# Finite Volume Discretisation with Polyhedral Cell Support

**Hrvoje Jasak**

**hrvoje.jasak@fsb.hr**

**FSB, University of Zagreb, Croatia**

# What is Discretisation?

Numerical Discretisation Method

- Generic transport equation can very rarely be solved analytically: this is why we resort to numerical methods
- **Discretisation** is a process of representing the differential equation we wish to solve by a set of algebraic expressions of equivalent properties (typically a matrix)
- Two forms of discretisation operators. We shall use a divergence operator as an example.
  - **Calculus.** Given a vector field $\mathbf{u}$, produce a scalar field of $\nabla_{\bullet}\mathbf{u}$
  - **Method.** For a given divergence operator $\nabla_{\bullet}$, create a set of matrix coefficients that represent $\nabla_{\bullet}\mathbf{u}$ for any given $\mathbf{u}$
- The Calculus form can be easily obtained from the Method (by evaluating the expression), but this is not computationally efficient

# Discretised Transport Equation

Discretisation Methodology: **Polyhedral Finite Volume Method**

1. We shall assemble the discretisation on a **per-operator** basis: visit each operator in turn and describe a strategy for evaluating the term explicitly and discretising it
2. Describe space and time: a **computational mesh** for the spatial domain and **time-steps** covering the time interval
3. Postulate spatial and temporal variation of $\phi$ required for a discrete representation of field data
4. Integrate the operator over a cell
5. Use the spatial and temporal variation to interpret the operator in discrete terms

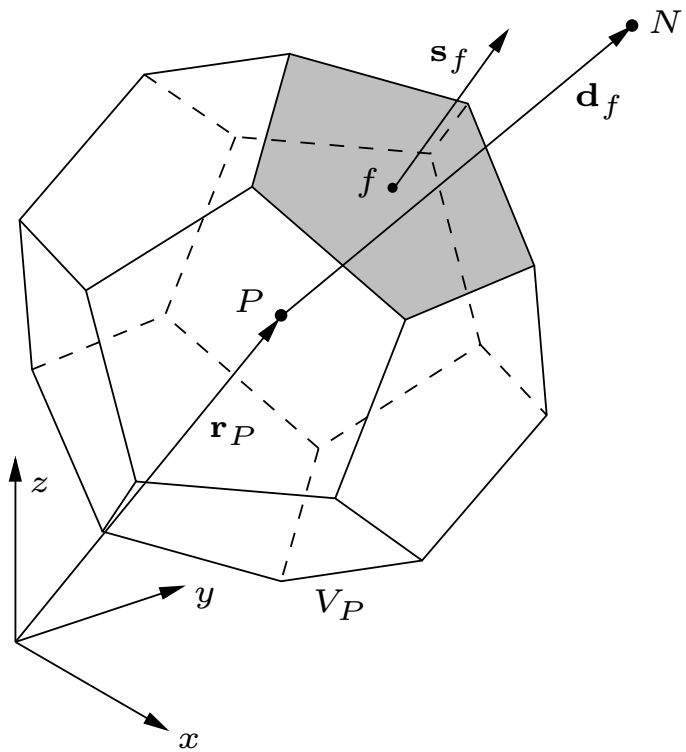
# Discretised Transport Equation

Representation of a Field Variable

- Equations we operate on work on fields: before we start, we need a discrete representation of the field
- Main solution variable will be stored in cell centroid: collocated cell-centred finite volume method. Boundary data will be stored on face centres of boundary faces
- For some purposes, *e.g.* face flux, different data is required – in this case it will be a field over all faces in the mesh
- Spatial variation can be used for interpolation in general: post-processing tools typically use point-based data.

