Solution of the Wave equation with Spectral Galerkin and Leapfrog

Term Project, Seminar for Applied Mathematics

Supervision: Prof. Dr. Ralph Hiptmair
Urs Zellweger

ETH Zurich, SS2006

Contents

1 Problem

The one-dimensional wave equation

$$\frac{\partial^2 u(x,t)}{\partial t^2} - \frac{\partial^2 u(x,t)}{\partial x^2} = 0 \tag{1}$$

with border conditions

$$u(0,t) = 0 \tag{2}$$

$$(\partial_x + \partial_t)u(1,t) = 0 \tag{3}$$

and initial conditions

$$u(x,0) = u_0(x) \tag{4}$$

$$\partial_t u(x,0) = v_0(x). \tag{5}$$

is to investigate for $x \in [0,1]$ and $t \in [0,T]$. To do this, a method called *Spectral Galerkin* has to be used to discretize the equation in space (x-dimension) and afterwards it is discretized in time with a routine called *dissipative Leapfrog*.

The variational formulation of the problem in eq. (1) is obtained by associating the wave equation with a bilinear form $B(u,v)$. This is done by multiplying eq. (1) by $v \in Y$ and integrating:

$$\int_0^1 \partial_t^2 u(x,t)v(x)dx \quad - \quad \int_0^1 \partial_x^2 u(x,t)v(x)dx = 0 \tag{6}$$

$$\int_0^1 \partial_t^2 u(x,t)v(x)dx \quad + \quad \int_0^1 \partial_x u(x,t)\partial_x v(x)dx - \partial_x u(1,t)v(1) = 0 \tag{7}$$

Here, the second equation follows with integration by parts. This corresponds to an equation $B(u,v) = 0\ \forall v \in Y$. A finite element method (FEM) starts with such a variational formulation of a boundary problem and generates approximate solutions for $u(x,t)$. This means that $u(x,t)$ is represented in terms of n basis functions of a specified function space X (evidently with dimension n):

$$u_n(x) \approx \sum_{i=1}^{n} a_i(t)N_i(x). \tag{8}$$

The N_i, $i = 1,\ldots,n$ are linearly independent functions of the space in which we expect our solution. The a_i's are coefficients which contain the time-dependency of the problem. If we choose our function space to be equal to the test space (i.e. $X = Y$) the method is referred to as Spectral Galerkin. In this space, we set the basis functions to be the *integrated Legendre Polynomials* on $\hat{\Omega} = [-1,1]$,

$$N_1 \quad = \quad \frac{1-x}{2} \tag{9}$$

$$N_2 \quad = \quad \frac{1+x}{2} \tag{10}$$

$$N_i \quad = \quad \sqrt{\frac{2i-3}{2}} \int_{-1}^{x} L_{i-2}(x)dx \text{ for } i = 3,\ldots,n. \tag{11}$$

With this setup, we can start to solve our problem.

First, we discretize the problem on the x-dimension. So we divide the interval $[0,1]$ into (not necessarily equally spaced) subintervals. That means, we divide the interval into nodes (nodal points),

$$0 = x_0 < x_1 < \ldots < x_M = 1 \tag{12}$$

and elements (open, disjoint intervals)

$$\Omega_j = (x_{j-1}, x_j) \text{ for } j = 1, \ldots, M. \tag{13}$$

Such a partition into open, disjoint subintervals is called a *mesh*. On each of these intervals, the variational formulation in eq. (7) has to be true. So we can reformulate the problem for every interval in $j = 1, \ldots, M-1$ like that

$$\int_{x_{j-1}}^{x_j} \partial_t^2 u(x,t) v(x) dx + \int_{x_{j-1}}^{x_j} \partial_x u(x,t) \partial_x v(x) dx = 0. \tag{14}$$

