1856-23

2007 Summer College on Plasma Physics

30 July - 24 August, 2007

Numerical methods and simulations. Lecture 3: Simulation of partial differential equations.

B. Eliasson

Institut Fuer Theoretische Physik IV
Ruhr-Universitaet Bochum

Bochum, Germany

Summer College on Plasma Physics
ICTP
Trieste, Italy
30 July – 24 August 2007

Course: **Numerical methods and simulations**

Tutor: Bengt Eliasson
E-mail: bengt@tp4.rub.de
Internet address: `www.tp4.rub.de/~bengt`

Lecture 3: Simulation of partial differential equations

- Will continue with simulations. Today partial differential equations
  - Korteweg–de Vries–Burgers' equation: A model for fluid flow with dissipation (e.g. collisions) and dispersion.

  Assignment: Modify the program to solve the cubic nonlinear Schrödinger equation.

We will learn how to simulate partial differential equations which depend both on space and time. As an example, we will simulate the Korteweg–de Vries–Burgers' equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + \frac{1}{2}\frac{\partial u^2}{\partial x} + \alpha\frac{\partial^3 u}{\partial x^3} - \beta\frac{\partial^2 u}{\partial x^2} = 0, \tag{1}$$

where α is the coefficient for the dispersion and β is the coefficient for the dissipation (viscosity).

To solve a partial differential equation numerically in time, we discretize the solution in both space and time so that the solution is only defined at discrete points, separated by the timestep Δt and space interval Δx. We will do the simulation from time $t = 0$ to $t = t_{end} = 12$, where t_{end} is the end time of the simulation. The space interval is from $x = 0$ to $x = L_x = 40$. We will use $N_t = 12000$ intervals in time and $N_x = 1000$ intervals in space, so that the timestep is $\Delta t = 0.001$ and the spatial grid size is $\Delta x = 0.04$.

In our numerical algorithm, time is discretized as $t = t^k = k\Delta t$, $k = 0, \ldots, N_t$, and space as $x = x_j = j\Delta x$, $j = 0, \ldots, N_x - 1$. We denote the solution $u(x_j, t^k) \equiv u_j^k$. We will use periodic boundary conditions so that $u(L,t) = u(0,t)$, or, for the discretized solution, $u_{Nx}^k = u_0^k$.

The spatial derivatives will be approximated with difference approximations, so that

$$\frac{\partial u(x_j)}{\partial x} \approx \frac{u_{j+1} - u_{j-1}}{2\Delta x} \tag{2}$$

and

$$\frac{\partial^2 u(x_j)}{\partial x^2} \approx \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{(\Delta x)^2} \tag{3}$$

for $j = 0, \ldots, N_x - 2$. The third-derivative in Eq. (1) is approximated by combining (2) and (3). At the boundaries we will use periodic boundary conditions so that

$$\frac{\partial u(x_0)}{\partial x} \approx \frac{u_1 - u_{Nx-1}}{2\Delta x} \tag{4}$$

$$\frac{\partial^2 u(x_0)}{\partial x^2} \approx \frac{u_1^k - 2u_0^k + u_{Nx-1}^k}{(\Delta x)^2} \tag{5}$$

for $j = 0$, and

$$\frac{\partial u(x_{Nx-1})}{\partial x} \approx \frac{u_0 - u_{Nx-2}}{2\Delta x} \tag{6}$$

$$\frac{\partial^2 u(x_{Nx-1})}{\partial x^2} \approx \frac{u_0^k - 2u_{Nx-1}^k + u_{Nx-2}^k}{(\Delta x)^2} \tag{7}$$

for $j = N_x - 1$.

We will use the Runge-Kutta algorithm for the time-stepping. We re-write the K-dV-Burgers equation as

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x} - \frac{1}{2}\frac{\partial u^2}{\partial x} + \alpha\frac{\partial^3 u}{\partial x^3} + \beta\frac{\partial^2 u}{\partial x^2} \equiv F(u,x,t). \tag{8}$$

In our simulation, $\mathbf{F}(\mathbf{u}, \mathbf{x}, t)$ will contain the discretized right-hand side of the K-dV-Burgers equation, where we use the difference approximations for the spatial derivatives, and we have denoted the unknowns $\mathbf{u} = [u_0\ u_1\ u_2\ \ldots\ u_{N_x-1}]$ and $\mathbf{x} = [x_0\ x_1\ x_2\ \ldots\ x_{N_x-1}]$.