# Nomenclature

Computational Cell




- This is a convex polyhedral cell boundary be a set of convex polygons
- Point $P$ is the computational point located at cell centroid $\mathbf{x}_P$. The definition of the centroid reads:

$$\int_{V_P} (\mathbf{x} - \mathbf{x}_P)\, dV = \mathbf{0}$$

# Nomenclature

Computational Cell

- Cell volume is denoted by $V_P$
- For the cell, there is one neighbouring cell across each face. Neighbour cell and cell centre will be marked with $N$.
- The face centre $f$ is defined in the equivalent manner, using the centroid rule:

$$\int_{S_f} (\mathbf{x} - \mathbf{x}_f)\, dS = \mathbf{0}$$

- Delta vector for the face $f$ is defined as

$$\mathbf{d}_f = \overline{PN}$$

- Face area vector $\mathbf{s}_f$ is a surface normal vector whose magnitude is equal to the area of the face. The face is numerically never flat, so the face centroid and area are calculated from the integrals.

$$\mathbf{s}_f = \int_{S_f} \mathbf{n}\, dS$$

# Nomenclature

Computational Cell

- The fact that the face centroid does not necessarily lay on the plane of the face is not worrying: we are dealing with surface-integrated quantities. However, we shall require the the cell centroid lays within the cell
- In practice, cell volume and face area calculated by decompositions into triangles and pyramids
- Types of faces in a mesh
  - **Internal face**, between two cells
  - **Boundary face**, adjacent to one cell only and pointing outwards of the computational domain
- When operating on a single cell, assume that all face area vectors $\mathbf{s}_f$ point outwards of cell $P$
- Discretisation is based on the integral form of the transport equation over each cell

$$\int_V \frac{\partial \phi}{\partial t}\, dV + \oint_S \phi\, (\mathbf{n}\bullet\mathbf{u})\, dS - \oint_S \gamma\, (\mathbf{n}\bullet\nabla\phi)\, dS = \int_V Q_v\, dV$$

# Spatial and Temporal Variation

Spatial Variation

- Postulating spatial variation of $\phi$: second order discretisation in space

$$\phi(\mathbf{x}) = \phi_P + (\mathbf{x} - \mathbf{x}_P) \bullet (\nabla\phi)_P$$

This expression is given for each individual cell. Here, $\phi_P = \phi(\mathbf{x}_P)$.

Temporal Variation

- Postulating linear variation in time: second order in time

$$\phi(t + \Delta t) = \phi^t + \Delta t \left(\frac{\partial \phi}{\partial t}\right)^t$$

where $\phi^t = \phi(t)$

Polyhedral Mesh Support

- In FVM, we have specified the “shape function” without reference to the actual cell shape (tetrahedron, prism, brick, wedge). The variation is always linear. Doing polyhedral Finite Volume should be straightforward!

# Volume and Surface Integrals

Evaluating Volume Integrals

$$\begin{aligned}\int_V \phi\, dV &= \int_V \left[\phi_P + (\mathbf{x}-\mathbf{x}_P)\bullet(\nabla\phi)_P\right] dV \\ &= \phi_P \int_V dV + (\nabla\phi)_P \bullet \int_V (\mathbf{x}-\mathbf{x}_P) dV \\ &= \phi_P V_P\end{aligned}$$

Evaluating Surface Integrals

- Surface integral splits into a sum over faces and evaluates in the same manner

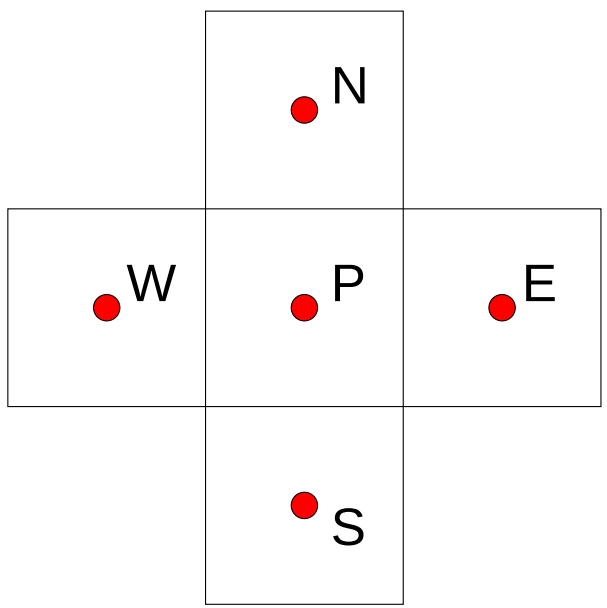
$$\begin{aligned}\oint_S \mathbf{n}\,\phi\, dS &= \sum_f \int_{S_f} \mathbf{n}\phi_f\, dS_f = \sum_f \int_{S_f} \mathbf{n}[\phi_f + (\mathbf{x}-\mathbf{x}_f)\bullet(\nabla\phi)_f]\, dS_f \\ &= \sum_f \mathbf{s}_f \phi_f\end{aligned}$$