For the interval $j = M$ we have to take care because of the last term in eq. (7):

$$\int_{x_{M-1}}^{x_M} \partial_t^2 u(x,t) v(x) dx + \int_{x_{M-1}}^{x_M} \partial_x u(x,t) \partial_x v(x) dx - \partial_x u(1,t) v(1) = 0. \tag{15}$$

Boundary conditions of *von Neumann* type have to be taken into account in the variational formulation. The second boundary condition in eq. (3) is of such a type. Inserting it into the last equation, we arrive with

$$\int_{x_{M-1}}^{x_M} \partial_t^2 u(x,t) v(x) dx + \int_{x_{M-1}}^{x_M} \partial_x u(x,t) \partial_x v(x) dx + \partial_t u(1,t) v(1) = 0. \tag{16}$$

In a next step, we can choose for every interval Ω_j a polynomial degree p_j, which is one factor to cause accuracy. According to this degree, in every interval are the coefficients a_i of the series in eq. (17) for $i = 1, \ldots, p_j$ to determine. Note that we approximate $u(x,t)$ for every interval seperately:

$$u_{p_j}^{(j)}(x) \approx \sum_{i=1}^{p_j} a_i^{(j)}(t) N_i(x). \tag{17}$$

Since the integrated Legendre polynomials in eq. (11) are created for the interval $[-1,1]$, we have to transform these $N_i(x)$ for each subinterval Ω_j. This can be done by the following mapping:

$$\begin{aligned} \Omega_j &= Q_j(\hat{\Omega}) && (18)\\ x = Q_j(\chi) &= \frac{1}{2}(1-\chi)x_{j-1} + \frac{1}{2}(1+\chi)x_j && (19)\\ \chi = Q_j^{-1}(x) &= \frac{2x - x_j - x_{j-1}}{x_j - x_{j-1}}. && (20) \end{aligned}$$

This results then in new basis functions for each subinterval by $\Psi_i^{(j)}(x) := N_i(Q_j^{-1}(x))$. There is again one exception, this time for the first interval, $j = 1$. Our first boundary

condition, eq. (2), a *Dirichlet* condition, is only fulfilled, if all basis functions vanish at $x = 0$. Since only $\Psi_1^{(1)}(0) \neq 0$ does not match this condition, we simply drop it in the first interval, leading to a smaller basis with dimension $p_1 - 1$.
Summarized, for the last interval $j = M$ there is the following equation to solve for every $k = 1, \dots, p_M$:

$$\begin{aligned}
\int\limits_{x_{M-1}}^{x_M} \partial_t^2 \sum_{i=1}^{p_M} a_i(t)\Psi_i^{(M)}(x)\Psi_k^{(M)}(x)dx + & \cdots \\
\int\limits_{x_{M-1}}^{x_M} \sum_{i=1}^{p_M} a_i(t)\Psi_i^{'(M)}(x)\Psi_k^{'(M)}(x)dx + & \cdots \\
\sum_{i=1}^{p_M} \partial_t a_i(t)\Psi_i^{(M)}(1)\Psi_k^{(M)}(1) & = 0 \qquad (21) \\
\sum_{i=1}^{p_M} \partial_t^2 a_i(t) \int\limits_{x_{M-1}}^{x_M} \Psi_i^{(M)}(x)\Psi_k^{(M)}(x)dx + & \cdots \\
\sum_{i=1}^{p_M} a_i(t) \int\limits_{x_{M-1}}^{x_M} \Psi_i^{'(M)}(x)\Psi_k^{'(M)}(x)dx + & \cdots \\
\sum_{i=1}^{p_M} \partial_t a_i(t)\Psi_i^{(M)}(1)\Psi_k^{(M)}(1) & = 0 \qquad (22)
\end{aligned}$$

Here, the prime indicates the partial derivative with respect to x, i.e. $\Psi' = \partial_x \Psi$. Note that the equation for the other subintervals ($j = 1, \dots, M-1$) is easily taken from the last equation, dropping the last term and substituting M by j. With the next little calculation, this can be simplified. We just compute the integrals...

