

An Introduction to PALM: Numerics and boundary conditions

Zingst, July 9, 2003

Overview

① Introduction

⇒ finite differences

② Temporal discretization

⇒ Time-step-schemes

③ Spatial discretization

⇒ Advection-schemes

④ Boundary conditions

⑤ Ensuring incompressibility

Introduction

⇒ Starting point: The Navier-Stokes equation

momentum equation:

$$\frac{\partial \bar{u}_i}{\partial t} = -\frac{\partial(\bar{u}_j \bar{u}_i)}{\partial x_j} - \frac{1}{\rho_0} \frac{\partial \bar{p}^*}{\partial x_i} - (\varepsilon_{ijk} f_j \bar{u}_k - \varepsilon_{i3k} f_3 u_{kgeo}) + \frac{g}{\theta_{v00}} (\bar{\theta}_v - \theta_{v00}) \delta_{i3} - \frac{\partial \tau_{ij}}{\partial x_j}$$

liquid water potential temperature:

$$\frac{\partial \bar{\theta}_l}{\partial t} = -\frac{\partial(\bar{u}_i \bar{\theta}_l)}{\partial x_i} - \frac{\partial W_i}{\partial x_i} + S_{\text{rad}} + S_{\text{prec}}$$

total water content:

$$\frac{\partial \bar{q}}{\partial t} = -\frac{\partial(\bar{u}_i \bar{q})}{\partial x_i} - \frac{\partial H_i}{\partial x_i} + S_{\text{prec}}$$

equation of continuity:

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0$$

The **Navier-Stokes equations** are forming a set of coupled, non-linear partial differential equations

Objective: to find a solution (numerically) taken into account given initial and boundary conditions

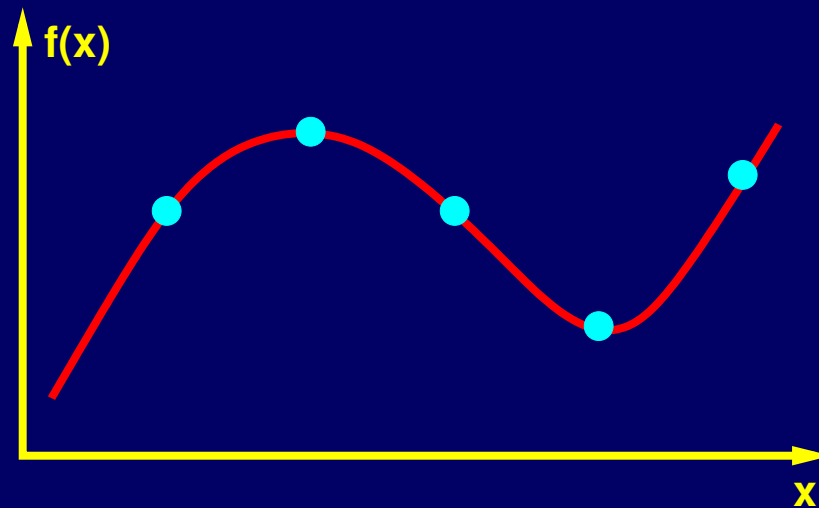
Problem: the equations cannot be solved directly
⇒ **Computers cannot integrate or differentiate!**

Introduction (I)

Possibility:

- ⇒ design methods to convert the original differential equations into a set of solvable algebraic equations
- ⇒ as a part of this task continuous functions must be represented by finite set of numbers
- ⇒ one basic strategie **⇒ the grid-point method**
 - ⇒ transformation of the spatial and temporal differential equations to difference equation

continuous functions **⇒** functions defined at discrete points



- ⇒ In PALM the finit difference method is used

Introduction – Finite Differences

⇒ finite differences are based on truncated Taylor's series:

$$\psi(x \pm \Delta x) = \psi(x) \pm \Delta x \frac{\partial \psi(x)}{\partial x} + \frac{(\Delta x)^2}{2} \frac{\partial^2 \psi(x)}{\partial x^2} \pm \dots \frac{(\Delta x)^n}{n!} \frac{\partial^n \psi}{\partial x^n} \dots$$

⇒ **Example:** finite-difference representation of derivatives:

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x + \Delta x) - \psi(x)}{\Delta x} + \mathcal{O}(\Delta x) \quad \text{forward differences}$$

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x) - \psi(x - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \quad \text{backward differences}$$

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x + \Delta x) - \psi(x - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x^2) \quad \text{central differences}$$

$$\left(\frac{\partial^2 \psi}{\partial x^2} \right) \approx \frac{\psi(x + \Delta x) - 2\psi(x) + \psi(x - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad \text{2nd derivatives}$$

Introduction – Finite Differences

⇒ finite differences are based on truncated Taylor's series:

$$\psi(x \pm \Delta x) = \psi(x) \pm \Delta x \frac{\partial \psi(x)}{\partial x} + \frac{(\Delta x)^2}{2} \frac{\partial^2 \psi(x)}{\partial x^2} \pm \dots \frac{(\Delta x)^n}{n!} \frac{\partial^n \psi}{\partial x^n} \dots$$

⇒ **Example:** finite-difference representation of derivatives:

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x + \Delta x) - \psi(x)}{\Delta x} + \mathcal{O}(\Delta x) \quad \text{forward differences}$$

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x) - \psi(x - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \quad \text{backward differences}$$

$$\left(\frac{\partial \psi}{\partial x} \right) \approx \frac{\psi(x + \Delta x) - \psi(x - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x^2) \quad \text{central differences}$$

$$\left(\frac{\partial^2 \psi}{\partial x^2} \right) \approx \frac{\psi(x + \Delta x) - 2\psi(x) + \psi(x - \Delta x))}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad \text{2nd derivatives}$$

truncation error



Discretization of the time-axis

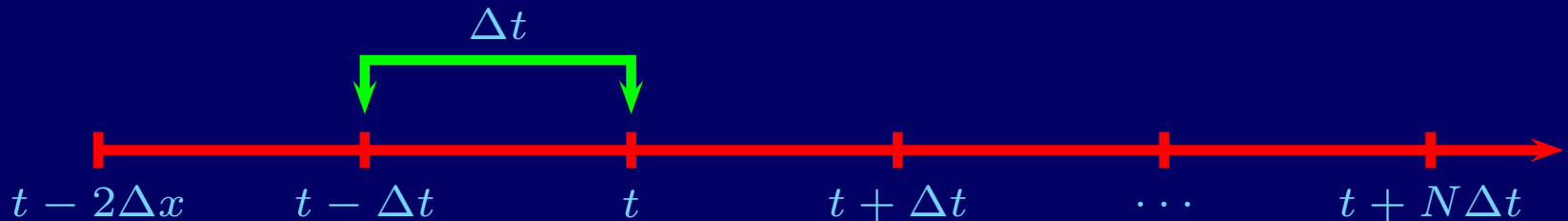
Objective:

⇒ Prediction of the temporal evolution of ψ :

$$\frac{\partial \psi(t)}{\partial t} = F(t)$$

(ψ any variable, F = forces affecting ψ)

⇒ for discretization of the above equation the time-axis is divided into (equidistant) segments with spacings Δt



⇒ based on the truncated Taylor series different time-step scheme could be defined

$$\psi(t \pm \Delta t) = \psi(t) \pm \Delta t \frac{\partial \psi(t)}{\partial t} + \frac{(\Delta t)^2}{2} \frac{\partial^2 \psi(t)}{\partial t^2} \pm \dots \frac{(\Delta t)^n}{n!} \frac{\partial^n \psi}{\partial t^n} \dots$$

Time-step schemes in PALM (I)

Euler-scheme:

- ☞ truncate the Taylor series at term of the first order
- ☞ forward differences:

$$\psi(t + \Delta t) = \psi(t) + \Delta t \cdot F(t)$$

- ☞ entirely unstable and diffusive

$$\frac{\partial \psi}{\partial t} \approx \frac{\psi(t + \Delta t) - \psi(t)}{\Delta t} + \mathcal{O}(\Delta t)$$

Leapfrog-scheme:

- ☞ scheme of second order accuracy
- ☞ central differences:

$$\psi(t + \Delta t) = \psi(t - \Delta t) + 2\Delta t \cdot F(t)$$

- ☞ unstable for large Δt ($\Delta t \stackrel{!}{\approx} 0.1 \cdot \Delta t_{\text{CFL}}$)
- ☞ "time-splitting" requires a weak time filter
(☞ Asselin Filter)

$$\frac{\partial \psi}{\partial t} \approx \frac{\psi(t + \Delta t) - \psi(t - \Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2)$$

Time-step schemes in PALM (II)

- in PALM the time-step schemes are written in compact notation

$$u_p(k, j, i) = (1 - at) * u_m(k, j, i) + at * u(k, j, i) + bt * dt_3d * tend(k, j, i)$$

- the switches at , bt distinguish between two time-step schemes:

$at = 1, bt = 1$: **Euler-scheme**

$$u_p(k, j, i) = u(k, j, i) + dt_3d * tend(k, j, i)$$

$at = 0, bt = 2$: **Leap-frog scheme**

$$u_p(k, j, i) = u_m(k, j, i) + 2 * dt_3d * tend(k, j, i)$$

NAMELIST-parameter	parameter kind	possible values
<code>timestep_scheme</code>	<code>&INIPAR</code>	'euler' 'leapfrog' (default) 'leapfrog+euler'

- the first time-step of each simulation is performed using the euler-scheme

Time-step schemes in PALM (III) – The future?

The Adams-Bashforth time-step scheme

$$\psi^{t+\Delta t} \approx \psi^t + \Delta t \left(\frac{3}{2} F^t - \frac{1}{2} F^{t-\Delta t} \right)$$

Advantages:

- no time-splitting appears
- compared to leap-frog method large time-steps could be used: Δt near Δt_{CFL}
- Derivation:

⇒ use truncated Taylor series up to term of 2nd order accuracy

$$\psi^{t+\Delta t} = \psi^t + \Delta t \frac{\partial \psi^t}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 \psi^t}{\partial t^2} + \mathcal{O}(\Delta t^3)$$

⇒ approximate the term of second derivative by backward differences:

$$\frac{\partial^2 \psi^t}{\partial t^2} = \frac{\partial}{\partial t} \left(\frac{\partial \psi^t}{\partial t} \right) \approx \frac{\partial}{\partial t} \left(\frac{\psi^t - \psi^{t-\Delta t}}{\Delta t} \right) = \frac{1}{\Delta t} \left(\frac{\partial \psi^t}{\partial t} - \frac{\partial \psi^{t-\Delta t}}{\partial t} \right)$$

Time-step schemes in PALM (VI) – The future?