- Assumption of linear variation of $\phi$ and selection of $P$ and $f$ in the centroid creates second-order discretisation

# Software Organisation

Cell and Face Addressing

- Assuming that $\phi_f$ depends on the values of $\phi$ in the two cells around the face, $P$ and $N$, let us attempt to calculate a surface integral for the complete mesh. Attention will be given on how the mesh structure influences the algorithm
- **Structured mesh**. Introducing compass notation: East, West, North, South
- The index of $E$, $W$, $N$ and $S$ can be calculated from the index of P: n + 1, n - 1, n + colDim, n - colDim



# Software Organisation

Structured and Body-Fitted Mesh

- Looping structure
  - Option 1: For all cells, visit East, West, North, South and sum up the values. Not too good: each face value calculated twice. Also, poor optimisation for vector computers we want to do a relatively short operation for lots and lots of cells
  - Option 2:
    * For all cells, do East face and add to $P$ and $E$
    * For all cells, do North face and add to $P$ and $N$

    Better, but stumbles on the boundary. Nasty tricks, like “zero-volume boundary cells” on the $W$ and $S$ side of the domain.
  - OK, I can do a box. How about implementing a boundary condition: separate discretisation on E, W, N and S boundary. Ugly and wasteful!

# Software Organisation

Block Structured Mesh

- Same kind of looping as above
- On connections between blocks, the connectivity is no longer “regular”, *e.g.* on the right side I can get a $N$ cell of another block
- Solution: repeat the code for discretisation and boundary conditions for all possible block-to-block connections
- Repeated code is very bad for your health: needs to be changed consistently, much more scope for errors, boring and difficult to keep running properly.

Tetrahedral Mesh

- In a tetrahedral mesh we cannot calculate the neighbouring indices because the mesh is irregular
- Cell-to-cell connectivity needs to be calculated during mesh generation or at the beginning of the simulation and **stored**
- Example: for each tetrahedron, store 4 indices of neighbour cells across 4 faces in order.

# Software Organisation

Unstructured Mesh

- We can treat a block structured mesh in the same manner: forget about blocks and store neighbour indices for each cell. Much better: no code duplication

Mixed cell types

- Re-use the unstructured mesh idea, but with holes: a tetrahedron only has 4 neighbours and a brick has got six
  - Option 1: For all cells, visit all neighbours. Woops: short loop inside a long loop AND all face values calculated twice
  - Option 2:
    * For all neighbours, up to max number of neighbours
    * For all cells
    * . . . do the work if there is a neighbour

    Works, but not too happy: I have to check if the neighbour is present

# Software Organisation

Polyhedral Mesh: Face Addressing

- Thinking about the above, all I want to do is to visit all cell faces and then all boundary faces. For internal face, do the operation and put the result into two cells around the face
- Orient face from $P$ to $N$: add to $P$ and subtract from $N$ (because the face area vector points the wrong way)
- Addressing slightly different: for each internal face, record the left and right (owner and neighbour) cell index. Owner will be the first one in the cell list
- Much cleaner, compact addressing, fast and efficient (some cache hit issues are hidden but we can work on that)
- Most importantly, it no longer matters how many faces there is in the cell: nothing special is required for polyhedral cells

