Numerical Simulation of Dynamic Systems XII

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

April 9, 2013

Hyperbolic PDEs

Let us now analyze the second class of PDE problems, the hyperbolic PDEs. The simplest specimen of this class of problems is the *wave equation* or *linear conservation law*:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \cdot \frac{\partial^2 u}{\partial x^2}$$

which can be easily converted to two first-order PDEs:

$$\begin{aligned} \frac{\partial u}{\partial t} &= v \\ \frac{\partial v}{\partial t} &= c^2 \cdot \frac{\partial^2 u}{\partial x^2} \end{aligned}$$

This time around, we need two initial conditions and two boundary conditions.

Hyperbolic PDEs II

One complete specification of such a model could be:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \quad ; \quad x \in [0,1] \quad ; \quad t \in [0,\infty)$$

$$u(x, t=0) = \sin\left(\frac{\pi}{2}x\right)$$

$$\frac{\partial u}{\partial t}(x, t=0) = 0.0$$

$$u(x=0, t) = 0.0$$

$$\frac{\partial u}{\partial x}(x=1, t) = 0.0$$

Hyperbolic PDEs III

The PDE model can be easily converted to a set of ODEs using the method of lines:

$$\begin{aligned}
u_1 &= 0.0 \\
\dot{u}_2 &= v_2 \\
&\dots \\
\dot{u}_n &= v_n \\
\dot{u}_{n+1} &= v_{n+1} \\
v_1 &= 0.0 \\
\dot{v}_2 &= n^2\left(u_3 - 2u_2 + u_1\right) \\
\dot{v}_3 &= n^2\left(u_4 - 2u_3 + u_2\right) \\
&\dots \\
\dot{v}_n &= n^2\left(u_{n+1} - 2u_n + u_{n-1}\right) \\
\dot{v}_{n+1} &= 2n^2\left(u_n - u_{n+1}\right)
\end{aligned}$$

initial conditions :

$$\begin{aligned}
u_2(0) &= \sin\left(\frac{\pi}{2n}\right) \\
&\dots \\
u_n(0) &= \sin\left(\frac{(n-1)\pi}{2n}\right) \\
u_{n+1}(0) &= \sin\left(\frac{\pi}{2}\right) \\
v_2(0) &= 0.0 \\
v_3(0) &= 0.0 \\
&\dots \\
v_n(0) &= 0.0 \\
v_{n+1}(0) &= 0.0
\end{aligned}$$

Hyperbolic PDEs IV

This is a *linear, time-invariant, inhomogeneous, $(2n)^{th}$-order, single-input system* of the type:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u$$

where:

$$\mathbf{A} = \begin{pmatrix} \mathbf{0}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} \\ \mathbf{A}_{21} & \mathbf{0}^{(\mathbf{n})} \end{pmatrix}$$

with:

$$\mathbf{A}_{21} = n^2 \begin{pmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 2 & -2 \end{pmatrix}$$

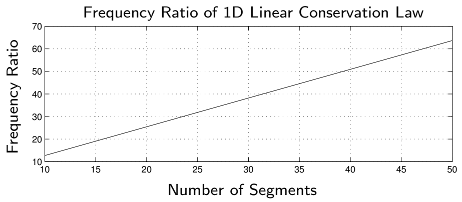
$\mathbf{A}$ is a band-structured matrix of dimensions $2n \times 2n$ with two separate non-zero bands.

Hyperbolic PDEs V

Let us look at the distribution of eigenvalues of the **A**-matrix in function of the grid width:

$n = 3$	$n = 4$	$n = 5$	$n = 6$
$\pm 1.5529j$	$\pm 1.5607j$	$\pm 1.5643j$	$\pm 1.5663j$
$\pm 4.2426j$	$\pm 4.4446j$	$\pm 4.5399j$	$\pm 4.5922j$
$\pm 5.7956j$	$\pm 6.6518j$	$\pm 7.0711j$	$\pm 7.3051j$
	$\pm 7.8463j$	$\pm 8.9101j$	$\pm 9.5202j$
		$\pm 9.8769j$	$\pm 11.0866j$
			$\pm 11.8973j$

Therefore:



Hyperbolic PDEs VI

- We notice that all eigenvalues of the wave equation converted to a set of ODEs are on the imaginary axis. The wave equation is totally undamped.
- We notice further that, whereas the eigenvalue pair of the converted ODE set with the lowest frequency component remains more or less at the same location, eigenvalues with ever increasing frequency components are added as the number of discretization points grows, i.e., as the discretization grid is made finer.
- **Hyperbolic PDEs converted to a set of equivalent ODEs using the method of lines always lead to marginally stable systems.**
- **Hence there are now two types of marginally stable systems: lumped parameter systems without or with little damping, and hyperbolic PDEs that become marginally stable in the conversion to ODEs.**

