Efficient High Order DG Method for Time-domain Maxwells Equations

Christoph Lehrenfeld, Joachim Schöberl

Center for Computational Engineering Sciences (CCES)
RWTH Aachen University

MAFELAP 2009, June 11, 2009

content

1. spatial discretization
   - DG formulation
   - dissipative approach
   - non-dissipative stabilization
2. the complete scheme
   - global time-stepping
   - computational aspects
3. local time-stepping
   - definition of a local time-stepping scheme
   - implementation aspects
   - numerical examples

spatial discretization

Maxwell Equations

in an isotropic, non-conducting medium

$$\begin{aligned} \varepsilon \frac{\partial E}{\partial t} &= \mathrm{curl} H \\ \nu \frac{\partial H}{\partial t} &= -\mathrm{curl} E \\ &+ \text{boundary conditions} \end{aligned}$$

model problem: wave equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathrm{div}(v) \\ \frac{\partial v}{\partial t} &= \nabla u \\ &+ \text{boundary conditions} \end{aligned}$$

Maxwell Equations

in an isotropic, non-conducting medium

$$\begin{aligned} \varepsilon \frac{\partial E}{\partial t} &= \mathrm{curl} H \\ \nu \frac{\partial H}{\partial t} &= -\mathrm{curl} E \\ &+ \text{boundary conditions} \end{aligned}$$

model problem: wave equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \mathrm{div}(v) \\ \frac{\partial v}{\partial t} &= \nabla u \\ &+ \text{boundary conditions} \end{aligned}$$

Discontinuous Galerkin Discretization

We approximate the continuous problem

$$\begin{aligned}\frac{\partial}{\partial t}(u,\xi)_\Omega &= (\mathrm{div}(v),\xi)_\Omega \\ \frac{\partial}{\partial t}(v,\eta)_\Omega &= (\nabla u,\eta)_\Omega + b.c.\end{aligned}$$

with discontinuous u_h and v_h:

$$\begin{aligned}\xi, u_h \in U_h &:= \{v \in L^2 : v|_T \in \mathcal{P}^{k+1}(T)\} \\ \eta, v_h \in [V_h]^3, V_h &:= \{v \in L^2 : v|_T \in \mathcal{P}^{k}(T)\}\end{aligned}$$

Discontinuous Galerkin Discretization

discrete problem ($\sum_T$):

$$\frac{\partial}{\partial t}(u_h, \xi)_T = (\mathrm{div}(v_h), \xi)_T = -(v_h, \nabla\xi)_T + (v_n^*, \xi)_{\partial T}$$
$$\frac{\partial}{\partial t}(v_h, \eta)_T = (\nabla u_h, \eta)_T = (\nabla u_h, \eta)_T + (u^* - u_h, \eta_n)_{\partial T}$$

How to choose the numerical fluxes u^* and v_n^*?

Discontinuous Galerkin Discretization

discrete problem ($\sum_T$):

$$\begin{aligned}
\frac{\partial}{\partial t}(u_h, \xi)_T &= (\mathrm{div}(v_h), \xi)_T &&= -(v_h, \nabla\xi)_T + (v_n^*, \xi)_{\partial T} \\
\frac{\partial}{\partial t}(v_h, \eta)_T &= (\nabla u_h, \eta)_T &&= (\nabla u_h, \eta)_T + (u^* - u_h, \eta_n)_{\partial T}
\end{aligned}$$

How to choose the numerical fluxes u^* and v_n^*?

central flux

If we choose

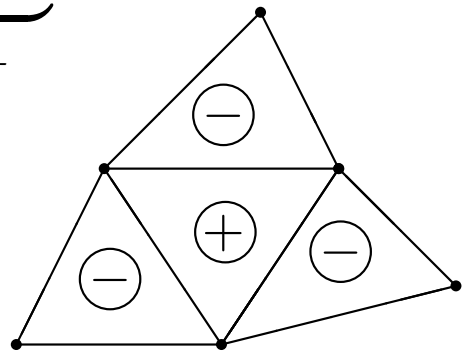
$$\begin{aligned} u^* &= \{\!\{u_h\}\!\} \\ v_n^* &= \{\!\{v_h\}\!\}_n \end{aligned}$$

we get

$$\begin{aligned} \frac{\partial}{\partial t}(u_h, \xi)_T &= -(v_h, \nabla \xi)_T + (\{\!\{v_h\}\!\}_n, \xi)_{\partial T} \\ \frac{\partial}{\partial t}(v_h, \eta)_T &= (\nabla u_h, \eta)_T - \underbrace{(\frac{1}{2}[\![u_h]\!], \eta_n)_{\partial T}}_{(u_h, \{\!\{\eta\}\!\}_n)_{\partial T}} \end{aligned}$$

with $[\![u_h]\!] = u_h^+ - u_h^-$

$\Rightarrow$ skewsymmetric r.h.s.



properties of the central flux

Writing

$$G_h(u_h, \eta) = (\nabla u_h, \eta)_T - (\frac{1}{2}[\![u_h]\!], \eta_n)_{\partial T}$$

we get the semi-discrete formulation

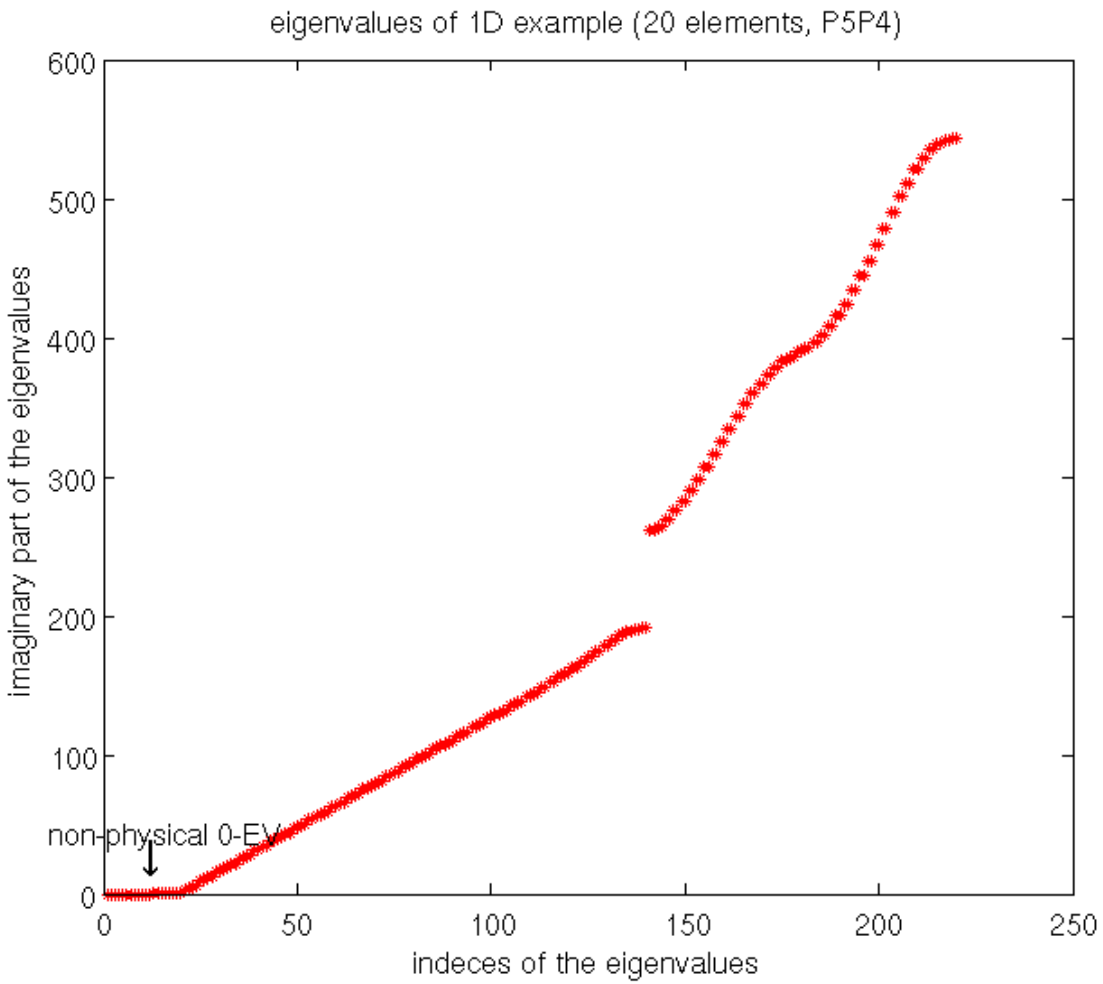
$$\frac{\partial}{\partial t} \begin{pmatrix} u_h \\ v_h \end{pmatrix} = \begin{pmatrix} & -G_h^T \\ G_h & \end{pmatrix} \begin{pmatrix} u_h \\ v_h \end{pmatrix}$$

