OCT'09: for a constant-coefficient transport equation discretized on a uniform grid, analysis/visualisation of numerical dispersion error inside domain, and analysis/visualisation/minimization of spurious reflection at boundaries due to mismatch in numerical phase error between scheme applied inside domain and scheme applied at boundary.
Adaptation of, but essentially the same as, similar analysis stuff in the MAPLE worksheet **testDDcollocated** from APR-AUG'09, except for $a_0 = 0$ where the adapted boundary scheme considered here turns out to be a major improvement (!).

FEB'10: we forgot to analyze the discretization of the (essential) boundary condition (oops!), i.e., it is worth trying to reduce the amplitude error at an inflow boundary by considering a discretization of the Dirichlet boundary condition that is more than just the two-point central discretization at the cell center corresponding with the boundary. **NOV'22**: this should **not** be: minimize (very small) spurious mode at inflow boundary, but must be: minimize amplitude error (per solution/Fourier mode) at inflow boundary with Dirichlet condition as a function of the location of that boundary with respect to the grid, mimicking a cut cell in 1D. NB, amplitude shift across Fourier modes can be compensated for by F-mode dependent amplitude correction of imposed Dirichlet value.
MAR'10: optimization of the discretization of the Dirichlet boundary condition now included in the construction and subsequent analysis and plotting of a numerical solution.
SEP'10: so far we analysed only 3-point boundary schemes as an alternative to the 2-point central boundary schemes that we always have been using. This gives (spurious reflection) errors at boundaries (much) smaller than the (phase) error inside the domain, except for the compact 4th-order scheme that we consider => consider 4-point boundary schemes for that case. **NOV'22**: 4th-order scheme irrelevant here, hence forget about that 4-th point.
JUL'11: revisited => some comments added and plot of phase error of scheme applied inside domain added to plots with boundary condition errors, for comparison.
JAN'19: revisited => a few corrections and some minor modifications.

JUL-OCT'22: update of the analysis/visualization of numerical phase error (and numerical amplitude error) of a constant-coefficient advection-diffusion(-reaction) coefficient discretized on a uniform grid using some central (FV) schemes. Also update/extension of analysis/optimization of discretizations at boundaries, both outflow and inflow.

NOV'23: quick extens1ion with plot of Fourier-mode analysis of a number of mass matrices of bc schemes, to verify to what extent they are blind for the Pi-mode.

We consider the constant-coefficient advection-diffusion equation $\frac{\partial c}{\partial t} + u \cdot \frac{\partial c}{\partial x} = d \cdot \frac{\partial^2 c}{\partial x^2}$ that will be discretized on a uniform grid using some central (FV) schemes

NB, because of presence of time derivative (also a term free of space derivatives) no need to consider reaction term.

..... In dimensionless form, with typical length scale $1/k$ and typical time scale $1/(k \cdot u)$: $k \cdot u \cdot \frac{\partial c}{\partial t} + (1/k) \cdot k \cdot u \cdot u \cdot \frac{1}{1/k} \cdot \frac{\partial c}{\partial x} = d \cdot \frac{1}{(1/k)^2} \cdot \frac{\partial^2 c}{\partial x^2}$

... quick note .... Pecell = u*Dx/d => d = u*Dx/Pecell

We consider **3-point central discretizations on a uniform grid**, which include FD/simple FV ($a_0 = 0$), (compatible/error consistent) piecewise linear vertex-centered FV ($a_0 = \frac{1}{8}$), (GFEM/OLOCS) piecewise parabolic vertex-centered FV ($a_0 = \frac{1}{6}$), and (box) piecewise linear cell-vertex FV ($a_0 = \frac{1}{4}$):

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\left(a_0 \cdot c_{i-1} + (1-2a_0) \cdot c_i + a_0 \cdot c_{i+1}\right) + u \cdot \frac{c_{i+1} - c_{i-1}}{2\Delta x} = d \cdot \frac{c_{i+1} - 2c_i + c_{i-1}}{\Delta x^2} .$$

We consider periodic solutions of the form $\exp\left(I\cdot\left(\omega\cdot t-k\cdot x+\phi\right)\right)$, hence we have:

$$I\cdot\omega\cdot\left(1-2\cdot a_0+2\cdot a_0\cdot\cos\left(k\cdot\Delta x\right)\right)+u\cdot\frac{I\cdot\sin\left(-k\cdot\Delta x\right)}{\Delta x}=-d\cdot\frac{\sin^2\left(-k\cdot\Delta x/2\right)}{\Delta x^2}$$ and, for $d=0$ (**zero** diffusion),

$$k=\frac{1}{\Delta x}\cdot\arctan\left(\frac{\omega\cdot\Delta x}{u}\cdot\frac{\left(1-2\cdot a_0\right)+2\cdot a_0\cdot\sqrt{1+\left(4\cdot a_0-1\right)\cdot\left(\frac{\omega\cdot\Delta x}{u}\right)^2}}{\sqrt{1+\left(4\cdot a_0-1\right)\cdot\left(\frac{\omega\cdot\Delta x}{u}\right)^2}+a_0\cdot\left(4\cdot a_0-2\right)\cdot\left(\frac{\omega\cdot\Delta x}{u}\right)^2}\right)$$ (see below for derivation).

We also briefly consider the **3-point piecewise cubic vertex-centered FV discretization on a uniform grid**, that in addition to unknown c_i per grid point also uses 2nd derivative/difference $c2_i$ per grid point. => Cubic interpolation per grid cell

$c(\xi)=(1-\xi)\cdot c_i+\xi\cdot c_{i+1}+\frac{\xi}{6}\cdot(\xi-1)\cdot(2-\xi)\cdot c2_i+\frac{\xi}{6}\cdot(\xi+1)\cdot(\xi-1)\cdot c2_{i+1}$, $x(\xi)=(1-\xi)\cdot x_i+\xi\cdot x_{i+1}$ (because of uniform grid), $i\le\xi\le i+1$, $\forall\ i$. => The error-consistent/compatible 4th-order accurate FV discretization (see also below):

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{8}\cdot\left(c_{i-1}+6\cdot c_i+c_{i+1}\right)-\frac{1}{384}\cdot\left(7\cdot c2_{i-1}+18\cdot c2_i+7\cdot c2_{i+1}\right)\right)+u\cdot\frac{c_{i+1}-c_{i-1}-\frac{1}{8}\cdot\left(c2_{i+1}-c2_{i-1}\right)}{2\Delta x}=d$$
$$\cdot\frac{c_{i+1}-2c_i+c_{i-1}-\frac{1}{24}\cdot\left(c2_{i+1}-2\,c2_i+c2_{i-1}\right)}{\Delta x^2}$$

.

with auxiliary equation:

$$a_2\cdot c_{i-1}+\left(1-2a_2\right)\cdot c_i+a_2\cdot c_{i+1}=\cdot c_{i+1}-2c_i+c_{i-1}\,.$$

NB, above expressions for piecewise cubic scheme are all from **`1Dadvdiffeq Fmodeanal_accuracyhigherorderschemes`**.

COMMENTS BELOW TO BE UPDATED!!!

FEB'10: at the left side (assume $u>0$) an **'essential' boundary condition** is required. For the optimization of that thing, consider the function fit:

$$\left(1-2\cdot a_0\right)\cdot c_i+a_0\cdot\left(c_{i-1}+c_{i+1}\right)=\int_{x_{i-0.5}}^{x_{i+0.5}}f\,\mathrm{d}x\,.$$

It seems like a good idea to design the boundary condition such that its discretization error matches as accurately as possible the error of the discretization inside the domain. Obtain this by optimizing the parameters b_1 and b_2 in:

$$\frac{c_0+c_1}{2}+b_1\cdot\frac{c_0-c_1}{2}+b_2\cdot\frac{c_0-2\cdot c_1+c_2}{2}=f_{0.5}\,.$$

SEP'10: for the 4th-order scheme ($a_0=\frac{1}{6}$), consider also: $\frac{c_0+c_1}{2}+b_2\cdot\frac{c_0-2\cdot c_1+c_2}{2}+b_3\cdot\frac{c_0-3\cdot c_1+3\cdot c_2-c_3}{2}=f_{0.5}\,.$

Would these optimized parameters be a good choice for the discretization of Dirichlet boundary conditions?

MAR'10: a number of issues here:

- Do not optimize the discretization of the Dirichlet boundary condition such that we have a minimum error at grid points or at cell centers, but such that we have minimum error integrated over the volumes (whence the optimization procedure proposed above). Especially important to obtain this when schemes with a clear discrete solution definition across the entire domain are considered, i.e., the (semi-)compatible schemes.
- For the inspection of the error that is due to the discretization of the Dirichlet boundary condition: look at the error in both grid points and cell centers, or better yet, at the error integrated over the entire volume (as we have implemented now!).
- On the other hand, given the fact that (nonlinear!) fluxes are evaluated at cell centers, it may perhaps be better to look at the effect of that error at cell centers. This would imply that $b_1=b_2=0$ is always the best parameter setting. In multi-D, however, although fluxes are evaluated at volume faces through cell centers, they are integrated across those faces. JUN'11: but then the boundary conditions are imposed at those faces and integrated across them as well, so no worries.
- For $a_0=\frac{1}{6}$ (=> piecewise parabolic fit through cell center values), the optimal discretization is obtained for $b_1=b_2\,(=b_3)=0$ => **exact** prescription of cell center value at boundary. This parameter setting is obtained from the analytic optimization that maximizes the order of accuracy of the discretization of the Dirichlet boundary condition (at least 3rd-order accurate if $b_1=0$ and $b_2=2\cdot a_0-\frac{1}{3}$, and at least 4th-order accurate if in addition $b_3=a_0-\frac{1}{6}$, see below). Is that analytic optimization perhaps (nearly) always the best choice?
- In the (re)construction of some numerical solution at the end of this worksheet, we apply the discretization of the Dirichlet boundary condition to the numerical solution **as a whole** (by computing and applying a correction factor afterwards), i.e., to the sum of the physical mode and the spurious mode. Since the spurious mode is usually close to a perfect wiggle, it is hardly felt by the discretization of the Dirichlet boundary condition **if** $b_1=b_2=0$. Hm! This means that the presence of spurious modes has a negative effect on the current optimization of the Dirichlet boundary-condition discretization (where the presence of spurious modes had to be ignored, because of the generally unknown size of these modes), although this effect is small if b_1 and b_2 are small => use **small** b_1 and b_2 !
- SEP'10: to get a clean measurement of the accuracy of the discretization of the Dirichlet boundary condition, we will compute its error by applying the condition to the physical mode **only** (the discretization has also been optimized that way, see previous item). Note, however, that in practice a Dirichlet boundary condition can only be applied to the numerical solution as a whole and hence will always be polluted by any spurious mode that may be present.

... START UPDATING THIS PART

... diffusion not yet included ...

At the right side (assume $u>0$) an **outflow boundary 'condition'** is required, i.e., a discretization of the model equation at the boundary. If the phase error of that thing is strongly different from the phase error of the scheme applied inside the domain, it may give quite some spurious reflections = generation of spurious modes,

which travel in opposite direction due to negative group speed. The remedy is: match those phase errors! (Which is equal to: minimize generation of spurious modes, e.g., by following the Vichnevetsky approach, 1987, IJNMF Vol.7, 409-452) To that end, consider the one-parameter scheme (maybe also that 2nd parameter a_1 ?):

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{c_I+c_{I+1}}{2}+a_1\cdot\frac{c_{I+1}-c_I}{2}+a_2\cdot\frac{c_{I+1}-2\cdot c_I+c_{I-1}}{2}\right)+u\cdot\frac{c_{I+1}-c_I}{\Delta x}=0\,.$$

NOTE: $a_2\neq 0$ introduces a 2nd-order dispersion effect/error and a 3rd-order dissipation error; $a_1\neq 0$ introduces a 2nd-order dispersion effect/error and a 1st-order dissipation error => $a_1\neq 0$ may not be that useful. On the other hand: if optimal value of a_2 (and a_1 ?) then minimal spurious reflections. => Minimal errors, dissipative or otherwise!
Inserting the solution mode in the boundary scheme we get, for $a_1=0$,
$I\cdot\omega\cdot\left(\frac{1+a_2}{2}+\frac{1-2\cdot a_2}{2}\exp(I\cdot k\cdot\Delta x)+\frac{a_2}{2}\exp(2\cdot I\cdot k\cdot\Delta x)\right)+\frac{u}{\Delta x}\cdot(1-\exp(I\cdot k\cdot\Delta x))=0$. Analysis of the scheme (see below) reveals that we should then take
$a_2=2\cdot a_0-\frac{1}{2}$ => negative dissipation error for $a_0\leq\frac{1}{4}$.

NB1, no need (SEP'10: *except* for the 4th-order scheme!) to add a higher-order term of the form $a_3\cdot(c_{I+1}-3\cdot c_I+3\cdot c_{I-1}-c_{I-2})$ to the numerical boundary condition (=> 4-point scheme => would be rather inconvenient), because a_1 and a_2 give already quite a lot of (± enough) room for optimization, taking into accour that spurious reflections only have to be low for values of $\frac{\omega\cdot\Delta x}{u}$ ($=k\cdot\Delta x=\frac{2\cdot\pi}{N}$, with N the number of grid cells per wave length) up to the point where the accuracy of the scheme applied inside the domain starts deteriorating, which is at the non-dim. wavenumber (phase error in pharentheses):

- $a_0=0$: $0.173=\frac{2\cdot\pi}{36.25}$ (-0.5%), $0.245=\frac{2\cdot\pi}{25.61}$ (-1%), $0.347=\frac{2\cdot\pi}{18.08}$ (-2%), $0.552=\frac{2\cdot\pi}{11.38}$ (-5%);
- $a_0=\frac{1}{8}$: $0.343=\frac{2\cdot\pi}{18.32}$ (-0.5%), $0.480=\frac{2\cdot\pi}{13.08}$ (-1%), $0.667=\frac{2\cdot\pi}{9.42}$ (-2%), $1.006=\frac{2\cdot\pi}{6.24}$ (-5%);
- $a_0=\frac{1}{6}$: $0.948=\frac{2\cdot\pi}{6.63}$ (-0.5%), $1.116=\frac{2\cdot\pi}{5.63}$ (-1%), $1.310=\frac{2\cdot\pi}{4.80}$ (-2%), $1.607=\frac{2\cdot\pi}{3.91}$ (-5%);
- $a_0=\frac{1}{4}$: $0.244=\frac{2\cdot\pi}{25.73}$ (0.5%), $0.344=\frac{2\cdot\pi}{18.25}$ (1%), $0.484=\frac{2\cdot\pi}{12.98}$ (2%), $0.752=\frac{2\cdot\pi}{8.35}$ (5%); (this one, Box scheme, is just for comparison).

These values have been obtained by using the computation of $k\cdot\Delta x$ at a certain phase error err that is specified somewhere below.

