Introduction to Parallel Programming for Multicore/Manycore Clusters

Part A1: FVM Code (PCG)

Kengo Nakajima
Information Technology Center
The University of Tokyo

Date	Hour	Content
	09:10-10:00	Introduction
February 14 (Tue), 2023	10:10-11:00	Finite Volume Method (FVM) (1/4)
	11:10-12:00	Finite Volume Method (FVM) (2/4)
	13:10-14:00	Finite Volume Method (FVM) (3/4)
	14:10-15:00	Finite Volume Method (FVM) (4/4)
	15:10-16:00	Introduction to OpenMP (1/4)
	16:10-17:00	Login to Odyssey
	09:10-10:00	Introduction to OpenMP (2/4)
	10:10-11:00	Introduction to OpenMP (3/4)
	11:10-12:00	Introduction to OpenMP (4/4)
February 15 (Wed), 2023	13:10-14:00	ICCG Method (1/3)
	14:10-15:00	ICCG Method (2/3)
	15:10-16:00	ICCG Method (3/3)
	16:10-17:00	Reordering (1/4)
February 16 (Thu), 2023	09:10-10:00	Reordering (2/4)
	10:10-11:00	Reordering (3/4)
	11:10-12:00	Reordering (4/4)
	13:10-14:00	Parallel FVM using OpenMP (1/4)
	14:10-15:00	Parallel FVM using OpenMP (2/4)
	15:10-16:00	Parallel FVM using OpenMP (3/4)
	16:10-17:00	Parallel FVM using OpenMP (4/4)

Part-A
PCG Solver
Point Jacobi Preconditioning
(Simple, Easy)

Part-B
ICCG Solver
Incomplete Cholesky Preconditioning
(More Complicated, Difficult)

Target of Part-A

- Material: PCG solver for sparse matrices derived from FVM applications (Finite Volume Method).
  - Point Jacobi/Diagonal Scaling Preconditioner
- Parallelization on a single node of Wisteria/BDEC-01 (Odyssey) using OpenMP
  - Data Placement
  - Methods for Matrix Storage

- Keywords
  - Finite Volume Method (FVM)
  - Sparse Matrices
  - PCG Method: **P**reconditioned **C**onjugate **G**radient

Files on PC

Files on WEB:
http://nkl.cc.u-tokyo.ac.jp/files/fvm.tar

```
>$ cd
>$ tar xvf fvm.tar
>$ cd fvm
```

Please confirm that following directories are created:

```
src-c, src-f, run
```

Call the fvm directory <$P-FVM>

PC

Odyssey

Paraview for Visualization

http://www.paraview.org/

Free Software
Windows, iOS, Unix/Linux

http://nkl.cc.u-tokyo.ac.jp/NTU2023W/ParaView.pdf

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- PCG Solver for Poisson's Equations
  - How to run
    - Data Structure
  - Program
    - Initialization
    - Coefficient Matrices
    - PCG

Target Application

- 3D Poisson Equation/Poisson's Equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

- Finite Volume Method (FVM)
  - Arbitrary Shape Meshes, Cell-Centered
  - "Direct" Finite Difference Method
- Boundary Conditions (B.C.) etc.
  - Dirichlet B.C., Volume Flux
- Preconditioned Iterative Solvers
  - Conjugate Gradient + Preconditioner

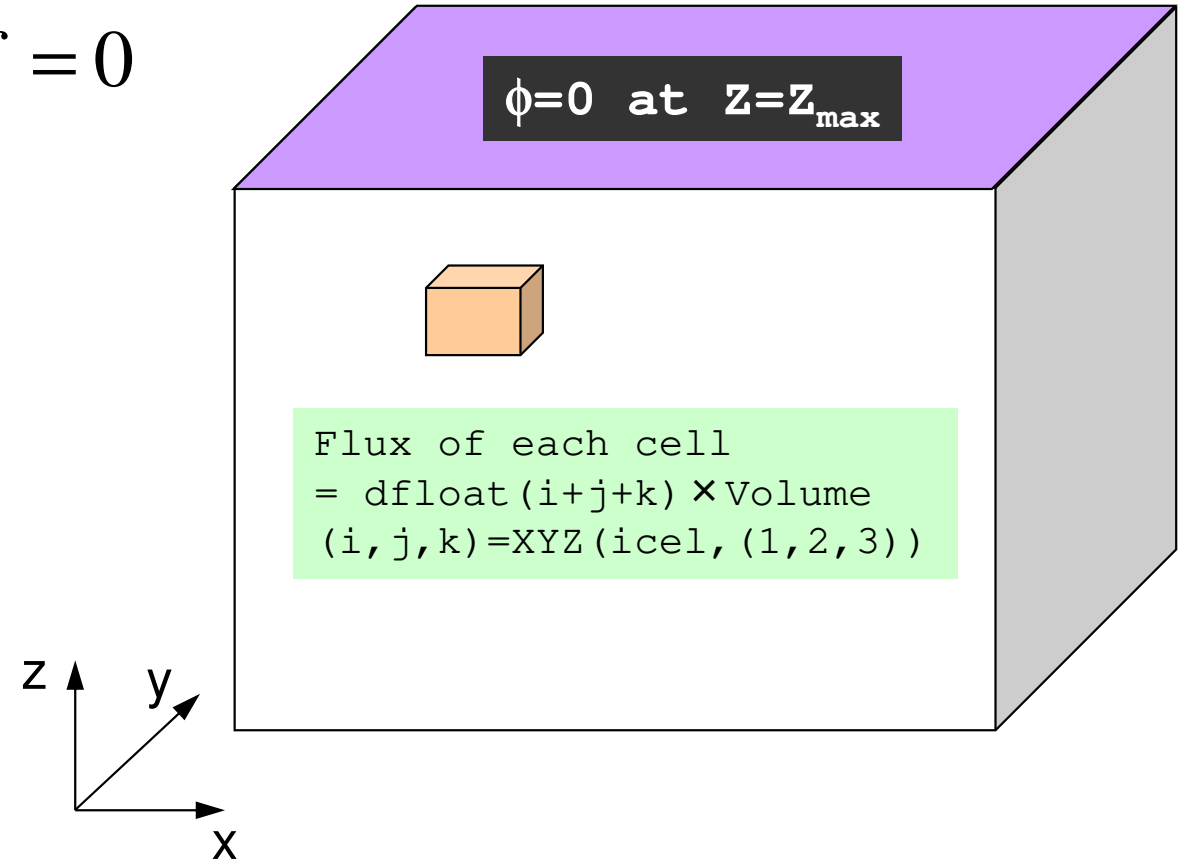
Target Problem: Variables are defined at cell-center's