which represents the structure of the continuous problem
($-\nabla$ is the adjoint operator to div)

$$\frac{\partial}{\partial t} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} & \text{div} \\ \nabla & \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

spatial discretizations preserving the antisymmetry of the PDE are energy-conserving

eigenvalue computation with the central flux



Nontrivial nonphysical kernel of G_h $\Rightarrow$ spurious modes.

a dissipative flux

A choice which avoids spurious modes is

$$\begin{aligned} u^* &= \{\!\{u_h\}\!\} - [\![v_h \cdot \mathbf{n}]\!] \\ v_n^* &= \{\!\{v_h\}\!\}_n - \frac{1}{2}[\![u_h]\!] \end{aligned}$$

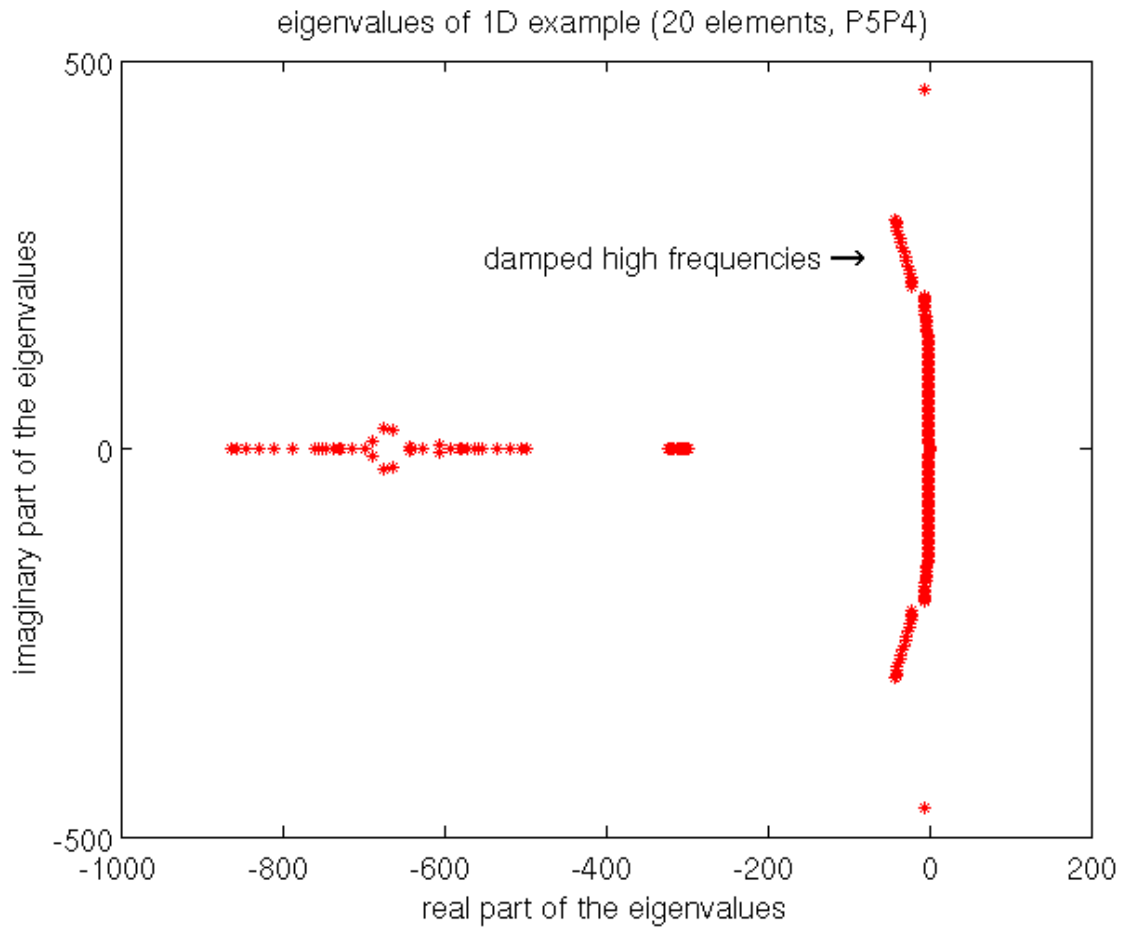
we get

$$\begin{aligned} \frac{\partial}{\partial t}(u_h, \xi)_T &= -(v_h, \nabla\xi)_T + (\{\!\{v_h\}\!\}_n - \tfrac{1}{2}[\![u_h]\!], \xi)_{\partial T} \\ \frac{\partial}{\partial t}(v_h, \eta)_T &= (\nabla u_h, \eta)_T - (\tfrac{1}{2}[\![u_h]\!] + [\![v_h \cdot \mathbf{n}]\!], \eta_n)_{\partial T} \end{aligned}$$

The difference to the central flux can be written as a sum over all facets:

$$-\sum_E \frac{1}{2}([\![u_h]\!], [\![\xi]\!])_E + ([\![v_h \cdot \mathbf{n}]\!], [\![\eta \cdot \mathbf{n}]\!])_E$$

eigenvalue computation with the dissipative flux



Nonphysical damping of high frequencies

- \+ avoids spurios modes
- – is not energy-conserving

problem in the frequency domain

The problem in frenquency domain reads

$$\begin{aligned} i\omega\hat{u} &= \mathrm{div}\hat{v} \\ i\omega\hat{v} &= \nabla\hat{u} \end{aligned}$$

and the discretized version reads (with the discrete operator G_h):

$$\begin{aligned} i\omega(\hat{u},\hat{\xi}) &= -(G_h\hat{\xi},\hat{v}) \\ i\omega(\hat{v},\hat{\eta}) &= (G_h\hat{u},\hat{\eta}) \end{aligned}$$

or

$$0 = (i\omega)^2(\hat{u},\hat{\xi}) + (G_h\hat{u}, G_h\hat{\xi}) + \sum_E \frac{\alpha}{h}([\![\hat{u}]\!],[\![\hat{\xi}]\!])_{\partial T}$$

$(G_h\hat{u}, G_h\hat{\xi}) \approx (\nabla u, \nabla\xi)$ is an symmetric, positive definite bilinearform
$\Rightarrow$ Add an Internior Penalty Term to avoid nontrivial kernel of G_h
(proposed by *Warburton/Hesthaven*)

problem in the frequency domain

The problem in frenquency domain reads

$$\begin{aligned} i\omega\hat{u} &= \operatorname{div}\hat{v} \\ i\omega\hat{v} &= \nabla\hat{u} \end{aligned}$$

and the discretized version reads (with the discrete operator G_h):

$$\begin{aligned} i\omega(\hat{u},\hat{\xi}) &= -(G_h\hat{\xi},\hat{v}) \\ i\omega(\hat{v},\hat{\eta}) &= (G_h\hat{u},\hat{\eta}) \end{aligned}$$

or

$$0 = (i\omega)^2(\hat{u},\hat{\xi}) + (G_h\hat{u}, G_h\hat{\xi}) + \sum_E \frac{\alpha}{h}([\![\hat{u}]\!],[\![\hat{\xi}]\!])_{\partial T}$$