Consistency Error

We chose once again an example of a system, for which the analytical solution is known:

$$u(x,t) = \frac{1}{2}\sin\left(\frac{\pi}{2}(x-t)\right) + \frac{1}{2}\sin\left(\frac{\pi}{2}(x+t)\right)$$

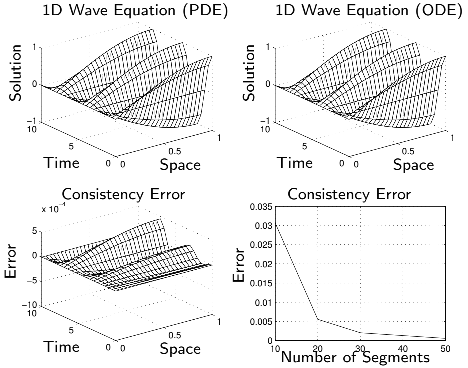
Hence we can compute the *consistency error* explicitly.

- Traditionally, the numerical PDE literature talks about the three facets: *stability*, *consistency*, and *convergence*. It is then customary to prove that any two of the three imply the third one, i.e., it is sufficient to look at any selection of two of the three.
- However, that way of reasoning is more conducive to fully discretized (*finite difference* or *finite element*) schemes, where the step size in time, h, is locked in a fixed relationship with the grid width in space, δx.
- Consequently, h and δx approach zero simultaneously.

In the context of the *method of lines* methodology, our approach may be more appealing.

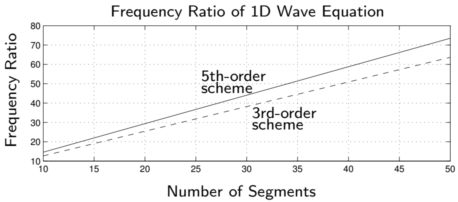
Consistency Error II

We compare the analytical solutions of the original PDE problem and the discretized ODE problem with each other and compute the consistency error:



Consistency Error III

Let us compare the frequency ratios of the third-order and fifth-order approximations of the spatial derivative:

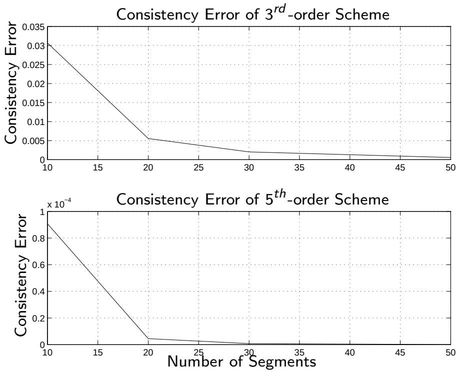


- The frequency ratio of the more accurate 5^{th}*-order scheme* is consistently higher than that of the less accurate 3^{rd}*-order scheme* for the same number of segments.

- Since the true PDE solution, corresponding to the solution with infinitely many infinitely dense discretization lines, has a frequency ratio that is infinitely large, we suspect that choosing a higher-order discretization scheme may indeed help with the reduction of the consistency error.

Consistency Error IV

Let us compare the consistency errors of the third-order and fifth-order approximations of the spatial derivative:



Consistency Error V

- The improvement from the 3^{rd}-order to the 5^{th}-order approximation is quite dramatic.
- The consistency error has been reduced by at least two orders of magnitude.
- **Using the 5^{th}-order scheme almost always pays off.**
- **The scheme should be implemented as a sequence of two 3^{rd}-order schemes with a Richardson corrector.**

Simulation of Hyperbolic PDEs

Comparing different ODE solvers with each other for simulating the wave equation discretized using a fifth-order approximation in space with 50 segments, we find:

h	RK3	IEX3	BI3
0.1	unstable	0.6782e-4	0.4947e-6
0.05	unstable	0.8668e-5	0.2895e-7
0.02	unstable	0.5611e-6	0.1324e-8
0.01	0.7034e-7	0.7029e-7	0.2070e-8
0.005	0.8954e-8	0.8791e-8	0.2116e-8
0.002	0.2219e-8	0.2145e-8	0.2120e-8
0.001	0.2127e-8	0.2119e-8	0.2120e-8

h	AB3	ABM3	AM3	BDF3
0.1	unstable	unstable	unstable	garbage
0.05	unstable	unstable	unstable	garbage
0.02	unstable	unstable	unstable	garbage
0.01	unstable	0.6996e-7	unstable	garbage
0.005	0.7906e-7	0.8772e-8	0.8783e-8	0.9469e-2
0.002	0.5427e-8	0.2156e-8	0.2149e-8	0.1742e-6
0.001	0.2239e-8	0.2120e-8	0.2120e-8	0.4363e-7

Simulation of Hyperbolic PDEs II

- Using a step size of $h = 0.001$, all seven 3rd-order accurate ODE solvers simulate the problem successfully. In fact, all of them with the exception of BDF3 are down to the level of the consistency error. Some of them are more efficient than others, but all of them are successful.