The Runge-Kutta algorithm then becomes

(1) $\mathbf{F}_1 \leftarrow \mathbf{F}(\mathbf{u}, t)$

(2) $\mathbf{F}_2 \leftarrow \mathbf{F}(\mathbf{u} + \Delta t \mathbf{F}_1/2, t + \Delta t/2)$

(3) $\mathbf{F}_3 \leftarrow \mathbf{F}(\mathbf{u} + \Delta t \mathbf{F}_2/2, t + \Delta t/2)$

(4) $\mathbf{F}_4 \leftarrow \mathbf{F}(\mathbf{u} + \Delta t \mathbf{F}_3, t + \Delta t)$

(5) $\mathbf{u} \leftarrow \mathbf{u} + (\Delta t/6)(\mathbf{F}_1 + 2\mathbf{F}_2 + 2\mathbf{F}_3 + \mathbf{F}_4)$

This gives the solution $\mathbf{u}$ at time $t+\Delta t$. The steps (1)–(5) are repeated with the new values of $\mathbf{u}$ until we have reached the end of the simulation.

We now write the main program "main.m" for our simulation. Select from the Matlab menu *File / New / M-file*, then from the menu of the new window, choose *File / Save as*, and Filename: `main.m`

We now write the program in the file:

```
==============================================

% The main program: main.m

clear

Nt=12000;  % Number of time steps
Nprints=200; % Number of times to save data and print the results

Lx=40;  % box length
Nx=1000;  % Number of x intervals
dx=Lx/Nx; % Delta x

x=(0:(Nx-1))*Lx/Nx; % The x variable
dt=0.001; % The time step

t=0; % Time starts at zero.


%%% The initial condition %%%%%%%%%
  for j=1:Nx
     u(j)=1+exp(-(x(j)+15)^2/5);
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Time-stepping using the Runge-Kutta algorithm
```

```
for j=1:Nt
  [u]=RungeKutta(u,t,dt,dx);
  if mod(j*Nprints,Nt)==0
    plot(x,u);
    axis([0 40 0 3]); % Set the axis scaling in the figure
    title('Velocity')
    pause(0.01); % Make a pause of 0.01 seconds to plot the solutions
  end
end

==========================================================================
```

Then choose from the menu *File / Save*.

Now we write the Runge-Kutta subroutine: From the Matlab menu, choose *File / New / M-file*, then choose from the menu of the new file *File / Save as*, and Filename: `RungeKutta.m`

In the new file, we write

```
==========================================================================

% The Runge-Kutta algorithm: RungeKutta.m
function [u]=RungeKutta(u,t,dt,dx)

  [F1_u]=F(u,t,dx);
  [F2_u]=F(u+0.5*dt*F1_u,t+0.5*dt,dx);
  [F3_u]=F(u+0.5*dt*F2_u,t+0.5*dt,dx);
  [F4_u]=F(u+dt*F3_u,t+dt,dx);

  u=u+dt/6*(F1_u+2*F2_u+2*F3_u+F4_u);

==========================================================================
```

Choose *File / Save*.

Next, we write the function that defines the right-hand side of the differential equation. From the Matlab menu, choose *File / New / M-file*, and then *File / Save as* and Filename: `F.m`

In the new file, we write

```
==========================================================================

% Definition of the differential equation: F.m
function [F_u]=f(u, t, dx)
```

```
  alpha=0.05;
  beta=0.05;

  F_u=-d1x(u.^2/2+u+alpha*d2x(u,dx),dx)+beta*d2x(u,dx);
```

==

Choose *File / Save.*

The spatial derivatives are defined in separate functions. The first derivative is defined in *d1x.m.* From the Matlab menu, choose *File / New / M-file*, and then *File / Save as* and Filename: `d1x.m` In the new file, we write

==

```
%Function for calculating d/dx: d1x.m
function d1x=d1x(y,dx)

N=length(y);

d1x(2:N-1)=(y(3:N)-y(1:N-2))/(2*dx);

d1x(1)=(y(2)-y(N))/(2*dx);
d1x(N)=(y(1)-y(N-1))/(2*dx);
```

==

Choose *File / Save.*

Finally, the second derivative is defined in *d2x.m.* From the Matlab menu, choose *File / New / M-file*, and then *File / Save as* and Filename: `d2x.m` In the new file, we write