Poisson Equation/ Poisson's Equation

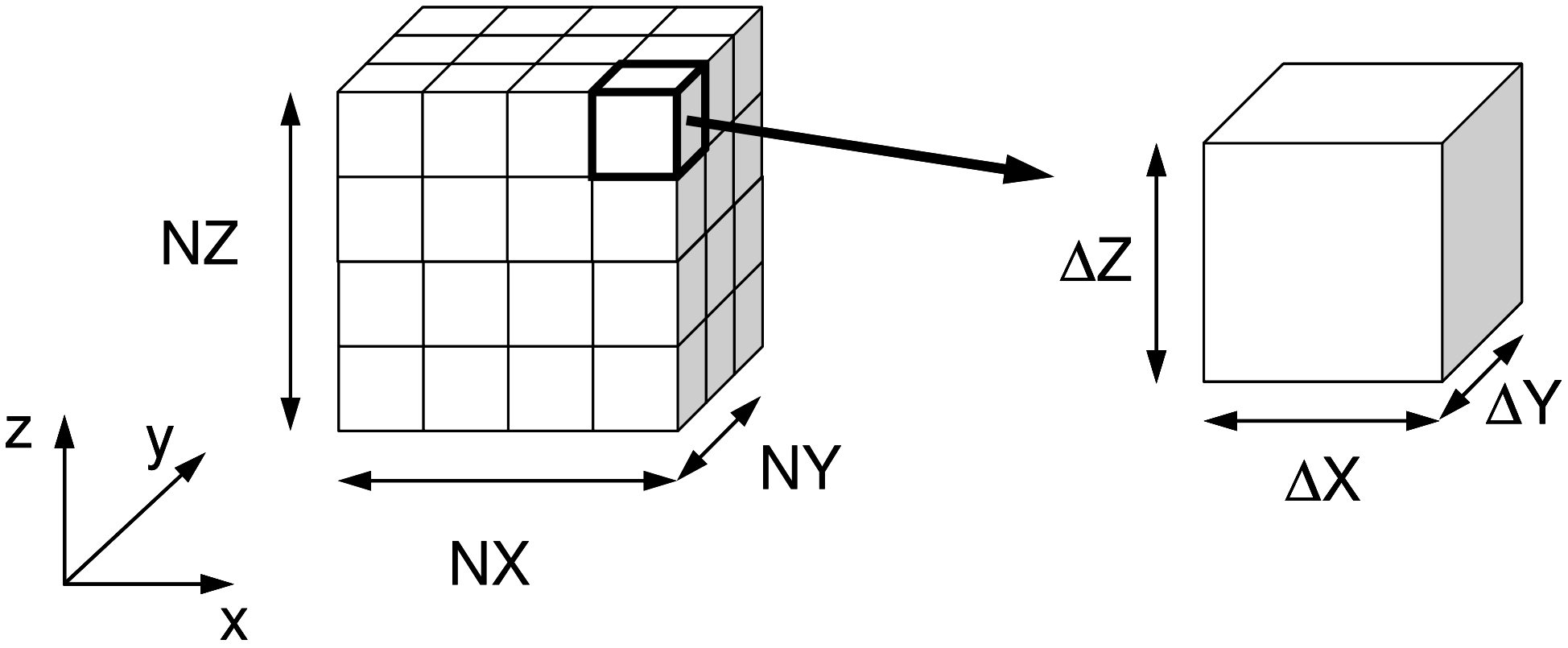
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

Boundary Conditions (B.C.) etc.

- Volume Flux
- $\phi = 0 @ Z = Z_{max}$



3D Structured Mesh
Internal data structure is “unstructured”