- As the step size becomes smaller, the higher-order terms in the Taylor-series expansion become less and less important. For sufficiently small step sizes, all integration algorithms behave either like forward or backward Euler.

- BDF3 performs a bit poorer than the other algorithms, because its *error coefficient* is considerably larger than that of its competitors. BDF algorithms perform generally somewhat poor in terms of accuracy in comparison with their peers of equal order.

Simulation of Hyperbolic PDEs III

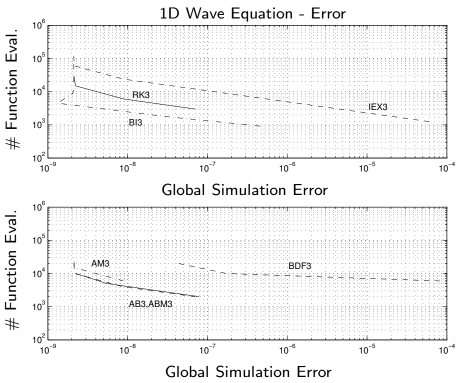
- It turns out that the problem is kind of "stiff," although it does not meet most of the conventional definitions of stiffness. The problem is "stiff" in the sense that all the algorithms with stability domains looping into the left-half plane are unable to produce solutions with the desired accuracy of 1.0%, since they are numerically unstable when a step size is used that would produce the desired accuracy otherwise.

- BDF3 doesn't suffer the same fate, but it eventually succumbs to *error accumulation* problems. As the step sizes grow too big, the computations become so inaccurate that the simulation error exceeds the simulation output in magnitude. Hence BDF3 starts accumulating numerical garbage.

- Only IEX3 and BI3 are capable of solving the problem successfully for large step sizes. Between the two, BI3 seems to work a little better, which is no big surprise. Being an *F-stable algorithm*, BI3 is earmarked for these types of applications.

Simulation of Hyperbolic PDEs IV

Representing the same information graphically:



- The *BI3 algorithm* is the one that is most effective in solving this problem.
- We did not try the *GE3 algorithm*. That algorithm would be expected to be quite efficient also for this problem.

Shock Waves

Let us now study a *more involved hyperbolic PDE problem*.

A thin tube of length $1\ \mathrm{m}$ is initially pressurized at $p_B = 1.1\ \mathrm{atm}$. The tube is located at sea level, i.e., the surrounding atmosphere has a pressure of $p_0 = 1.0\ \mathrm{atm} = 760.0\ \mathrm{Torr} = 1.0132 \cdot 10^5\ \mathrm{N\ m^{-2}}$. The current temperature is $T = 300.0\ \mathrm{K}$.

At time zero, the tube is opened at one of its two ends. We wish to determine the pressure at various places inside the tube as functions of time.

As the tube is opened, air rushes out of the tube, and a *rarefaction wave* enters the pipe. Had the initial pressure inside the pipe been smaller than the outside pressure, air would have rushed in, and a *compression wave* would have formed.

Shock Waves II

The problem can be mathematically described by a set of first-order hyperbolic PDEs:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} &= -v \cdot \frac{\partial \rho}{\partial x} - \rho \cdot \frac{\partial v}{\partial x} \\
\frac{\partial v}{\partial t} &= -v \cdot \frac{\partial v}{\partial x} - \frac{a}{\rho} \\
\frac{\partial p}{\partial t} &= -v \cdot a - \gamma \cdot p \cdot \frac{\partial v}{\partial x} \\
a &= \frac{\partial p}{\partial x} + \frac{\partial q}{\partial x} + f \\
q &= \begin{cases} \beta \cdot \delta x^2 \cdot \rho \cdot \left(\frac{\partial v}{\partial x}\right)^2 & ; \quad \frac{\partial v}{\partial x} < 0.0 \\ 0.0 \quad ; \quad \frac{\partial v}{\partial x} \geq 0.0 \end{cases} \\
f &= \frac{\alpha \cdot \rho \cdot v \cdot |v|}{\delta x}
\end{aligned}$$

where $\rho(x,t)$ denotes the *gas density* inside the tube at position x and time t, $v(x,t)$ denotes the *gas velocity*, and $p(x,t)$ denotes the *gas pressure*. The quantity a was pulled out into a separate algebraic equation, since the same quantity is used in two places within the model.