NB2, see MAPLE worksheet **testDDcollocated** for the optimization of parameters equivalent to a_1, a_2 and a_3. There we have considered the boundary scheme:

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{c_I+c_{I+1}}{2}+(\beta_0-\beta_1+\beta_2)\cdot\frac{c_{I+1}-c_I}{2}\right)+u\cdot\left((1-\beta_0-\beta_1)\cdot\frac{c_{I+1}-c_I}{\Delta x}+\beta_0\cdot\frac{3\cdot c_{I+1}-4\cdot c_I+c_{I-1}}{2\cdot\Delta x}+\beta_1\cdot\frac{3\cdot c_I-4\cdot c_{I-1}+c_{I-2}}{2\cdot\Delta x}\right)=0\,.$$

Together with the inner scheme this can be combined to (for derivation of these coefficients see that other **transpeq-analysisdiscretizationinsidedomain@boundaries**):

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{c_I+c_{I+1}}{2}+\frac{\beta_2}{1+\beta_0-3\cdot\beta_1}\cdot\frac{c_{I+1}-c_I}{2}+2\cdot\frac{\beta_1+\left(\beta_0-3\cdot\beta_1\right)\cdot a_0}{1+\beta_0-3\cdot\beta_1}\cdot\frac{c_{I+1}-2\cdot c_I+c_{I-1}}{2}-2\cdot\frac{\beta_1\cdot a_0}{1+\beta_0-3\cdot\beta_1}\cdot\frac{c_{I+1}-3\cdot c_I+3\cdot c_{I-1}-c_{I-2}}{2}\right)+u\cdot\frac{c_{I+1}-c_I}{\Delta x}=0$$

.

In other words, by taking $a_1=\frac{\beta_2}{1+\beta_0-3\cdot\beta_1}$ ($=0$ if $\beta_2=0$ => correct!), $a_2=2\cdot\frac{\beta_1+\left(\beta_0-3\cdot\beta_1\right)\cdot a_0}{1+\beta_0-3\cdot\beta_1}$ and $a_3=-2\cdot\frac{\beta_1\cdot a_0}{1+\beta_0-3\cdot\beta_1}$, the boundary scheme considered here is identical with the one considered in **testDDcollocated** .

We immediately see why in **testDDcollocated** the case $a_0=0$ was a deteriorated case. Then a_3 always zero, and only the parameter combinations $\frac{\beta_2}{1+\beta_0-3\cdot\beta_1}$ and $\frac{\beta_1}{1+\beta_0-3\cdot\beta_1}$ are relevant. Furthermore, we have a singularity at $\beta_0-3\cdot\beta_1=-1$. => Current boundary scheme is better, also because it involves the extension of the mass matrix at the boundary. An extension of the space discretization of the time derivative at boundaries, always involving only (simple combinations of) the primary variables (what about source terms?), is generally much easier to realize/implement than an extension of the space discretization of the space derivative(s) (there is often more than one) at boundaries, involving combinations of primary variables and other variables that may be quite complex.

OCT'22: **what about source terms?** Now mentioned separately, because of its possible importance. On the other hand, source terms usually very small locally, like at boundaries.

OCT'22: analysis below shows that best discretization of model equation at outflow boundary is the one whose accuracy mimics as closely as possible (within practical limits of course :-) the accuracy of the discretization used inside the domain. This would suggest to discretize source terms a la the time derivative using a similar mass matrix.

```
> restart:
  with(plots):
  # assume( u > 0, d > 0, a0 > 0, 1+beta0-3*beta1 > 0 );   # aux represents ω·Δx/u
> # dummy statement to allow skipping next one
```

SEP'22: construction of the error-consistent/compatible **4th-order accurate** vertex-centered FV discretization of the advection-diffusion equation, taken from **1DadvdiffeqFmodeanal_accuracyhigherorderschemes** .

1. Define cubic polynomials in left and right grid cell using variable and its 2nd derivative at grid points => continuous function and 2nd

derivative => in general no continuous 1st derivative.

2. Use cubic polynomials to get space discretization of advection-diffusion equation by means of integration over finite volume.

3. Auxiliary equation to obtain 2nd derivative at grid points from variable at grid points.

Concerning parameter a_2 in the piecewise cubic scheme that determines how the second derivative at the grid points depends on the grid-point values (text taken from **Fmodeanalysis1Dperturbation_update** which text was taken from **1DadvdiffeqFmodeanal_accuracyhigherorderschemes**):

- $a_2 = 9/40 = 108/480$ gives equal 4th-order accuracy in advection and diffusion;
- $a_2 = 47/240 = 94/480$ gives 6th-order accuracy in (linear) advection; from **construction_schemes_compact_form**: $\beta > 47/240$ ($\beta < 47/240$) => leading (lagging) phase error;
- $a_2 = 1/6 = 80/480$ gives cubic spline fit;
- $a_2 = 0$ gives simplest 2nd-difference approximation;
- $a_2 = 7/32 = 105/480$ gives simplest mass matrix (and about the smallest wiggles at both sides of a steep gradient, which is obtained for $\beta = 6.9/32 = 103.5/480$).

```
> cbarL := (1+xi)*ccc - xi*cm1 + xi/6*(xi^2+3*xi+2)*c2cc + xi/6*(1-xi^2)*c2m1;
  cbarR := (1-xi)*ccc + xi*cp1 - xi/6*(xi^2-3*xi+2)*c2cc + xi/6*(xi^2-1)*c2p1;
  advdiffHIGH := I*omega * kDx/k * ( int(cbarL, xi=-1/2..0) + int(cbarR, xi=0..1/2) )
                 + u * ( eval(cbarR,xi=1/2) - eval(cbarL,xi=-1/2) )
                 = d * ( eval(diff(cbarR,xi),xi=1/2)*k/kDx - eval(diff(cbarL,xi),xi=-1/2)*k/kDx ):
  advdiffHIGH := collect( advdiffHIGH, [omega,u,d,kDx,k] );
  auxHIGH := a2*c2m1 + (1-2*a2)*c2cc + a2*c2p1 = cm1 - 2*ccc + cp1;
```

$$cbarL := (1+\xi)\,ccc - \xi\,cm1 + \frac{1}{6}\,\xi\,(\xi^2+3\,\xi+2)\,c2cc + \frac{1}{6}\,\xi\,(-\xi^2+1)\,c2m1$$

$$cbarR := (1-\xi)\,ccc + \xi\,cp1 - \frac{1}{6}\,\xi\,(\xi^2-3\,\xi+2)\,c2cc + \frac{1}{6}\,\xi\,(\xi^2-1)\,c2p1$$

$$advdiffHIGH := \frac{I\,\omega\,kDx\left(-\frac{3}{64}\,c2cc - \frac{7}{384}\,c2m1 + \frac{3}{4}\,ccc + \frac{1}{8}\,cm1 - \frac{7}{384}\,c2p1 + \frac{1}{8}\,cp1\right)}{k} + u\left(\frac{1}{2}\,cp1 - \frac{1}{16}\,c2p1 - \frac{1}{2}\,cm1 + \frac{1}{16}\,c2m1\right)$$

$$= \frac{\left(-2\,ccc + cp1 + \frac{1}{12}\,c2cc - \frac{1}{24}\,c2p1 + cm1 - \frac{1}{24}\,c2m1\right)k\,d}{kDx}$$

$$auxHIGH := a2\,c2m1 + (1-2\,a2)\,c2cc + a2\,c2p1 = cm1 - 2\,ccc + cp1$$

```
> # continue
```

The 'standard' 3-point central discretizations of the advection-diffusion equation in FV form.

```
> ### d := u*kDx/k*Pecellinv;
  advdiffLOW := I*omega * kDx/k * ( a0*(cm1 + cp1) + (1-2*a0) * ccc )
```

```
                + u * ( (cp1 + ccc)/2 - (ccc + cm1)/2 )
                = d * ( (cp1 - ccc)*k/kDx - (ccc - cm1)*k/kDx ):
```

Fourier-mode transform of discretizations of advection-diffusion equation in FV form (+- taken from **1DadvdiffeqFmodeanal_accuracyhigherorderschemes**)

```
> # ccc := 1:  cm1 := ccc * exp(I*kDx):  cp1 := ccc * exp(-I*kDx):
  #          c2m1 := c2cc * exp(I*kDx):  c2p1 := c2cc * exp(-I*kDx):
  ccc := 1:  cm1 := ccc * ( cos(kDx) + I*sin(kDx) ):  cp1 := ccc * ( cos(kDx) - I*sin(kDx) ):
           c2m1 := c2cc * ( cos(kDx) + I*sin(kDx) ):  c2p1 := c2cc * ( cos(kDx) - I*sin(kDx) ):
  c2cc := solve( auxHIGH, c2cc );
  omegaLOW  := unapply( collect(simplify(solve(advdiffLOW,  omega)),[u,d]), [u,d,kDx,a0] );
  omegaHIGH := unapply( collect(simplify(solve(advdiffHIGH, omega)),[u,d]), [u,d,kDx,a2] );
  seriesLOW  := simplify( series(omegaLOW(u,d,kDx,a0),  kDx, 7) );  # 2nd-order accurate advection and diffusion for any value of a0
  seriesHIGH := simplify( series(omegaHIGH(u,d,kDx,a2), kDx, 7) );  # 4th-order accurate advection and diffusion for any value of a2
  leadLOWu  := eval( op(3, convert(seriesLOW,polynom)),  [k=1,u=1,d=0] );
  leadLOWpu := eval( op(4, convert(seriesLOW,polynom)),  [k=1,u=1,d=0] );
  leadHIGHu := eval( op(3, convert(seriesHIGH,polynom)), [k=1,u=1,d=0] );
  err := 0.01;
  Nfd   := evalf( 2*Pi / op(1,[solve(abs(eval(leadLOWu,a0=0))=err,kDx)]) );      # number of points per wavelength for err advection error when a0 = 0  (=> leading error -kDx^2/6 )
  Nlin  := evalf( 2*Pi / op(1,[solve(abs(eval(leadLOWu,a0=1/8))=err,kDx)]) );    # number of points per wavelength for err advection error when a0 = 1/8  (=> leading error -kDx^2/24 )
  Npar  := evalf( 2*Pi / op(1,[solve(abs(eval(leadLOWpu,a0=1/6))=err,kDx)]) );  # number of points per wavelength for err advection error when a0 = 1/6  (=> leading error -kDx^4/180 )
  Ncubs := evalf( 2*Pi / op(1,[solve(abs(eval(leadHIGHu,a2= 80/480))=err,kDx)]) );  # number of points required per wavelength for err advection error and a2 = 1/6  (cubic spline)
  Ncub  := evalf( 2*Pi / op(1,[solve(abs(eval(leadHIGHu,a2=108/480))=err,kDx)]) );  # number of points required per wavelength for err advection error and a2 = ··· (may be rough approximation!!!)
  # kDxLOW := unapply( collect(simplify(solve(advdiffLOW, kDx)),[u,d]), [u,d,omega,a0] );
  # kDxHIGH := unapply( collect(simplify(solve(advdiffHIGH, kDx)),[u,d]), [u,d,omega,a2] );
```

$$c2cc := \frac{2\,(\cos(kDx) - 1)}{2\cos(kDx)\,a2 - 2\,a2 + 1}$$

$$omegaLOW := (u, d, kDx, a0) \to \frac{k \sin(kDx)\, u}{kDx\,(2\,a0\cos(kDx) + 1 - 2\,a0)} - \frac{I\,k\,(2\cos(kDx)\,k - 2\,k)\,d}{kDx^2\,(2\,a0\cos(kDx) + 1 - 2\,a0)}$$

$$omegaHIGH := (u, d, kDx, a2) \to -\frac{8\,I\,k\,(24\,I\cos(kDx)\sin(kDx)\,a2\,kDx - 3\,I\cos(kDx)\sin(kDx)\,kDx - 24\,I\sin(kDx)\,a2\,kDx + 15\,I\sin(kDx)\,kDx)\,u}{kDx^2\,(48\,a2\cos(kDx)^2 + 96\,a2\cos(kDx) - 7\cos(kDx)^2 - 144\,a2 + 22\cos(kDx) + 81)}$$

$$- \frac{8\,I\,k\,(48\cos(kDx)^2\,a2\,k - 2\cos(kDx)^2\,k - 96\cos(kDx)\,a2\,k + 28\cos(kDx)\,k + 48\,a2\,k - 26\,k)\,d}{kDx^2\,(48\,a2\cos(kDx)^2 + 96\,a2\cos(kDx) - 7\cos(kDx)^2 - 144\,a2 + 22\cos(kDx) + 81)}$$

$$seriesLOW := k\,u + I\,k^2\,d + \left(-\frac{1}{6}\,k\,u + u\,k\,a0 - \frac{1}{12}\,I\,k^2\,d + I\,d\,k^2\,a0\right) kDx^2 + \left(\frac{1}{120}\,k\,u - \frac{1}{4}\,u\,k\,a0 + u\,k\,a0^2 + \frac{1}{360}\,I\,k^2\,d - \frac{1}{6}\,I\,d\,k^2\,a0 + I\,d\,k^2\,a0^2\right) kDx^4 + O(kDx^6)$$

$$seriesHIGH := k\,u + I\,k^2\,d + \left(\frac{1}{24}\,u\,a2\,k - \frac{47}{5760}\,k\,u - \frac{1}{24}\,I\,d\,k^2\,a2 + \frac{61}{5760}\,I\,k^2\,d\right) kDx^4 + O(kDx^6)$$

$$leadLOWu := \left(-\frac{1}{6} + a0\right) kDx^2$$

$$leadLOWpu := \left(\frac{1}{120} - \frac{1}{4}\,a0 + a0^2\right) kDx^4$$

$$leadHIGHu := \left(\frac{1}{24}\,a2 - \frac{47}{5760}\right) kDx^4$$

$$err := 0.01$$

$$Nfd := 25.65099660$$

$$Nlin := 12.82549830$$

$$Npar := 5.424525339$$

$$Ncubs := 3.709785141$$

$$Ncub := 3.709785141$$

```
> # dummy statement to allow skipping next one
```

SEP'22: plot of discretization behavior, i.e., dimensionless $\omega \cdot \Delta x / u$ (advection) and $\omega \cdot \Delta x^2 / \left(I \cdot k^2 \cdot d\right)$ (diffusion) as a function of dimensionless wave number $k\Delta x$.

```
> plotphase := plot(
    [ kDx, simplify( omegaLOW(u,0,kDx,0)*kDx/(k*u) ),        # 3-point FD, 2nd-order
      simplify( omegaLOW(u,0,kDx,1/12)*kDx/(k*u) ),          # α = 1 / 12, 2nd-order
      simplify( omegaLOW(u,0,kDx,1/8)*kDx/(k*u) ),           # piecewise linear vertex-centered FV, 2nd-order
      simplify( omegaLOW(u,0,kDx,1/6)*kDx/(k*u) ),           # piecewise parabolic vertex-centered FV (GFEM), 4th-order
      simplify( omegaLOW(u,0,kDx,1/4)*kDx/(k*u) ),           # piecewise linear cell-vertex FV (box), 2nd-order
```

```
    simplify( omegaHIGH(u,0,kDx, 80/480)*kDx/(k*u) ),     # piecewise cubic spline vertex-centered FV, 4th-order
    simplify( omegaHIGH(u,0,kDx,108/480)*kDx/(k*u) ) ],   # another piecewise cubic vertex-centered FV, 4th-order
  kDx=0..Pi, 0..Pi, numpoints=100, thickness=2, size=[458,400], # size=[700,400],
  color=["Black", "Red", "Coral", "ForestGreen", "MediumBlue", "DarkOrange", "LimeGreen",
"CornflowerBlue"],
  linestyle=[2,1,1,1,1,1,1,1], thickness=2, legendstyle=[font=[Times,11],location=right],
#   legend=["", "3-point FD", typeset(a[0],"=1/12"), "vertex-cent pwise lin", "vertex-cent pwise parab",
"cell-vertex pwise lin",
#          "vertex-cent cubic spline", "vertex-cent pwise cubic" ],
  labels=[typeset("\t       ",k*Dx),"phase\n\n\n\n\n\n\n\n\n"], labelfont=[Times,11],
  axis=[thickness=2], axesfont=[Times,11],
  tickmarks=[ [0=0, Pi/10="", Pi/5=typeset(Pi,"/5"), 3*Pi/10="", 2*Pi/5=typeset("2",Pi,"/5"), 5*Pi/10="",
3*Pi/5=typeset("3",Pi,"/5"),
                7*Pi/10="", 4*Pi/5=typeset("4",Pi,"/5"), 9*Pi/10="", Pi=typeset(Pi) ],
             [0="0.0", 0.25="", 0.5="0.5", 0.75="", 1.0="1.0", 1.25="", 1.5="1.5", 1.75="", 2="2.0",
2.25="", 2.5="2.5", 2.75="", 3="3.0"]]
         );
```