$(G_h\hat{u}, G_h\hat{\xi}) \approx (\nabla u, \nabla\xi)$ is an symmetric, positive definite bilinearform
$\Rightarrow$ Add an Internior Penalty Term to avoid nontrivial kernel of G_h
(proposed by *Warburton/Hesthaven*)

non-dissipative stabilization approach

With $\frac{\alpha}{h}[\![\hat{u}]\!] = i\omega\hat{v}^F$ we get a new unknown living only on the facets.
We introduce this variable analogously on the time domain level

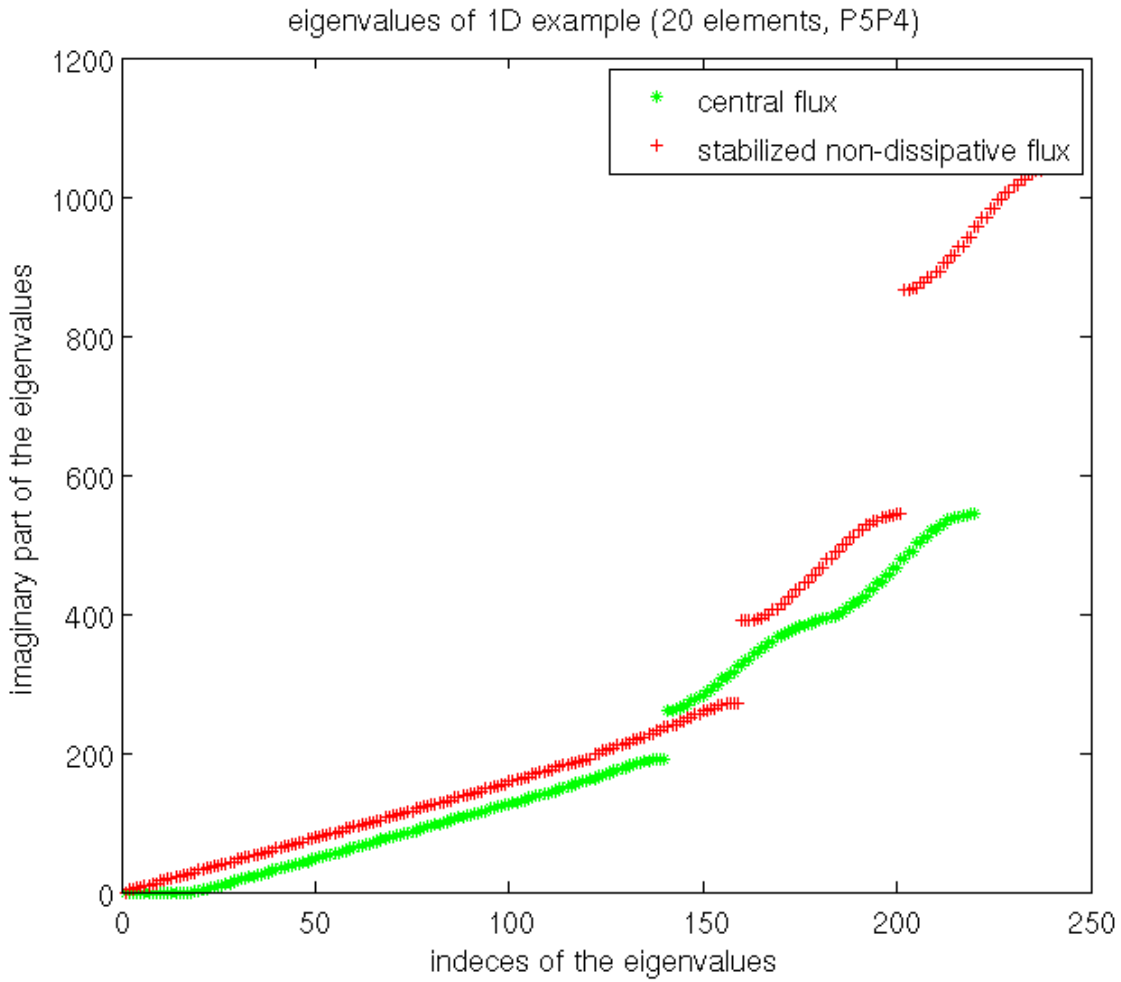
$$v_h^F \in V_h^F := \{v_h^F \in L^2(\mathcal{F}) : v|_E \in \mathcal{P}^k(E)\}$$

$$\frac{\partial}{\partial t}(v_h^F, \xi^F) = (\frac{\alpha}{h}[\![u_h]\!], \xi^F) \qquad \forall \xi^F \in V_h^F$$

and get the new non-dissipative spatial discretization
$(\sum_T \cdots , \sum_E \cdots)$

$$\begin{aligned}
\frac{\partial}{\partial t}(u_h, \xi)_T &= -(v_h, \nabla\xi)_T + (\{\!\{v_h\}\!\}_n, \xi)_{\partial T} - (v_h^F, [\![\xi]\!])_E \\
\frac{\partial}{\partial t}(v_h, \eta)_T &= (\nabla u_h, \eta)_T - (\frac{1}{2}[\![u_h]\!], \eta_n)_{\partial T} \\
\frac{h}{\alpha}\frac{\partial}{\partial t}(v_h^F, \xi^F)_E &= ([\![u_h]\!], \xi^F)_E
\end{aligned}$$

eigenvalue computation with the non-dissipative stabilized flux



+ the zero eigenvalues vanished
+ discretization is still energy-conserving

discretization of Maxwell's equation

Transfering the ideas of the wave equation to Maxwell's equation, we get ($\sum_T \cdots$, $\sum_E \cdots$)

$$\begin{aligned}
\frac{\partial}{\partial t}(\varepsilon E_h, e)_T &= (H_h, \operatorname{curl} e)_T + (\{\!\{H_h\}\!\} \times \mathbf{n}, e)_{\partial T} + (H_h^F \times \mathbf{n}, [\![e]\!])_E \\
\frac{\partial}{\partial t}(\nu H_h, h)_T &= -(\operatorname{curl} E_h, h)_T + (\frac{1}{2}[\![E_h]\!] \times \mathbf{n}, h)_{\partial T} \\
\frac{h}{\alpha}\frac{\partial}{\partial t}(H_h^F, h^F)_E &= ([\![E_h]\!] \times \mathbf{n}, h^F)_E
\end{aligned}$$

where $\alpha \sim p(p+1)$

the complete scheme

global time-stepping

For the investigations of the time-stepping scheme we consider the semi-discrete problem

$$\frac{\partial}{\partial t}\begin{pmatrix} M_\varepsilon E \\ M_\nu H \end{pmatrix} = \begin{pmatrix} -G_h^T H \\ G_h E \end{pmatrix}$$