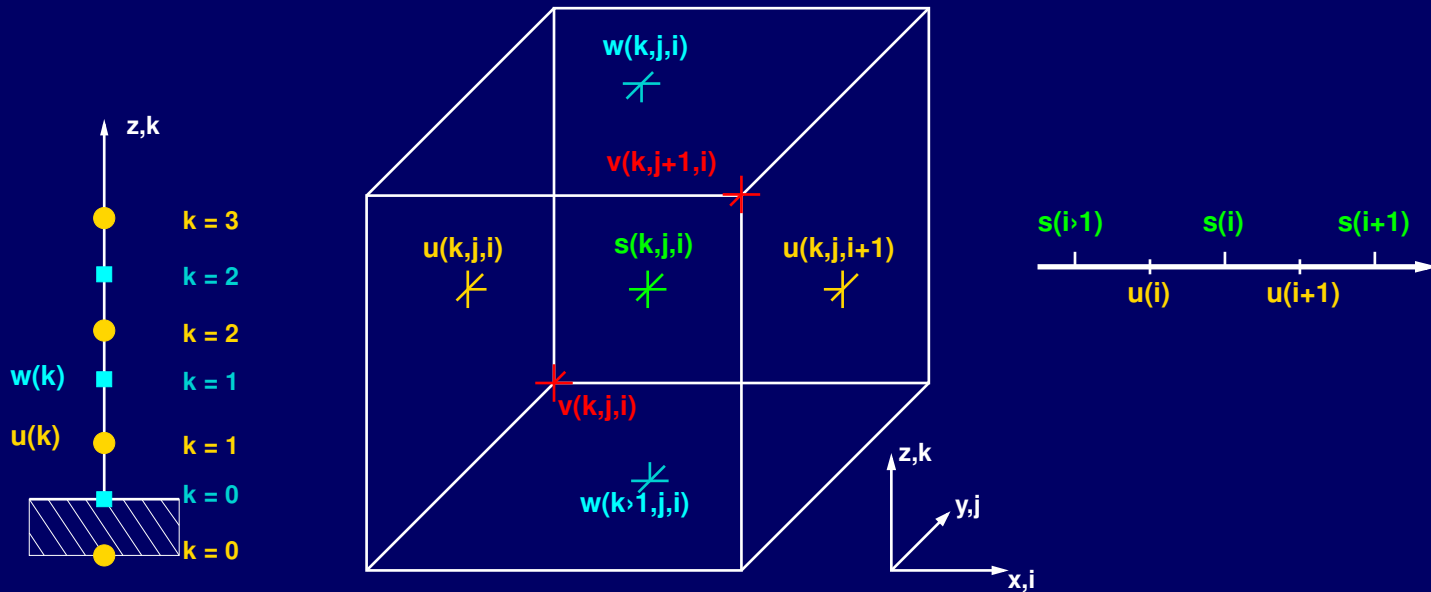
The Runge-Kutta time-step scheme (general form)

$$\begin{aligned}\tilde{\psi}^{t+\alpha} &= \psi^t + \alpha \Delta t F(\psi^t) \\ \psi^{t+\Delta t} &= \psi^t + \beta \Delta t F(\tilde{\psi}^{t+\alpha}) + (1 - \beta) \Delta t F(\psi^t)\end{aligned}$$

Advantages:

- ☞ no time-splitting appears
- ☞ large time-steps are possible

Spatial discretization – The numerical grid



- ⇒ in space the equations are discretized on an Arakawa-C grid
 - ⇒ regular staggered grid
- ⇒ scalar variables (e.g. θ , p^* , e , K_m , K_h) are defined in the cell centers
- ⇒ velocity components (u , v , w) are shifted by a half of the grid spacing
- ⇒ spacing are equidistant

Spatial discretization - advection terms (1)

⇒ the investigation will be focused on the discretization of the advection term:

$$\frac{\partial \psi}{\partial t} = -u_k \frac{\partial \psi}{\partial x_k} + S$$

(respectively: $-\frac{\partial(u_k \psi)}{\partial x_k}$ in case of an incompressible flow)

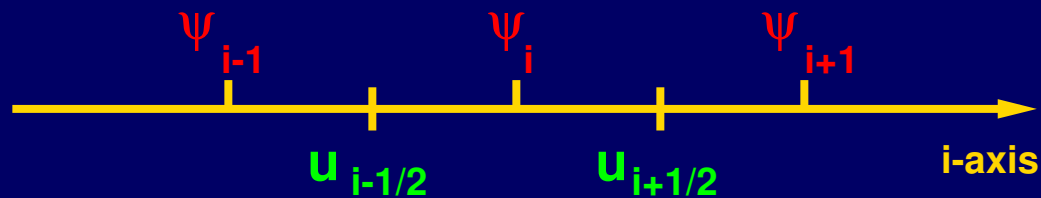
⇒ advection schemes used in PALM:

- scheme after Picsek-Williams
- Upstream-scheme
- Upstream-Spline-scheme
- Bott-Chlond-Schema (skalar variables only)

NAMELIST-parameter	parameter kind	possible values
<code>scalar_advec</code>	&INIPAR	'pw-scheme' (default) 'bc-scheme' 'ups-scheme'
<code>impulse_advec</code>	&INIPAR	'pw-scheme' (default) 'ubs-scheme'

Advection scheme of Piacsek und Williams

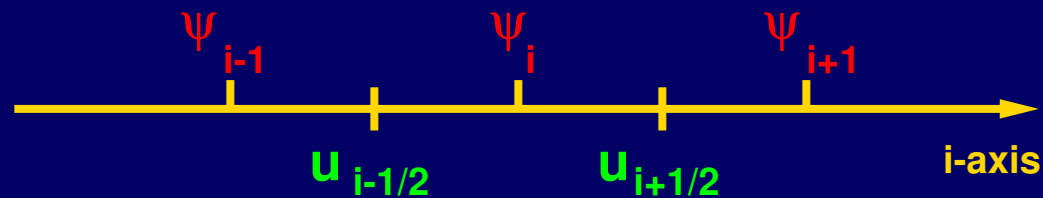
- ☞ default advection scheme in PALM
- ☞ scheme C3 after Piacsek and Williams (1970, J. Comput. Phys., 6, 392)
- ☞ scheme of 2nd order accuracy
- ☞ conservation of integrals of linear and quadratic quantities
- ☞ requires incompressibility \implies flux form of advection term
- ☞ low computational costs
- ☞ used in combination with the leap-frog time-step-scheme



$$\frac{\partial}{\partial x}(u \psi) \Big|_i = \frac{1}{2\Delta x} \left(u_{i+\frac{1}{2}} \psi_{i+1} - u_{i-\frac{1}{2}} \psi_{i-1} \right)$$

Advection scheme of Piacsek und Williams

- ☞ default advection scheme in PALM
- ☞ scheme C3 after Piacsek and Williams (1970, J. Comput. Phys., 6, 392)
- ☞ scheme of 2nd order accuracy
- ☞ conservation of integrals of linear and quadratic quantities
- ☞ requires incompressibility \implies flux form of advection term
- ☞ low computational costs
- ☞ used in combination with the leap-frog time-step-scheme



$$\left. \frac{\partial}{\partial x} (u \psi) \right|_i = \frac{1}{2\Delta x} \left(u_{i+\frac{1}{2}} \psi_{i+1} - u_{i-\frac{1}{2}} \psi_{i-1} \right)$$

Annotation: In case of velocity advection (e.g., $\psi = u$), $u_{i+\frac{1}{2}}$ and $u_{i-\frac{1}{2}}$ have to be obtained by linear interpolation