Shock Waves III

The two quantities, q and f, are artificial, as their dependence on δx shows. Clearly, δx is not a physical quantity, but is introduced only in the process of converting the (small) set of PDEs into a (large) set of ODEs. q denotes the *pseudo viscous pressure*, and f denotes the *frictional resistance*. These quantities were introduced by Richtmyer and Morton in order to smoothen out numerical problems with the solution.

γ is the *ratio of specific heat constants*, a non-dimensional constant with a value of $\gamma = c_p/c_v = 1.4$. α and β are non-dimensional numerical fudge factors. We shall initially assign the following values to them: $\alpha = \beta = 0.1$. The "ideal" (i.e., undamped) problem has $\alpha = \beta = 0.0$.

Introduction of the two dissipative terms is not a bad idea, since the "ideal" solution does not represent a physical phenomenon in any true sense. Phenomena without any sort of dissipation belong allegedly in the world that we may enter after we die. They certainly don't form any part of this universe.

Shock Waves IV

The *initial conditions* are:

$$\begin{aligned}\rho(x, t = 0.0) &= \rho_B \\ v(x, t = 0.0) &= 0.0 \\ p(x, t = 0.0) &= p_B\end{aligned}$$

where ρ_B is determined by the *equation of state* for ideal gases:

$$\rho_B = \frac{p_B \cdot M_{\mathrm{air}}}{R \cdot T}$$

with $T = 300.0$ denoting the *absolute temperature* (measured in Kelvin), $R = 8.314\ \mathrm{J\ K^{-1}\ mole^{-1}}$ the *gas constant*, and $M_{\mathrm{air}} = 28.96\ \mathrm{g\ mole^{-1}}$ the *average molar mass of air*.

Shock Waves V

The *boundary conditions* are:

$$v(x = 0.0, t) = 0.0$$
$$\rho(x = 1.0, t) = \rho_0$$
$$\begin{cases} v(x = 1.0, t) = -\sqrt{\frac{2(p_0 - p(x=1.0,t))}{\rho(x=1.0,t)}} \ ; \quad v(x = 1.0, t) < 0.0 \\ p(x = 1.0, t) = p_0 \ ; \quad v(x = 1.0, t) \geq 0.0 \end{cases}$$

The air velocity, $v(x, t)$, is zero at the closed end ($x = 0.0$). The air density, $\rho(x, t)$, is equal to the ambient air density, ρ_0, at the open end ($x = 1.0$). The third boundary condition is formulated differently depending on the current air flow direction. If air is flowing out of the tube ($v \geq 0.0$), we set the air pressure, $p(x, t)$, equal to the ambient air pressure, p_0. If air is rushing in ($v < 0.0$), we formulate an equation for the air velocity at the open end as a function of the pressure gradient and the air density at that location.

Shock Waves VI

We converted all spatial derivatives by means of *second-order accurate central differences* using the formula:

$$\left.\frac{\partial u}{\partial x}\right|_{x=x_i} \approx \frac{1}{2\delta x} \cdot (u_{i+1} - u_{i-1})$$

where we gained one order of approximation accuracy due to symmetry, with the exception of locations near the boundaries, where we used *second-order accurate biased differences*:

$$\frac{\partial u}{\partial x}(x = x_1, t) \approx \frac{1}{2\delta x} \cdot (-u_3 + 4u_2 - 3u_1)$$
$$\frac{\partial u}{\partial x}(x = x_{n+1}, t) \approx \frac{1}{2\delta x} \cdot (3u_{n+1} - 4u_n + u_{n-1})$$

The variable u can stand for either ρ, v, p, or q.

Shock Waves VII

I coded a Matlab function that realizes the computation of the first spatial derivative including special treatment of boundary conditions:

$$\mathbf{u_x} = \text{partial}(\mathbf{u}, \delta x, bc, bctype);$$

where bc denotes the location of the boundary condition:

$$\begin{aligned} bc &= -1 : \text{boundary condition specified at left boundary} \\ bc &= 0 \quad : \text{no boundary condition specified} \\ bc &= +1 : \text{boundary condition specified at right boundary} \end{aligned}$$

and $bctype$ determines the type of boundary condition:

$$\begin{aligned} bctype &= 0 : \text{Neumann boundary condition} \\ bctype &= 1 : \text{Dirichlet boundary condition} \end{aligned}$$

Shock Waves VIII

Now the discretized model can be formulated as follows (in pseudo-Matlab code):

```
function [$xdot$] = st_eq($x$, $t$)
    %
    % $State - space\ model\ of\ shock - tube\ problem$
    %
    $n$ = round(length($x$)/3);
    $n_1$ = $n$ + 1;
    $\delta x$ = 1/$n$;
    %
    % $Constants$
    %
    $R$ = 8.314;
    %
    % $Physical\ parameters$
    %
    $Temp$ = 300;
    $Mair$ = 0.02896;
    $p_0$ = 1.0132e5;
    $\rho_0$ = $p_0 * Mair/(R * Temp)$;
    $\gamma$ = 1.4;
    %
    % $Fudge\ factors$
    %
    global $\alpha$ $\beta$
    %
```