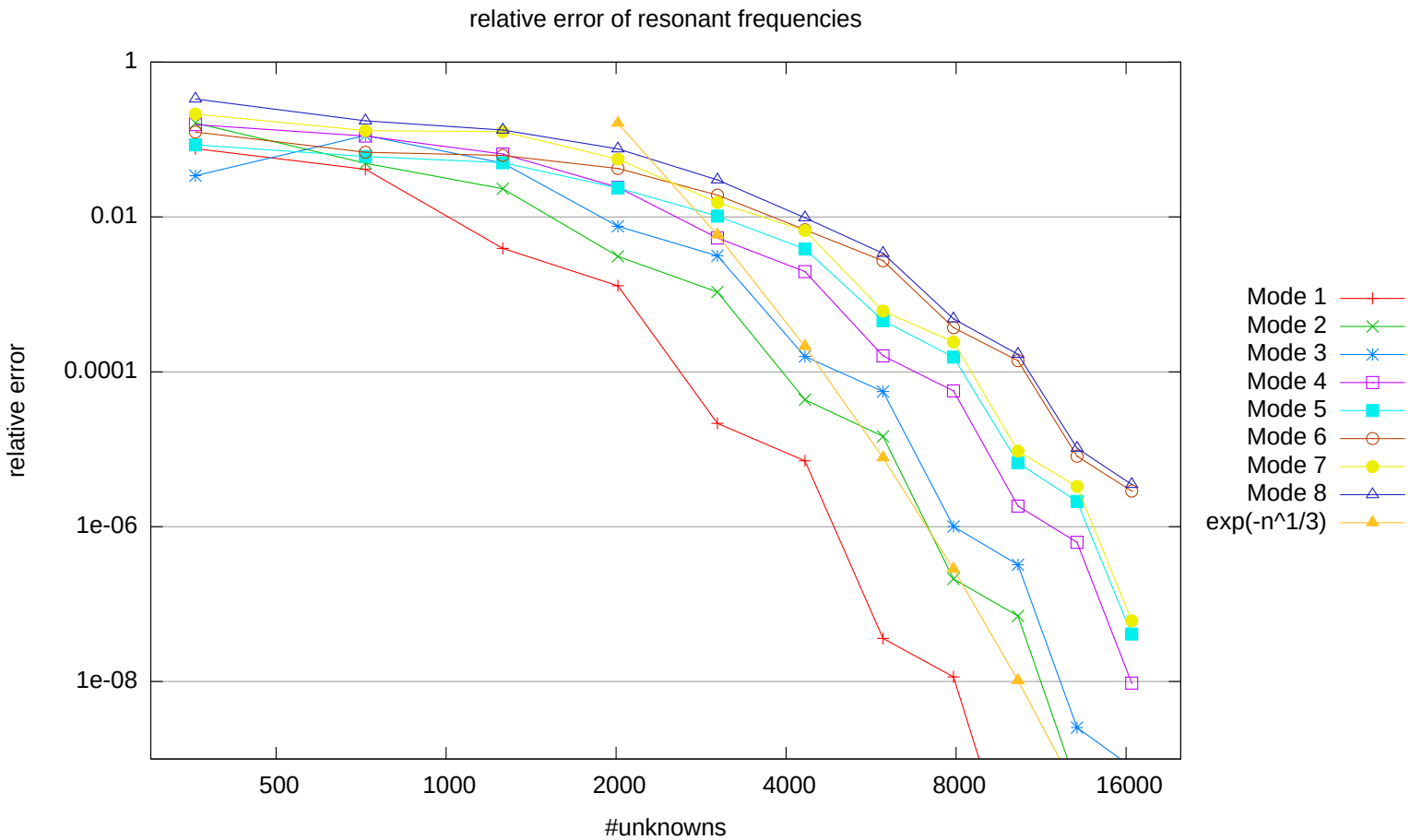
To maintain the energy-conserving property of our spatial discretization we use a time-stepping scheme that is also energy-conserving in a discrete sense. Here we use the symplectic euler / leap frog:

$$\begin{aligned} H^{n+1} &= H^n + M_\nu^{-1} G_h(E_h^n) \\ E^{n+1} &= E^n - M_\varepsilon^{-1} G_h^T(H_h^{n+1}) \end{aligned} \qquad \begin{aligned} H^{n+\frac{1}{2}} &= H^{n-\frac{1}{2}} + M_\nu^{-1} G_h(E_h^n) \\ E^{n+1} &= E^n - M_\varepsilon^{-1} G_h^T(H_h^{n+\frac{1}{2}}) \end{aligned}$$

numerical example

As an example of the opportunities of a higher order discretization, we will look at the error of the smallest resonance frequencies (eigenvalues) for a cube vacuum resonator with side length $1mm$ and PEC boundary conditions.

- 6 elements
- p-refinement



computational aspects

From point of view of implementation we have

- full polynomials of degree $k+1$ for E_h
- full polynomials of degree k and additional facet unknowns for H_h
- **block-diagonal mass matrices**

For further optimization we

- don't assemble global matrices G_h or $-G_h^T$
- but implement the operations $M_\nu^{-1} G_h(E_h^k)$ and $-M_\varepsilon^{-1} G_h^T(H_h^k)$

The complexity of elementwise operations is dominated by

- evaluating gradients
- evaluating trace terms

For both we use non-standard techniques on **tetrahedra** elements leading to fast computations.

local time-stepping

For explicit schemes the stability limit of the global time step is limited by the material parameters, the mesh size and the polynomial order.

- global time step depends (more or less) on the smallest element.

Idea of local time-stepping:

- use different step sizes for different elements

So the global time step is only limited by the size of the largest elements.

For a local time-stepping scheme the mesh elements are divided in different classes representing different time steps:

- Class 0 is the class with the largest time step

  ⋮

- Class N is the class with the smallest time step

So we need an algorithm making the different classes compatible.
We use only classes such that time steps of smaller classes are a certain multiple integer of a larger class.

recursive leap frog

One possible algorithm proposed by *Montseny et al.* (2008) is the *recursive leap-frog method* which is based on the time integration method *leap frog* or a *symplectic euler*:

1. ComputeH(0,Δt)
2. ComputeE(0,Δt)

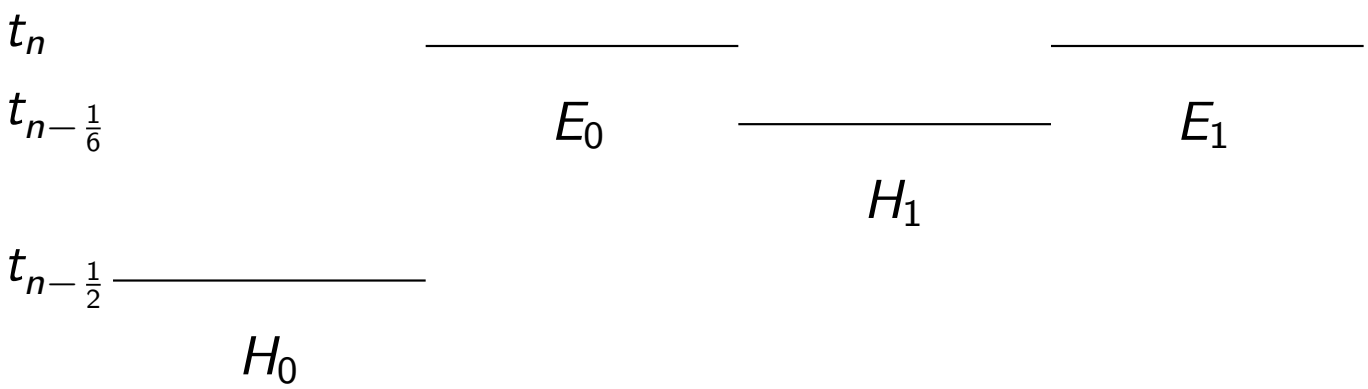
with the recursively defined functions

ComputeH(N,Δt):	ComputeE(N,Δt):
UpdateH(N,Δt)	UpdateE(N,Δt)
ComputeH(N+1,$\frac{1}{3}\Delta t$)	ComputeE(N+1,$\frac{1}{3}\Delta t$)
ComputeE(N+1,$\frac{1}{3}\Delta t$)	ComputeH(N+1,$\frac{1}{3}\Delta t$)
ComputeH(N+1,$\frac{1}{3}\Delta t$)	ComputeE(N+1,$\frac{1}{3}\Delta t$)

where UpdateH and UpdateE are the actual computation and updates.