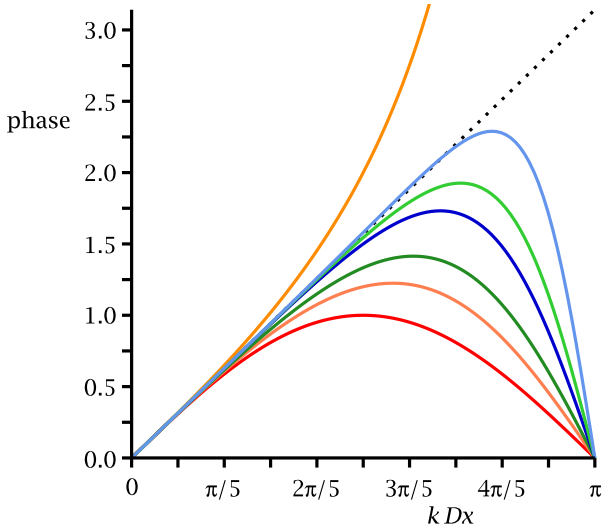
$plotphase := PLOT(\ldots)$

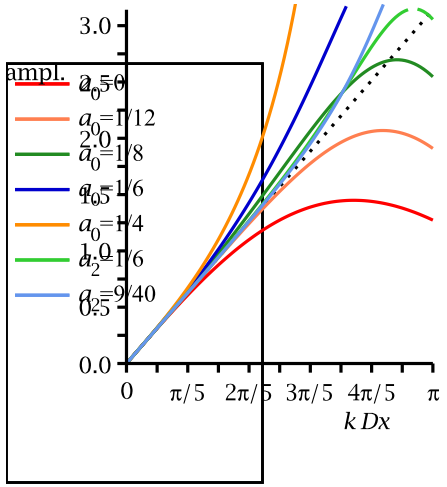
```
> plotampl := plot(
  [ kDx, simplify( omegaLOW(0,d,kDx,0)*kDx/(I*k^2*d) ),       # 3-point FD, 2nd-order
    simplify( omegaLOW(0,d,kDx,1/12)*kDx/(I*k^2*d) ),        # α = 1 / 12, 4th-order
    simplify( omegaLOW(0,d,kDx,1/8)*kDx/(I*k^2*d) ),         # piecewise linear vertex-centered FV, 2nd-order
    simplify( omegaLOW(0,d,kDx,1/6)*kDx/(I*k^2*d) ),         # piecewise parabolic vertex-centered FV (GFEM), 2nd-order
    simplify( omegaLOW(0,d,kDx,1/4)*kDx/(I*k^2*d) ),         # piecewise linear cell-vertex FV (box), 2nd-order
    simplify( omegaHIGH(0,d,kDx, 80/480)*kDx/(I*k^2*d) ),    # piecewise cubic spline vertex-centered FV, 4th-order
    simplify( omegaHIGH(0,d,kDx,108/480)*kDx/(I*k^2*d) ) ],  # another piecewise cubic vertex-centered FV, 4th-order
  kDx=0..Pi, 0..Pi, numpoints=100, thickness=2, size=[555,400], # size=[650,400],
  color=["Black", "Red", "Coral", "ForestGreen", "MediumBlue", "DarkOrange", "LimeGreen",
"CornflowerBlue"],
  linestyle=[2,1,1,1,1,1,1,1], thickness=2, legendstyle=[font=[Times,11],location=right],
#   legend=[ "", "3-point FD", typeset(a[0],"=1/12"), "vertex-cent pwise lin", "vertex-cent pwise parab",
"cell-vertex pwise lin",
#          "vertex-cent cubic spline", "vertex-cent pwise cubic" ],
  legend=[ "", typeset(a[0],"=0"), typeset(a[0],"=1/12"), typeset(a[0],"=1/8"), typeset(a[0],"=1/6"),
typeset(a[0],"=1/4"),
```

```
          typeset(a[2],"=1/6"), typeset(a[2],"=9/40") ],
  labels=[typeset("\t       ",k*Dx),"ampl.\n\n\n\n\n\n\n\n\n"], labelfont=[Times,11],
  axis=[thickness=2], axesfont=[Times,11],
  tickmarks=[ [0=0, Pi/10="", Pi/5=typeset(Pi,"/5"), 3*Pi/10="", 2*Pi/5=typeset("2",Pi,"/5"), 5*Pi/10="",
3*Pi/5=typeset("3",Pi,"/5"),
                7*Pi/10="", 4*Pi/5=typeset("4",Pi,"/5"), 9*Pi/10="", Pi=typeset(Pi) ],
              [0="0.0", 0.25="", 0.5="0.5", 0.75="", 1.0="1.0", 1.25="", 1.5="1.5", 1.75="", 2="2.0",
2.25="", 2.5="2.5", 2.75="", 3="3.0"]]
          );
```

$plotampl := PLOT(\ldots)$

```
> display( convert( [plotphase,plotampl], Array )  );
```






```
> # continue/dummy statement to allow skipping next one
```

SEP'22: plot of discretization error in advection and in diffusion.

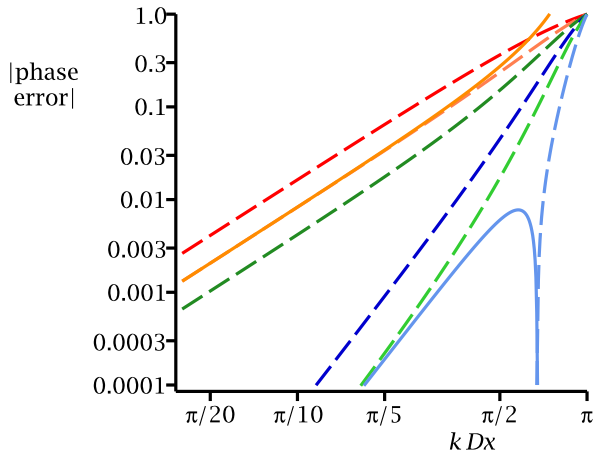
```
> plotphaseerr := loglogplot(
    [   simplify( omegaLOW(u,0,kDx,0)/(k*u) ) - 1,            # 3-point FD, 2nd-order
      - simplify( omegaLOW(u,0,kDx,0)/(k*u) ) + 1,
        simplify( omegaLOW(u,0,kDx,1/12)/(k*u) ) - 1,         # α = 1 / 12, 2nd-order
      - simplify( omegaLOW(u,0,kDx,1/12)/(k*u) ) + 1,
        simplify( omegaLOW(u,0,kDx,1/8)/(k*u) ) - 1,          # piecewise linear vertex-centered FV, 2nd-order
      - simplify( omegaLOW(u,0,kDx,1/8)/(k*u) ) + 1,
        simplify( omegaLOW(u,0,kDx,1/6)/(k*u) ) - 1,          # piecewise parabolic vertex-centered FV (GFEM), 4th-order
      - simplify( omegaLOW(u,0,kDx,1/6)/(k*u) ) + 1,
        simplify( omegaLOW(u,0,kDx,1/4)/(k*u) ) - 1,          # piecewise linear cell-vertex FV (box), 2nd-order
      - simplify( omegaLOW(u,0,kDx,1/4)/(k*u) ) + 1,
        simplify( omegaHIGH(u,0,kDx, 80/480)/(k*u) ) - 1,     # piecewise cubic spline vertex-centered FV, 4th-order
      - simplify( omegaHIGH(u,0,kDx, 80/480)/(k*u) ) + 1,
        simplify( omegaHIGH(u,0,kDx,108/480)/(k*u) ) - 1,     # another piecewise cubic vertex-centered FV, 4th-order
      - simplify( omegaHIGH(u,0,kDx,108/480)/(k*u) ) + 1 ],
    kDx=0.04*Pi..Pi, 0.0001..1, numpoints=500, thickness=2, size=[458,400], # size=[650,400],
    color=["Red", "Red", "Coral", "Coral", "ForestGreen", "ForestGreen", "MediumBlue", "MediumBlue",
  "DarkOrange", "DarkOrange",
           "LimeGreen", "LimeGreen", "CornflowerBlue", "CornflowerBlue"],
    linestyle=[1,3,1,3,1,3,1,3,1,3,1,3,1,3], thickness=2, legendstyle=[font=[Times,11],location=right],
  #   legend=[ "3-point FD", "", typeset(a[0],"=1/12"), "", "vertex-cent pwise lin", "", "vertex-cent pwise
  parab", "", "cell-vertex pwise lin", "",
  #           "vertex-cent cubic spline", "", "vertex-cent pwise cubic", "" ],
    labels=[typeset("\t       ",k*Dx)," |phase\n  error|\n\n\n\n\n\n\n\n\n\n"], labelfont=[Times,11],
    axis=[thickness=2], axesfont=[Times,11],
    tickmarks=[[Pi/100=typeset(Pi,"/100"),Pi/50=typeset(Pi,"/50"),Pi/20=typeset(Pi,"/20"),
                Pi/10=typeset(Pi,"/10"),Pi/5=typeset(Pi,"/5"),Pi/2=typeset(Pi,"/2"),Pi=typeset(Pi)],
  #              [.0001="0.0001",.0002="0.0002",.0005="0.0005",.001="0.001",.002="0.002",.005="0.005",
  #                .01="0.01",.02="0.02",.05="0.05",.1="0.1",.2="0.2",.5="0.5",1="1.0"]],
              [.0001="0.0001",.0003="0.0003",.001="0.001",.003="0.003",.01="0.01",.03="0.03",.1="0.1",.3=
  "0.3",1="1.0"]]
            );
```

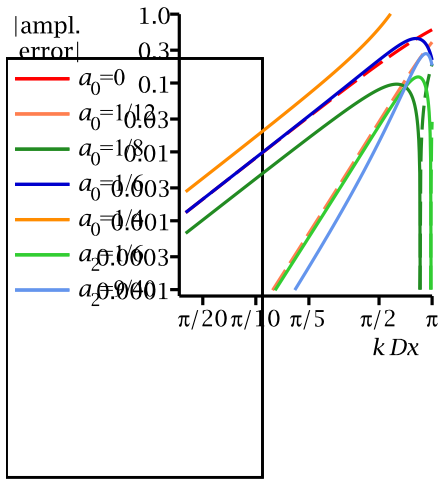
plotphaseerr := *PLOT*(...)

```
> plotamplerr := loglogplot(
    [   simplify( omegaLOW(0,d,kDx,0)/(I*k^2*d) ) - 1,            #  3-point FD, 2nd-order
      - simplify( omegaLOW(0,d,kDx,0)/(I*k^2*d) ) + 1,
        simplify( omegaLOW(0,d,kDx,1/12)/(I*k^2*d) ) - 1,         #  α = 1 / 12, 2nd-order
      - simplify( omegaLOW(0,d,kDx,1/12)/(I*k^2*d) ) + 1,
        simplify( omegaLOW(0,d,kDx,1/8)/(I*k^2*d) ) - 1,          #  piecewise linear vertex-centered FV, 2nd-order
      - simplify( omegaLOW(0,d,kDx,1/8)/(I*k^2*d) ) + 1,
        simplify( omegaLOW(0,d,kDx,1/6)/(I*k^2*d) ) - 1,          #  piecewise parabolic vertex-centered FV (GFEM), 4th-order
      - simplify( omegaLOW(0,d,kDx,1/6)/(I*k^2*d) ) + 1,
        simplify( omegaLOW(0,d,kDx,1/4)/(I*k^2*d) ) - 1,          #  piecewise linear cell-vertex FV (box), 2nd-order
      - simplify( omegaLOW(0,d,kDx,1/4)/(I*k^2*d) ) + 1,
        simplify( omegaHIGH(0,d,kDx, 80/480)/(I*k^2*d) ) - 1,     #  piecewise cubic spline vertex-centered FV, 4th-order
      - simplify( omegaHIGH(0,d,kDx, 80/480)/(I*k^2*d) ) + 1,
        simplify( omegaHIGH(0,d,kDx,108/480)/(I*k^2*d) ) - 1,     #  another piecewise cubic vertex-centered FV, 4th-order
      - simplify( omegaHIGH(0,d,kDx,108/480)/(I*k^2*d) ) + 1 ],
    kDx=0.04*Pi..Pi, 0.0001..1, numpoints=500, thickness=2, size=[555,400], # size=[650,400],
    color=["Red", "Red", "Coral", "Coral", "ForestGreen", "ForestGreen", "MediumBlue", "MediumBlue",
  "DarkOrange", "DarkOrange",
           "LimeGreen", "LimeGreen", "CornflowerBlue", "CornflowerBlue"],
    linestyle=[1,3,1,3,1,3,1,3,1,3,1,3,1,3], thickness=2, legendstyle=[font=[Times,11],location=right],
  #   legend=[ "3-point FD", "", typeset(a[0],"=1/12"), "", "vertex-cent pwise lin", "", "vertex-cent pwise
  parab", "", "cell-vertex pwise lin", "",
  #           "vertex-cent cubic spline", "", "vertex-cent pwise cubic", "" ],
    legend=[ typeset(a[0],"=0"), "", typeset(a[0],"=1/12"), "", typeset(a[0],"=1/8"), "", typeset(a[0],"=
  1/6"), "", typeset(a[0],"=1/4"), "",
             typeset(a[2],"=1/6"), "", typeset(a[2],"=9/40"), "" ],
    labels=[typeset("\t       ",k*Dx)," |ampl.\n  error|\n\n\n\n\n\n\n\n\n\n"], labelfont=[Times,11],
    axis=[thickness=2], axesfont=[Times,11],
    tickmarks=[[Pi/100=typeset(Pi,"/100"),Pi/50=typeset(Pi,"/50"),Pi/20=typeset(Pi,"/20"),
                Pi/10=typeset(Pi,"/10"),Pi/5=typeset(Pi,"/5"),Pi/2=typeset(Pi,"/2"),Pi=typeset(Pi)],
  #              [.0001="0.0001",.0002="0.0002",.0005="0.0005",.001="0.001",.002="0.002",.005="0.005",
  #                .01="0.01",.02="0.02",.05="0.05",.1="0.1",.2="0.2",.5="0.5",1="1.0"]],
              [.0001="0.0001",.0003="0.0003",.001="0.001",.003="0.003",.01="0.01",.03="0.03",.1="0.1",.3=
  "0.3",1="1.0"]]
            );
```

plotamplerr := PLOT(...)

```
> display( convert( [plotphaseerr,plotamplerr], Array )  );
```






```
> # continue
```

INTERMEZZO: determination of a_1, a_2, a_3 as a function of β_0, β_1, β_2 (and a_0). NOT REQUIRED/CONSIDERED ANYMORE => REMOVED

Preparatory work for construction of next procedures for $a_0 = \frac{1}{4}$, which is a bit of a degenerate case. NOT REQUIRED/CONSIDERED ANYMORE => REMOVED

```
> advdiffLOW := advdiffLOW;
  advdiff := collect( simplify(I*eval(advdiffLOW,[omega=aux*u*k/kDx,d=0])/u), aux );
```

$$advdiffLOW := \frac{I\,\omega\,kDx\,(2\,a0\cos(kDx)+1-2\,a0)}{k} - I\,u\sin(kDx) = d\left(\frac{(\cos(kDx)-I\sin(kDx)-1)\,k}{kDx} - \frac{(1-\cos(kDx)-I\sin(kDx))\,k}{kDx}\right)$$

$$advdiff := (-2\,a0\cos(kDx) + 2\,a0 - 1)\,aux + \sin(kDx) = 0$$

Compute dimensionless wave number $k\cdot\Delta x$ for 3-point central discretizations on uniform grid (see above).
NOTE: is a subset of the previous procedures but with a_0 still a free parameter, to be used in the plotting of results below.