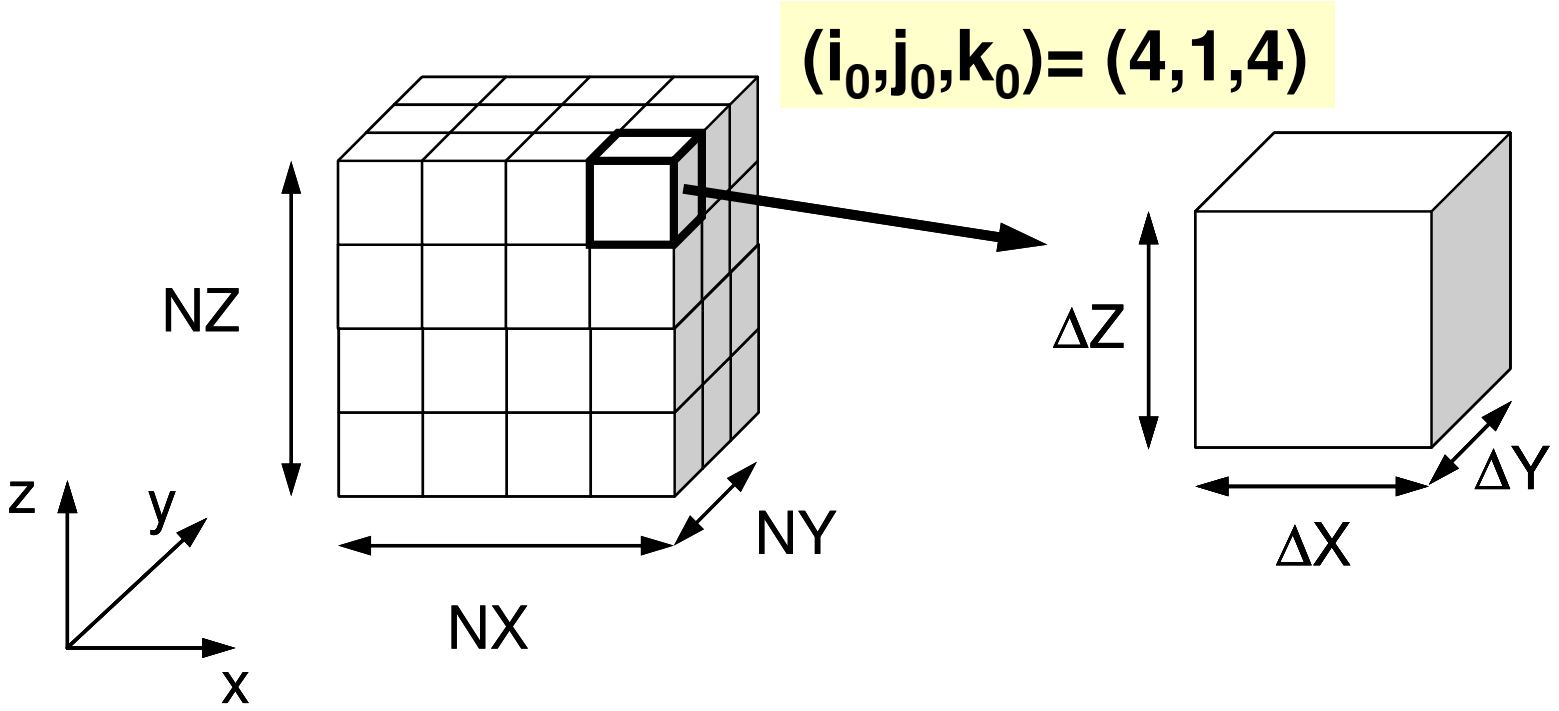
Volume Flux f

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

$$f = dfloat(i_0 + j_0 + k_0)$$

$$i_0 = XYZ(icel,1),$$
$$j_0 = XYZ(icel,2),$$
$$k_0 = XYZ(icel,3)$$

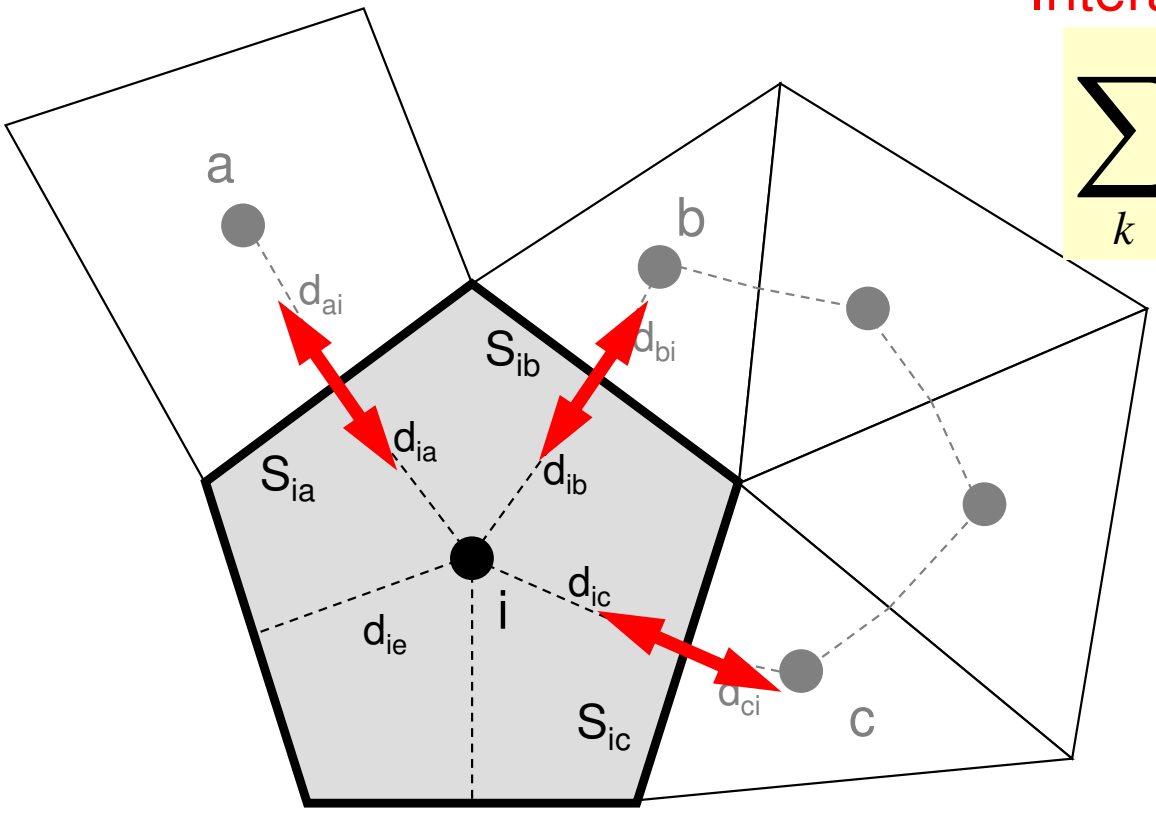
$XYZ(icel, k)$(k=1,2,3)
Index for location of finite-difference mesh in X-/Y-/Z-axis.



Poisson Equation by Finite Volume Method (FVM)

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

Conservation of Fluxes through Surfaces



Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

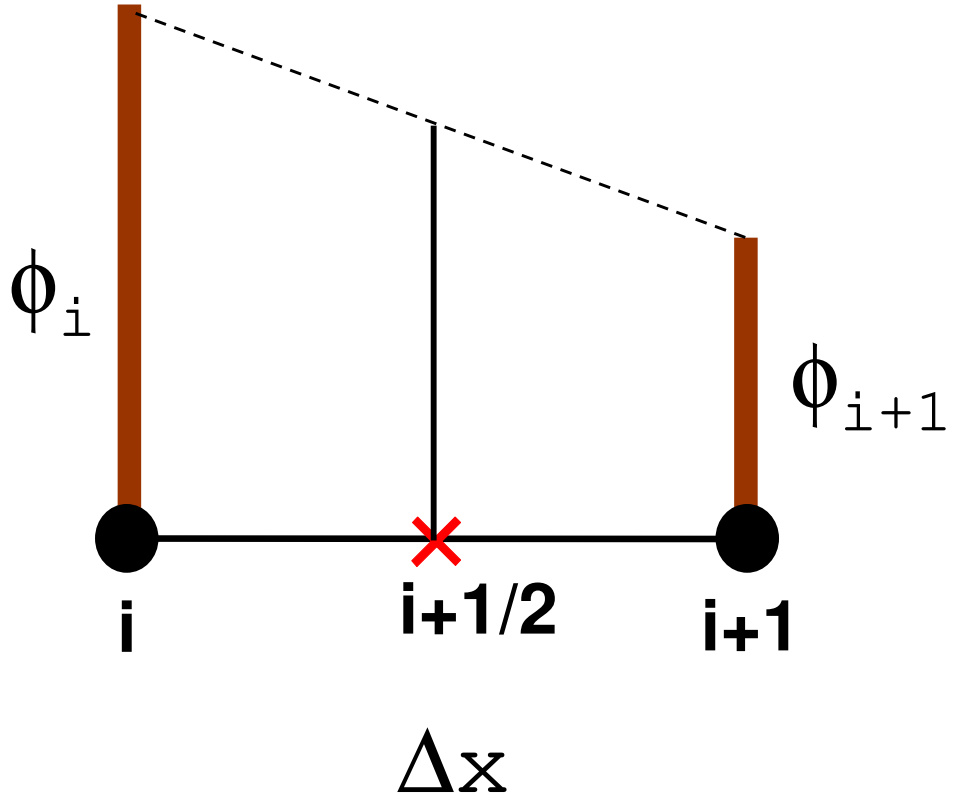
Volume Flux

V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Finite Difference Method (FDM)
(有限)差分法:巨視的微分
macroscopic differentiation

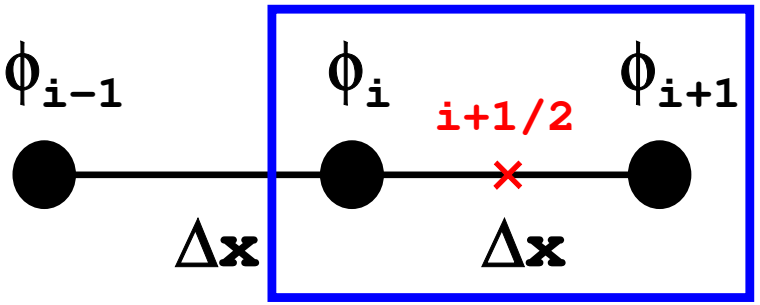
$$\left(\frac{d\phi}{dx}\right)_{i+1/2} \approx \frac{\phi_{i+1}-\phi_i}{\Delta x}$$

$$\left(\frac{d\phi}{dx}\right)_{i+1/2} = \lim_{\Delta x \to 0} \frac{\phi_{i+1}-\phi_i}{\Delta x}$$