Shock Waves IX

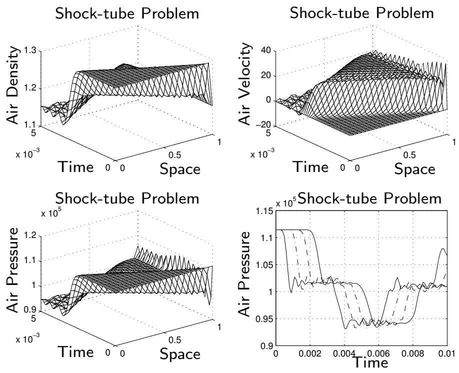
```
% Unpack individual state vectors from total state vector
%
$\rho \ = \ [\ x(1:n)\ ;\ \rho_0\ ];$
$v \ = \ [\ 0\ ;\ x(n_1:2*n)\ ];$
$p \ = \ x(n_1+n:n_1+2*n);$
%
% Calculate nonlinear boundary condition
%
if $v(n1) \ < \ 0,$
    $v(n_1) \ = \ -\mathrm{sqrt}(\max([2*(p_0 \ - \ p(n_1))/\rho(n_1), 0]));$
else
    $p(n_1) \ = \ p_0;$
end
%
% Calculate spatial derivatives
%
$\rho_x \ = \ \mathrm{partial}(\rho, \delta x, +1, +1);$
$v_x \ = \ \mathrm{partial}(v, \delta x, -1, +1);$
$p_x \ = \ \mathrm{partial}(p, \delta x, +1, +1);$
%
% Calculate algebraic quantities
%
$f \ = \ \alpha*(\rho \ .* \ v \ .* \ \mathrm{abs}(v))/\delta x;$
```

Shock Waves X

```
    $q \;=\; \mathrm{zeros}(n1, 1);$
    for $i = 1 : n_1$,
        if $v_x(i) \;<\; 0$,
            $q(i) \;=\; \beta * (\delta x^2) * \rho(i) * (v_x(i)^2);$
        end,
    end
    $q_x \;=\; \mathrm{partial}(q, \delta x, -1, +1);$
    $a \;=\; p_x \;+\; q_x \;+\; f;$
    %
    % Calculate temporal derivatives
    %
    $\rho_t \;=\; -(v \;.*\; \rho_x) \;-\; (\rho \;.*\; v_x);$
    $v_t \;=\; -(v \;.*\; v_x) \;-\; (a \;./\; \rho);$
    $p_t \;=\; -(v \;.*\; a) \;-\gamma * (p \;.*\; v_x);$
    %
    % Pack individual state derivatives into total state derivative vector
    %
    $xdot \;=\; [\; \rho_t(1:n) \;;\; v_t(2:n1) \;;\; p_t \;];$
return
```

Shock Waves XI

The resulting set of 151 nonlinear ODEs was simulated across 0.01 sec using the *RKF4/5 algorithm* with step-size control.



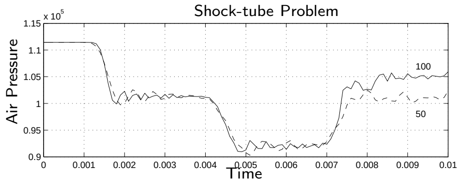
Shock Waves XII

- The bottom right curve shows the air pressure as a function of time. The solid curve depicts the pressure $20\ \mathrm{cm}$ away from the closed end, the dashed line shows the pressure $40\ \mathrm{cm}$ away, the dot-dashed line $60\ \mathrm{cm}$ away, and the dotted line $80\ \mathrm{cm}$ away.
- As the end of tube opens, the point closest to the opening experiences the rarefaction wave first. The points further into the tube experience the wave later. From the bottom right graph, it can be concluded that the wave travels through the tube with a constant wave-front velocity of roughly $35\ \mathrm{cm}$ per $0.001\ \mathrm{sec}$, or $350.0\ \mathrm{m\ sec}^{-1}$. This is the correct value of the *velocity of sound* at sea level and at a temperature of $T = 300\ \mathrm{K}$. Thus, our simulation seems to be working fine.
- As the *rarefaction wave* reaches the closed end of the tube, the inertia of the flowing air creates a vacuum. The air flows further, but cannot be replaced by more air from the left. Consequently, the air pressure now sinks below that of the outside air.
- As the vacuum reaches the open end of the tube, a new wave is created, this time a *compression wave*, that races back into the tube.