# Gauss' Theorem

Gauss' Theorem in Finite Volume Discretisation

- Gauss' theorem is a tool we will use for handing the volume integrals of divergence and gradient operators
- Divergence form

$$\int_{V_P} \nabla \bullet \mathbf{a} \, dV = \oint_{\partial V_P} d\mathbf{s} \bullet \mathbf{a}$$

- Gradient form

$$\int_{V_P} \nabla \phi \, dV = \oint_{\partial V_P} d\mathbf{s} \, \phi$$

- Note how the face area vector operates from the same side as the gradient operator: fits with our definition of te gradient of for a vector field
- In the rest of the analysis, we shall look at the problem face by face. A diagram of a face is given below for 2-D. Working with vectors will ensure no changes are required when we need to switch from 2-D to 3-D.

# Matrix Assembly

From Discretisation to Linear System of Equations

- Assembling the terms from the discretisation method: Thus, the value of the solution in a point depends on the values around it: this is always the case. For each computational point, we will create an equation

$$a_P x_P + \sum_N a_N x_N = b$$

where N denotes the neighbourhood of a computational point

  - Every time $x_P$ depends on itself, add contribution into $a_P$
  - Every time $x_N$ depends on itself, add contribution into $a_N$
  - Other contributions into $b$

# Linear System of Equations

Nomenclature

- Equations form a **linear system** or a matrix

$$[A][x] = [b]$$

  where $[A]$ contain matrix coefficients, $[x]$ is the value of $x_P$ in all cells and $[b]$ is the right-hand-side

- $[A]$ is potentially very big: N cells $\times$ N cells
- This is a **square matrix**: the number of equations equals the number of unknowns
- . . . but very few coefficients are non-zero. The matrix connectivity is always local, potentially leading to storage savings if a good format can be found

# Implicit and Explicit Methods

Solution Advancement Method

- **Explicit method**: $x_P^n$ depends on the **old** neighbour values $x_N^o$
  - Visit each cell, and using available $x^o$ calculate

$$x_P^n = \frac{b - \sum_N a_N x_N^o}{a_P}$$

  - No additional information needed
  - Fast and efficient; however, poses the **Courant number limitation**: the information about boundary conditions is propagated very slowly and poses a limitation on the time-step size
- **Implicit method**: $x_P^n$ depends on the **new** neighbour values $x_N^n$

$$x_P^n = \frac{b - \sum_N a_N x_N^n}{a_P}$$

  - Each cell value of $x$ for the “new” level depends on others: all equations need to be solved simultaneously

# Operator Discretisation

Discretising Operators

- **The Story So Far...**
  - Split the space into cells and time into time steps
  - Assembled a discrete description of a continuous field variable
  - Postulated spatial and temporal variation of the solution for second-order discretisation
  - Generated expressions for evaluation of volume and surface integrals
- We shall now use this to assemble the discretisation of the differential operators
  1. Rate of change term
  2. Gradient operator
  3. Convection operator
  4. Diffusion operators
  5. Source and sink terms

# Temporal Derivative

First Derivative in Time

- Time derivative captures the rate-of-change of $\phi$. We only need to handle the volume integral.
- Defining time-step size $\Delta t$
- $t_{new} = t_{old} + \Delta t$, defining time levels $\phi^n$ and $\phi^o$

$$\phi^o = \phi(t = t_{old})$$

$$\phi^n = \phi(t = t_{new})$$

- Temporal derivative, first and second order approximation

$$\frac{\partial \phi}{\partial t} = \frac{\phi^n - \phi^o}{\Delta t}$$

$$\frac{\partial \phi}{\partial t} = \frac{\frac{3}{2}\phi^n - 2\phi^o + \frac{1}{2}\phi^{oo}}{\Delta t}$$

