.

The call to this function gives the following data :

	timesteps		**cpu - time**		$\mathbf{h}_{tmax}$	
N	*ode*23	*ode*23*s*	*ode*23	*ode*23*s*	*ode*23	*ode*23*s*
10	345	104	0.1479	0.1944	0.0169	0.1473
20	1295	132	0.4730	0.4108	0.0043	0.1563
40	5114	169	1.9056	0.8883	0.0010	0.1522

We can easily see that, for all the factors taken into account, *ode*23*s* is better than its explicit counterpart.

First, the number of time steps. The explicit method needs much more steps to reach $\tau = 2$, this is because the problem is stiff. We can even compare this number with the theoretical number needed for stability in the case of a constant step size. For $N = 10$, the maximal stable stepsize would be $h_t = \frac{(0.1)^2}{2} = 0.005$ so the number of steps to reach $\tau = 2$ would be $n = \frac{2}{0.005} = 400$. We can see that *ode*23 is just a little better whereas *ode*23*s* divides this number by 4.

Concerning the cpu - time, *ode*23*s* is an obvious winner only when $N = 40$. For $N = 10$, *ode*23 is even a little bit better. This is mainly because the Gaussian elmination of the linear system takes time, but this will be solved in part e.

Finally, the maximal timestep looks approximately constant for *ode*23*s*. This is not the case for the explicit method. It has to reduce considerably the timestep in order to avoid instability.

e) Improvements

Ceci est la partie e

f) Visualization

Ceci est la partie f