Shock Waves XIII

I ended the simulation at $t = 0.01$ sec, since shortly thereafter, the Runge-Kutta algorithm would finally give up on me, and die with an error message!

We need to ask ourselves, how accurate are the results? To answer this question, I repeated the simulation with 100 segments.



- The simulation results are visibly different. **Moreover, the differences grow over time.** Is this a consistency error, or simply the result of an inaccurate simulation?

Shock Waves XIV

- To answer this question, I repeated the same experiment once more, this time using the *BI4 algorithm* that is supposed to work for this problem at least as well as the RK algorithm.

- The simulation results are indistinguishable by naked eye. Whereas the largest relative distance between the air pressure with 50 and 100 segments:

$$err = \frac{\max(\max(\text{abs}(p_{100} - p_{50})))}{\max([\|p_{100}\|, \|p_{50}\|])}$$

  is $err = 7.5726e-4$, the largest relative distance between the air pressure with 50 segments comparing the two different integration algorithms is $err = 1.2374e-7$, and with 100 segments, it is $err = 6.3448e-7$.

- Hence the *simulation error* is smaller than the *consistency error* by three orders of magnitude. Evidently, we are not faced with a *simulation problem* at all, but rather with a *modeling problem*. The simulation is as accurate as can be expected.

Shock Waves XV

The *BI4 algorithm* turned out to be at least as accurate as the *RKF4/5 algorithm* on this problem. **Yet BI4 was disappointingly inefficient in spite of using step sizes that were quite a bit larger than those employed by RKF4/5.**

The inefficiency is caused by the computation of the *Jacobian matrix*, which was approximated numerically:

```
function [J] = jacobian(x, t)
    %
    % Jacobian of shock − tube problem
    %
    n = length(x);
    J = zeros(n, n);
    xd_ref = st_eq(x, t);
    for i = 1 : n,
        x_new = x;
        if abs(x(i)) < 1.0e − 6,
            x_new(i) = 0.05;
        else
            x_new(i) = 1.05 ∗ x(i);
        end,
        xd_new = st_eq(x, t);
        J(:, i) = (xd_new − xd_ref)/(x_new(i) − x(i));
    end
return
```

Shock Waves XVI

- Every single Jacobian, which is being computed once per integration step, requires 152 additional function evaluations in the case of a 50-segment simulation, and 302 additional function evaluations in the case of a 100-segment simulation.
- The overhead is atrocious and kills the efficiency of the algorithm.
- We shall have to do something about the size of these matrices. This problem shall be tackled in the next chapter.

What can we do to reduce the *consistency error*?

From our previous discussions, we know the answer to this question. If we increase the approximation order of the spatial derivatives by two, the consistency error is expected to decrease by two orders of magnitude.

Shock Waves XVII

To this end, I modified the partial function to use fourth-order accurate central differences instead of the previously used second-order accurate central differences:

$$\left.\frac{\partial u}{\partial x}\right|_{x=x_i} \approx \frac{1}{12\delta x} \cdot (-u_{i+2} + 8u_{i+1} - 8u_{i-1} + u_{i-2})$$

and near the boundaries:

$$\begin{aligned}
\frac{\partial u}{\partial x}(x = x_1, t) &\approx \frac{1}{12\delta x} \cdot (-3u_5 + 16u_4 - 36u_3 + 48u_2 - 25u_1) \\
\frac{\partial u}{\partial x}(x = x_2, t) &\approx \frac{1}{12\delta x} \cdot (u_5 - 6u_4 + 18u_3 - 10u_2 - 3u_1) \\
\frac{\partial u}{\partial x}(x = x_n, t) &\approx \frac{1}{12\delta x} \cdot (3u_{n+1} + 10u_n - 18u_{n-1} + 6u_{n-2} - u_{n-3}) \\
\frac{\partial u}{\partial x}(x = x_{n+1}, t) &\approx \frac{1}{12\delta x} \cdot (25u_{n+1} - 48 * u_n + 36u_{n-1} - 16u_{n-2} + 3u_{n-3})
\end{aligned}$$

Shock Waves XVIII

with the following modifications for Neumann boundary conditions:

$$\begin{aligned}
\frac{\partial u}{\partial x}(x = x_1, t) &\approx 0.0 \\
\frac{\partial u}{\partial x}(x = x_2, t) &\approx \frac{1}{12\delta x} \cdot (-u_4 + 8u_3 + u_2 - 8u_1) \\
\frac{\partial u}{\partial x}(x = x_n, t) &\approx \frac{1}{12\delta x} \cdot (8u_{n+1} - u_n - 8u_{n-1} + u_{n-2}) \\
\frac{\partial u}{\partial x}(x = x_{n+1}, t) &\approx 0.0
\end{aligned}$$

Shock Waves XIX

- We simulated the system using **RKF4/5**.
- Unfortunately, the experiment failed miserably. The integration step size had to be reduced by three orders of magnitude to values around $h = 10^{-8}$, in order to obtain a numerically stable solution, and the results are still incorrect.
- *Shift-out* killed us.
- With step sizes that small, the higher order terms of the Taylor-series expansion become irrelevant, and RKF4/5 behaves just like forward Euler. Consequently, also the stability domain of the method shrinks to that of forward Euler, which is totally useless with eigenvalues of the Jacobian spread up and down along the imaginary axis of the complex $\lambda \cdot h$-plane.
- Unfortunately, **BI4** didn't work any better. The *roundoff errors* still killed us.