$$M_{ik}^{(j)} = \int\limits_{x_{j-1}}^{x_j} \Psi_i^{(j)}(x)\Psi_k^{(j)}(x)dx = \frac{x_j - x_{j-1}}{2}\int\limits_{-1}^{1} N_i(x)N_k(x)dx \qquad (23)$$

$$K_{ik}^{(j)} = \int\limits_{x_{j-1}}^{x_j} \Psi_i^{'(j)}(x)\Psi_k^{'(j)}(x)dx = \frac{2}{x_j - x_{j-1}}\int\limits_{-1}^{1} N_i(x)N_k(x)dx \qquad (24)$$

$$P_{ik}^{(M)} = \Psi_i^{(M)}(1)\Psi_k^{(M)}(1) \qquad (25)$$

... and can rewrite eq. (22) as

$$M^{(j)}\partial_t^2 \vec{a}^{(j)} + K^{(j)}\vec{a}^{(j)} + \delta_{jM}P^{(M)}\partial_t \vec{a}^{(j)} = 0. \qquad (26)$$

This is valid for every interval. It is a system with p_j unknown coefficients a_i and the same amount of equations. As a next step, we want to collect all these equations into one system for all intervals. That means we want to end up with an equation like

$$M\partial_t^2 \vec{a} + K\vec{a} + P\partial_t \vec{a} = 0. \qquad (27)$$

To do this, we have to regard that corresponding functions of neighboring elements must be joined at the nodes. Since we have only two basis functions which are *external* *and nodal* (that means not vanishing at one of the two borders), there is only one basis function associated with node j, given by

$$\Phi_1^{(j)} = \begin{cases} \Psi_2^{(j)}(x) & : \quad x \in \Omega_j \\ \Psi_1^{(j+1)}(x) & : \quad x \in \Omega_{j+1} \\ 0 & : \quad \text{else} \end{cases} \tag{28}$$

In the assembly of the matrices in eq. (27) and in the calculation of the coefficients, we have to include this. Then we end up with the following matrix for K,

$$K = \begin{pmatrix} K^{(1)} & & & 0 \\ & \ddots & & \\ & & K^{(j)} & \\ 0 & & & \ddots \end{pmatrix} \tag{29}$$

and analogously for M. P has only a contribution for the last interval, hence is has only a matrixpart in the lower right corner.

Now after motivating eq. (27), we want to solve it in a last step. As already mentioned in the beginning, this equation can be solved by a routine called dissipative Leapfrog. For this, we rewrite it (note that superscript now indicates the time-step, i.e. $k = 1 \Rightarrow \vec{a}^{(k)} = \vec{a}(dt)$ and so on):

$$M\frac{\vec{a}^{(k+1)} - 2\vec{a}^{(k)} + \vec{a}^{(k-1)}}{(\Delta t)^2} + K\vec{a}^{(k)} + P\frac{\vec{a}^{(k+1)} - \vec{a}^{(k-1)}}{2\Delta t} = 0. \tag{30}$$

Rearranging the terms, it follows that:

$$(M + \frac{\Delta t}{2}P)\vec{a}^{(k+1)} = (2M - (\Delta t)^2 K)\vec{a}^{(k)} + (\frac{\Delta t}{2}P - M)\vec{a}^{(k-1)}. \tag{31}$$

To solve this equation, we need two initial coefficient vectors, $\vec{a}^{(0)}$ and $\vec{a}^{(1)}$. The first one, $\vec{a}^{(0)}$, we get from a projection of the starting condition in eq. (4) onto our basis. From the projection of the second initial condition in eq. (5), we get $\partial_t a^{(0)}$. With these two and the equation

$$\frac{u^{(-1)} - u^{(1)}}{2\Delta t} = \partial_t u(0), \tag{32}$$

we are able to calculate the first step in the Leapfrog method and can then continue iteratively with the eq. (31).