Advection scheme of Piacsek und Williams – Example

Example: A piece of code taken from `advec_u_pw.f90`

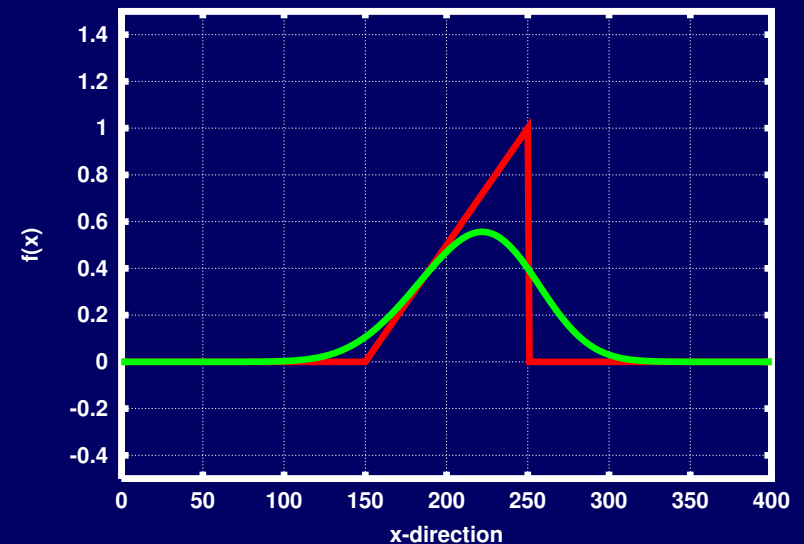
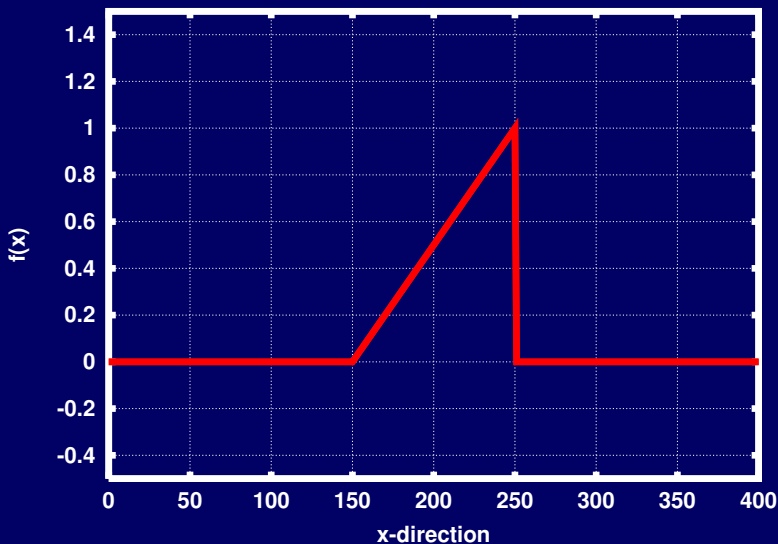
```
gu = 2.0 * u_gtrans      ! Galilei-transformation
gv = 2.0 * v_gtrans      ! Galilei-transformation
DO  i = nxl, nxr
  DO  j = nys, nyn
    DO  k = nzb+1, nzt
      tend(k,j,i) = tend(k,j,i) - 0.25 * (
        ( u(k,j,i+1) * ( u(k,j,i+1) + u(k,j,i) - gu )
        - u(k,j,i-1) * ( u(k,j,i) + u(k,j,i-1) - gu ) ) * ddx &
      + ( u(k,j+1,i) * ( v(k,j+1,i) + v(k,j+1,i-1) - gv )
        - u(k,j-1,i) * ( v(k,j,i) + v(k,j,i-1) - gv ) ) * ddy &
      + ( u(k+1,j,i) * ( w(k,j,i) + w(k,j,i-1) )
        - u(k-1,j,i) * ( w(k-1,j,i) + w(k-1,j,i-1) ) )
        * ddzw(k)
      )
    ENDDO
  ENDDO
ENDDO
```

Upstream-scheme (I)

- ☞ discretization of advection terms by forward/backward differences
- ☞ the space derivative is evaluated upwind from the concerning grid-point

$$u \frac{\partial \psi}{\partial x} \Big|_i = \begin{cases} u \frac{\psi_i - \psi_{i-1}}{\Delta x} & \text{if } u > 0.0 \\ u \frac{\psi_{i+1} - \psi_i}{\Delta x} & \text{if } u \leq 0.0 \end{cases}$$

- ☞ used in combination with the Euler time-step-scheme
- ☞ mass conserving, but large numerical diffusion



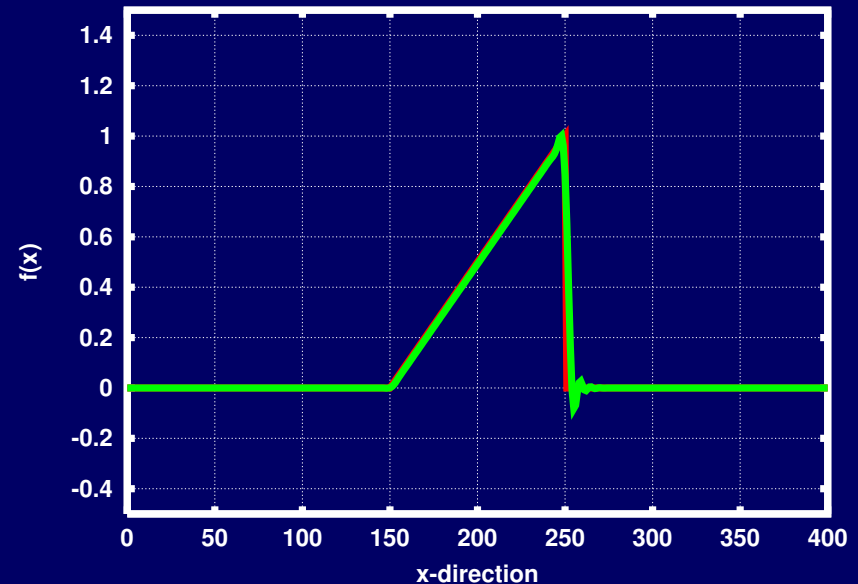
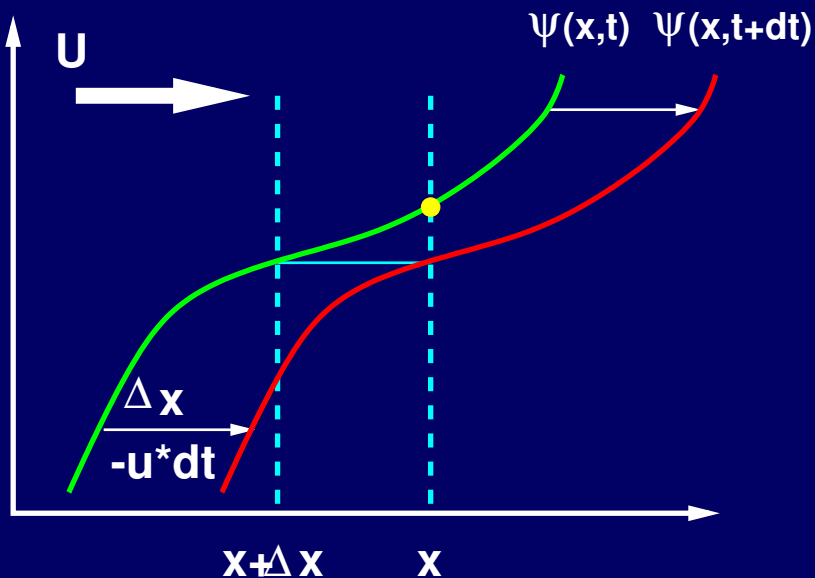
Upstream-spline scheme