# Matrix Coefficients

First Derivative in Time

- Thus, with the volume integral:

$$\int_V \frac{\partial \phi}{\partial t}\, dV = \frac{\phi^n - \phi^o}{\Delta t} V_P$$

$$\int_V \frac{\partial \phi}{\partial t}\, dV = \frac{\frac{3}{2}\phi^n - 2\phi^o + \frac{1}{2}\phi^{oo}}{\Delta t} V_P$$

Temporal Derivative

- Calculus: given $\phi^n$, $\phi^o$ and $\Delta t$ create a field of the time derivative of $\phi$
- Method: matrix representation. Since $\frac{\partial \phi}{\partial t}$ in cell $P$ depends on $\phi_P$, the matrix will only have a diagonal contribution and a source
  - Diagonal coefficient: $a_P = \frac{V_P}{\Delta t}$
  - Source contribution: $r_P = \frac{V_P\, \phi^o}{\Delta t}$

# Gradient Calculation

Evaluating the Gradient

- How to evaluate a gradient of a given field: Gauss Theorem

$$\int_{V_P} \nabla\phi \, dV = \oint_{\partial V_P} d\mathbf{s} \, \phi$$

- Discretised form splits into a sum of face integrals

$$\oint_S \mathbf{n}\phi \, dS = \sum_f \mathbf{s}_f \phi_f$$

- It still remains to evaluate the face value of $\phi$. Consistently with second-order discretisation, we shall assume linear variation between P and N

$$\phi_f = f_x \phi_P + (1 - f_x)\phi_N$$

where $f_x = \overline{fN}/\overline{PN}$

- Gradient evaluation almost exclusively used as a calculus operation

# Gradient Calculation

Other Forms of Gradient: Least Square

- Consider cell centre $P$ and a cluster of points around it $N$. Fit a plane:

$$e_N = \phi_N - (\phi_P + \mathbf{d}_N \bullet (\nabla\phi)_P)$$

- Minimising the weighted error

$$e_P^2 = \sum_N (w_N e_N)^2 \quad \text{where} \quad w_N = \frac{1}{|\mathbf{d}_N|}$$

yields a second-order **least-square form of gradient**:

$$(\nabla\phi)_P = \sum_N w_N^2 \, \mathbf{G}^{-1} \bullet \mathbf{d}_N (\phi_N - \phi_P)$$

- $\mathbf{G}$ is a $3 \times 3$ symmetric matrix:

$$\mathbf{G} = \sum_N w_N^2 \, \mathbf{d}_N \mathbf{d}_N$$

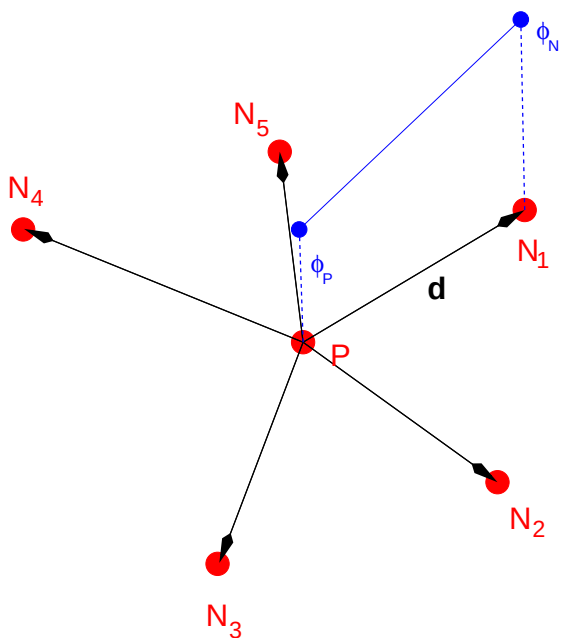
# Gradient Calculation

Other Forms of Gradient: Cell- and Face-Limited Gradient

- Gradient reconstruction may lead to local over- or under-shoots in reconstructed field:

$$min_N(\phi_N) \leq \phi_P + \mathbf{d}_N \bullet (\nabla\phi)_P \leq max_N(\phi_N)$$