NOTE: for $a_0 = 0$ we have sort of deteriorated case, in the sense that we then have $k_{spur}\cdot\Delta x = \pi - k_{phys}\cdot\Delta x \Rightarrow \sin\left(k_{spur}\cdot\Delta x\right) = \sin\left(k_{phys}\cdot\Delta x\right)$ and $\cos\left(k_{spur}\cdot\Delta x\right) = -\cos\left(k_{phys}\cdot\Delta x\right)$.

```
> # assume( aux^2*(1-4*a0) < 1 );   # restrict analysis to frequencies of propagating modes (higher frequencies pertain to evanescent modes)
  kDxLOW := solve( advdiff, kDx ):
  kDxphy := simplify( kDxLOW[1] );   # wave number of physical solution mode
  kDxspu := simplify( kDxLOW[2] );   # wave number of spurious solution mode
  kDx_phy := convert( simplify( series(kDxphy,aux,8) ), polynom ); # k·Δx – value of physical mode (high-order expansion to ensure small difference with exact, which is ...)
  kDx_spu := convert( simplify( series(kDxspu,aux,8) ), polynom ); # k·Δx – value of spurious mode (... *very* important when comparing analytic results with num.results)
  kDx_spu := Pi + (-1+4*a0)*aux + (-1/6+a0-16/3*a0^3)*aux^3          # NOTE: didn't find a way to evaluate (and get rid of)
                + (-3/40+3/4*a0-2*a0^2+64/5*a0^5)*aux^5              # that csgn function with very complex argument in
                + (-5/112+5/8*a0-3*a0^2+5*a0^3-256/7*a0^7)*aux^7 ;   # result of previous expression => use copy of result
```

$$kDxphy := \arctan\left(\frac{aux\left(2\sqrt{4\,a0\,aux^2 - aux^2 + 1}\;a0 - 2\,a0 + 1\right)}{4\,a0^2\,aux^2 + 1}, \frac{4\,a0^2\,aux^2 - 2\,a0\,aux^2 + \sqrt{4\,a0\,aux^2 - aux^2 + 1}}{4\,a0^2\,aux^2 + 1}\right)$$

$$kDxspu := \arctan\left(-\frac{aux\left(2\sqrt{4\,a0\,aux^2 - aux^2 + 1}\;a0 + 2\,a0 - 1\right)}{4\,a0^2\,aux^2 + 1}, -\frac{-4\,a0^2\,aux^2 + 2\,a0\,aux^2 + \sqrt{4\,a0\,aux^2 - aux^2 + 1}}{4\,a0^2\,aux^2 + 1}\right)$$

$$kDx\_phy := aux + \left(-a0 + \frac{1}{6}\right)aux^3 + \left(2\,a0^2 - \frac{3}{4}\,a0 + \frac{3}{40}\right)aux^5 + \left(-5\,a0^3 + 3\,a0^2 - \frac{5}{8}\,a0 + \frac{5}{112}\right)aux^7$$

$$kDx\_spu := \operatorname{csgn}\left(\frac{\mathrm{I}\left(2\,\mathrm{I}\sqrt{4\,a0\,aux^2 - aux^2 + 1}\;a0\,aux - 4\,a0^2\,aux^2 + 2\,\mathrm{I}\,a0\,aux + 2\,a0\,aux^2 - \mathrm{I}\,aux + \sqrt{4\,a0\,aux^2 - aux^2 + 1}\right)}{4\,a0^2\,aux^2 + 1}\right)\pi + (4\,a0 - 1)\,aux + \left(a0 - \frac{16}{3}\,a0^3 - \frac{1}{6}\right)aux^3 + \left(-2\,a0^2 + \frac{3}{4}\,a0 + \frac{64}{5}\,a0^5 - \frac{3}{40}\right)aux^5 + \left(-\frac{5}{112} - \frac{256}{7}\,a0^7 + \frac{5}{8}\,a0 + 5\,a0^3 - 3\,a0^2\right)aux^7$$

$$kDx\_spu := \pi + (4\,a0 - 1)\,aux + \left(a0 - \frac{16}{3}\,a0^3 - \frac{1}{6}\right)aux^3 + \left(-2\,a0^2 + \frac{3}{4}\,a0 + \frac{64}{5}\,a0^5 - \frac{3}{40}\right)aux^5 + \left(-\frac{5}{112} - \frac{256}{7}\,a0^7 + \frac{5}{8}\,a0 + 5\,a0^3 - 3\,a0^2\right)aux^7$$

```
> # dummy statement to allow skipping next one
```

Compute for a certain scheme the numerical wavenumber $k_{phy}\cdot\Delta x$ at a certain phase error err, i.e., the error in numerical phase speed = numerical propagation spee u_{num}. Procedure: compute non-dim. frequency $\omega\cdot\Delta x / u_{exa} = aux$ at that phase error, using the fact that for exact wavenumber $k_{exa}\cdot\Delta x$ we have

$k_{exa}\cdot\Delta x=\omega\cdot\Delta x/u_{exa}=aux$. Check result afterwards by verifying that $u_{num}/u_{exa}-1=k_{exa}\cdot\Delta x/\left(k_{phy}\cdot\Delta x\right)-1=err$.

```
> # dummy statement to allow skipping next one
```

Plot $k\cdot\Delta x$ and phase approximation error for staggered scheme and for collocated scheme for several values of a_0 as a function of $\omega\cdot\Delta x/u$ (should be a close approximation of a straight line).

THIS TO BE REMOVED!!! (I THINK)

FEB'10: choose parameter b_2 (and b_1) in **essential boundary condition** to minimize mismatch between error in inner domain and error at boundary => $b_2=2\cdot a_0-\frac{1}{3}$ (and $b_1=0$).

SEP'10: parameter b_3 added => optimal value $b_3=a_0-\frac{1}{6}$.

```
> solinner := simplify(  int( exp(I*kDx*i), i=-1/2..1/2 ) / ( (1-2*a0) + a0*(exp(-I*kDx) +exp(I*kDx)) )  ):
  solbound := simplify(  exp(I*kDx/2) / ( (1+b1+b2+b3)/2 + (1-b1-2*b2-3*b3)/2 * exp(I*kDx)
                                        + (b2+3*b3)/2 * exp(I*kDx*2) - b3/2 * exp(I*kDx*3) )  ):
  solinnerexpand := simplify( series(solinner, kDx, 5) );
  auxaux := series( 6/8 + 1/8 * ( exp(-I*kDx) + exp(I*kDx) ), kDx, 4 );  # CHECK: piecewise parabolic postprocessing
```

correction that we would apply if $a_0=\frac{1}{6}$

```
  CHECK := series( auxaux * subs(a0=1/6, solinnerexpand), kDx, 4 );
  solboundexpand := simplify( series(solbound, kDx, 4) );
  solve( [ b1 = 0, 1/8 + 1/2*b2 - 1/4*b1^2 = -1/24 + a0,  # equations obtained by setting solboundexpand = solinnerexpand
           b2/4 - b3/2 + b1*5/48 + b1*b2/2 - b1^3/8 = 0 ], [ b1, b2, b3 ] );
```

$$solinnerexpand := 1+\left(-\frac{1}{24}+a0\sim\right)kDx^2+\mathrm{O}\left(kDx^4\right)$$

$$auxaux := 1-\frac{1}{8}kDx^2+\mathrm{O}\left(kDx^4\right)$$

$$CHECK := 1+\mathrm{O}\left(kDx^4\right)$$

$$solboundexpand := 1+\frac{1}{2}\,\mathrm{I}\,b1\,kDx+\left(\frac{1}{8}+\frac{1}{2}\,b2-\frac{1}{4}\,b1^2\right)kDx^2+\left(\frac{5}{48}\,\mathrm{I}\,b1+\frac{1}{4}\,\mathrm{I}\,b2-\frac{1}{2}\,\mathrm{I}\,b3+\frac{1}{2}\,\mathrm{I}\,b1\,b2-\frac{1}{8}\,\mathrm{I}\,b1^3\right)kDx^3+\mathrm{O}\left(kDx^4\right)$$

$$\left[\left[b1=0,\ b2=-\frac{1}{3}+2\,a0\sim,\ b3=-\frac{1}{6}+a0\sim\right]\right]$$

```
> epsexact := solbound / solinner - 1;
```

```
epsexpand := simplify( series(epsexact, kDx, 5) );
solve( [ b1/2 = 0, -a0 + 1/6 + b2/2 - b1^2/4 = 0 ,   # equations obtained by setting epsexpand = 0 (?why did we include this totally equivalent derivation?)
         b2/4 - b3/2 + b1/8 + b1*b2/2 - b1^3/8 - b1*a0/2 = 0 ], [ b1, b2, b3 ] );
```

$$epsexact := -\frac{e^{\frac{1}{2}\,I\,kDx}\,kDx\,(2\,a0\sim\cos(kDx)+1-2\,a0\sim)}{(e^{I\,kDx}\,b1+2\,e^{I\,kDx}\,b2+3\,e^{I\,kDx}\,b3-e^{2\,I\,kDx}\,b2-3\,e^{2\,I\,kDx}\,b3+b3\,e^{3\,I\,kDx}-e^{I\,kDx}-b1-b2-b3-1)\sin\left(\frac{1}{2}\,kDx\right)}-1$$

$$epsexpand := \frac{1}{2}\,I\,b1\,kDx+\left(\frac{1}{6}+\frac{1}{2}\,b2-\frac{1}{4}\,b1^2-a0\sim\right)kDx^2+\left(-\frac{1}{2}\,I\,b1\,a0\sim+\frac{1}{8}\,I\,b1+\frac{1}{4}\,I\,b2-\frac{1}{2}\,I\,b3+\frac{1}{2}\,I\,b1\,b2-\frac{1}{8}\,I\,b1^3\right)kDx^3+O(kDx^4)$$

$$\left[\left[b1=0,\ b2=-\frac{1}{3}+2\,a0\sim,\ b3=-\frac{1}{6}+a0\sim\right]\right]$$

```
> # dummy statement to allow skipping next one
```

▶ Investigate the use of a small $b_1 \neq 0$ and a b_2 slightly different from 'optimal' (?) value $2\cdot a_0 - 1/3$ for $a_0 = 0$ and $b_3 = 0$.

```
> # dummy statement to allow skipping next one
```

▶ Investigate the use of a small $b_1 \neq 0$ and a b_2 slightly different from 'optimal' (?) value $2\cdot a_0 - \frac{1}{3}$ for $a_0 = \frac{1}{8}$ and $b_3 = 0$.

```
> # dummy statement to allow skipping next one
```

▶ Investigate the use of a small $b_1 \neq 0$ and a b_2 slightly different from 'optimal' (?) value $2\cdot a_0 - \frac{1}{3}$ for $a_0 = \frac{1}{6}$ and $b_3 = 0$.

```
> # dummy statement to allow skipping next one
```

▶ Investigate the use of a b_2 and a b_3 slightly different from the 'optimal' (?) values $2\cdot a_0 - \frac{1}{3}$ and $a_0 - \frac{1}{6}$ for $a_0 = \frac{1}{6}$ and $b_1 = 0$.

END THIS TO BE REMOVED!!! (I THINK)

Discretization at outflow boundary = outflow boundary 'condition'

Choose parameter a_2 (and a_1) in **numerical boundary scheme** to minimize spurious reflection (ratio of spurious mode and physical mode) => $a_2 = 2\cdot a_0 - 1/2$ (and $a_1 = 0$).

SEP'10: parameter a_3 added => optimal value $a_3 = a_0 - 1/4$.
These are the values that also give the best accuracy match (see below). With this setting, the first, second (sep'10: and third) term in the expansion of the spurious reflection coefficient vanish.

Choosing a slightly different value of a_2 (with still $a_1 = 0$, and now also $a_3 = 0$), with the intent to get a low spurious reflection over a larger frequency range, is not useful. We cannot use a small non-zero second term, which is imaginary, to compensate for the first non-zero (= third) term in the expansion, which is real. => Introduction of parameter a_1 (first term vanishes for optimal value $a_1 = 0$) besides a_2, even though this introduces a (small) low-order error. => Small non-zero real first term in the expansion. => Possibility to compensate partially for the real third term (and the higher-order odd-numbered other real terms). => Tune parameters a_2 and a_1 such that the higher-order even-numbered (imaginary) terms in the expansion cancel to some extent against a small second term and the higher-order odd-numbered (real) terms in the expansion cancel to some extent against a small first term.
SEP'10: similar story for optimization/tuning of parameter a_2 and new parameter a_3.
Below we consider the optimization of a_2 and a_1 (=> still a convenient 3-point boundary scheme) for $a_0 = 0$, $a_0 = 1/8$, and $a_0 = 1/6$ => spurious reflection $\ll$ numerical phase error, except for higher-order compact scheme $a_0 = 1/6$. Hence, SEP'10, for $a_0 = 1/6$ we now also consider the optimization of a_2 and a_3 (=> less convenient 4-point boundary scheme).
NOTE: low reflection never possible if frequency is fairly (=unrealistically) large. Should not be a problem in practice, where unrealistically large frequencies cq. very short modes should be adequately suppressed to avoid large phase errors.
NB: spurious reflection coefficient ρ determined here differs from the one in "testDDcollocated.mw" by a factor I (?). Didn't find a real explanation, but cause seems to be in the use of different (but equivalent) expressions. => Different squareroots, possibly different evaluation of squareroots => different signs => different phases (I think).