It appears, that the choice of Δx and Δt are related very sensitive. The condition, which has to be satisfied by Δt depending on Δx is known as the *CFL condition*. If it is not satisfied, an exponential blow-up in the computed solution is observable. The upper limit for Δt is given through

$$\frac{(\Delta t)^2}{4}\lambda_{\max} \leq 1, \tag{33}$$

where $\lambda_{\max}$ is the biggest eigenvalue of the generalized eigenvalue problem $Kx = \lambda Mx$, with the matrices M, K as defined above.

2 Numerical experiments

2.1 Time step vs. polynomial degree

The width of the time step depends as already mentioned on the degree of the polynomial. In figure 1 is the dependency of the inverse time step plotted against the polynomial degree for different numbers of intervals. The MATLAB function `polyfit`

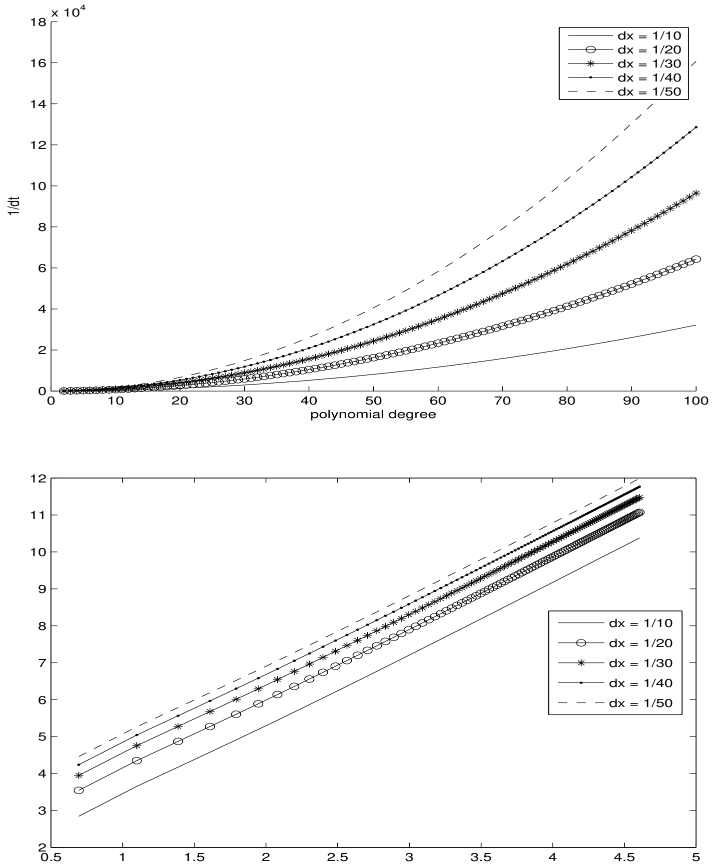

Figure 1: Inverse time step versus polynomial degree, normal and logarithmic scales.

calculates in the average for the relation

$$\log(1/dt) = \alpha + \beta \cdot \log(\deg) \tag{34}$$

a coefficient $\beta = 1.9416$. This means, that

$$1/dt \propto \deg^{1.9416}. \tag{35}$$

Evidently, the exact connection between dt and deg depends also on the interval width dx.

2.2 Error development

In this subsection, the development of the error of the numerical solution in comparison with the exact analytic solution is investigated. For this reason, we look at three different functions, the hat-function $\hat{h}$, the $\cos^2$-function and the Gaussian bell-shaped curve, and measure the errors in L^2 norm, depending on dx, dt and the polynomial degree. The analytic solutions of the three mentioned waves are simple superpositions. Since we know, that $f(x \pm t)$ solves eq. (1) for any function f, we can make the superposition $F(\xi) = f(\xi) - f(-\xi)$ and this for $\xi = x \pm t$. Obviously, this solves the wave equation and in addition, the border conditions are fulfilled automatically. In figure 2, the three functions are depicted at time $t = 0$.