- This is important for bounded variables, especially when gradients are used in further discretisation or coupling terms
- Solution: based on the gradient, calculate min and max neighbourhood value and apply gradient limiter to preserve bounds in cell centres



# Convection Operator

Convection Operator

- Convection term captures the transport by convective velocity
- Convection operator splits into a sum of face integrals (integral and differential form)

$$\oint_S \phi(\mathbf{n}\bullet\mathbf{u})dS = \int_V \nabla\bullet(\phi\mathbf{u})\,dV$$

- Integration follows the same path as before

$$\oint_S \phi(\mathbf{n}\bullet\mathbf{u})dS = \sum_f \phi_f(\mathbf{s}_f\bullet\mathbf{u}_f) = \sum_f \phi_f F$$

where $\phi_f$ is the face value of $\phi$ and

$$F = \mathbf{s}_f\bullet\mathbf{u}_f$$

is the **face flux**: measure of the flow through the face

- In order to close the system, we need a way of evaluating $\phi_f$ from the cell values $\phi_P$ and $\phi_N$: **face interpolation**

# Face Interpolation

Face Interpolation Scheme for Convection

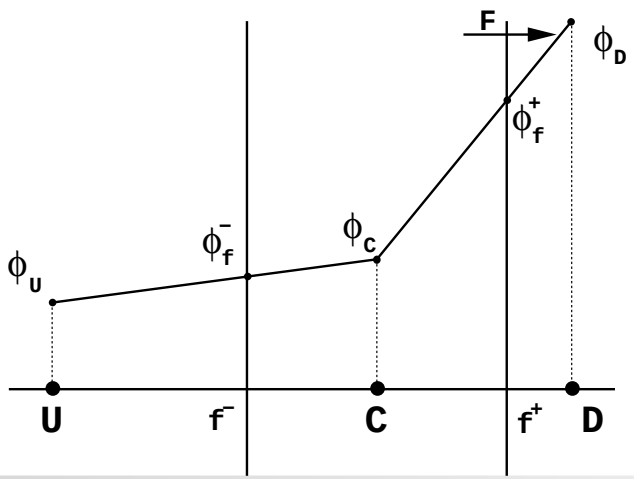
- Simplest face interpolation: **central differencing**. Second-order accurate, but causes oscillations

$$\phi_f = f_x \phi_P + (1 - f_x)\phi_N$$

- Upwind differencing: taking into account the **transportive property** of the term: information comes from upstream. No oscillations, but smears the solution

$$\phi_f = max(F, 0)\,\phi_P + min(F, 0)\,\phi_N$$

- There exists a large number of schemes, trying to achieve good accuracy without causing oscillations: *e.g.* TVD, and NVD families: $\phi_f = f(\phi_P, \phi_N, F, \ldots)$



# Matrix Coefficients

Convection Discretisation

- In the convection term, $\phi_f$ depends on the values of $\phi$ in two computational points: $P$ and $N$.
- Therefore, the solution in $P$ will depend on the solution in $N$ and vice versa, which means we've got an **off-diagonal coefficient** in the matrix. In the case of central differencing on a uniform mesh, a contribution for a face $f$ is
  - Diagonal coefficient: $a_P = \frac{1}{2}F$
  - Off-diagonal coefficient: $a_N = \frac{1}{2}F$
  - Source contribution: in our case, nothing. However, some other schemes may have additional (gradient-based) correction terms
  - Note that, in general the $P$-to-$N$ coefficient will be different from the $N$-to-$P$ coefficient: the matrix is asymmetric
- Upwind differencing
  - Diagonal coefficient: $a_P = max(F, 0)$
  - Off-diagonal coefficient: $a_N = min(F, 0)$