⇒ Semi-Lagrangian interpolation scheme

⇒ approximate $\psi(x)$ by $\hat{\psi}(x)$ using cubic-spline functions

⇒ estimate the displacement distance $\Delta\tilde{x} = \begin{cases} -u \cdot \Delta t & \text{if } u > 0 \\ u \cdot \Delta t & \text{if } u < 0 \\ 0 & \text{else} \end{cases}$

⇒ calculate $\psi(x)^{t+\Delta t} = \hat{\psi}(x + \tilde{x})^t$



Advection scheme of Bott and Chlond

⇒ Bott-Chlond scheme is based on the finite difference presentation of the advection equation:

$$\frac{\partial \psi}{\partial t} = -\frac{\partial}{\partial x}(u \psi) \quad \Rightarrow \quad \psi_i^{t+\Delta t} = \psi_i^t - \frac{\Delta t}{\Delta x} \left(F_{i+1/2}^n - F_{i-1/2}^n \right)$$

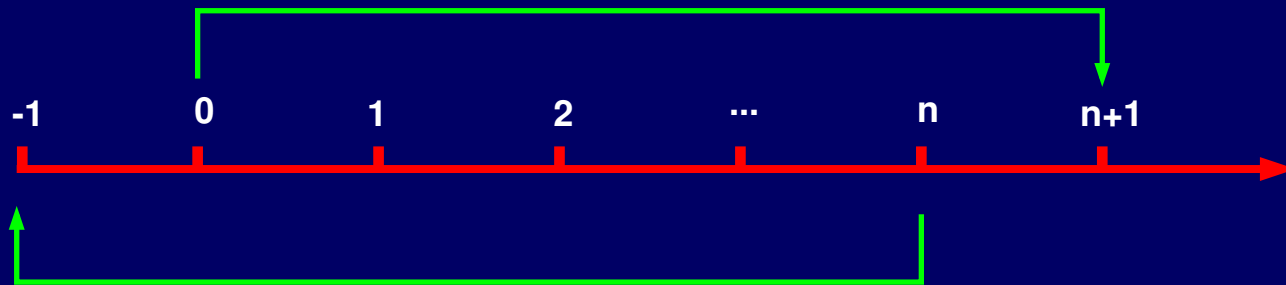
with F^n being the fluxes through the right and left boundary of the grid box

- ⇒ in contrast to the upstream method (assumes constant values of ψ in a grid box) ψ is fitted by polynomials
- ⇒ fitting is realized by using volume conserving polynomials in 'weak regions'
- ⇒ in regions of strong gradients exponential functions are used
- ⇒ yields to a conserving and monotonic advection scheme

Boundary conditions (I)

□ lateral boundary conditions

⇒ lateral boundary conditions are cyclic



$$\psi(-1) = \psi(n)$$

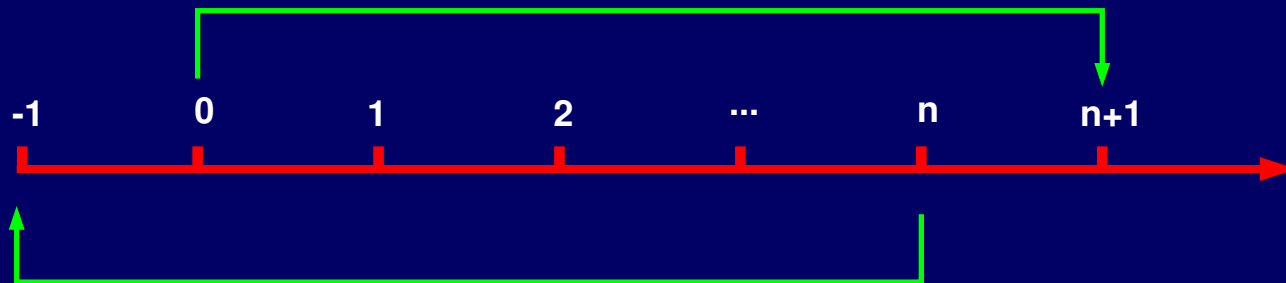
$$\psi(n+1) = \psi(0)$$

⇒ applied to all variables

Boundary conditions (I)

□ lateral boundary conditions

⇒ lateral boundary conditions are cyclic



$$\psi(-1) = \psi(n)$$

$$\psi(n+1) = \psi(0)$$

⇒ applied to all variables

Annotation: Currently non-cyclic boundary conditions will be implemented to PALM
(Diploma-thesis Micha Gryschka)