Shock Waves XX

- **Why did all simulation attempts fail after a little more than 0.01 seconds of simulated time?**

- In flow simulations (and in real flow experiments), it can happen that the top of the wave travels faster than the bottom of the wave. When this happens, the wave will eventually topple over, and at this moment, the wave front becomes infinitely steep. The flow is no longer *laminar*; it has now become *turbulent*.

- This is what happens in our shock-tube problem as subsequent versions of rarefaction and compression waves chase after each other back and forth through the tube at ever shorter time intervals. No wonder that the bottom of the three-dimensional plots of the shock-tube simulation look like the bottom of a water fall.

- The method of lines method doesn't work for simulating turbulent flows. There exist other simulation techniques (such as *vortex methods*) that work well for simulating models with very high Reynolds numbers (above 100 or 1000), and that don't work at all for laminar flows.

Shock Waves XXI

- **Reynolds numbers between 1.0 (transition from laminar to turbulent flow) and 100, is where the real research in numerical solution of hyperbolic PDE problems is to be found. Until this day, we don't have any decent simulation methods that can deal appropriately with turbulent flows at low Reynolds numbers.**

Upwind Discretization

One very simple and appealing way to alleviate the numerical problems in simulating hyperbolic PDEs is to use *upwind discretization*.

The idea of upwind discretization is trivial. As a wave travels through space, e.g. from left to right, it may make sense to use in the computation of the spatial derivatives more points from the "past" of the wave, i.e., from the direction, where the wave comes from, i.e., from the "upwind" direction. Rather than using *central differences*, we now use *biased differences*, biased in the upwind direction.

Many wave propagation problems can be formulated in the following way:

$$\frac{\partial u}{\partial t} + v \cdot \frac{\partial u}{\partial x} = 0.0$$

The velocity v determines the direction of flow of the wave. If $v > 0$, the wave moves from left to right. If $v < 0$, it moves from right to left.

Upwind Discretization II

The upwind discretization scheme can thus be implemented e.g. as follows:

$$\frac{\partial u}{\partial x}(x = x_i, t) \approx \begin{cases} (3u_i - 4u_{i-1} + u_{i-2})/(2\delta x) & , \quad v \gg 0 \\ (u_{i+1} - u_{i-1})/(2\delta x) & , \quad v \approx 0 \\ (-u_{i+2} + 4u_{i+1} - 3u_i)/(2\delta x) & , \quad v \ll 0 \end{cases}$$

Looking at the shock-tube problem with $\alpha = \beta = 0.0$:

$$\frac{\partial \rho}{\partial t} = -v \cdot \frac{\partial \rho}{\partial x} - \rho \cdot \frac{\partial v}{\partial x}$$
$$\frac{\partial v}{\partial t} = -v \cdot \frac{\partial v}{\partial x} - \frac{1}{\rho} \cdot \frac{\partial p}{\partial x}$$
$$\frac{\partial p}{\partial t} = -v \cdot \frac{\partial p}{\partial x} - \gamma \cdot p \cdot \frac{\partial v}{\partial x}$$

we recognize three traveling waves each with a correction term.

Upwind Discretization III

We encoded the upwind formulae in the Matlab function:

$$\mathbf{u_x} \;= \text{upwindv}(\mathbf{u}, \delta x, bc, bctype, \mathbf{fdirv});$$

where **fdirv** is a vector of flow directions.

Carver proposed in his **Forsim-VI** manual, from where we borrowed the shock-tube example, to compute the spatial derivatives of the ρ-variable by upwind discretization. I did the same, and it worked, but didn't accomplish much, i.e., it did not resolve any of the numerical difficulties encountered earlier.

I then tried to also compute the spatial derivatives of the v- and p-variables by upwind discretization, but this didn't help at all. The numerical problems actually got worse.

Upwind Discretization IV

- **Simulation hyperbolic PDEs is still more of an art than a science.**

- We are still far from a situation, where we can formulate a hyperbolic PDE problem by just writing down the model and then press a button to get the model simulated.