# Diffusion Operator

Diffusion Operator

- Diffusion term captures the gradient transport
- Integration same as before

$$\oint_S \gamma(\mathbf{n}\bullet\nabla\phi)dS = \sum_f \int_{S_f} \gamma(\mathbf{n}\bullet\nabla\phi)\, dS$$
$$= \sum_f \gamma_f\, \mathbf{s}_f\bullet(\nabla\phi)_f$$

- $\gamma_f$ evaluated from cell values using central differencing
- Evaluation of the face-normal gradient. If $\mathbf{s}$ and $\mathbf{d}_f = \overline{PN}$ are aligned, use difference across the face

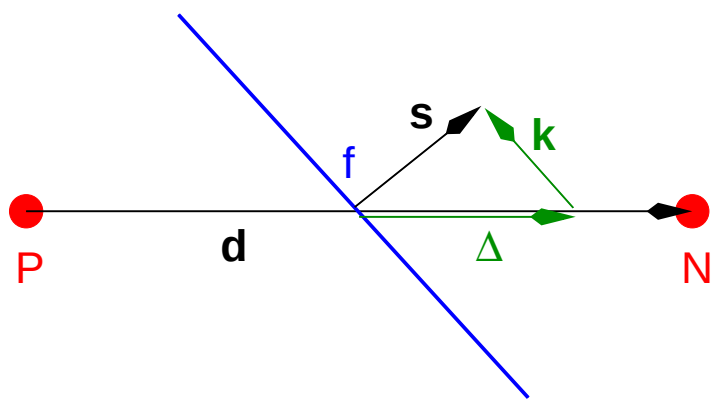
$$\mathbf{s}_f\bullet(\nabla\phi)_f = |\mathbf{s}_f|\frac{\phi_N - \phi_P}{|\mathbf{d}_f|}$$

- This is the component of the gradient in the direction of the $\mathbf{d}_f$ vector
- For non-orthogonal meshes, a correction term may be necessary

# Diffusion Operator

Matrix Coefficients

- For an orthogonal mesh, a contribution for a face $f$ is
  - Diagonal value: $a_P = -\gamma_f \frac{|\mathbf{s}_f|}{|\mathbf{d}_f|}$
  - Off-diagonal value: $a_N = \gamma_f \frac{|\mathbf{s}_f|}{|\mathbf{d}_f|}$
  - Source contribution: for orthogonal meshes, nothing. Non-orthogonal correction will produce a source
- The $P$-to-$N$ and $N$-to-$P$ coefficients are identical: **symmetric matrix**. This is an important characteristic of the diffusion operator
- For **non-orthogonal meshes**, a correction is added to compensate for the angle between the face area and $\overline{PN}$ vectors



# Diffusion Operator

Limiting Non-Orthogonal Correction in a Laplacian

- Decomposition of face gradient into “orthogonal component” and “non-orthogonal correction” depends on mesh quality: mesh non-orthogonality is measured from $\overline{PN}$ and $\mathbf{s}_f$
- Mathematically, a Laplacian is a perfect operator: smooth, bounded, self-adjoint. Its discretisation yields a symmetric matrix
- In contrast, non-orthogonal correction is explicit, unbounded and unsigned
- Limited non-orthogonal correction: explicit part clipped to be smaller than its implicit counterpart, base on the current solution

$$\lambda \frac{|\mathbf{s}_f|}{|\mathbf{d}_f|}(\phi_N - \phi_P) > \mathbf{k}_f \bullet \nabla(\phi)_f$$

where $\lambda$ is the limiter value

# Source and Sink Terms

Source and Sinks

- Source and sink terms are local in nature

$$\int_V q_v \, dV = q_v V_P$$

- In general, $q_v$ may be a function of space and time, the solution itself, other variables and can be quite complex. In complex physics cases, the source term can carry the main interaction in the system. Example: complex chemistry mechanisms. We shall for the moment consider only a simple case.
- Typically, linearisation with respect to $\phi$ is performed to promote stability and boundedness

$$q_v(\phi) = q_u + q_d \phi$$

where $q_d = \frac{\partial q_v(\phi)}{\partial \phi}$ and for cases where $q_d < 0$ (sink), treated separately