2nd Order Differentiation in FDM Taylor Series Expansion

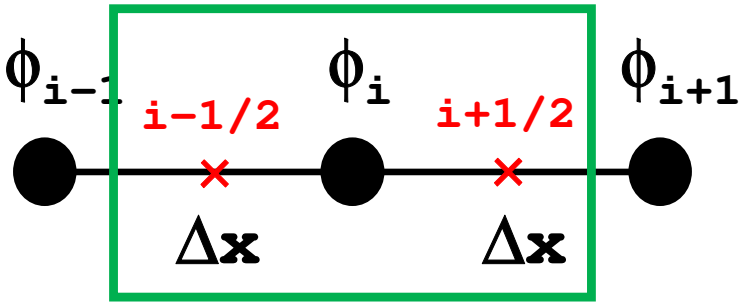
- **Approximate Derivative at × (center of *i* and *i+1*)**



$$\left(\frac{d\phi}{dx}\right)_{i+1/2} \approx \frac{\phi_{i+1}-\phi_i}{\Delta x}$$

Δx→0: Real Derivative

- **2nd-Order Diff. at *i***

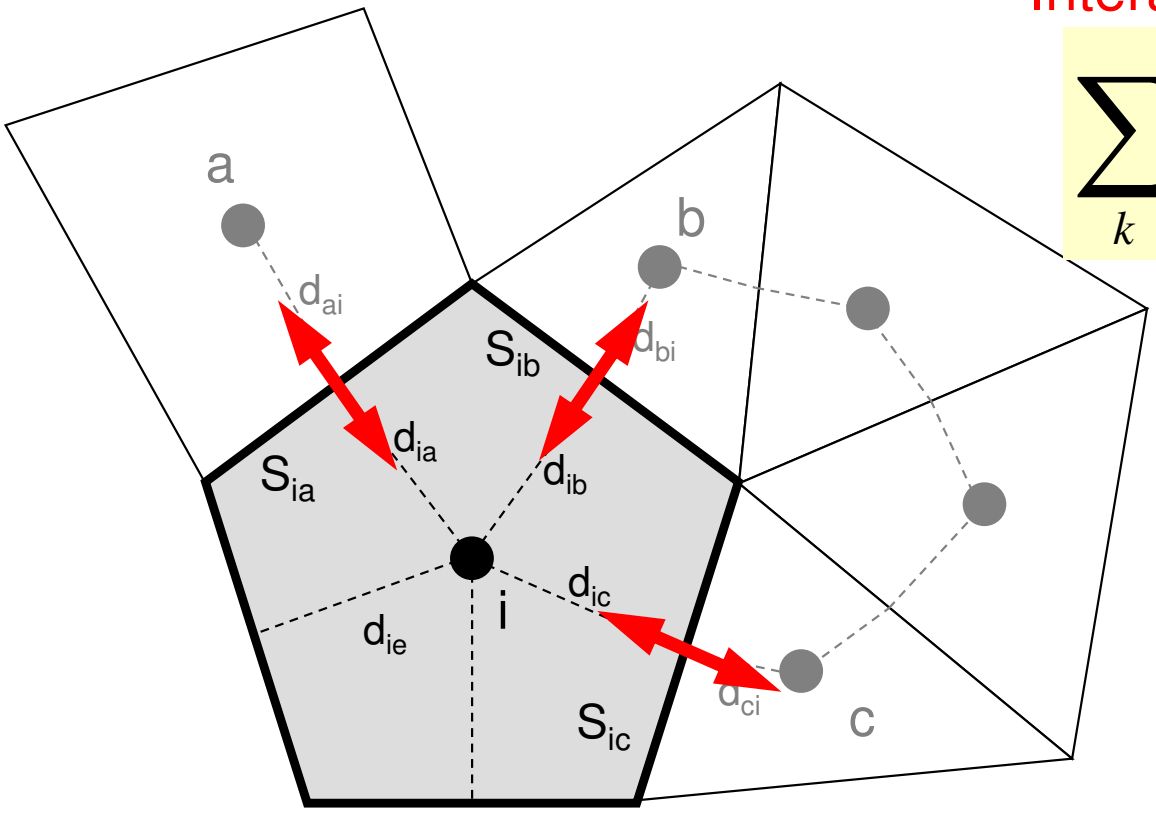


$$\left(\frac{d^2\phi}{dx^2}\right)_i \approx \frac{\left(\frac{d\phi}{dx}\right)_{i+1/2} - \left(\frac{d\phi}{dx}\right)_{i-1/2}}{\Delta x} = \frac{\frac{\phi_{i+1}-\phi_i}{\Delta x} - \frac{\phi_i-\phi_{i-1}}{\Delta x}}{\Delta x} = \frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2}$$

Poisson Equation by Finite Volume Method (FVM)

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

Conservation of Fluxes through Surfaces



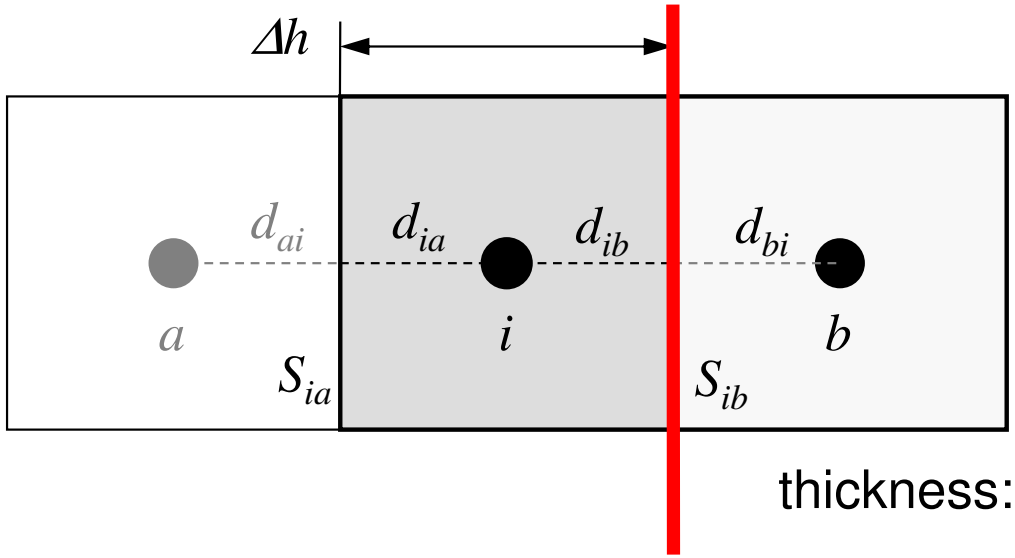
Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux

V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Comparison with 1D FDM (1/3)



Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

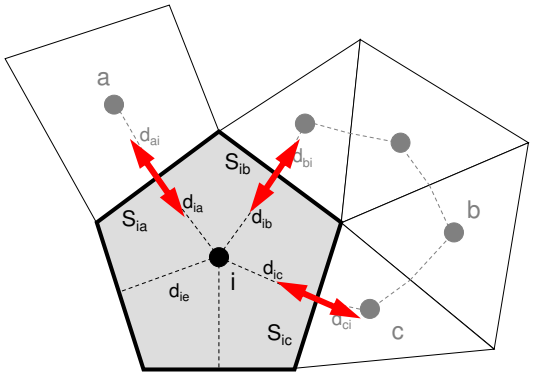
Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

Flux through this surface: Qs_{ib}

$$Qs_{ib} = -\frac{\phi_b - \phi_i}{d_{ib} + d_{bi}} \cdot S_{ib}$$

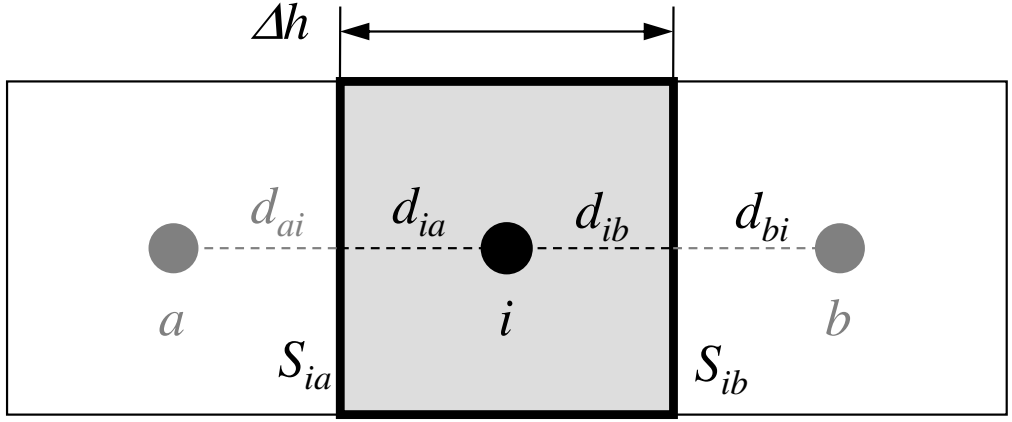
Fourier's Law

Flux through a surface
= - (gradient of potential)



$$\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} (\phi_k - \phi_i) + V_i \dot{Q}_i = 0$$

Comparison with 1D FDM (2/3)



Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

thickness: 1

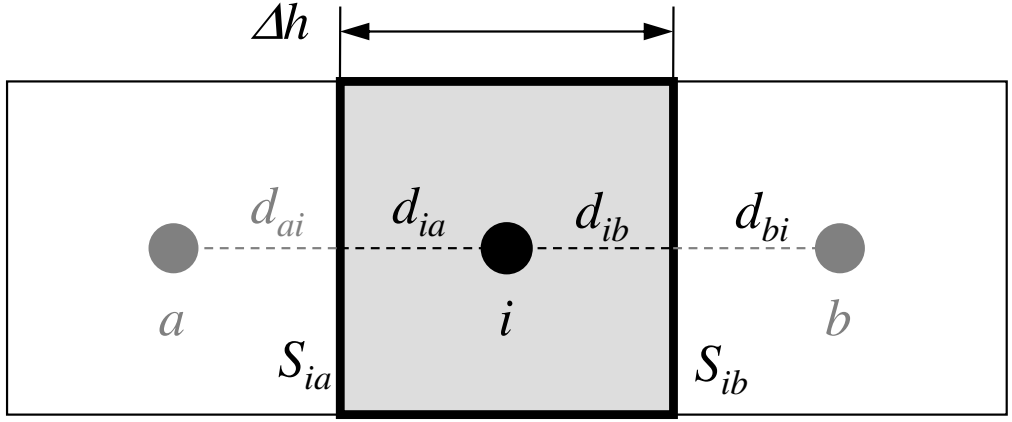
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k - \phi_i) + V_i \dot{Q}_i = 0$$

Divided by V_i:

$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k - \phi_i) + \dot{Q}_i = 0$$

considering this part

Comparison with 1D FDM (3/3)



Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

thickness: 1

$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i)$$

$$= \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\Delta h}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}(\phi_k-\phi_i)$$

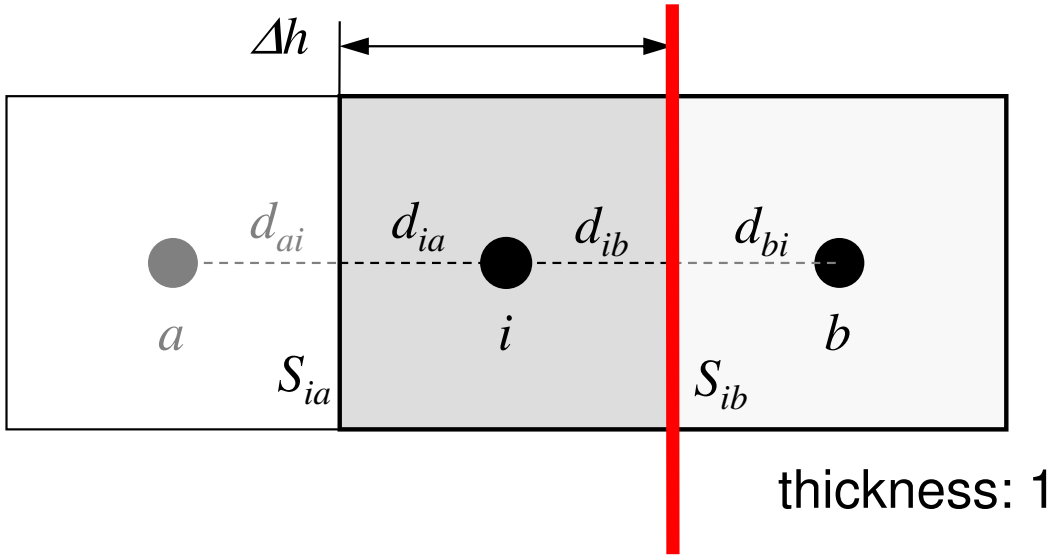
$$= \frac{1}{(\Delta h)^2}(\phi_a-\phi_i)+\frac{1}{(\Delta h)^2}(\phi_b-\phi_i) = \boxed{\frac{\phi_a-2\phi_i+\phi_b}{(\Delta h)^2}}$$

for i-th cell
linear equations

Heat Equation (1/3)