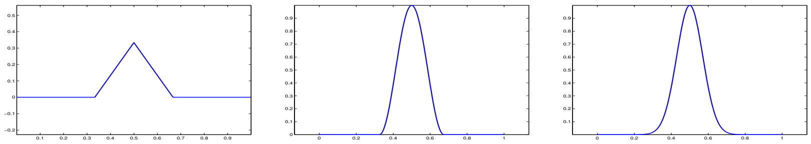

Figure 2: A hat-function, a cosine peak squared and a gaussian bell-shape.

2.2.1 Hat function

The hat function $\hat{h} \in C^0$ at time $t = 0$ is given by

$$\hat{h} = \begin{cases} 0 & : \quad x \in [0, \frac{1}{3}] \\ x - \frac{1}{3} & : \quad x \in (\frac{1}{3}, \frac{1}{2}] \\ \frac{2}{3} - x & : \quad x \in (\frac{1}{2}, \frac{2}{3}] \\ 0 & : \quad x \in (\frac{2}{3}, 1] \end{cases} \tag{36}$$

The simulated results for $dx = 0.1$, $dt = 0.01$, $\deg = 4$ and 180 time steps are depicted in the 3D figure. The following figures show the L^2 error development in time.

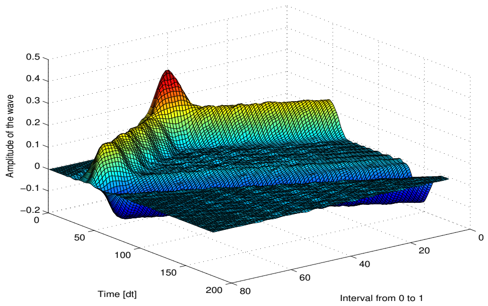

Figure 3: Simulation of the time development of a hat-function for 180 time steps.

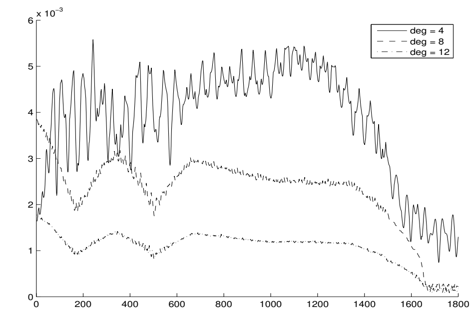

Figure 4: Hat-function: L^2 error for different degrees ($dx = 0.1, dt = 0.001$, 1800 time steps).

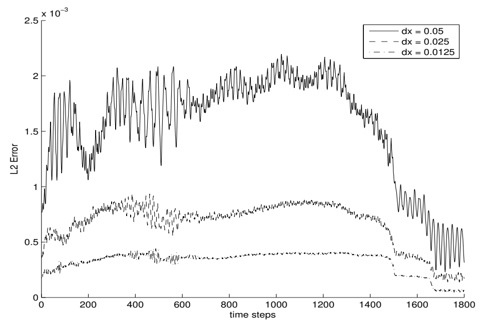

Figure 5: Hat-function: L^2 error for different interval widths ($dt = 0.001, \deg = 4$, 1800 time steps).

2.2.2 $\cos^2$ function

We look at the following function $c \in C^1$ (at time $t = 0$):

$$c = \begin{cases} 0 & : \quad x \in [0, \frac{1}{2} - \frac{\pi}{18}] \\ \cos\left(9(x - \frac{1}{2})\right) & : \quad x \in (\frac{1}{2} - \frac{\pi}{18}, \frac{1}{2} + \frac{\pi}{18}) \\ 0 & : \quad x \in [\frac{1}{2} + \frac{\pi}{18}, 0] \end{cases} \tag{37}$$

The simulated results for $dx = 0.1$, $dt = 0.01$, $\deg = 4$ and 180 time steps are depicted in the 3D figure. The following figures show the L^2 error development in time.