Boundary conditions (II)

□ bottom boundary condition:

⇒ describes a physical/real boundary (impermeable wall)

⇒ u, v, w : Dirichlet-conditions ($\psi = 0$)

due to the staggered grid: $w(0) = 0, \quad u(0) = -u(1), \quad v(0) = -v(1)$

⇒ e, p^*, K_m, K_h : Neumann-conditions ($\partial\psi/\partial z = 0$)

⇒ sub-grid turbulent fluxes (Dirichlet-conditions):

1. prescribed by the user
2. calculated via Monin-Obukhov theory

⇒ θ, q :

☞ prescribed fluxes: Neumann-conditions

☞ Monin-Obukhov theory: Dirichlet-conditions

Boundary conditions (II)

□ top boundary condition:

⇒ u, v, w : Dirichlet-conditions ($\psi = \psi_0$)

⇒ e : Neumann-conditions ($\partial\psi/\partial z = 0$)

⇒ θ, q : temporal constant gradients – ‘Neumann-conditions’
($\partial\psi/\partial z = \partial\psi/\partial z|_0$)

Ensuring incompressibility (I)

- ⇒ governing equations of PALM require incompressibility
- ⇒ incompressibility is reached by a predictor-corrector method
 - ① prediction of a preliminary momentum field by taking into account the pressure from the previous time step

$$\tilde{u}_i^{t+\Delta t} = u_i^{t-\Delta t} + 2\Delta t \cdot (\dots) + \frac{\Delta t}{\rho_0} \frac{\partial p^{*t}}{\partial x_i}$$

- ② the final momentum field has to guarantee mass conservation (fulfilling the equation of continuity):

$$\frac{\partial u_i^{t+\Delta t}}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\tilde{u}_i^{t+\Delta t} - \frac{\Delta t}{\rho_0} \frac{\partial p^{*t}}{\partial x_i} \right) \stackrel{!}{=} 0$$

- ③ resulting in a Poisson-equation for the perturbation pressure

$$\frac{\partial^2 p^*}{\partial x_i^2} = \frac{\rho_0}{\Delta t} \frac{\partial \tilde{u}_i^{t+\Delta t}}{\partial x_i}$$

Ensuring incompressibility (II)

☞ the Poisson equation form a set of linear equations:

$$\frac{\partial^2 p^*}{\partial x_i^2} = f \quad \Leftrightarrow \quad \mathbf{A} \cdot \vec{p}^* = \vec{f}$$

with $\vec{p}^*, \vec{f} \in R^n$, $\mathbf{A} = D - L - R \in R^{n \times n}$

☞ two general possibilities are provided by PALM to solve the Poisson-equation

- ① direct solving by means of a Fast-Fourier-transformation (FFT)
- ② iterative solving
 - (a) successive over-relaxation-solver (SOR-solver)
 - (b) multi-grid scheme

NAMELIST-parameter	parameter kind	possible values
<code>psolver</code>	<code>&INIPAR</code>	'poisfft' ('poisfft_hybrid', default) 'multigrid' 'sor'

Ensuring incompressibility (III) – FFT-solver

Procedure:

- ① discretization of the Poisson-equation by central differences
- ② 2D discrete FFT in both horizontal directions
- ③ solving the resulting tridiagonal set of linear equations
- ④ inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure
- ⑤ correction of the preliminary velocity field, e.g. $u_i^{t+\Delta t} = \tilde{u}_i^{t+\Delta t} - \Delta t \frac{p_i^* - p_{i-1}^*}{\Delta x}$

Annotations:

- ☞ highly effective solver
- ☞ due to non-locality of the FFT, transposition are required on distributed machines
- ☞ the use is linked to periodic boundary conditions and uniform grids

Ensuring incompressibility (III) – iterative solvers

Basic idea of iterative solvers:

Poisson equation is transformed to a fixed point problem: $\vec{p}^{k+1} = \mathbf{T} \cdot \vec{p}^k + \vec{c}^k$

Implementation:

depending of the structure of the matrix \mathbf{T} and vector \vec{c} different iterative solvers can be defined, e.g.: Jacobi-scheme (2D-uniform grid):

$$p_{i,j}^{k+1} = \frac{1}{4} \cdot (p_{i-1,j}^k + p_{i+1,j}^k + p_{i,j-1}^k + p_{i,j+1}^k - \Delta x^2 f(i, j, k))$$

Starting from a first guess the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned}\vec{p}^1 &= \mathbf{T} \cdot \vec{p}^0 + \vec{c}^0 \\ \vec{p}^2 &= \mathbf{T} \cdot \vec{p}^1 + \vec{c}^1 \\ &\vdots \\ \vec{p}^k &= \mathbf{T} \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= \mathbf{T} \cdot \vec{p}^k + \vec{c}^k\end{aligned}$$