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+Q=0$$

λ: Thermal Conductivity



Δh x Δh Square Mesh

Surface Area: $S_{ik}=\Delta h$

Volume: $V_i=\Delta h^2$

Distance (Ctr.-Suf): $d_{ij}=\Delta h/2$

Heat Flux through this surface: Qs_{ib}

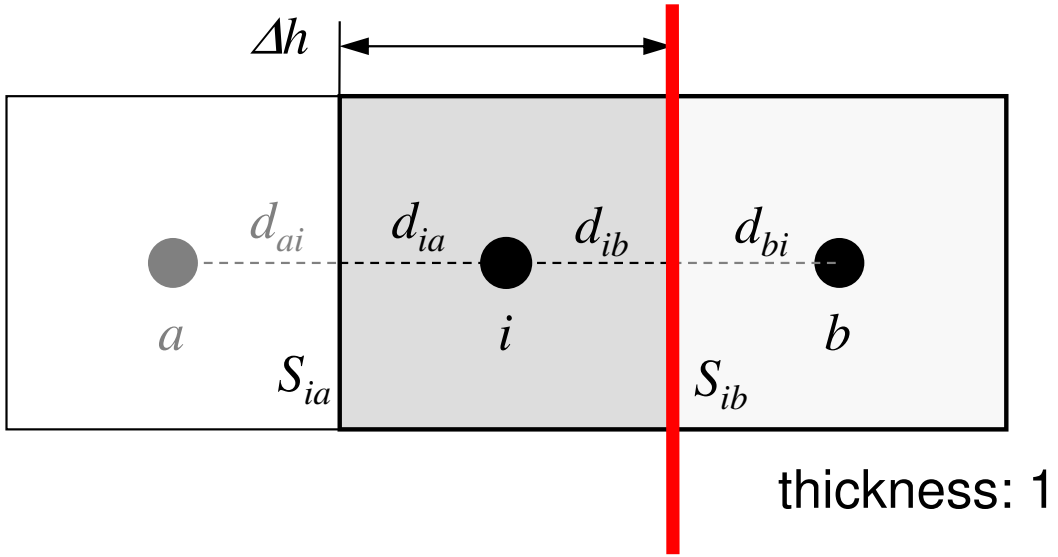
$$Qs_{ib}=-\lambda\frac{T_b-T_i}{\Delta h}\cdot S_{ib}$$

$$\lambda_i=\lambda_b=\lambda$$

Heat Equation (2/3)

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+Q=0$$

λ: Thermal Conductivity



Δh x Δh Square Mesh

Surface Area: $S_{ik}=\Delta h$

Volume: $V_i=\Delta h^2$

Distance (Ctr.-Suf): $d_{ij}=\Delta h/2$

Heat Flux through this surface: Qs_{ib}

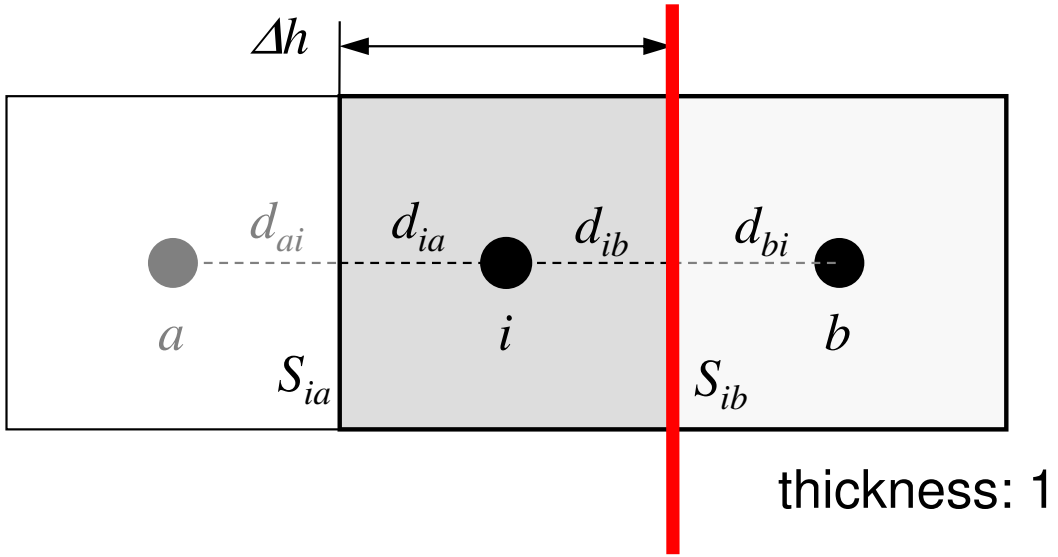
$$Qs_{ib}=-\lambda\frac{T_b-T_i}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}\cdot S_{ib}=-\frac{T_b-T_i}{\left[\left(\frac{\Delta h}{2}\right)/\lambda\right]+\left[\left(\frac{\Delta h}{2}\right)/\lambda\right]}\cdot S_{ib}$$

$$\lambda_i=\lambda_b=\lambda$$

Heat Equation (3/3)

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+Q=0$$

λ: Thermal Conductivity



Δh x Δh Square Mesh

Surface Area: $S_{ik}=\Delta h$

Volume: $V_i=\Delta h^2$

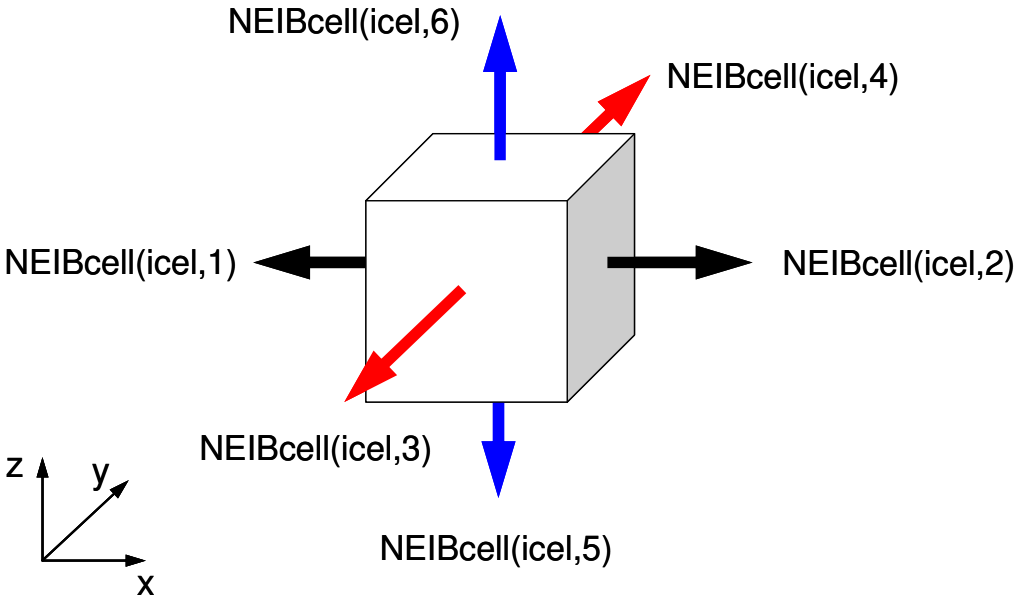
Distance (Ctr.-Suf): $d_{ij}=\Delta h/2$

Heat Flux through this surface: Qs_{ib}

$$Qs_{ib}=-\frac{T_b-T_i}{\left[\left(\frac{\Delta h}{2}\right)/\lambda_i\right]+\left[\left(\frac{\Delta h}{2}\right)/\lambda_b\right]}\cdot S_{ib}$$