2.2.3 Gaussian function

The well known Gaussian function $g \in C^\infty$ at time $t = 0$ is given by

$$G = a \cdot \exp^{-b(x-c)^2}, \tag{38}$$

where we choose for our example $a = 1$, $b = 100$ (so that the curve vanishes fast enough) and $c = 0.5$ (such that the maximum of the bell-shape is in the middle of the

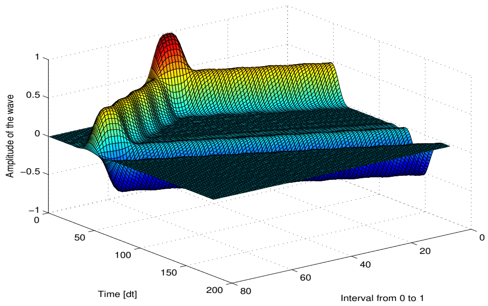

Figure 6: Simulation of the time development of a $\cos^2$-peak for 180 time steps.

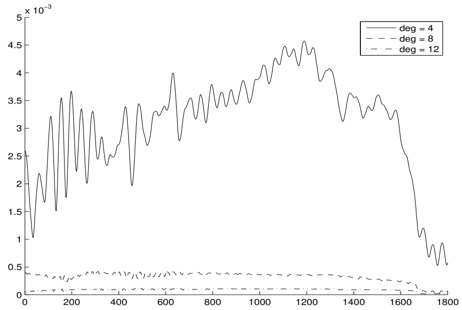

Figure 7: cos^2-peak: L^2 error for different degrees ($dx = 0.1, dt = 0.001$, 1800 time steps).

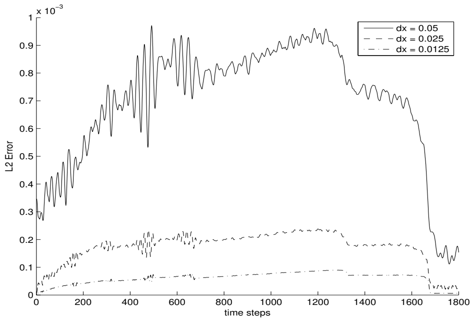

Figure 8: cos^2-peak: L^2 error for different interval widths ($dt = 0.001, \deg = 4$, 1800 time steps).

interval $[0,1]$. The simulated results for $dx = 0.1$, $dt = 0.01$, $\deg = 4$ and 180 time steps are depicted in the 3D figure. The following figures show the L^2 error development in time.

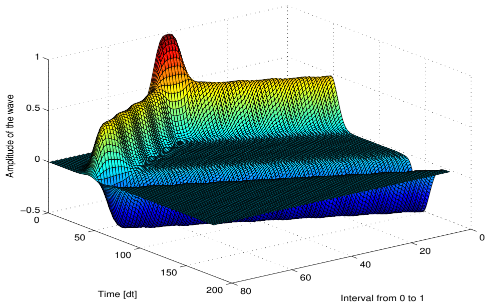

Figure 9: Simulation of the time development of a Gaussian for 180 time steps.

Figure 10: Gaussian function: L^2 error for different degrees ($dx = 0.1, dt = 0.001$, 1800 time steps).

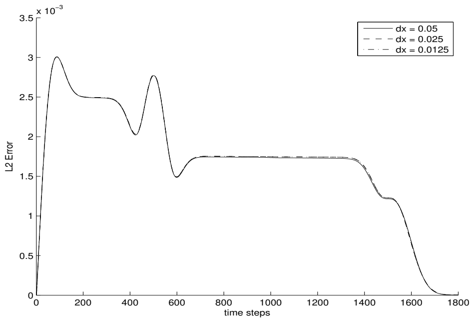

Figure 11: Gaussian function: L^2 error for different interval widths ($dt = 0.001, \mathrm{deg} = 4$, 1800 time steps).