Matrix Coefficients

- Source and sink terms do not depend on the neighbourhood
  - Diagonal value created for $q_d < 0$: “boosting diagonal dominance”
  - Explicit source contribution: $q_u$

# Numerical Boundary Conditions

Implementation of Boundary Conditions

- Boundary conditions will contribute the the discretisation through the prescribed boundary behaviour
- Boundary condition is specified for the whole equation
- . . . but we will study them term by term to make the problem simpler

# Dirichlet Condition

Dirichlet Condition: Fixed Boundary Value

- Boundary condition specifies $\phi_f = \phi_b$
- Convection term: fixed contribution $F\,\phi_b$. Source contribution only
- Diffusion term: need to evaluate the near-boundary gradient

$$\mathbf{n}_\bullet(\nabla\phi)_b = \frac{\phi_b - \phi_P}{|\mathbf{d}_b|}$$

  This produces a source and a diagonal contribution

- What about source, sink, rate of change?

# Neumann Condition

Neumann and Gradient Condition

- Boundary condition specifies the near-wall gradient $\mathbf{n}_\bullet(\nabla\phi)_b = g_b$
- Convection term: evaluate the boundary value of $\phi$ from the internal value and the known gradient

$$\phi_b = \phi_P + \mathbf{d}_b{}_\bullet(\nabla\phi)_b = \phi_P + |\mathbf{d}_b|g_b$$

  Use the evaluated boundary value as the face value. This creates a source and a diagonal contribution

- Diffusion term: boundary-normal $g_b$ gradient can be used directly. Source contribution only

Mixed Condition

- Combination of the above
- Very easy: $\alpha$ times Dirichlet plus $(1-\alpha)$ times Neumann

Geometric and Coupled Conditions

- Symmetry plane condition is enforces using the mirror-image of internal solution
- Cyclic and periodic boundary conditions couple near-boundary cells to cells on another boundary

# Time Advancement

Advancing the Solution in Time

- Two basic types of time advancement: Implicit and explicit schemes. Properties of the algorithm critically depend on this choice, but both are useful under given circumstances
- There is a number of methods, with slightly different properties, *e.g.* fractional step methods,
- Temporal accuracy depends on the choice of scheme and time step size
- Steady-state simulations
  - If equations are linear, this can be solved in one go! (provided the discretisation is linear as well!)
  - For non-linear equations or special discretisation practices, **relaxation methods** are used, which show characteristics of time integration (we are free to re-define the meaning of time

# Time Advancement

Explicit Schemes

- The algorithm uses the calculus approach, sometimes said to operate on residuals
- In other words, the expressions are evaluated using the currently available $\phi$ and the new $\phi$ is obtained from the time term
- **Courant number limit** is the major limitation of explicit methods: information can only propagate at the order of cell size; otherwise the algorithm is unstable
- Quick and efficient, no additional storage
- Very bad for elliptic behaviour

Implicit Schemes

- The algorithm is based on the method: each term is expressed in matrix form and the resulting linear system is solved
- A new solution takes into account the new values in the complete domain: ideal for elliptic problems
- Implicitness removed the Courant number limitation: we can take larger time-steps
- Substantial additional storage: matrix coefficients!

# Equation Discretisation

Assembling Equations

- The equation we are trying to solve is simply a collection of terms: therefore, assemble the contribution from
- **Initial condition**. Specifies the initial distribution of $\phi$
- . . . and we are ready to look at examples!

Examples: Convection Differencing Schemes

- Testing differencing schemes on standard profiles
- Simple second-order discretisation: upwind differencing, central differencing, blended differencing, NVD schemes
- First-order scheme: Upwind differencing. Take into account the transport direction
- Exercise: now does all this relate to the discretisation of the Euler equation described in the previous lectures?

Examples: Stability and Boundedness

- Positive and negative diffusion coefficient
- Temporal discretisation: first and second-order, implicit or explicit discretisation