OCT'22: horror story and just for comparison: Neumann condition at outflow boundary

```
> reflNeu := solve(  1 - exp(I*kDxphy) + reflaux * ( 1 - exp(I*kDxspu) ) = 0, reflaux  );
  # reflNeu := simplify( reflNeu );
```

$$reflNeu := \frac{2\,I\sqrt{4\,a0\,aux^2 - aux^2 + 1}\;a0\,aux - 2\,I\,a0\,aux - 2\,a0\,aux^2 + I\,aux + \sqrt{4\,a0\,aux^2 - aux^2 + 1} - 1}{2\,I\,a0\,aux + 2\,a0\,aux^2 - I\,aux + 2\,I\sqrt{4\,a0\,aux^2 - aux^2 + 1}\;a0\,aux + 1 + \sqrt{4\,a0\,aux^2 - aux^2 + 1}}$$

```
> reflgen := solve(  I*aux * ( (1+a1+a2+a3)/2 + (1-a1-2*a2-3*a3)/2 * exp(I*kDxphy)
                             + (a2+3*a3)/2 * exp(I*kDxphy*2) - a3/2 * exp(I*kDxphy*3) )
                   + 1 - exp(I*kDxphy)
       + reflaux * ( I*aux * ( (1+a1+a2+a3)/2 + (1-a1-2*a2-3*a3)/2 * exp(I*kDxspu)
                             + (a2+3*a3)/2 * exp(I*kDxspu*2) - a3/2 * exp(I*kDxspu*3) )
                   + 1 - exp(I*kDxspu) ) = 0, reflaux  ):
  reflgen := simplify( reflgen ):
  reflgenexpand := simplify( series(reflgen, aux, 5) );
  solve( [ op(1,reflgenexpand)=0, op(3,reflgenexpand)=0, op(5,reflgenexpand)=0 ], [ a1, a2, a3 ] );  # equations obtained by setting reflgenexpand = 0
```

$$reflgenexpand := -\frac{1}{4}\,a1\,aux^2 + \left(-\frac{1}{2}\,I\,a0 + \frac{1}{8}\,I + \frac{1}{4}\,I\,a2 + \frac{1}{2}\,I\,a0\,a1 - \frac{1}{4}\,I\,a1 + \frac{1}{8}\,I\,a1^2 + \frac{1}{4}\,I\,a1\,a2 + \frac{1}{2}\,I\,a1\,a3\right) aux^3 + \left(\frac{1}{16}\,a1 - \frac{1}{4}\,a2 + \frac{1}{2}\,a3 + \frac{3}{4}\,a0 + \frac{1}{2}\,a2\,a3 + \frac{1}{4}\,a1^2\,a2 + \frac{1}{2}\,a1^2\,a3 + \frac{1}{4}\,a1\,a2^2 + a1\,a3^2 + \frac{1}{4}\,a0\,a1^2 + \frac{1}{4}\,a2^2 + \frac{1}{16}\,a1^3 - \frac{1}{8}\,a1^2 - a0\,a1\,a3 + a1\,a2\,a3 - a0^2 - \frac{1}{8} - a0\,a3\right) aux^4 + O(aux^5)$$

$$\left[\left[a1 = 0,\ a2 = 2\,a0 - \frac{1}{2},\ a3 = a0 - \frac{1}{4}\right]\right]$$

CONCLUSION from the order-of-accuracy analysis: taking a_1 (the first-order upwind component) different from zero is **not** useful!

```
> # dummy statement to allow skipping next one
```

Choose parameter a_2 (and a_1) in **numerical boundary scheme** to match optimally the (order of) accuracy of the scheme used inside the domain => $a_2 = 2 \cdot a_0 - 1/2$ (and $a_1 = 0$) **again**.
SEP'10: parameter a_3 added => optimal value $a_3 = a_0 - 1/4$ **again**.

NB1, this solution is obtained by equating the first 3 terms in the series expansion of frequency ω as a function of wave number k for discretization inside domain and at boundary.
NB2, below, we now substitute this solution in the series expansion for discretization at boundary. => First 3 terms are clearly equal to those of the series expansion for discretization inside domain.
NB3, compute frequency ω as a function of wave number k (much easier than the inverse!) for discretization inside domain and at boundary (see above).

```
> auxinnerphys := - sin(-kDxph) / ( 1-2*a0 + 2*a0*cos(kDxph) );   # nondim. freq. pertaining to physical mode in inner scheme
  auxinnerphys := simplify( series( auxinnerphys, kDxph, 7 ) );
  auxboundphys := - (1 - exp(I*kDxph))                            # nondim. freq. pertaining to physical mode in boundary scheme
                  / ( (1+a1+a2+a3)/2 + (1-a1-2*a2-3*a3)/2 * exp(I*kDxph)
                      + (a2+3*a3)/2 * exp(I*kDxph*2) - a3/2 * exp(I*kDxph*3) ) / I;
  ### a3 := a0 - 1/4;  a2 := 2*a0 - 1/2;  a1 := 0;                # value of a3, a2 (and a1) giving best accuracy match
  auxboundphys := simplify( series( subs( a3=a0-1/4, a2=2*a0-1/2, a1=0, auxboundphys ), kDxph, 7 ) );
  collect( convert(auxboundphys,polynom)                          # => perfect match for 2nd-, 3rd-, and 4th-order term (*and* 5th-order term (and higher)
           - convert(auxinnerphys,polynom), kDxph );   #      if a0 = 1/4, of course, because this is box scheme => degenerate spurious mode)
```

$$auxinnerphys := \frac{\sin(kDxph)}{1 - 2\,a0 + 2\,a0\cos(kDxph)}$$

$$auxinnerphys := kDxph + \left(-\frac{1}{6} + a0\right) kDxph^3 + \left(\frac{1}{120} - \frac{1}{4}\,a0 + a0^2\right) kDxph^5 + O(kDxph^7)$$

$$auxboundphys := \frac{I\left(1 - e^{I\,kDxph}\right)}{\frac{1}{2} + \frac{1}{2}\,a1 + \frac{1}{2}\,a2 + \frac{1}{2}\,a3 + \frac{1}{2}\,(1 - a1 - 2\,a2 - 3\,a3)\,e^{I\,kDxph} + \frac{1}{2}\,(a2 + 3\,a3)\,e^{2\,I\,kDxph} - \frac{1}{2}\,a3\,e^{3\,I\,kDxph}}$$

$$auxboundphys := kDxph + \left(-\frac{1}{6} + a0\right) kDxph^3 + \left(-\frac{13}{240} + a0^2\right) kDxph^5 + \left(-\frac{1}{16}\,I + \frac{1}{4}\,I\,a0\right) kDxph^6 + O\left(kDxph^7\right)$$

$$\left(-\frac{1}{16}\,I + \frac{1}{4}\,I\,a0\right) kDxph^6 + \left(-\frac{1}{16} + \frac{1}{4}\,a0\right) kDxph^5$$

Investigate the use of a small $a_1 \neq 0$ => Quite useful!!!

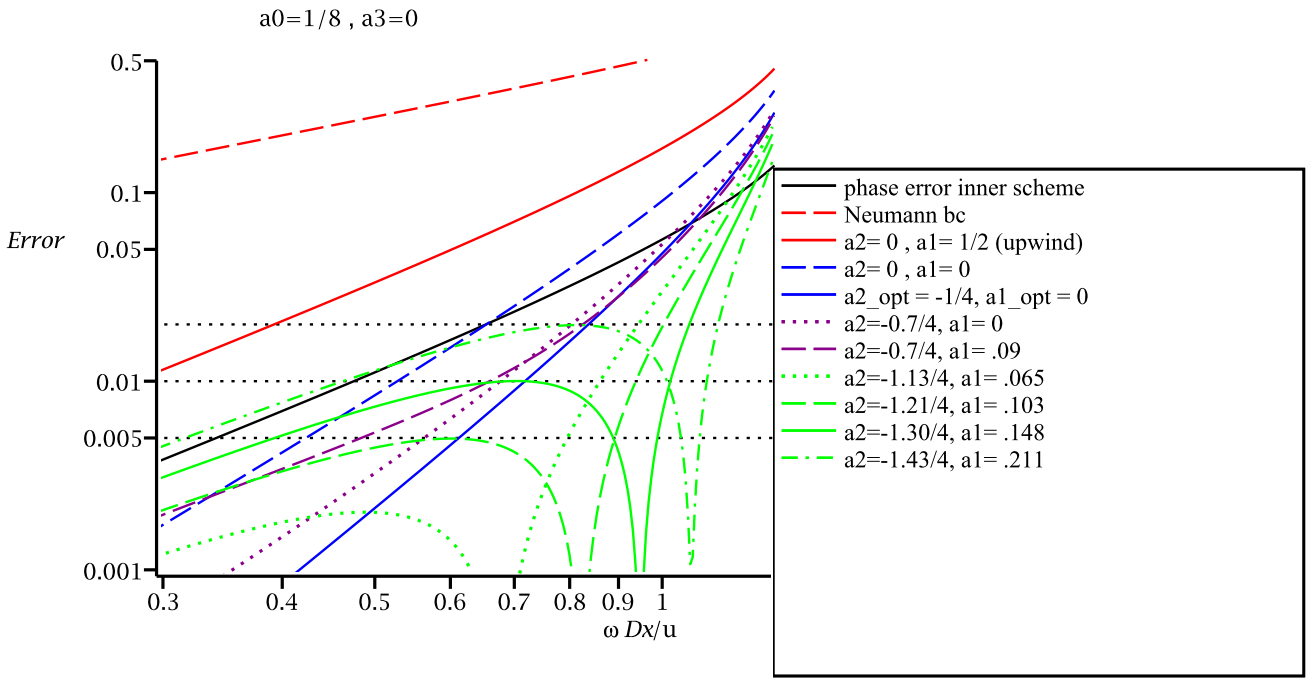
Only consider the compatible/error-consistent 2nd-order piecewise linear vertex-centered FV scheme, i.e., $a_0 = 1/8$.

Investigate the use of a small $a_1 \neq 0$ and a a_2 slightly different from 'optimal' (?) value $2 \cdot a_0 - 1/2$ for $a_0 = 1/8$ and $a_3 = 0$.

```
> refl := simplify( subs(a0=1/8,a3=0,reflgen) ):   # investigate for a0 = 1/8 and a3 = 0
  typeplot:="ln";
  if typeplot="lin" then  auxplot :=       plot:  auxmin := 0:     errmin := 0:      errmax := 0.5:
                    else  auxplot := loglogplot:  auxmin := 0.01:  errmin := 0.001:  errmax := 0.5:  end if:
  errorlevel  := auxplot( [0.005, 0.01, 0.02],                                color=["Black"], thickness=1,
  linestyle=[2] ):
  plotref     := auxplot( - aux/subs(a0=1/8,kDxphy) + 1,      aux=0..3, 0..1, color=["Black"], thickness=1,
  linestyle=[1], numpoints=200, legend="phase error inner scheme" ):
  plotNeu     := auxplot( abs(subs(a0=1/8,reflNeu)),          aux=0..3, 0..1, color=["Red"],   thickness=1,
  linestyle=[3], legend="Neumann bc" ):
  plotupw     := auxplot( abs(subs(a2= 0     ,a1=1/2 ,refl)), aux=0..3, 0..1, color=["Red"],   thickness=1,
  linestyle=[1], legend="a2= 0 , a1= 1/2 (upwind)" ):
  plot00      := auxplot( abs(subs(a2= 0     ,a1= 0  ,refl)), aux=0..3, 0..1, color=["Blue"],  thickness=1,
  linestyle=[3], legend="a2= 0 , a1= 0" ):
  plotoptopt  := auxplot( abs(subs(a2=-1/4   ,a1= 0  ,refl)), aux=0..3, 0..1, color=["Blue"], thickness=1,
  linestyle=[1], legend="a2_opt = -1/4, a1_opt = 0" ):
  plottry1    := auxplot( abs(subs(a2=-0.7/4 ,a1= 0  ,refl)), aux=0..3, 0..1, color=["DarkMagenta"],
  thickness=2, linestyle=[2], legend="a2=-0.7/4, a1= 0" ):
  plotest1    := auxplot( abs(subs(a2=-0.7/4 ,a1=.09 ,refl)), aux=0..3, 0..1, color=["DarkMagenta"],
  thickness=1, linestyle=[3], legend="a2=-0.7/4, a1= .09" ):
  plottry2    := auxplot( abs(subs(a2=-1.13/4,a1=.065,refl)), aux=0..3, 0..1, color=[green], thickness=2,
```

```
linestyle=[2], legend="a2=-1.13/4, a1= .065", numpoints=500 ):  # error <= 0.002
plotest2    := auxplot( abs(subs(a2=-1.21/4,a1=.103,refl)), aux=0..3, 0..1, color=[green], thickness=1,
linestyle=[3], legend="a2=-1.21/4, a1= .103", numpoints=500 ):  # error <= 0.005
plotest3    := auxplot( abs(subs(a2=-1.30/4,a1=.148,refl)), aux=0..3, 0..1, color=[green], thickness=1,
linestyle=[1], legend="a2=-1.30/4, a1= .148", numpoints=500 ):  # error <= 0.01
plotest4    := auxplot( abs(subs(a2=-1.43/4,a1=.211,refl)), aux=0..3, 0..1, color=[green], thickness=1,
linestyle=[4], legend="a2=-1.43/4, a1= .211", numpoints=500 ):  # error <= 0.02
display( [ errorlevel, plotref, plotNeu, plotupw, plot00, plotoptopt, plottry1, plotest1, plottry2,
plotest2, plotest3, plotest4 ], # , plotlegends ] ,
        view=[0.3..1.3,errmin..errmax], size=[700,400], symbolsize=12, labels=[typeset("\t\t\t",omega*Dx,
"/u"),typeset(Error,"\n\n\n\n\n\n")],
        labelfont=[Times,11], axis=[thickness=2], axesfont=[Times,11], titlefont=[Times,11], legendstyle=
[font=[Times,11],location=right],
        title="a0=1/8 , a3=0                                                                    " );
```

typeplot := "ln"




```
> # dummy statement to allow skipping next one
```

```
> eval( xi^2 - xi + 1/6, xi=1/2 );
```

$$-\frac{1}{12}$$

```
> # QUICK ADDITION
```

```
> restart:
```

```
with(plots):
> FVEmass := 1/8*exp(-I*kDx) + 3/4 + 1/8*exp(I*kDx);   # For comparison: F-mode transform of average amplitude of FVE inner scheme
```

$$FVEmass := \frac{1}{8}\, e^{-I\,kDx} + \frac{3}{4} + \frac{1}{8}\, e^{I\,kDx}$$

```
> FVEerror := (exp(I*kDx) - exp(-I*kDx)) / 2 / FVEmass - I*kDx;   # F-mode transform of error in FVE advection (error of about same amplitude in FVE diffusion)
```

$$FVEerror := \frac{1}{2}\, \frac{e^{I\,kDx} - e^{-I\,kDx}}{\frac{1}{8}\, e^{-I\,kDx} + \frac{3}{4} + \frac{1}{8}\, e^{I\,kDx}} - I\,kDx$$

```
> BClin := (1-xi)*1 + xi*exp(I*kDx);   # F-mode transform of linear interpolation at boundary
```

$$BClin := 1 - \xi + \xi\, e^{I\,kDx}$$

```
> BCparGP := BClin - (1-2*exp(I*kDx)+exp(2*I*kDx))/2 * (1-xi)*xi;   # F-mode transform of quadratic interpolation at boundary through Grid Points
```

$$BCparGP := 1 - \xi + \xi\, e^{I\,kDx} - \frac{1}{2}\, \left(1 - 2\, e^{I\,kDx} + e^{2\,I\,kDx}\right)\, (1 - \xi)\, \xi$$

```
> [ eval(BCparGP,xi=0), eval(BCparGP,xi=1/2), eval(BCparGP,xi=1), eval(BCparGP,xi=3/2), eval(BCparGP,xi=2) ];
# just inspecting/checking: correct!
[ eval(diff(BCparGP,xi),xi=1/2), eval(diff(BCparGP,xi),xi=3/2) ];   # just checking: correct!
```

$$\left[1,\ \frac{3}{8} + \frac{3}{4}\, e^{I\,kDx} - \frac{1}{8}\, e^{2\,I\,kDx},\ e^{I\,kDx},\ -\frac{1}{8} + \frac{3}{4}\, e^{I\,kDx} + \frac{3}{8}\, e^{2\,I\,kDx},\ e^{2\,I\,kDx}\right]$$

$$\left[-1 + e^{I\,kDx},\ -e^{I\,kDx} + e^{2\,I\,kDx}\right]$$

```
> BCparCC := BClin + (1-2*exp(I*kDx)+exp(2*I*kDx))/2 * (xi-1/2)^2;   # F-mode transform of quadratic interpolation at boundary through Cell Centers
```