- with each iteration step k the improved solution converges towards the exact solution, the residual is a measure for the error of intermediate solution: $\vec{r}^k = \vec{c}^k - \mathbf{T} \cdot \vec{p}^k$
- iterative scheme are 'local schemes' \Rightarrow exchange of information takes places only between neighboring grid-points
- the convergence depends on the number of grid-points (implicit on the grid-spacing)
 $\mu = 1 - \mathcal{O}(1/n)$ ($0 < \mu < 1$)
the larger μ the slower the residual will be reduces by an iteration-step

with regard to the number of grid-points typically used in PALM the convergence of all iterative solvers is uneconomical! (SOR-solver will be used for tests only)

Ensuring incompressibility (IV) – Multi-grid-method

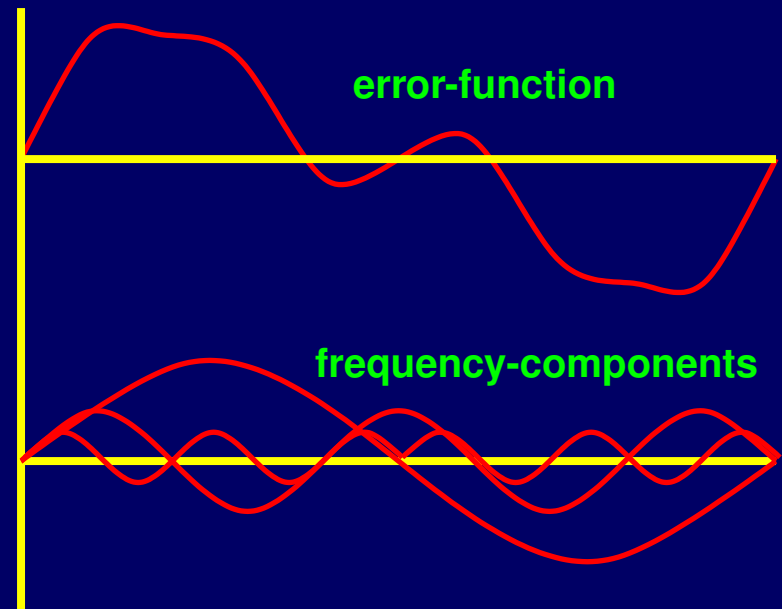
Motivation:

- ☞ standard iteration methods have (very) slow speed of convergence
- ☞ iterative methods show a frequency-dependent reduction of the residual \Rightarrow low frequencies are reduced slower than high-frequencies (due to locality)

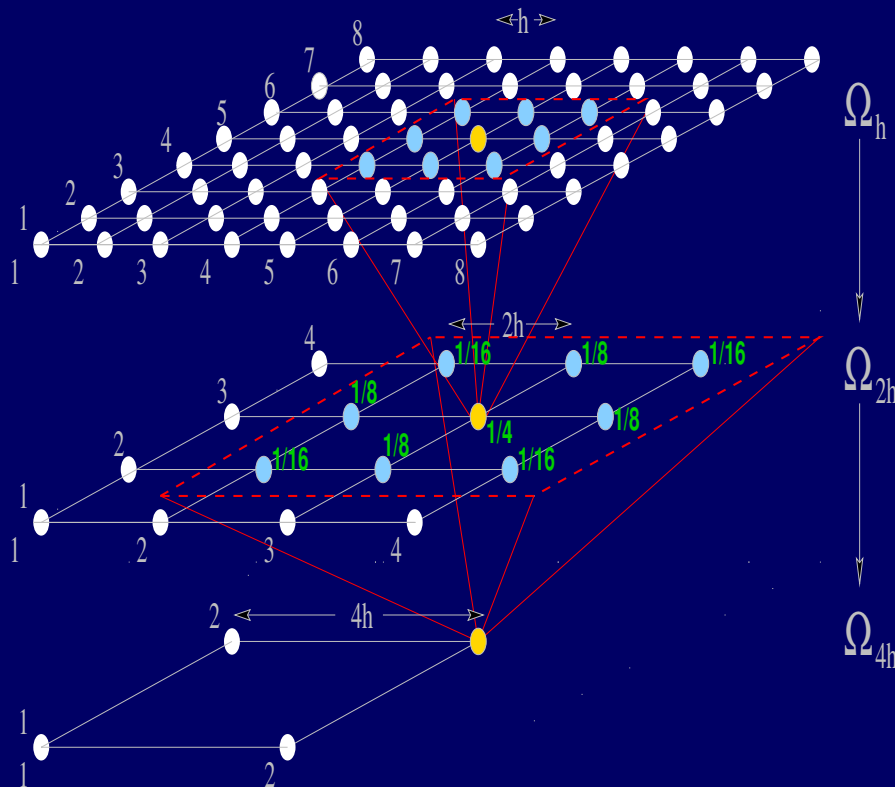
Idea:

reducing the errors-frequencies on grids with different grid-spacing

- ☞ errors of low frequency are reduced on coarse grids
- ☞ errors of high frequency are reduced on fine grids



Ensuring incompressibility (V) – Multi-grid-method



- ⇒ on each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations
- ⇒ the solution is transmitted the next coarser grid-level where it is used as the first guess to solve the fixed point problem
- ⇒ this procedure is performed up to the coarsest grid-level containing two grid-points in each direction
- ⇒ from the coarsest grid-level the procedure is passed in backward order to get the final solution