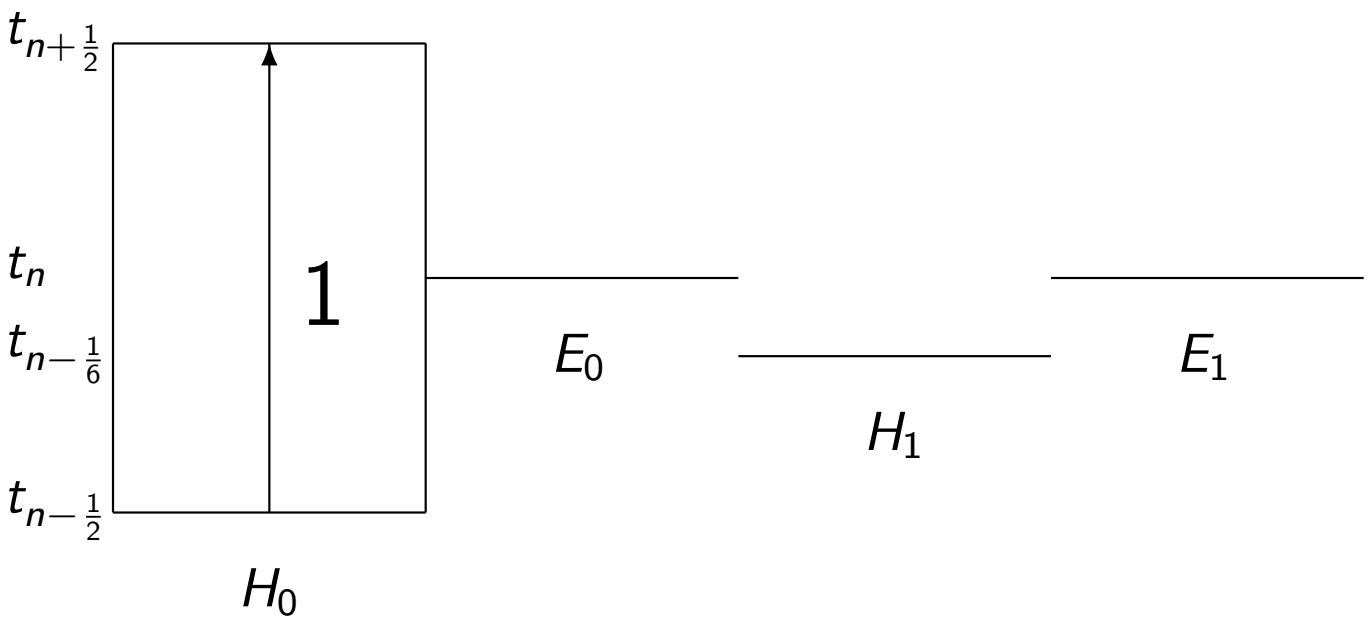
recursive leap frog example

class 0: ComputeH(0,Δt)



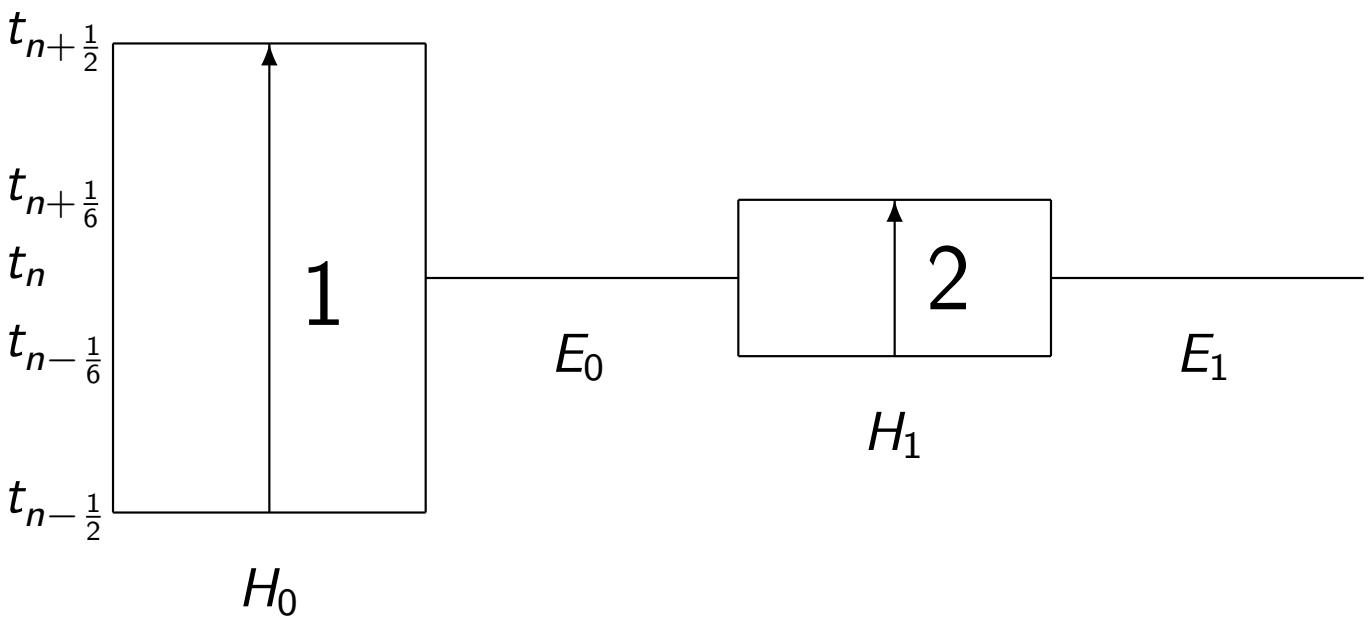
recursive leap frog example

class 0: ComputeH(0,Δt) $\rightarrow$ UpdateH(0,Δt)



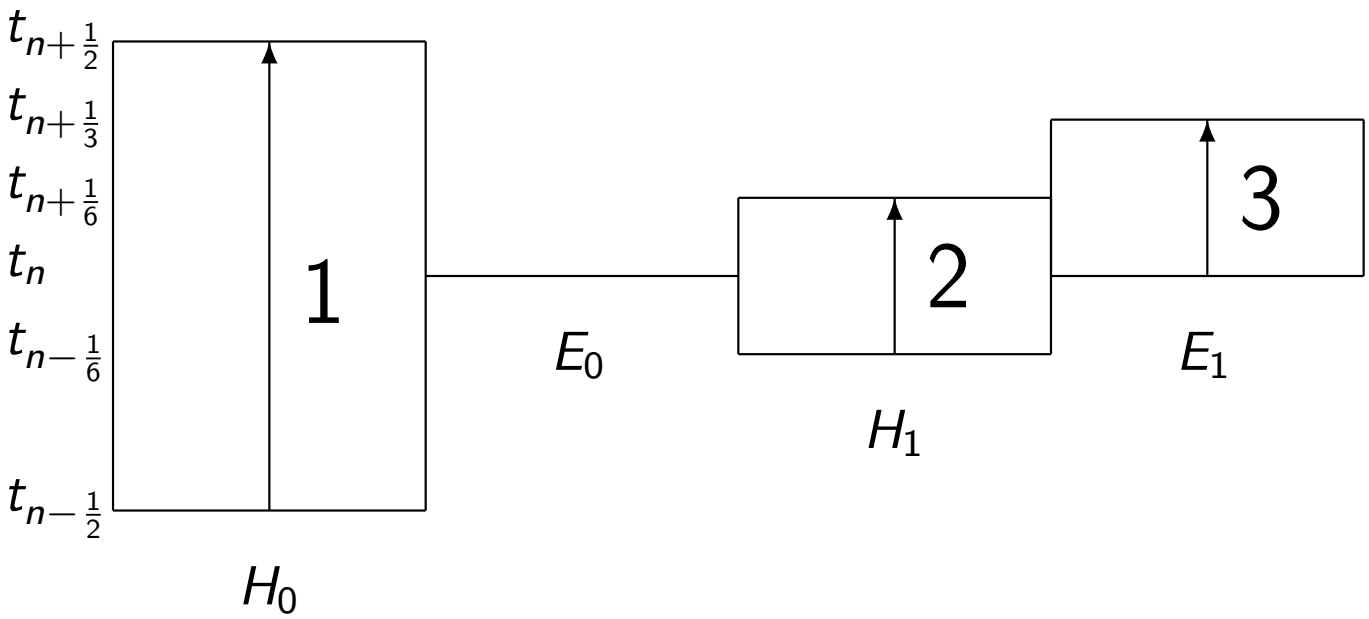
recursive leap frog example

class 0: ComputeH(0,Δt) $\rightarrow$ ComputeH(1,$\frac{1}{3}\Delta t$)
class 1: ComputeH(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateH(1,$\frac{1}{3}\Delta t$)