$$BCparCC := 1 - \xi + \xi\, e^{I\,kDx} + \frac{1}{2}\, \left(1 - 2\, e^{I\,kDx} + e^{2\,I\,kDx}\right)\, \left(\xi - \frac{1}{2}\right)^2$$

```
> [ eval(BCparCC,xi=0), eval(BCparCC,xi=1/2), eval(BCparCC,xi=1), eval(BCparCC,xi=3/2), eval(BCparCC,xi=2) ];
# just inspecting/checking: correct!
[ eval(diff(BCparCC,xi),xi=1/2), eval(diff(BCparCC,xi),xi=3/2) ];   # just checking: correct!
```

$$\left[\frac{9}{8} - \frac{1}{4}\, e^{I\,kDx} + \frac{1}{8}\, e^{2\,I\,kDx},\ \frac{1}{2} + \frac{1}{2}\, e^{I\,kDx},\ \frac{1}{8} + \frac{3}{4}\, e^{I\,kDx} + \frac{1}{8}\, e^{2\,I\,kDx},\ \frac{1}{2}\, e^{I\,kDx} + \frac{1}{2}\, e^{2\,I\,kDx},\ \frac{1}{8} - \frac{1}{4}\, e^{I\,kDx} + \frac{9}{8}\, e^{2\,I\,kDx}\right]$$

$$\left[-1 + e^{I\,kDx},\ -e^{I\,kDx} + e^{2\,I\,kDx}\right]$$

```
> BCparderiv := 1/3 * BCparGP + 2/3 * BCparCC;   # construction of BCpar
BCpar := BClin + (1-2*exp(I*kDx)+exp(2*I*kDx))/2 * (xi^2 - xi + 1/6);   # F-mode transform of best quadratic fit at boundary (equal to previous)
```

```
simplify( BCparderiv - BCpar );   # just checking: correct!
int( BCpar, xi=1/2..3/2);   # just checking: correct!
```

$$BCparderiv := 1 - \xi + \xi\, e^{I kDx} - \frac{1}{6}\left(1 - 2\, e^{I kDx} + e^{2\,I kDx}\right)(1-\xi)\,\xi + \frac{1}{3}\left(1 - 2\, e^{I kDx} + e^{2\,I kDx}\right)\left(\xi - \frac{1}{2}\right)^2$$

$$BCpar := 1 - \xi + \xi\, e^{I kDx} + \frac{1}{2}\left(1 - 2\, e^{I kDx} + e^{2\,I kDx}\right)\left(\xi^2 - \xi + \frac{1}{6}\right)$$

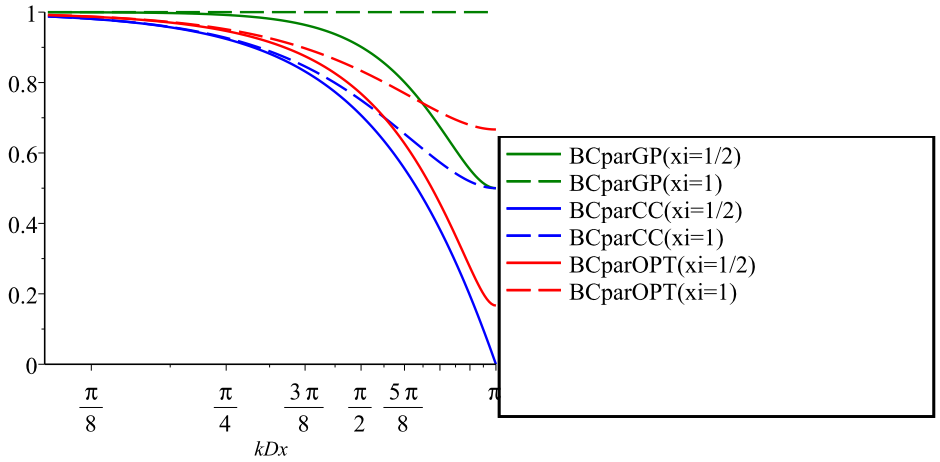
$$0$$

$$\frac{1}{8} + \frac{3}{4}\, e^{I kDx} + \frac{1}{8}\, e^{2\,I kDx}$$

```
> [ eval(BCpar,xi=1/2), eval(BCpar,xi=1), eval(BCpar,xi=3/2) ];   # just inspecting
```

$$\left[\frac{11}{24} + \frac{7}{12}\, e^{I kDx} - \frac{1}{24}\, e^{2\,I kDx},\ \frac{1}{12} + \frac{5}{6}\, e^{I kDx} + \frac{1}{12}\, e^{2\,I kDx},\ -\frac{1}{24} + \frac{7}{12}\, e^{I kDx} + \frac{11}{24}\, e^{2\,I kDx}\right]$$

```
> # logplot( [ FVEmass,
#         abs(eval(BClin,xi=1/2)), abs(eval(BClin,xi=1)), abs(eval(BCparGP,xi=1/2)), abs(eval(BCparGP,xi=1)
), abs(eval(BCparCC,xi=1/2)), abs(eval(BCparCC,xi=1)),
#         abs(eval(BCpar,xi=1/2)), abs(eval(BCpar,xi=1)), abs(eval(BCpar,xi=3/2)) ], kDx=0.1*Pi..1.0*Pi,
legendstyle=[location=right], size=[500,300],
#       linestyle=[1,1,3,1,3,1,3,1,3,2], color=["Black","Red","Red","Green","Green","Blue","Blue","Red",
"Red","Red"],
#       legend=["FVEmass", "BClin(xi=1/2)", "BClin(xi=1)", "BCparGP(xi=1/2)", "BCparGP(xi=1)", "BCparCC(xi=
1/2)", "BCparCC(xi=1)",
#               "BCparOPT(xi=1/2)", "BCparOPT(xi=1)", "BCparOPT(xi=3/2)"] );
#  DEC'23: BClin(xi=1/2) plot coincides with BCparCC(xi=1/2); BClin(xi=1) plot coincides with BCparGP(xi=1) => remove?
#  DEC'23: BCparOPT(xi=1/2) plot coincides with BCparOPT(xi=3/2) => remove
#  DEC'23: FVEmass plot coincides with BCparCC(xi=1), but compares volume value with point value => ??? => remove
semilogplot( [ abs(eval(BCparGP,xi=1/2)), abs(eval(BCparGP,xi=1)), abs(eval(BCparCC,xi=1/2)), abs(eval
(BCparCC,xi=1)),
        abs(eval(BCpar,xi=1/2)), abs(eval(BCpar,xi=1)) ], kDx=0.1*Pi..1.0*Pi, legendstyle=[location=right],
size=[500,300],
      linestyle=[1,3,1,3,1,3,1,3,2], color=["Green","Green","Blue","Blue","Red","Red","Red"],
      legend=["BCparGP(xi=1/2)", "BCparGP(xi=1)", "BCparCC(xi=1/2)", "BCparCC(xi=1)",
              "BCparOPT(xi=1/2)", "BCparOPT(xi=1)"] );
```



DEC'23: jump at volume boundaries

```
> jumpparGP  := simplify( eval(BCparGP,xi=1/2) - eval(BCparGP,xi=3/2) * exp(-I*kDx) );
  jumpparCC  := simplify( eval(BCparCC,xi=1/2) - eval(BCparCC,xi=3/2) * exp(-I*kDx) );
  jumpparOPT := simplify( eval(BCpar,xi=1/2)   - eval(BCpar,xi=3/2)   * exp(-I*kDx) );
```

$$jumpparGP := -\frac{3}{8} + \frac{3}{8}\,e^{1\,kDx} - \frac{1}{8}\,e^{2\,1\,kDx} + \frac{1}{8}\,e^{-1\,kDx}$$

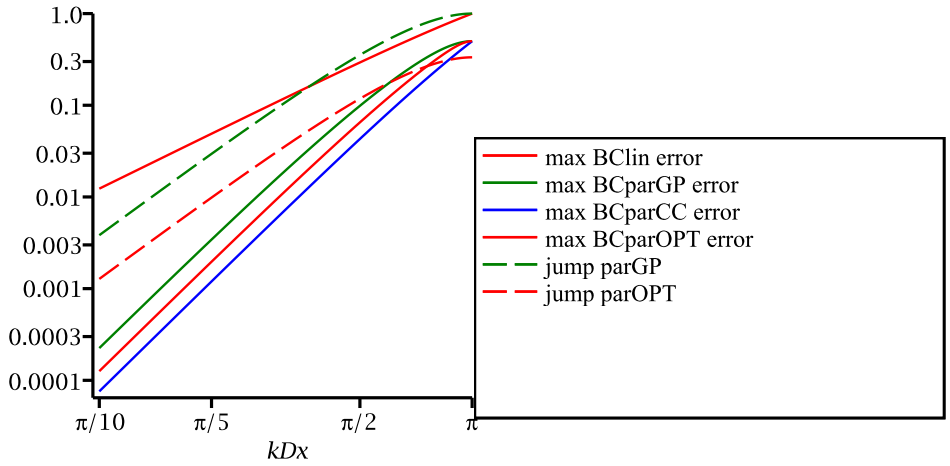
$$jumpparCC := 0$$

$$jumpparOPT := -\frac{1}{8} + \frac{1}{8}\,e^{1\,kDx} - \frac{1}{24}\,e^{2\,1\,kDx} + \frac{1}{24}\,e^{-1\,kDx}$$

DEC'23: quick plot of max. amplitude difference/error of the boundary condition interpolations that we consider and jump at volume boundaries if present.

```
> loglogplot( [ abs( abs(eval(BClin,xi=1/2))   - abs(eval(BClin,xi=1))   ), abs( abs(eval(BCparGP,xi=1/2)) -
  abs(eval(BCparGP,xi=1)) ),
                abs( abs(eval(BCparCC,xi=1/2)) - abs(eval(BCparCC,xi=1)) ), abs( abs(eval(BCpar,xi=1/2))   -
  abs(eval(BCpar,xi=1))   ),
                abs(jumpparGP), abs(jumpparOPT) ],
              kDx=0.1*Pi..1.0*Pi, legendstyle=[location=right], size=[500,300],
```

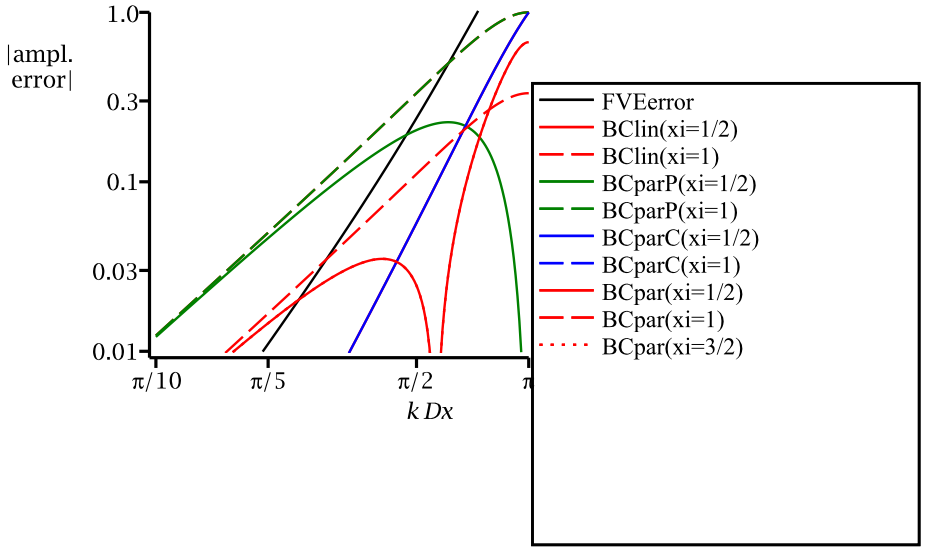
```
      linestyle=[1,1,1,1,3,3], color=["Red","Green","Blue","Red","Green","Red"],
      legend=["max BClin error","max BCparGP error", "max BCparCC error", "max BCparOPT error", "jump
parGP", "jump parOPT"], labelfont=[Times,11],
      axis=[thickness=2], axesfont=[Times,11],
      tickmarks=[[Pi/100=typeset(Pi,"/100"),Pi/50=typeset(Pi,"/50"),Pi/20=typeset(Pi,"/20"),
                  Pi/10=typeset(Pi,"/10"),Pi/5=typeset(Pi,"/5"),Pi/2=typeset(Pi,"/2"),Pi=typeset(Pi)],
                 [.0001="0.0001",.0003="0.0003",.001="0.001",.003="0.003",.01="0.01",.03="0.03",.1="0.1",.3=
"0.3",1="1.0"]] );
```



Plot of error behavior: compare bc amplitude with FVEmass. Are we looking at the correct error?

```
> loglogplot( [ abs(FVEerror), abs(abs(eval(BClin,xi=1/2)/FVEmass)-1), abs(abs(eval(BClin,xi=1)/FVEmass)-1),
                              abs(abs(eval(BCparP,xi=1/2)/FVEmass)-1), abs(abs(eval(BCparP,xi=1)/FVEmass)-1),
                              abs(abs(eval(BCparC,xi=1/2)/FVEmass)-1), abs(abs(eval(BCparC,xi=1)/FVEmass)-1),
                              abs(abs(eval(BCpar,xi=1/2)/FVEmass)-1), abs(abs(eval(BCpar,xi=1)/FVEmass)-1),
  abs(abs(eval(BCpar,xi=3/2)/FVEmass)-1) ],
    kDx=0.1*Pi..Pi, 0.01..1, legendstyle=[location=right], size=[500,300],
    linestyle=[1,1,3,1,3,1,3,1,3,2], color=["Black","Red","Red","Green","Green","Blue","Blue","Red","Red",
```

```
"Red"],
  legend=["FVEerror", "BClin(xi=1/2)", "BClin(xi=1)", "BCparP(xi=1/2)", "BCparP(xi=1)", "BCparC(xi=1/2)",
"BCparC(xi=1)", "BCpar(xi=1/2)", "BCpar(xi=1)", "BCpar(xi=3/2)"],
  labels=[typeset("\t       ",k*Dx)," |ampl.\n  error|\n\n\n\n\n\n\n\n\n\n"], labelfont=[Times,11],
  axis=[thickness=2], axesfont=[Times,11],
  tickmarks=[[Pi/100=typeset(Pi,"/100"),Pi/50=typeset(Pi,"/50"),Pi/20=typeset(Pi,"/20"),
              Pi/10=typeset(Pi,"/10"),Pi/5=typeset(Pi,"/5"),Pi/2=typeset(Pi,"/2"),Pi=typeset(Pi)],
#             [[.0001="0.0001",.0002="0.0002",.0005="0.0005",.001="0.001",.002="0.002",.005="0.005",
#                 .01="0.01",.02="0.02",.05="0.05",.1="0.1",.2="0.2",.5="0.5",1="1.0"]],
            [.0001="0.0001",.0003="0.0003",.001="0.001",.003="0.003",.01="0.01",.03="0.03",.1="0.1",.3=
"0.3",1="1.0"]]
 );
```




```
> # END QUICK ADDITION
```

```
> # NOV'23: STUFF BELOW SEEMS TO BE ABOUT ACTUAL SOLUTION RECONSTRUCTION; OBSOLETE?
```

SPECIFY:

• type of discretization by specifying a_0 and a_1, a_2, a_3, b_1, b_2, b_3:

$0 \le a_0 \le \frac{1}{4}$: weight of piecewise linear discretization on collocated grid (includes finite difference, $a_0 = 0$; compatible, $a_0 = \frac{1}{8}$; Galerkin finite element, $a_0 = \frac{1}{6}$; box, $a_0 = \frac{1}{4}$).

