Applied Numerical Methods - Lab 4

GOYENS Florentin & WEICKER David

6th 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 $ode23$ and the implicit method $ode23s$. 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 $ode23$ is used and the second is for $ode23s$. 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	$ode23$	$ode23s$	$ode23$	$ode23s$	$ode23$	$ode23s$
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, $ode23s$ 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 $ode23$ is just a little better whereas $ode23s$ divides this number by 4.

Concerning the cpu - time, $ode23s$ is an obvious winner only when $N = 40$. For $N = 10$, $ode23$ 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 $ode23s$. This is not the case for the explicit method. It has to reduce considerably the timestep in order to avoid instability.

e) Improvements

In this section, we will try to improve the efficiency of $ode23s$. Indeed, this method has, at each timestep, to solve a system and uses a Gaussian elimination. However, our system is tridiagonal so this is not optimal. It is in fact possible to "tell" Matlab a bit more about our system so that the resolution will be more efficient. Two ways are used here.

First, we can tell him which entries of the jacobian are non zeros. It will increase the efficiency because Matlab will know what computations are not necessary (because somothing times zero is always zero). This is done by the *odeset* property called $JPattern$. We can obtain information by calling the same function as in part d but with a different argument :

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

Second, we can tell Matlab what is the jacobian of our system. In our case, it is a constant matrix. This will greatly help Matlab in his resolution of the system. To see this, we can call :

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

To see the effect of the combined properties, we can call :

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

The following table contains the cpu - time for $N = 10, 20, 40$ for different combinations of the *odeset* properties with the method $ode23s$. The column $'both'$ means that the two properties were set.

CPU - time ($ode23s$)

N	Gaussian	sparse	jacobian	both
10	0.1944	0.1097	0.0504	0.0560
20	0.4108	0.1476	0.0667	0.0697
40	0.8883	0.1803	0.0910	0.0992

We can see that $JPattern$ improves the compution time and $Jacobian$ enven more. Using both $Jacobian$ and $JPattern$ seems to not have a greater effect than using $Jacobian$ alone since the CPU - times are really close. But the computation time can be reduced by a almost a factor 10 for $N = 40$.

Finally, we can look at the same table as in part e and see if something other than the cpu - time changes when we set the properties. The following table is obtained when the two properties are set :

	timesteps		**cpu - time**		$\mathbf{h}_{tmax}$	
N	$ode23$	$ode23s$	$ode23$	$ode23s$	$ode23$	$ode23s$
10	345	103	0.1217	0.0560	0.0169	0.1473
20	1295	132	0.4383	0.0697	0.0043	0.1563
40	5114	173	1.8093	0.0992	0.0010	0.1523

We can see that, the cpu - time aside, all the factors remain almost the same. For $ode23$, this is because this is an explicit method and so those two properties will have little influence. Indeed, in an explicit method, no system has to be solved. Instead, a matrix multiplication is used. The resolution will thus be a little bit faster because the jacobian is constant but the gain will not be as large as for an implicit method.

f) Visualization

Ceci est la partie f