==

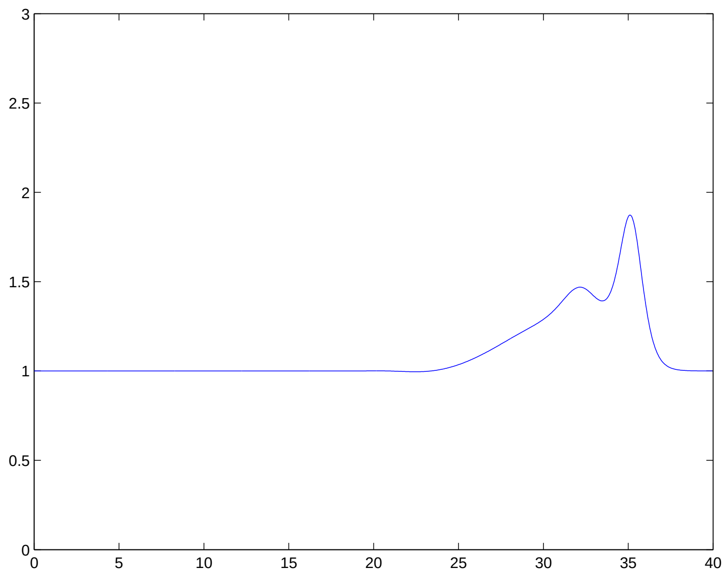
```
% Function for calculating d^2/dx^2: d2x.m.
function d2x=d2x(y,dx)

N=length(y);

d2x(2:N-1)=(y(3:N)-2*y(2:N-1)+y(1:N-2))/dx^2;

d2x(1)=(y(2)-2*y(1)+y(N))/dx^2;
d2x(N)=(y(1)-2*y(N)+y(N-1))/dx^2;
```

==

Choose *File / Save*.

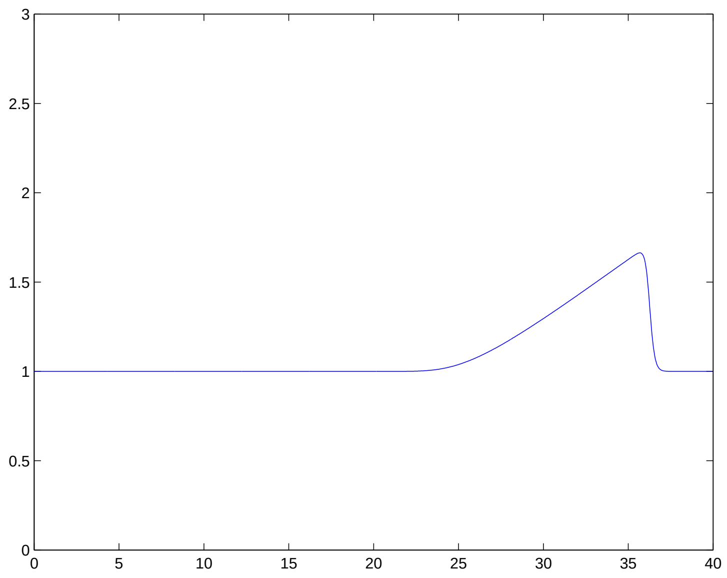
We now run the program. In the Matlab window, type in `main` (Enter).

The result is



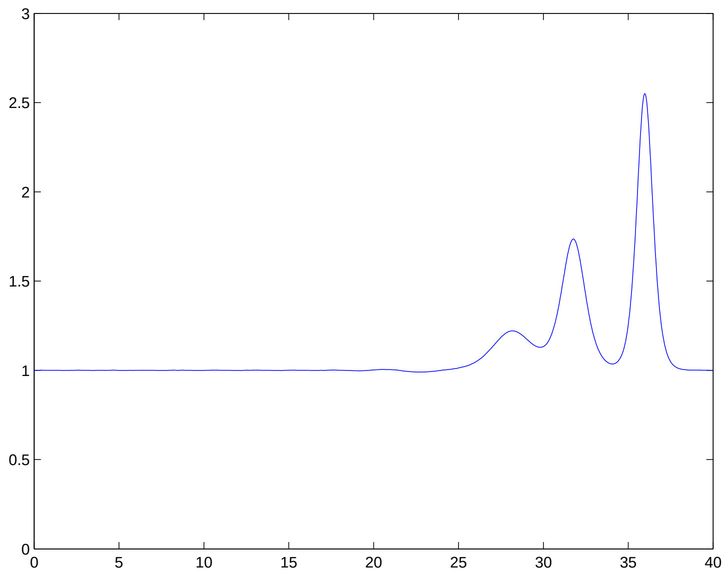
We see that the pulse self-steepens and at the end has an oscillatory shock structure with two maxima.