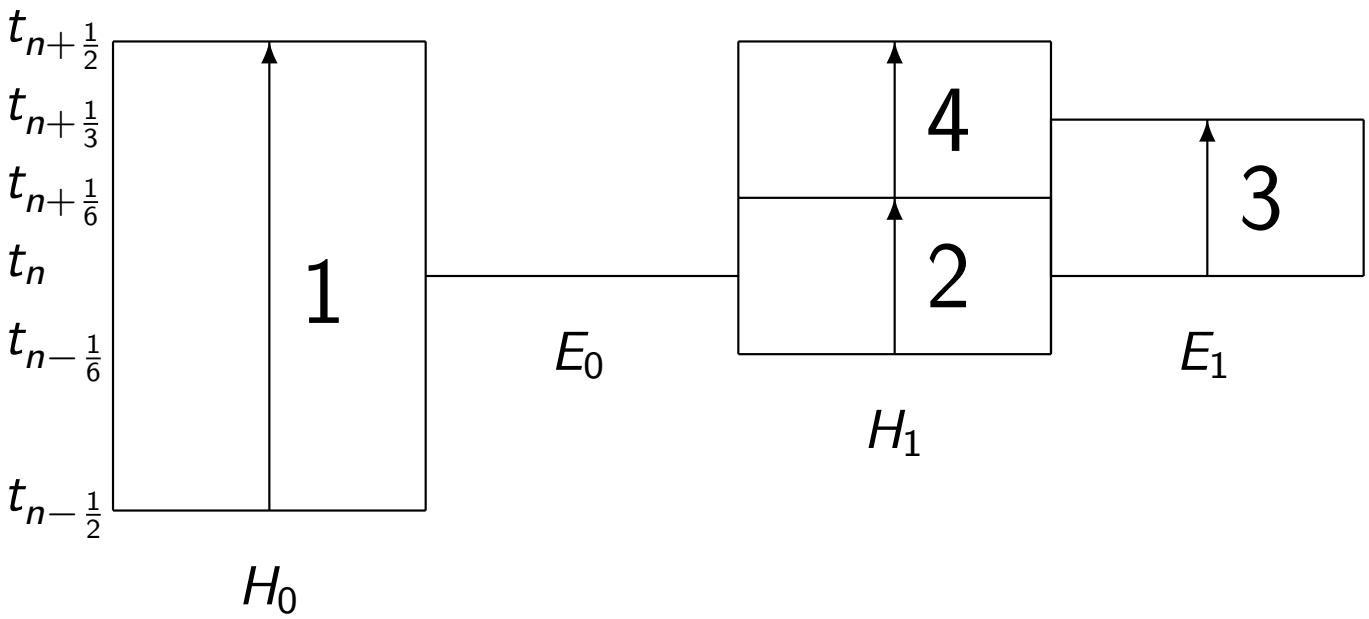
recursive leap frog example

class 0: ComputeH(0,Δt) $\rightarrow$ ComputeE(1,$\frac{1}{3}\Delta t$)
class 1: ComputeE(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateE(1,$\frac{1}{3}\Delta t$)



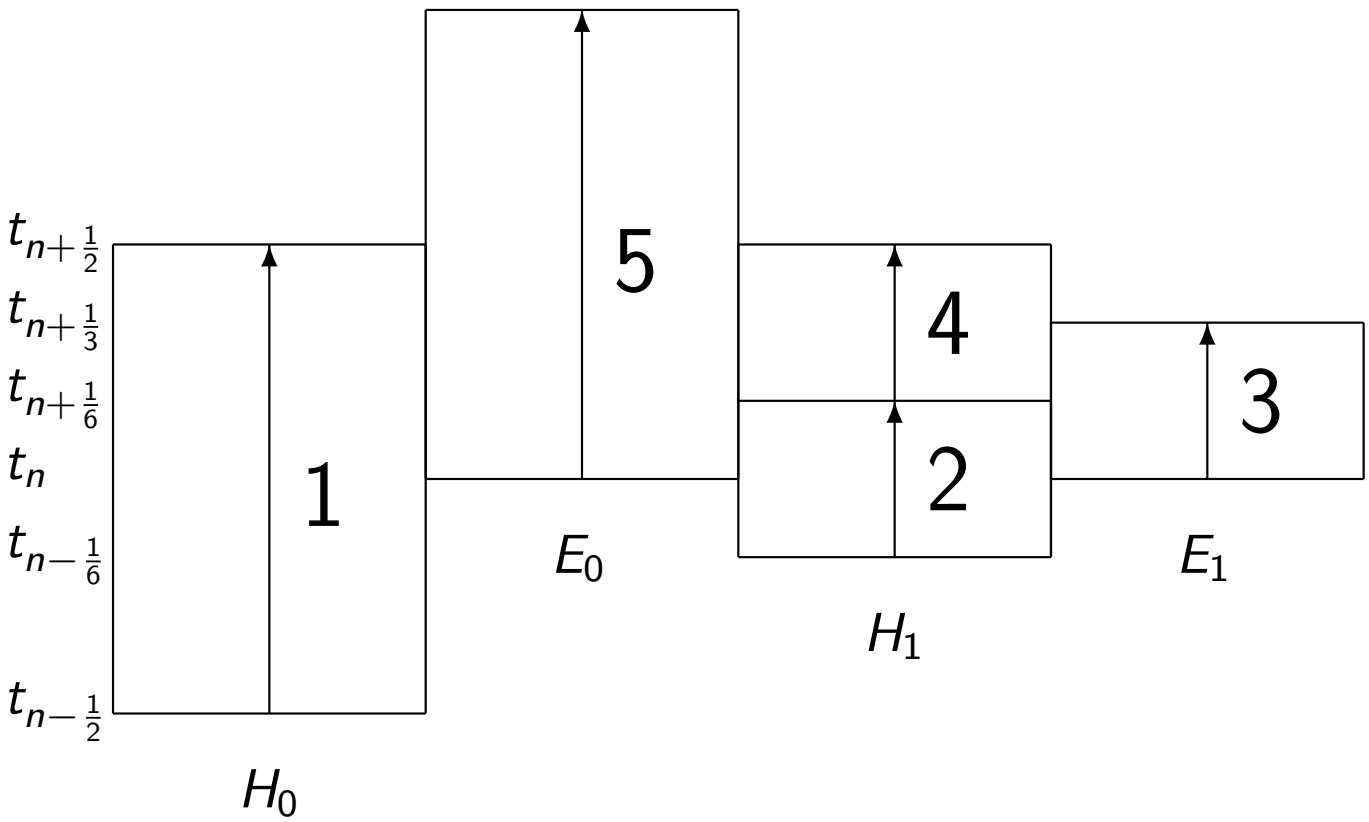
recursive leap frog example

class 0: ComputeH(0,Δt) $\rightarrow$ ComputeH(1,$\frac{1}{3}\Delta t$)
class 1: ComputeH(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateH(1,$\frac{1}{3}\Delta t$)