- Unfortunately, the numerical solution depends heavily on the right combination of algorithms employed, and it is not easy to know beforehand, which algorithms to use. This requires much experience and often quite a bit of experimentation.

- *The simulation of hyperbolic PDEs is to this day the discipline of kings among applied mathematicians. Most mathematicians have meanwhile left the field of numerical ODE solutions, considering the problem solved once and for all (although it is not). Hyperbolic PDEs are nowadays their preferred playground.*

Grid-width Control

How can we make the solution more accurate without paying too much for it?

We already know that it is generally a bad idea to reduce the consistency error by decreasing the grid width. It is much more effective to increase the approximation order of the spatial discretization scheme, whenever possible. Yet, the shock-tube problem has demonstrated that this approach may not always work.

A more narrow grid may be needed in order to accurately compute a wave front. It seems intuitively evident that a more narrow grid width should be used where the absolute spatial gradient is large, thus:

$$\delta x_i(t) \propto \left| \frac{\partial u}{\partial x}(x = x_i, t) \right|^{-1}$$

In the context of hyperbolic PDEs, this unfortunately suggests use of an *adaptively moving grid*, since the narrowly spaced regions of the grid should follow the wave fronts through space and time.

Grid-width Control II

Mack Hyman published some interesting work relating to this issue. He proposed the following:

- We basically operate on a *fixed grid* as before.
- However, we want to make sure that:

$$\delta x_i(t) \cdot \left| \frac{\partial u}{\partial x}(x = x_i, t) \right| \leq k_{\mathrm{max}}$$

- If the absolute spatial gradient grows at some point in space and time, we must reduce the local grid size in order to keep the above inequality satisfied.
- We do this by inserting a new auxiliary grid point in the middle between two existing points.

Grid-width Control III

- We should do this before the consistency error grows too large. It makes sense to look at the quantity:

$$\frac{1}{h}\left(\left|\frac{\partial u}{\partial x}(x=x_i,t=t_k)\right|-\left|\frac{\partial u}{\partial x}(x=x_i,t=t_{k-1})\right|\right)\approx\frac{d}{dt}\left(\left|\frac{\partial u}{\partial x}(x=x_i,t)\right|\right)$$

- If the inequality is in danger of not being satisfied any longer and if the temporal gradient of the absolute spatial gradient is positive, we insert a new grid point.

- On the other hand, if the inequality shows a sufficiently small value and if furthermore the temporal gradient is negative, neighboring auxiliary grid points can be thrown out again.

- The new grid point solutions are computed using spatial interpolation. These solutions are then used as initial conditions for the subsequent integration of the newly activated differential equations over time. When a grid point is thrown out again, so is the differential equation that accompanies it.

Grid-width Control IV

- The entire process is completely transparent to the user. Only those solution points are reported, for which a solution had been requested. The actually used basic grid width (determined using true grid-width control at time zero) and the auxiliary grid points that are introduced and removed during the simulation are internal to the algorithm, and the casual user doesn't need to be made aware of their existence.

- This is analogous to the concepts of *communication points* and a *communication interval*, which are disjoint from the concepts of the *step size* and the *sampling rate* introduced when discussing integration across time.

Conclusions

- In this presentation, we have looked at the simulation of *hyperbolic PDEs in a single space dimension*.
- We have discovered that there is a strong interaction between the efficiency of the simulation and the combination of numerical algorithms used to achieve it.
- The simulation of hyperbolic PDEs still defies the desire to fully automate the process of converting the model to a form that can be successfully simulated. The user requires a lot of insight in order to choose the best possible combination of algorithms, and often insight is not enough. The modeling and simulation environment should offer a fairly large number of algorithms that can be conveniently and easily combined so that the user can experiment with them.
- This was the approach taken in **Forsim-VI**, a Fortran-coded software environment for the numerical simulation of PDEs.

Conclusions II

- However, other environments have chosen a different approach.
- The currently most successful environment for the simulation of PDEs is **Comsol** (formerly **Femlab**).
- Comsol offers templates for a fairly large variety of PDE problems. Each template is simulated by a set of algorithms well suited for the specific type of PDE problem.
- The user selects the appropriate template and adjusts it to his or her needs by entering the correct coefficients in the form of template parameters.
- Comsol is not as general as Forsim, because it may happen that a user doesn't find any template that matches his or her specific needs, but if and when a suitable template is available, Comsol is much easier to use and in most cases quite efficient in its simulations.

References

1. Dshabarow, F., F.E. Cellier, and D. Zimmer (2008), "Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters," *Proc. 6th International Modelica Conference*, Bielefeld, Germany, Vol.2, pp.683-690.

2. Dshabarow, Farid (2007), *Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters*, MS Thesis, Dept. of Computer Science, ETH Zurich, Switzerland.