3 Program code

The described procedure is implemented in MATLAB. The code is structed in a few subroutines and a calling main program, `waveq.m`, which is fed with the parameters.

The subroutines consist of two parts: 1) Already existing routines which could be used like a library (`localbasisfunction.m`, `legendrepoly.m`, `gauleg.m`, `assemload.m`, `assemblematrix.m`) and 2) routines which had to be written like the main program (`leapfrog.m`, `N_at_one.m`,`function_values.m` `visualization.m`,`ErrorCalculation.m`). In the following, the already existing code is decribed very short. In a part after it, the new routines are explained.

3.1 Main routine `waveq.m`

The program is then called by the command `waveq(dx,dt,deg, time_steps)`, where dx is the width of the intervals in space, i.e. $dx = (x_j - x_{j-1})$. The second parameter dt sets the width in time space, which depends on dx and the degree of the polynomials (CFL condition). The number of time steps has been set with the last input parameter. `deg` is the degree of the polynomials. Theoretically, one could choose for every interval another degree, but for simplicity the program takes for every interval the same degree, i.e. $p = \deg \cdot (1, \dots, 1)$.
The main routine is then actually just calling subroutines:

```
galueg
assemblematrix → rigide_local, masselocale
assemload → gauleg, localbasisfunction → legendrepoly
N_at_one
leapfrog
function_values
visualization
ErrorCalculation
```

These subroutines are explained in the next two sections as mentioned. In the end, the initial wave and its propagation is visualized.

3.2 Already existing subroutines

`localbasisfunction.m - [B,C] = localbasisfunction(x,p)`
This routine uses `legendrepoly.m`. The input parameters are defined as follows:
$\mathbf{x} = (\mathbf{x_0}, \dots, \mathbf{x_M})$, a vector containing nodes in the interval $[0,1]$. dim$= (1, M+1)$
$\mathbf{p} = (\mathbf{p_1}, \dots, \mathbf{p_M})$, a vector containing the polynomial degree for each subelement of the interval. dim$= (1, M)$
This routine then creates two matrices B and C, which contain the following values:

$$B = \begin{pmatrix} N_1(x_0) & N_1(x_1) & \dots & N_1(x_M) \\ \vdots & & & \vdots \\ N_{p+1}(x_0) & \dots & \dots & N_{p+1}(x_M) \end{pmatrix} \tag{39}$$

$$C = \begin{pmatrix} N_1'(x_0) & N_1'(x_1) & \dots & N_1'(x_M) \\ \vdots & & & \vdots \\ N_{p+1}'(x_0) & \dots & \dots & N_{p+1}'(x_M) \end{pmatrix} \tag{40}$$

Here, the prime means d/dx and $N_i(x)$ denotes as in the text before the integrated Legendre polynomials (basis functions).

`legendrepoly.m` – *L = legendrepoly(x,p)*
This routine generates a matrix L, containig the values of the Legendre polynomials at the nodes of x:

$$L = \begin{pmatrix} L_0(x_0) & L_0(x_1) & \dots & L_0(x_M) \\ \vdots & & & \vdots \\ L_{p_{\max}}(x_0) & \dots & \dots & L_{p_{\max}}(x_M) \end{pmatrix} \tag{41}$$

Here, $p_{\max} = \max\{p_1, \dots, p_M\}$ and L_i is the ith Legendre polynomial.

`gauleg.m` – *quad = gauleg(a,b,n,tol)*
The inputs for this functions are:
a and b are broders of an interval $[a,b]$, a priori they are in $[-1,1]$.
n is the number of abscisses and weights one wants to calculate (the higher, the more accuracy). `tol` is the accuracy for x and w. Default is `eps`, the highest possible correctness of the machine the program is running on.
The function `gauleg.m` then creates a struct `quad` which contains two vectors, x and w. They are called by `quad.x()` and `quad.w()`, respectively. The vector x contains abscisses created with the Gaussian rule (integrating leads to a $2n-1$ accuracy). The vector w contains the corresponding weights.