• type of boundary handling:
$k_{per} = 1$: enforce periodic solution;
$k_{per} = 0$: apply boundary conditions.

• grid size Δx and number of volumes/inner grid points $itot$;

• frequency ω ;

• phase φ of right-going wave at left boundary grid point (for normalization; likewise, amplitude of right-going wave at left boundary grid point set to 1).

NOTE: the grid points are $x_0, x_1, \ldots, x_{itot}, x_{itot+1}$, with $x_i - x_{i-1} = \Delta x$ and x_0 and x_{itot+1} virtual grid points, hence grid with $itot$ (inner) volumes and 2 boundary volumes.

```
> a0 := 1/6;
  # see below for (automatic) setting of the β's
  kper := 0;
  # Dx := 10/40;
  # Dx := 10/20;
  # Dx := 10/14;
  # Dx := 10/10;
  Dx := 10/7;   # very coarse grid (only for 4th-order accurate schemes with a0 = 1/6)
  # Dx := 10/6;   # very coarse grid (only for 4th-order accurate schemes with a0 = 1/6)
  itot := 20/Dx;
  omega := 9/10 * u;   # CAREFUL: to ensure real wave numbers, there is an upper limit! (which is ω = u if a0 = 0 and Δx = 1 => use ω = 9/10 · u in tests)
  phi := -0.0: # 4/20:
```

$$a0 := \frac{1}{6}$$

$$kper := 0$$

$$Dx := \frac{10}{7}$$

$$itot := 14$$

$$\omega := \frac{9}{10}\, u\sim$$

- 'automatic' setting of parameters a_1, a_2, a_3 in numerical boundary scheme (values obtained from above figures with investigation of effect of these parameters):
  a_1 : weight of first-order upwind shift of time derivative;
  a_2 : weight of second-order upwind shift of time derivative;
  a_3 : weight of third-order upwind shift of time derivative.
- MAR'10: 'automatic' setting of parameters b_1, b_2, b_3 in essential (Dirichlet) boundary condition (values obtained from above figures with investigation of effect of these parameters):
  b_1 : weight of first-order redistribution of imposed value;
  b_2 : weight of second-order redistribution of imposed value;
  b_3 : weight of third-order redistribution of imposed value.

```
> a1 := 0:  a2 := 0:  a3 := 0:  b1 := 0:  b2 := 0:  b3 := 0:  # initialization
  if a0 = 0 then      # 2nd-order finite difference scheme
    a2 := -1/2;      a1 := 0;      # optimal (spurious reflection < 0.2 % ) for ω·Δx/u <≲ 0.40 (highest order of accuracy)
  #   a2 := -1.15/2;   a1 := .14;    # optimal (spurious reflection < 0.5 % ) for ω·Δx/u ≲ 0.70
  #   a2 := -1.20/2;   a1 := .195;   # optimal (spurious reflection ≲ 1.0 % ) for ω·Δx/u < 0.79
    b2 := -1/3;      b1 := 0;      # optimal (amplitude error < 0.2 % ) for k·Δx < 0.29 (highest order of accuracy)
  #   b2 := -1.05/3;   b1 := .049;   # optimal (amplitude error < 0.5 % ) for k·Δx < 0.62
  #   b2 := -1.10/3;   b1 := .0785;  # optimal (amplitude error < 1.0 % ) for k·Δx < 0.78
  elif a0 = 1/8 then  # 2nd-order piecewise linear compatible
    a2 := -1/4;      a1 := 0;      # optimal (spurious reflection < 0.2 % ) for ω·Δx/u ≲ 0.49 (highest order of accuracy)
  #   a2 := -1.13/4;   a1 := .065;   # optimal (spurious reflection < 0.2 % ) for ω·Δx/u ≲ 0.74
  #   a2 := -1.21/4;   a1 := .103;   # optimal (spurious reflection < 0.5 % ) for ω·Δx/u ≲ 0.89
  #   a2 := -1.30/4;   a1 := .148;   # optimal (spurious reflection < 1.0 % ) for ω·Δx/u ≲ 1.02
```

```
#   a2 := -1.43/4;   a1 := .211;   # optimal (spurious reflection < 2.0 % ) for ω·Δx/u < 1.15
#   b2 := -1/12;     b1 := 0;      # optimal (amplitude error < 0.2 % ) for k·Δx ≲ 0.45  (highest order of accuracy)
  b2 := -1.12/12;  b1 := .017:   # optimal (amplitude error < 0.2 % ) for k·Δx < 0.70
#   b2 := -1.22/12;  b1 := .031;   # optimal (amplitude error < 0.5 % ) for k·Δx ≲ 0.92
#   b2 := -1.35/12;  b1 := .0505;  # optimal (amplitude error < 1.0 % ) for k·Δx ≤ 1.13
#   b2 := -1.56/12;  b1 := .0825;  # optimal (amplitude error < 2.0 % ) for k·Δx <≲ 1.36
elif a0 = 1/6 then   # 4th-order compact/piecewise parabolic semi-compatible
#   a2 := -1/6;      a1 := 0;      # optimal (spurious reflection < 0.2 % ) for ω·Δx/u ≤ 0.55 (highest order of accuracy)
#   a2 := -1.16/6;   a1 := .053;   # optimal (spurious reflection < 0.2 % ) for ω·Δx/u ≲ 0.83
#   a2 := -1.26/6;   a1 := .086;   # optimal (spurious reflection < 0.5 % ) for ω·Δx/u ≤ 1.02
#   a2 := -1.37/6;   a1 := .123;   # optimal (spurious reflection < 1.0 % ) for ω·Δx/u < 1.18
#   a2 := -1.53/6;   a1 := .179;   # optimal (spurious reflection < 2.0 % ) for ω·Δx/u < 1.34
#   b2 := 0;         b1 := 0;      # optimal (amplitude error < 0.2 % ) for k·Δx ≤ 0.76  (highest order of accuracy; exact if post-processing is
applied)
#   b2 := -0.32/24;  b1 := .006;   # optimal (amplitude error < 0.2 % ) for k·Δx < 1.09
#   b2 := -0.53/24;  b1 := .013;   # optimal (amplitude error < 0.5 % ) for k·Δx ≲ 1.32
#   b2 := -0.75/24;  b1 := .0245;  # optimal (amplitude error < 1.0 % ) for k·Δx < 1.51
#   b2 := -1.14/24;  b1 := .0455;  # optimal (amplitude error < 2.0 % ) for k·Δx < 1.72
#   a2 := -1/6;      a3 := -1/12;  # higher-order optimal (spurious reflection < 0.2 % ) for ω·Δx/u < 0.710 (highest order of accuracy)
#   a2 := -.727/6;   a3 := -.106;  # higher-order optimal (spurious reflection < 0.2 % ) for ω·Δx/u < 1.022
#   a2 := -.594/6;   a3 := -.117;  # higher-order optimal (spurious reflection < 0.5 % ) for ω·Δx/u ≤ 1.194
  a2 := -.448/6;   a3 := -.129;  # higher-order optimal (spurious reflection < 1.0 % ) for ω·Δx/u < 1.329
#   a2 := -.246/6;   a3 := -.146;  # higher-order optimal (spurious reflection < 2.0 % ) for ω·Δx/u < 1.460
#   b2 := 0;         b1 := 0;      # higher-order optimal (amplitude error < 0.2 % ) for k·Δx < 0.760  (highest order of accuracy; exact if post-
processing is applied)
```

```
#   b2 := -.19/24;  b3 := -.0100; # higher-order optimal (amplitude error < 0.2 % ) for k·Δx < 1.253
#   b2 := -.14/24;  b3 := -.0170; # higher-order optimal (amplitude error < 0.5 % ) for k·Δx < 1.513
  b2 :=  .03/24;  b3 := -.0249; # higher-order optimal (amplitude error < 1.0 % ) for k·Δx < 1.707
#   b2 :=  .35/24;  b3 := -.0354; # higher-order optimal (amplitude error < 2.0 % ) for k·Δx < 1.895
elif a0 = 1/4 then  # 2nd-order box scheme
  a2 := 0;  a1 := 0;            # optimal is a2 = a1 = 0  (box scheme, spurious reflection = 0 % )
  b2 := 0;  b1 := 0;            # not optimal (optimization not investigated)
end if:
# a2 := 2*a0 - 1/2:  a3 = a0 - 1/4:  a1 := 0:  # apply optimal value of a2 and a3 if a1 = 0
# b2 := 2*a0 - 1/3:  b3 = a0 - 1/6:  b1 := 0:  # apply optimal value of b2 and b3 if b1 = 0
# a2 := 0:  a3 := 0:  a1 := 0:  # apply 2-point central box scheme
# b2 := 0:  b3 := 0:  b1 := 0:  # apply 2-point central scheme
the_a_values := [ a0, a1, a2, a3 ];  # just for output
the_b_values := [ a0, b1, b2, b3 ];  # just for output
```

$$the\_a\_values := \left[\frac{1}{6},\ 0,\ -0.07466666667,\ -0.129\right]$$

$$the\_b\_values := \left[\frac{1}{6},\ 0,\ 0.001250000000,\ -0.0249\right]$$

Compute the wave numbers k_{exact}, k_{phys}, k_{spur}, and the number of grid cells per wave N_{exact}, N_{phys}, N_{spur}.

NOTE: we solve the equation backwards, i.e., *against* the direction of transport. This works for central discretization schemes, but usually not for upwind schemes where we may have a strongly damped spurious mode in transport direction. => Strong growth of (round-off) errors if computed backwards. => Highly unstable.

```
> kDxexact := evalf( omega*Dx/u );  Nexact := evalf( 2*Pi/kDxexact );
  kDxspurmax := evalf( Pi );
  auxmax := piecewise( 1-4*a0 = 0, infinity, evalf(1 / sqrt(1 - 4*a0)) );
  aux := omega*Dx/u;
  kDxph := kDxphys( aux, a0 );  Nphys := evalf( 2*Pi/Re(kDxph) );
  kDxsp := kDxspur( aux, a0 );  Nspur := evalf( 2*Pi/Re(kDxsp) );
```

$$kDxexact := 1.285714286$$

$$Nexact := 4.886921904$$

$$kDxspurmax := 3.141592654$$

$$auxmax := 1.732050808$$

$$aux := \frac{9}{7}$$

$$kDxph := 1.312183419$$

$$Nphys := 4.788343776$$

$$kDxsp := 2.639192808$$

$$Nspur := 2.380722351$$

Only physical mode (well, not if boundary condition is applied at right boundary), hence ...

By the way, position of right boundary is at $\frac{x_{itot} + x_{itot+1}}{2}$ and is irrelevant if periodic solution is forced but relevant if boundary condition is applied, in the sense tha it determines the relative phase between the different modes. Since we are mainly interested in relative amplitudes and not really in relative phases, this is not something to worry about.

```
> c[itot] := exp( I*(phi-kDxph*(itot-1/2)) );
  if kper=1 then
```

enforce periodic solution

```
    c[itot+1] := exp( I*(phi-kDxph*(itot+1/2)) );
    c[itot-1] := exp( I*(phi-kDxph*(itot-3/2)) );   # just for checking (recomputed later)
  else
```

determine value c_{itot+1} at virtual grid point by solving numerical boundary condition and discretization for 1st and 2nd inner volume

```
    eqbc := I*omega * ( (1+a1+a2+a3)/2 * cvirt + (1-a1-2*a2-3*a3)/2 * c[itot] + (a2+3*a3)/2 * cin2 - a3/2 *
  cin3 ) * Dx
            + u * ( cvirt - c[itot] ) = 0;
    eqin1 := I*omega * ( (1-2*a0) * c[itot] + a0 * (cin2 + cvirt  ) ) * Dx + u/2 * ( cvirt   - cin2 ) = 0;  #
```

1st inner volume

```
    eqin2 := I*omega * ( (1-2*a0) * cin2    + a0 * (cin3 + c[itot]) ) * Dx + u/2 * ( c[itot] - cin3 ) = 0;  #
```

2nd inner volume

```
    sol := solve( [ eqbc, eqin1, eqin2 ], [ cvirt, cin2, cin3 ] );
    c[itot+1] := simplify( op(2,op(1,op(1,sol))) );
    c[itot-1] := simplify( op(2,op(2,op(1,sol))) );   # just for checking (recomputed later)
  end if:
  eqin1 := I*omega * ( (1-2*a0) * c[itot] + a0 * (c[itot-1] + c[itot+1]) ) * Dx + u/2 * ( c[itot+1] - c
  [itot-1] );   # CHECK!
```

MAR'10: add an $itot+1$ volume (for piecewise parabolic post-processing only)

```
  eqin0 := I*omega * ( (1-2*a0) * c[itot+1] + a0 * (c[itot] + c0) ) * Dx + u/2 * ( c0 - c[itot] ) = 0:
  sol := solve( [eqin0], [c0] ):
  c[itot+2] := simplify( op(2,op(1,op(1,sol))) );
```

$$c_{14} := 0.4220601247 + 0.9065678414\,I$$

$$eqin1 := \left(-4.\,10^{-10} + 1.\,10^{-10}\,I\right) u\sim$$

$$c_{16} := 0.07961695905 - 1.003302064\,I$$

Discretization in inner domain, skipping last volume (already taken care of in previous execution group) and adding a zeroth volume (for piecewise parabolic post-processing only).
NOTE: without the extension with a zeroth volume, a piecewise parabolic extrapolation would be required at the left side of the left domain in case of the use of a semi compatible piecewise parabolic discretization, in order to be able to apply the required post-processing (see below). This would not be correct because that side is not a boundary, since we compute the solution from right to left. To avoid errors in the normalization of the solution later on, which uses the solution at that left side, we extend the discretization across that side. We do apply, if applicable, a piecewise parabolic extrapolation at the right side of the right domain, also if that side is not treated as a boundary, because it has no consequence on the overall result.

```
> for i from itot-1 by -1 to 0 do
    eqin := I*omega * ( (1-2*a0) * c[i] + a0 * (c0 + c[i+1]) ) * Dx + u/2 * ( c[i+1] - c0 ) = 0;
    sol := solve( [eqin], [c0] );
    c[i-1] := simplify( op(2,op(1,op(1,sol))) );
  end do:
  c[1];  c[0];  c[-1];  # just for inspection; NB, average of c_o and c_1 should roughly be equal to 1
```

$$0.7968893597 - 0.6123453205\,I$$
$$0.7924499393 + 0.6109790892\,I$$
$$-0.3837304046 + 0.9232084136\,I$$

MAR'10: if not periodic solution, 'apply' imposed boundary condition $c = \exp(-i\cdot\varphi)$ by computing a correcting scaling factor for the solution already determined (application of correction factor below).

```
> if kper=0 then
    bcvalue := (1+b1+b2+b3)/2 * c[0] + (1-b1-2*b2-3*b3)/2 * c[1] + (b2+3*b3)/2 * c[2] - b3/2 * c[3]:
    corr := exp(I*phi) / bcvalue:
    abs( corr ):
  end if;
```

$$bcvalue := 0.8161305484 + 0.005807853478\,I$$
$$corr := 1.225232118 - 0.008719154836\,I$$
$$1.225263142$$