Now change to `alpha=0` and `beta=0.05` in *F.m* and run the program again. The result is



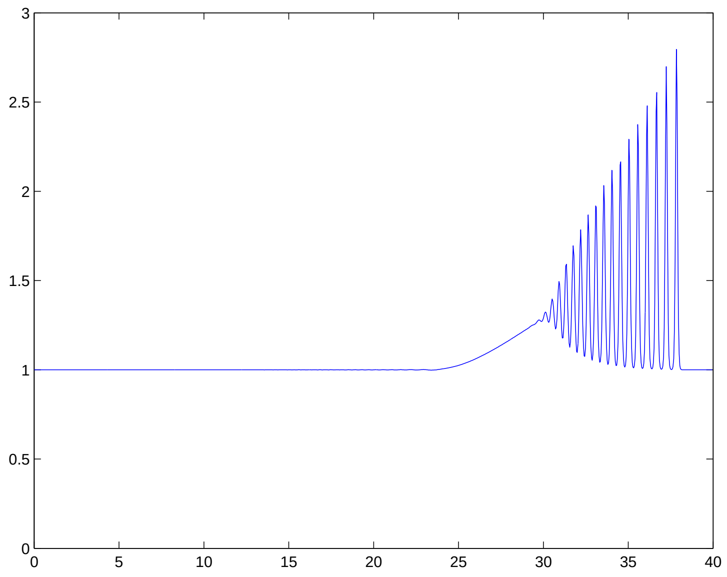
At the end we see a monotonic shock structure.

Now change to `alpha=0.05` and `beta=0.0` in *F.m* and run the program again. The result is



In this case, the solution breaks up into solitary waves (K-dV solitons).

Finally, change to `alpha=0` and `beta=0` in *F.m* and run the program again. The result is

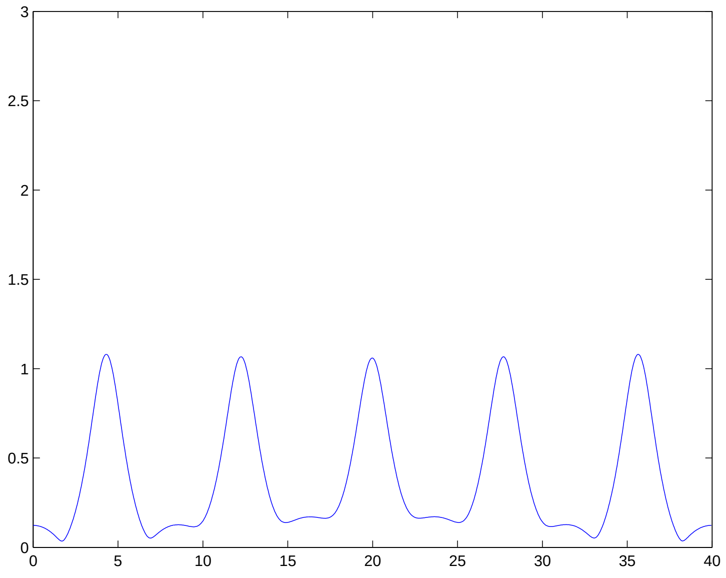


Here, we see very small-scale oscillations at the end. This is an **unphysical numerical effect** due to *numerical dispersion.* For nonlinear problems, it is often necessary to introduce viscosity into the problem in order to make the numerical method to work and to avoid oscillations and numerical instability.

Assignment: Modify the program so that it solves the cubic Schrödinger equation

$$i\left(\frac{\partial u}{\partial t}+\frac{\partial u}{\partial x}\right)+\frac{1}{2}\frac{\partial^2 u}{\partial x^2}+|u^2|u=0, \tag{9}$$

with the initial conditions $u = 0.5+0.01\cos(2\pi x/40)$. Solve the system from $t = 0$ to $t = 100$ (use `Nt=50000` and `dt=0.002`) and plot $|u|$. In Matlab, π is written `pi`, and $|u|$ is written `abs(u)`, and the imaginary unit is written `i`. At the end of the simulation, the result is



The solution first undergoes a modulational instability and then start to oscillate both in space and time, a phenomenon known as *Fermi–Pasta–Ulam* (FPU) oscillations.