`assemload.m` – *[FK,FM]=assemload(x,p)*
The inputs for this functions are:

`assemblematrix.m` – *[K,M]=assemblematrix(x,p)*
The inputs for this functions are:

3.3 New subroutines

`N_at_one.m` – *P = N_at_one(p)*
This routine is written to implement the last term of eq. (7), $\partial_t u(1,t)v(1)$. The input is: $\mathbf{p} = (\mathbf{p_1}, \dots, \mathbf{p_M})$, a vector containing the polynomial degree for each subelement of the interval. dim$= (1,M)$
The output of the routine is a matrix `P` with the following content:

$$P = \begin{pmatrix} N_1'(1)N_1(1) & N_2'(1)N_1(1) & \dots & N_{p+1}'(1)N_1(1) \\ \vdots & & & \vdots \\ N_1'(1)N_{p+1}(1) & \dots & \dots & N_{p+1}'N_{p+1}(1) \end{pmatrix} \tag{42}$$

`leapfrog.m` – *Coeff = leapfrog(M, A, a_0, da_0, steps)*
This routine is written to solve eq. (27). The inputs are:
M, a matrix, as explained in the first section. dim$= (n,n)$
A, a matrix, as explained in the first section. dim$= (n,n)$
$\mathbf{a_0}$, a vector containing the coefficients of $u_0(x) = \sum a_i(0)N_i(x)$. dim$= (1,n)$
$\mathbf{da_0}$, a vector containing the coefficients of $v_0(x) = \sum da_i(0)N_i(x)$. dim$= (1,n)$
`steps`, an integer of how many time steps should be evaluated.
The output of the routine is then a matrix `Coeff`, which contains the coefficients for

every time step:

$$\text{Coeff} = \begin{pmatrix} a_1^0 & a_2^0 & \dots & a_n^0 \\ a_1^1 & a_2^1 & \dots & a_n^1 \\ a_1^2 & a_2^2 & \dots & a_n^2 \\ \vdots & & & \vdots \\ a_1^{\text{steps}} & \dots & \dots & a_n^{\text{steps}} \end{pmatrix} \tag{43}$$

As mentioned, the first row of this matrix corresponds to the vector a_0. The second row can then be evaluated by bla.

`function_values.m` – *[U, X_all] = function_values(u0, Coeff, Y, x, p)*
This routine is written to store the values of the function u at different times. The inputs are all explained in the functions above. The outputs are:

$$U = \begin{pmatrix} u(X_0,0) & u(X_1,0) & \dots & u(X_M,0) \\ u(X_0,dt) & u(X_1,dt) & \dots & u(X_M,dt) \\ u(X_0,2dt) & u(X_1,2dt) & \dots & u(X_M,2dt) \\ \vdots & & & \vdots \\ u(X_0,\text{ts}dt) & u(X_1,\text{ts}dt) & \dots & u(X_M,\text{ts}dt) \end{pmatrix} \tag{44}$$

$$X_a ll = (X_1,\dots,X_M) \tag{45}$$

`visualization.m` – *F = visualization(X_all, U)*
This routine is written to plot the movie of the wave and its evolution. It creates a matrix `F` which contains all the frames to be played. To play the movie use the MATLAB command `movie(F, how_many_times, frames_per_second)`.

`ErrorCalculation.m` – *[H1,L2] = ErrorCalculation(u_orig, du_orig, u_appr, du_appr, Y, W, x, p)*
This routine calculates the L^2 error between the original function (analytic solution) `u_orig` and the approximated function (FEM solution) `u_appr`. Analogously, the H^1 error is calculated for `du_orig` and `du_appr`.