Procedure to compute an approximation of the average amplitude of the solution over a volume, using trapezium rule.
Rather exaggerated reconstruction procedure if numerical solution is (assumed) piecewise linear or piecewise constant, but intermediate values also used (below) for plotting of numerical solution where it is quite convenient to have same procedure for all schemes.

```
> amp := proc( cwest, ccent, ceast, a0 )  local aux;  global cl, clllc, clc, clccc, cc, crccc, crc, crrrc, cr;
          cl := (cwest + ccent) / 2;
          cr := (ceast + ccent) / 2;
  #         if a0 = 0 then    # finite difference: assume piecewise constant reconstruction if a_0 = 0
  #           cc := ccent;  cl := cc;  cr := cc;  clc := cc;  crc := cc;
  #           clllc := cc;  clccc := cc;  crccc := cc;  crrrc := cc;
          if a0 = 1/6 then  # elif a0 = 1/6 then  # piecewise parabolic reconstruction if
```

$$a_0 = \frac{1}{6}$$

```
      clllc := (49*cwest + 78*ccent +    ceast) / 128;
      clc   := ( 9*cwest + 22*ccent +    ceast) /  32;
      clccc := (25*cwest + 94*ccent +  9*ceast) / 128;
      cc    := (   cwest +  6*ccent +    ceast) /   8;
      crccc := ( 9*cwest + 94*ccent + 25*ceast) / 128;
      crc   := (   cwest + 22*ccent +  9*ceast) /  32;
      crrrc := (   cwest + 78*ccent + 49*ceast) / 128;
    else                    # (assume) piecewise linear reconstruction for all other values of a0
      cc    := ccent;
      clc   := (cl + cc) / 2;
      clllc := (cl + clc) / 2;
      clccc := (cc + clc) / 2;
      crc   := (cr + cc) / 2;
      crccc := (cc + crc) / 2;
      crrrc := (cr + crc) / 2;
    end if;
    aux := ( abs(cl) + 2*abs(clllc) + 2*abs(clc) + 2*abs(clccc) + 2*abs(cc)
           + 2*abs(crccc) + 2*abs(crc)+ 2*abs(crrrc) + abs(cr) ) / 16:
  end:
```

Test the procedure just defined [approximation, exact value, and relative error] for the piecewise linear scheme ($a_0 = 1/8$) and a piecewise linear function (above numerical integration is exact), and for the piecewise parabolic scheme ($a_0 = 1/6$) and a piecewise parabolic function (above numerical integration is not exact).

```
> [ amp(0,1,0,1/8), int(1+x, x=-1/2..0) + int(1-x, x=0..1/2), 1 - (int(1+x,x=-1/2..0) + int(1-x,x=0..1/2)) /
  amp(0,1,0,1/8) ];
  [ amp(0,1,0,1/6), int(3/4-x^2, x=-1/2..1/2), 1 - int(3/4-x^2, x=-1/2..1/2) / amp(0,1,0,1/6) ];
```

$$\left[\frac{3}{4}, \frac{3}{4}, 0\right]$$

$$\left[\frac{85}{128}, \frac{2}{3}, -\frac{1}{255}\right]$$

Compute the coordinate of the grid points for plot of amplitudes per finite volume, and the coordinate of the points for plot of numerical solution.
Put result in vectors in order to be able to plot them using command **pointplot** (trick to be able to plot things using command **plot** didn't work).
The index of a Maple vector starts at 1 => **add 1 to grid point index!!!**

```
> xamp := Vector(itot+2):  xsol := Vector(9*itot):
  xcoor := - Dx/2:
```

```
  xamp[1] := xcoor:
  for i from 1 to itot do
    xcoor := xcoor + Dx:
    xamp[i+1] := xcoor:
    xsol[9*i-8] := xcoor - Dx/2:  xsol[9*i-7] := xcoor - 3*Dx/8:  xsol[9*i-6] := xcoor - Dx/4:
    xsol[9*i-5] := xcoor - Dx/8:  xsol[9*i-4] := xcoor:           xsol[9*i-3] := xcoor + Dx/8:
    xsol[9*i-2] := xcoor + Dx/4:  xsol[9*i-1] := xcoor + 3*Dx/8:  xsol[9*i]   := xcoor + Dx/2:
  #   xsol[9*i-8] := xsol[9*i-8] + Dx/10:  xsol[9*i] := xsol[9*i] - Dx/10:
  end do:
  xamp[itot+2] := xcoor + Dx:
```

Split solution in physical part and spurious part.
Put result in vectors in order to be able to plot them using command **`pointplot`** (trick to be able to plot things using command **`plot`** didn't work).
The index of a vector starts from 1 => **add 1 to grid point index!!!**
NOTE: splitting assumes uniform conditions, i.e., constant wave number, hence a uniform grid.

```
> camp := Vector(itot+2):  cphysamp := Vector(itot+2):  cspuramp := Vector(itot+2):
  cre  := Vector(9*itot):  cphysre  := Vector(9*itot):  cspurre  := Vector(9*itot):
    cleft := c[-1] = cphy * exp(I*kDxph) + cspu * exp(I*kDxsp):  crght := c[0] = cphy + cspu:
    sol := solve( [cleft,crght], [cphy,cspu] ):
  cphys[0]  := op(2,op(1,op(1,sol))) * corr:  # cphys required later for determination of Dirichlet b.c. error => use index to store values!
  ccentspur := op(2,op(2,op(1,sol))) * corr:
  cphys[-1] := cphys[0]  * exp(I*kDxph):
  cwestspur := ccentspur * exp(I*kDxsp):
    cleft := c[0] = cphy * exp(I*kDxph) + cspu * exp(I*kDxsp):  crght := c[1] = cphy + cspu:
    sol := solve( [cleft,crght], [cphy,cspu] ):
  cphys[1]  := op(2,op(1,op(1,sol))) * corr:
  ceastspur := op(2,op(2,op(1,sol))) * corr:
  cphys[0]  := ( cphys[0]  + cphys[1]  * exp(I*kDxph) ) / 2;
  ccentspur := ( ccentspur + ceastspur * exp(I*kDxsp) ) / 2:
    cleft := c[1] = cphy * exp(I*kDxph) + cspu * exp(I*kDxsp):  crght := c[2] = cphy + cspu:
    sol := solve( [cleft,crght], [cphy,cspu] ):
  cphys[2]  := op(2,op(1,op(1,sol))) * corr:
  cnextspur := op(2,op(2,op(1,sol))) * corr:
  cphys[1]  := ( cphys[1]  + cphys[2]  * exp(I*kDxph) ) / 2;
  ceastspur := ( ceastspur + cnextspur * exp(I*kDxsp) ) / 2:
  camp[1]     := amp( c[-1]*corr, c[0]*corr, c[1]*corr, a0 );
```

```
cphysamp[1] := amp( cphys[-1],  cphys[0],  cphys[1],  a0 );
cspuramp[1] := amp( cwestspur,  ccentspur, ceastspur, a0 );
for i from 1 to itot do
  cwestspur := ccentspur;  ccentspur := ceastspur;  ceastspur := cnextspur;
    cleft := c[i+1] = cphy * exp(I*kDxph) + cspu * exp(I*kDxsp):  crght := c[i+2] = cphy + cspu:
    sol := solve( [cleft,crght], [cphy,cspu] );
  cphys[i+2] := op(2,op(1,op(1,sol))) * corr:
  cnextspur  := op(2,op(2,op(1,sol))) * corr:
  cphys[i+1] := ( cphys[i+1] + cphys[i+2] * exp(I*kDxph) ) / 2:
  ceastspur  := ( ceastspur  + cnextspur  * exp(I*kDxsp) ) / 2:
  camp[i+1] := amp( c[i-1]*corr, c[i]*corr, c[i+1]*corr, a0 );
  cre[9*i-8] := Re(cl):  cre[9*i-7] := Re(cllc):  cre[9*i-6] := Re(clc):  cre[9*i-5] := Re(clccc):
  cre[9*i-4] := Re(cc):  cre[9*i-3] := Re(crccc):  cre[9*i-2] := Re(crc):  cre[9*i-1] := Re(crrrc):  cre[9*
i] := Re(cr):
  cphysamp[i+1] := amp( cphys[i-1], cphys[i], cphys[i+1], a0 );
  cphysre[9*i-8] := Re(cl):  cphysre[9*i-7] := Re(cllc):  cphysre[9*i-6] := Re(clc):  cphysre[9*i-5] := Re
(clccc):
  cphysre[9*i-4] := Re(cc):  cphysre[9*i-3] := Re(crccc):  cphysre[9*i-2] := Re(crc):  cphysre[9*i-1] := Re
(crrrc):  cphysre[9*i] := Re(cr):
  cspuramp[i+1] := amp( cwestspur, ccentspur, ceastspur, a0 );
  cspurre[9*i-8] := Re(cl):  cspurre[9*i-7] := Re(cllc):  cspurre[9*i-6] := Re(clc):  cspurre[9*i-5] := Re
(clccc):
  cspurre[9*i-4] := Re(cc):  cspurre[9*i-3] := Re(crccc):  cspurre[9*i-2] := Re(crc):  cspurre[9*i-1] := Re
(crrrc):  cspurre[9*i] := Re(cr):
end do:
camp[itot+2]     := amp( c[itot]*corr, c[itot+1]*corr, c[itot+2]*corr, a0 );
cphysamp[itot+2] := amp( cphys[itot],  cphys[itot+1],  cphys[itot+1],  a0 );
cspuramp[itot+2] := amp( ccentspur, ceastspur, cnextspur, a0 );
xleft := (xamp[1] + xamp[2]) / 2;  xright := (xamp[itot+1] + xamp[itot+2]) / 2;
```

$$cphys_0 := 0.9804965775 + 0.7407639742\,\mathrm{I}$$

$$cphys_1 := 0.9668823750 - 0.7584479388\,\mathrm{I}$$

$$camp_1 := 0.9867541220$$

$$cphysamp_1 := 0.9878966140$$

$$cspuramp_1 := 0.001785039128$$

$$camp_{16} := 0.9884915196$$

$$cphysamp_{16} := 1.113541437$$

$$cspuramp_{16} := 0.001785039163$$

$$xleft := 0$$

$$xright := 20$$

```
> plotamp1  := pointplot( xamp, camp, connect=true, color=black, linestyle=solid );
  plotamp2  := pointplot( xamp, camp, symbol=solidcircle, color=black, linestyle=solid );
  plotre1   := pointplot( xsol, cre, connect=true, color=black, linestyle=solid );
  plotspurre1 := pointplot( xsol, cspurre, connect=true,    color=[red], thickness=4, linestyle=dot );  # ... or
  thickness=2, linestyle=dot
  ### plotspurre2 := pointplot( xsol, cspurre, symbol=solidbox, color=[red] );
  plotphysre1 := pointplot( xsol, cphysre, connect=true,        color=[green], thickness=4, linestyle=dot );
  ### plotphysre2 := pointplot( xsol, cphysre, symbol=soliddiamond, color=[green] );
  plotexact := plot( cos(-kDxexact/Dx*xx+phi-0.0), xx=xleft..xright, color=[blue], linestyle=dot );
```

$$plotamp1 := PLOT(\ldots)$$

$$plotamp2 := PLOT(\ldots)$$

$$plotre1 := PLOT(\ldots)$$

$$plotspurre1 := PLOT(\ldots)$$

$$plotphysre1 := PLOT(\ldots)$$

$$plotexact := PLOT(\ldots)$$

Discretization error, reflection coefficient, and Dirichlet boundary-condition error in total solution (=> error polluted by presence of spurious mode) and in physical mode only (=> clean error measure).
SEP'10: the applied solution normalization factor *corr* has been obtained by applying Dirichlet boundary condition, hence difference between amplitude of solution and 1 indicates Dirichlet boundary condition error. Be aware that this difference does *not* include any phase error information, i.e., does not measure the (probably very small) difference between the phase of the imposed periodic function and the phase at the boundary of the numerical solution.
SEP'10: the 'clean' error measure looks at the combined error of phase and amplitude, by taking the absolute value of the difference between the normalized numerical solution at the boundary as approximated by the discretization of the Dirichlet boundary condition (physical mode only) and 1. However, the same Dirichlet boundary condition discretization was used to obtain solution normalization factor *corr* obtained from the *total* solution, so the phase error is likely not to be measured accurately It is not so clear how to do that properly either; using another *corr* determined from the physical mode would cancel the phase error (not the amplitude error, because that one is obtained by comparing the amplitude at the boundary with the amplitude inside the domain). => Simply use difference between the normalized absolute value of the numerical solution at the boundary as approximated by the discretization of the Dirichlet boundary condition (physical mode only) and 1 instead? That one has the nice advantage that error is signed, indicating the under-/overapplication of the Dirichlet condition.

```
> #   c[itot+1] := exp( I*(phi-kDxph/2) );
  #   c[itot-1] := exp( I*(phi+kDxph*3/2) );  # just for checking (recomputed later)
  errphase   := kDxexact/kDxph - 1;  # relative error in phase
  Nphys      := evalf( 2*Pi/Re(kDxph) );  # number of grid cells per physical wave length
```

```
Nspur      := evalf( 2*Pi/Re(kDxsp) );   # number of grid cells per spurious wave mode
reflspur   := [ evalf( cspuramp[1] / cphysamp[1] ), evalf( cspuramp[itot] / cphysamp[itot] ) ];  # should be
twice the same value, CHECK
errampphys := [ cphysamp[1] - 1 , cphysamp[itot] - 1 ];  # should be twice the same value, CHECK
# errbcphys  := [ abs( ( (1+b1+b2+b3)/2 * cphys[0] + (1-b1-2*b2-3*b3)/2 * cphys[1] + (b2+3*b3)/2 * cphys[2]
#                     - b3/2 * cphys[3] ) * exp(I*phi) / cphysamp[1] - 1 ),  # should be twice the same value, CHECK
#                 abs( ( (1+b1+b2+b3)/2 * cphys[itot-2] + (1-b1-2*b2-3*b3)/2 * cphys[itot-1] + (b2+3*b3)/2 *
cphys[itot]
#                     - b3/2 * cphys[itot+1] ) * exp(I*(phi+kDxph*(itot-2))) / cphysamp[itot-1] - 1 )  ];
errbcphys  := [ abs(   (1+b1+b2+b3)/2 * cphys[0] + (1-b1-2*b2-3*b3)/2 * cphys[1] + (b2+3*b3)/2 * cphys[2]
                    - b3/2 * cphys[3] )              / cphysamp[1] - 1  ,  # should be twice the same value, CHECK
                abs(   (1+b1+b2+b3)/2 * cphys[itot-2] + (1-b1-2*b2-3*b3)/2 * cphys[itot-1] + (b2+3*b3)/2 *
cphys[itot]
                    - b3/2 * cphys[itot+1] )                                  / cphysamp[itot-1] - 1    ];
```

$$errphase := -0.0201718240$$

$$Nphys := 4.788343776$$

$$Nspur := 2.380722351$$

$$reflspur := [0.001806908843, 0.001806908737]$$

$$errampphys := [-0.0121033860, -0.0121033861]$$

$$errbcphys := [0.011963066, 0.011963066]$$

```
> display( [ plotexact, plotamp1, plotamp2, plotre1, plotspurre1, plotphysre1 ] ,
         view=[0..xright,-12/10..13/10],
#          view=[0..xright/3,7/10..11/10],
         symbolsize=11, thickness=2, labels=[" ",""], # labels=["\t\t\t\tx","ampl\n\n"], ### axis=
[tickmarks=[10, subticks=4]],
         title="                  BLACK: total wave, grid point position                  GREEN: right-going
wave                 RED: spurious 'reflected' wave                 BLUE: exact solution" );
```

BLACK: total wave, grid point position GREEN: right-going wave RED: spurious 'reflected' wave
BLUE: exact solution