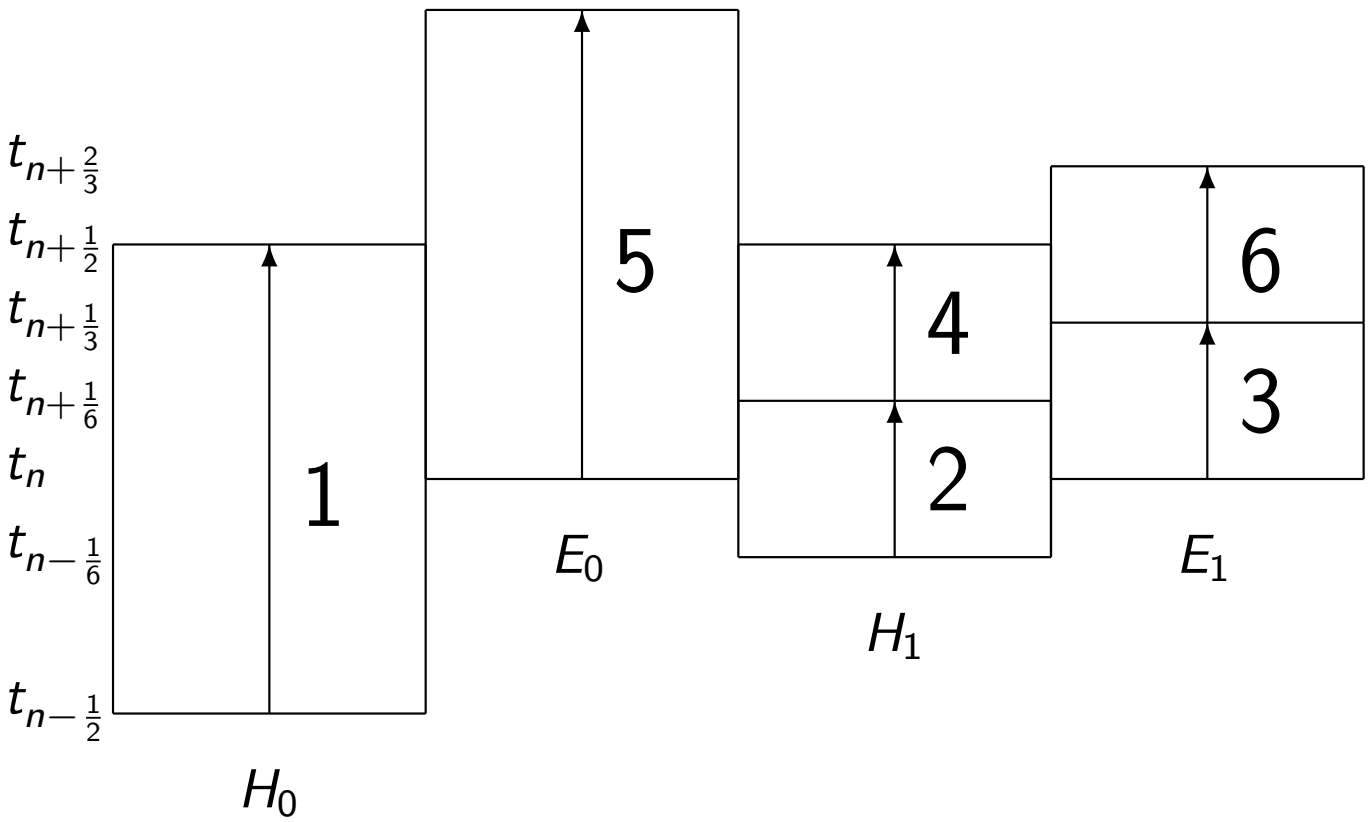
recursive leap frog example

class 0: ComputeE(0,Δt) $\rightarrow$ UpdateE(0,$\frac{1}{3}\Delta t$)



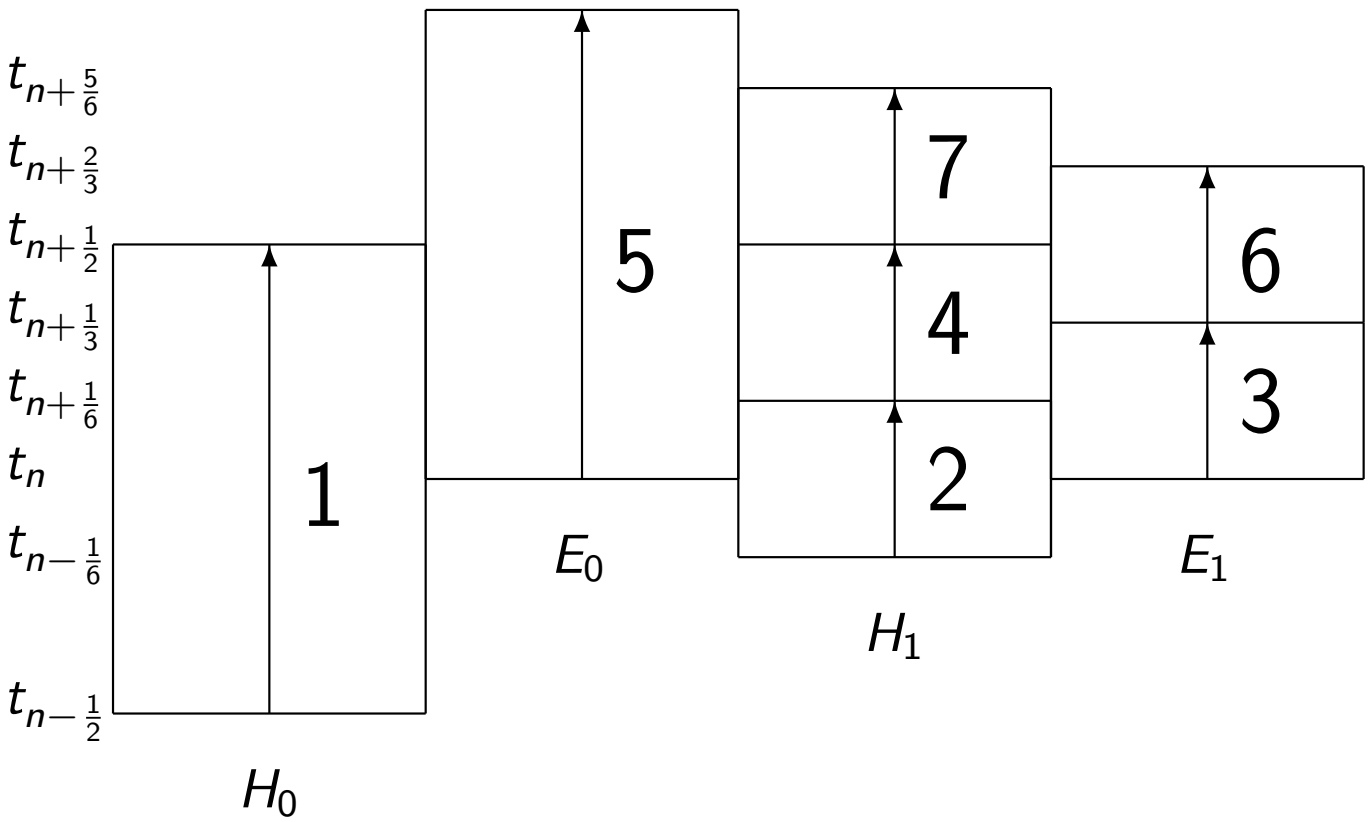
recursive leap frog example

class 0: ComputeE(0,Δt) $\rightarrow$ ComputeE(1,$\frac{1}{3}\Delta t$)
class 1: ComputeE(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateE(1,$\frac{1}{3}\Delta t$)



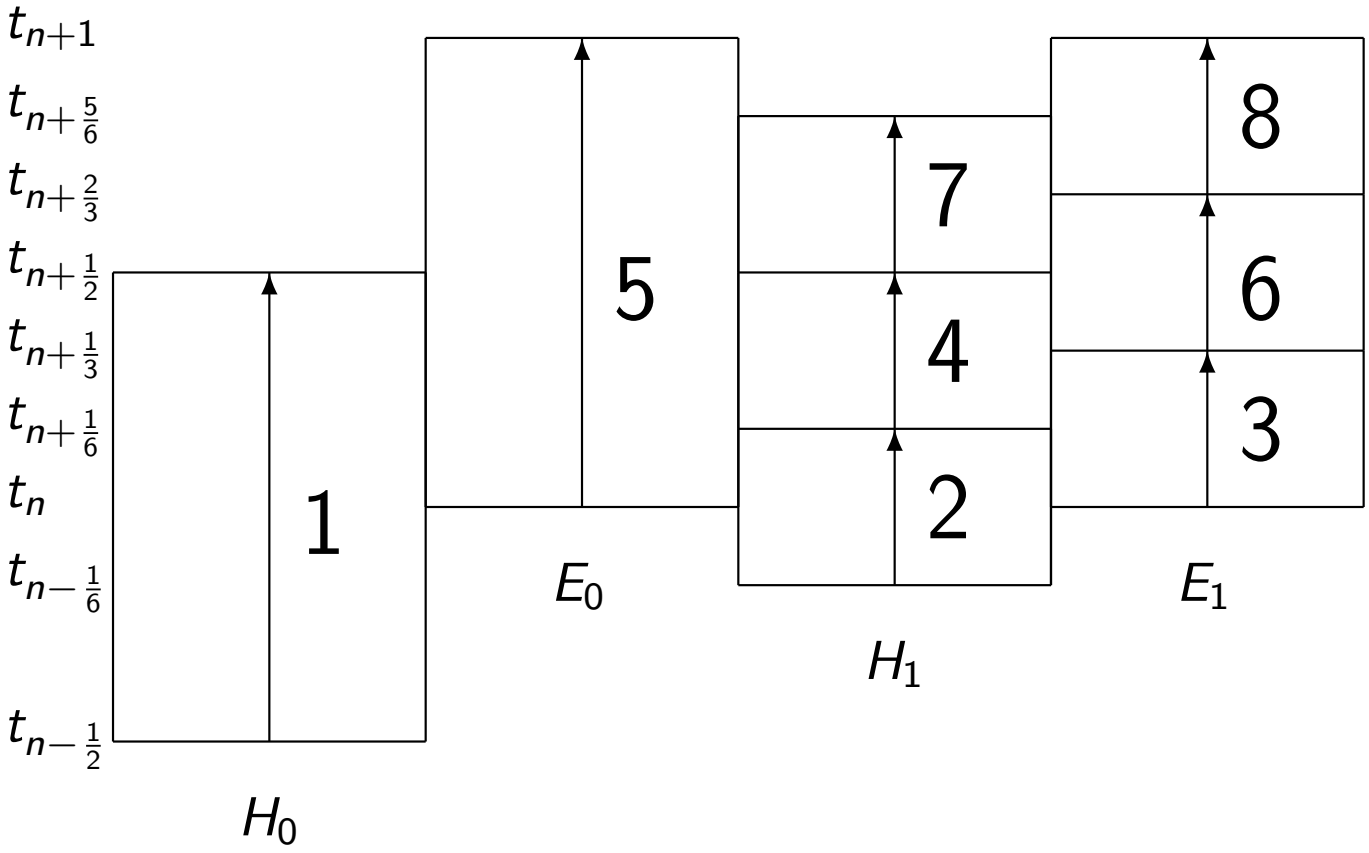
recursive leap frog example

class 0: ComputeE(0,Δt) $\rightarrow$ ComputeH(1,$\frac{1}{3}\Delta t$)
class 1: ComputeH(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateH(1,$\frac{1}{3}\Delta t$)



recursive leap frog example

class 0: ComputeE(0,Δt) $\rightarrow$ ComputeE(1,$\frac{1}{3}\Delta t$)
class 1: ComputeE(1,$\frac{1}{3}\Delta t$) $\rightarrow$ UpdateE(1,$\frac{1}{3}\Delta t$)



implementation of a local time-stepping scheme

The implementation of a local time-stepping scheme for a DG method is straight-forward:

- The loop over all elements gets replaced by a loop over all elements of the actual class.
- The loop over all facets gets replaced by a loop over all facets of the actual class. (some facets may be in two classes)

So a code which features local time-stepping needs

1. local time step estimates for every element
2. a subroutine dividing elements and facets into several classes
3. an estimate for the global time step

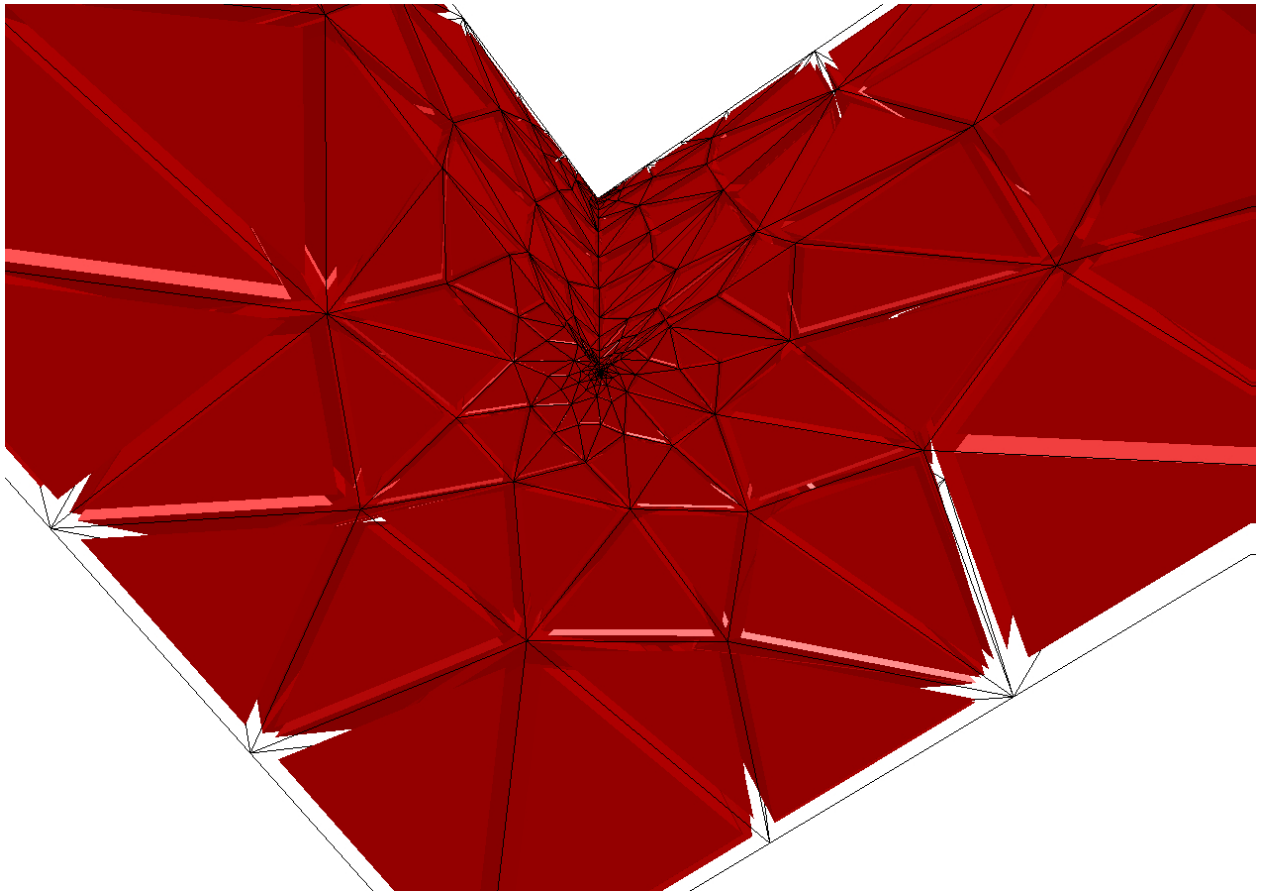
3 An estimate for the global time step can be obtained with a few steps of a power iteration for the matrix A of

$$\begin{pmatrix} E_h \\ H_h \end{pmatrix}^{k+1} = A \cdot \begin{pmatrix} E_h \\ H_h \end{pmatrix}^{k}$$

2 A subroutine dividing elements and facets into several classes is also easy to implement

1 Good estimates for the maximum local time step are not so easy to get

one example for the local time-stepping scheme



The Algorithm divides the elements and facets into 7 classes:

	total	**0**	**1**	**2**	**3**	**4**	**5**	**6**
elements	1467	131	341	305	260	227	155	48
facets	3264	351	790	745	653	553	386	128

Order 4

	max time step	elements each (large) step	facets each (large) step
w/o lts	$4.10526 \cdot 10^{-9}$	1467	3264
w lts	$2.18564 \cdot 10^{-6}$	101963	258960
factor	532.399	69.50	79.3382

$\Rightarrow$ Speedup ≈ 7
with just rough estimates for the local time step
($\Delta t_{local} \sim h +$ one smoothing step)

Thank you for your attention!