$\lambda_i \neq \lambda_b$

in 3D



$$\frac{\phi_{neib(icel,1)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib(icel,2)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib(icel,3)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib(icel,4)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib(icel,5)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib(icel,6)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

Linear Equations

$$\frac{\phi_{neib(icel,1)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib(icel,2)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$
$$\frac{\phi_{neib(icel,3)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib(icel,4)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$
$$\frac{\phi_{neib(icel,5)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib(icel,6)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

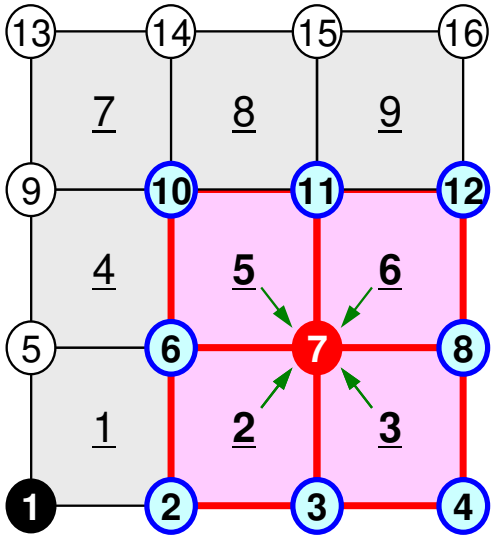
$$\sum_k \frac{S_{icel-k}}{d_{icel-k}}\left(\phi_k-\phi_{icel}\right)=-f_{icel}V_i$$

$$-\left[\sum_k \frac{S_{icel-k}}{d_{icel-k}}\right]\phi_{icel}+\left[\sum_k \frac{S_{icel-k}}{d_{icel-k}}\phi_k\right]=-f_{icel}V_i \quad (icel=1,\,N)$$

Diagonal Off-Diagonal

$$\longrightarrow \quad [A]\{\phi\}=\{f\}$$

Sparse Coef. Matrix in FEM Many "0's"



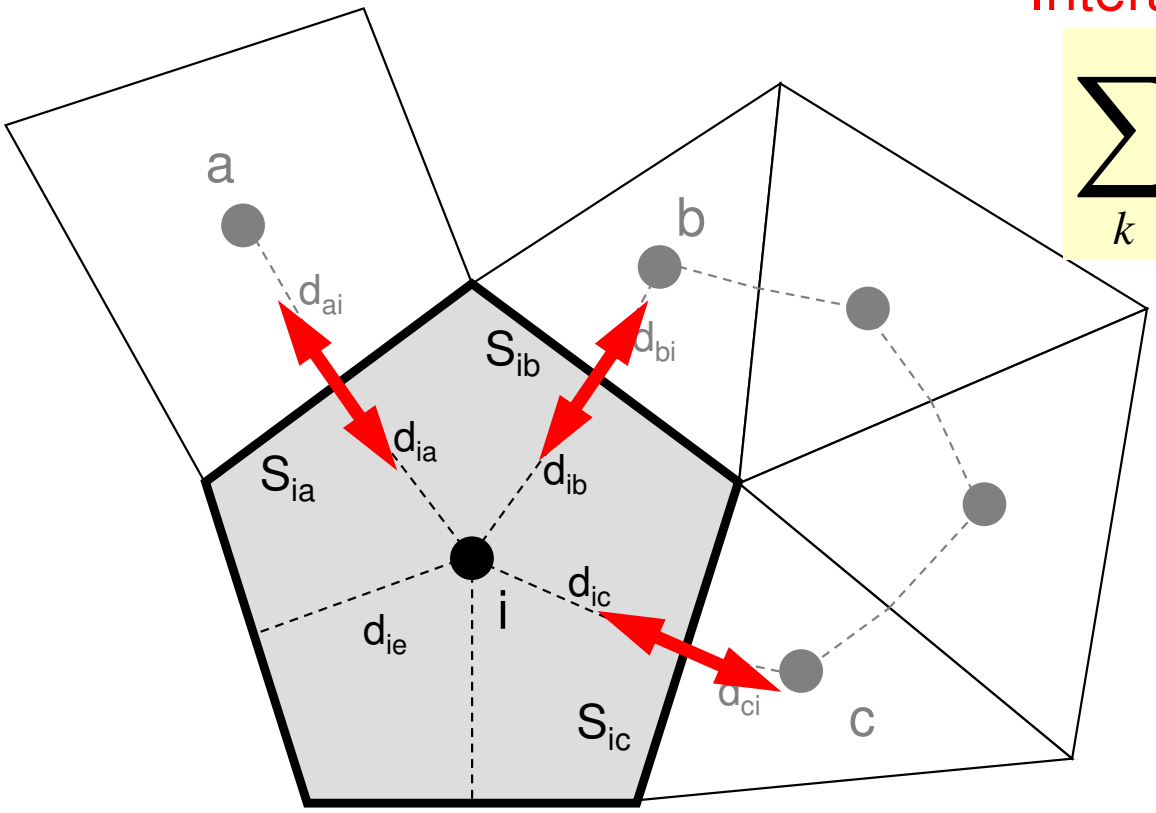
$$[K]\{\Phi\} = \{F\}$$

$$\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16} \end{Bmatrix}
=
\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16} \end{Bmatrix}$$

Coefficient Matrices for FVM are sparse

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

Only neighboring cells are considered



Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux

V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Sparse Matrix for FVM

- Sparse Matrix
  - Many “0”’s
- Storing all components (e.g. $\mathrm{A}(i,j)$) is not efficient for sparse matrices
  - $\mathrm{A}(i,j)$ is suitable for dense matrices

$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16} \end{Bmatrix}
=
\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16} \end{Bmatrix}
$$

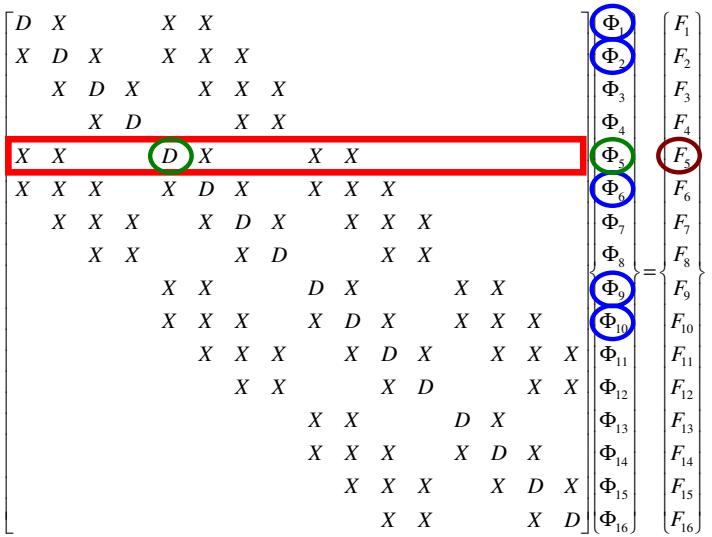
- Number of non-zero off-diagonal components is O(100) in FVM (only 6 in this case)
  - If number of unknowns is 10^8 :
    - $\mathrm{A}(i,j)$: $O(10^{16})$ words ~ $O(10^{17})$ bytes for DP: 100PB (400 x Odyssey)
    - Actual Non-zero Comp.: $O(10^9)$ words: 10GB (Odyssey: 32GB/node)
- Only (really) non-zero off-diag. components should be stored on memory

Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

Diag (i)	Diagonal Components (REAL, i=1~N)
Index(i)	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item(k)	Non-Zero Off-Diagonal Components (Corresponding Column ID) (INT, k=1, index(N))
AMat(k)	Non-Zero Off-Diagonal Components (Value) (REAL, k=1, index(N))

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= Diag(i)*X(i)
  do k= Index(i-1)+1, Index(i)
    Y(i)= Y(i) + Amat(k)*X(Item(k))
  enddo
enddo
```



Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

```
{Q}=[A]{P}

for(i=0;i<N;i++){
      W[Q][i] = Diag[i] * W[P][i];
      for(k=Index[i];k<Index[i+1];k++){
            W[Q][i] += AMat[k]*W[P][Item[k]];
      }
}
```

Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
{Y}= [A]{X}

do j= 1, N
  Y(j)= 0.d0
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

Compressed Row Storage (CRS)

	1	2	3	4	5	6	7	8
1	1.1	2.4	0	0	3.2	0	0	0
2	4.3	3.6	0	2.5	0	3.7	0	9.1
3	0	0	5.7	0	1.5	0	3.1	0
4	0	4.1	0	9.8	2.5	2.7	0	0
5	3.1	9.5	10.4	0	11.5	0	4.3	0
6	0	0	6.5	0	0	12.4	9.5	0
7	0	6.4	2.5	0	0	1.4	23.1	13.1
8	0	9.5	1.3	9.6	0	3.1	0	51.3

Compressed Row Storage (CRS): Fortran

	1	2	3	4	5	6	7	8
1	1.1 ①	2.4 ②			3.2 ⑤			
2	4.3 ①	3.6 ②		2.5 ④		3.7 ⑥		9.1 ⑧
3			5.7 ③		1.5 ⑤		3.1 ⑦	
4		4.1 ②		9.8 ④	2.5 ⑤	2.7 ⑥		
5	3.1 ①	9.5 ②	10.4 ③		11.5 ⑤		4.3 ⑦	
6			6.5 ③			12.4 ⑥	9.5 ⑦	
7		6.4 ②	2.5 ③			1.4 ⑥	23.1 ⑦	13.1 ⑧
8		9.5 ②	1.3 ③	9.6 ④		3.1 ⑥		51.3 ⑧

```
N= 8
```

Diagonal Components

```
Diag(1)=  1.1
Diag(2)=  3.6
Diag(3)=  5.7
Diag(4)=  9.8
Diag(5)= 11.5
Diag(6)= 12.4
Diag(7)= 23.1
Diag(8)= 51.3
```

Compressed Row Storage (CRS)

	1	2	3	4	5	6	7	8
1	1.1 ①		2.4 ②			3.2 ⑤			
2	3.6 ②	4.3 ①			2.5 ④		3.7 ⑥		9.1 ⑧
3	5.7 ③					1.5 ⑤		3.1 ⑦	
4	9.8 ④		4.1 ②			2.5 ⑤	2.7 ⑥		
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③				4.3 ⑦	
6	12.4 ⑥			6.5 ③				9.5 ⑦	
7	23.1 ⑦		6.4 ②	2.5 ③			1.4 ⑥		13.1 ⑧
8	51.3 ⑧		9.5 ②	1.3 ③	9.6 ④		3.1 ⑥		

Compressed Row Storage (CRS)

						# Non-Zero Off-Diag.	
							index(0)= 0
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(1)= 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2)= 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3)= 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4)= 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5)= 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6)= 17
7	23.1 ⑦	6.4 ②	2.5 ③	1.4 ⑥	13.1 ⑧	4	index(7)= 21
8	51.3 ⑧	9.5 ②	1.3 ③	9.6 ④	3.1 ⑥	4	index(8)= 25

NPLU= 25
(=index(N))

index(i-1)+1th~index(i)th
Non-Zero Off-Diag. Components corresponding to i-th row

Compressed Row Storage (CRS)

						# Non-Zero Off-Diag.	
							index(0)= 0
1	1.1 ①	2.4 ②,1	3.2 ⑤,2			2	index(1)= 2
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6	4	index(2)= 6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8			2	index(3)= 8
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11		3	index(4)= 11
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15	4	index(5)= 15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17			2	index(6)= 17
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21	4	index(7)= 21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25	4	index(8)= 25

NPLU= 25
(=index(N))

index(i-1)+1th~index(i)th
Non-Zero Off-Diag. Components corresponding to i-th row

Compressed Row Storage (CRS)

1	1.1 ①	2.4 ②,1	3.2 ⑤,2		
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8		
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11	
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17		
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25

```
Example:
item( 7)= 5, AMAT( 7)= 1.5
item(19)= 3, AMAT(19)= 2.5
```

Compressed Row Storage (CRS)

1	1.1 ①	2.4 ②,1	3.2 ⑤,2		
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8		
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11	
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17		
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25

Diag (i)	Diagonal Components (REAL, i=1~N)
Index(i)	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item(k)	Non-Zero-Off-Diagonal Components (Corresponding Column ID) （INT, k=1, index(N)）
AMat(k)	Non-Zero-Off-Diagonal Components (Value) （ REAL, k=1, index(N) ）

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```

Sparse Matrix: Only non-zero components are stored

Indirect Access: Memory-Bound

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    kk= item(k)
    Y(i)= Y(i) + AMAT(k)*X(kk)
  enddo
enddo
```

Dense Matrix: Continuous Access

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
{Y}= [A]{X}

do j= 1, N
  Y(j)= 0.d0
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

- Background
  - Finite Volume Method
  - **Preconditioned Iterative Solvers**
- PCG Solver for Poisson's Equations
  - How to run
    - Data Structure
  - Program
    - Initialization
    - Coefficient Matrices
    - PCG

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations **Ax=b** is the most important and **expensive** part of various types of scientific computing.
  - for both linear and nonlinear applications
- Various types of methods proposed & developed.
  - for dense and sparse matrices
  - classified into ***direct*** and ***iterative*** methods
- Dense Matrices:密行列: Globally Coupled Problems
  - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices:疎行列: Locally Defined Problems
  - FEM, FVM, FDM, DEM, MD (solid), BEM w/FMM

Direct Method
直接法

- Gaussian Elimination/LU Factorization
  - compute A^{-1} directly (or equivalent operations)

Good

- Robust for wide range of applications.
- Good for both dense and sparse matrices

Bad

- More expensive than iterative methods (memory, CPU)
  - not scalable

What is Iterative Method ?
反復法

Linear Equations
連立一次方程式

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\mathbf{A} \qquad \mathbf{x} \qquad \mathbf{b}$$

Initial Solution
初期解

$$\mathbf{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix}$$

Starting from a initial vector $\mathrm{x}^{(0)}$, iterative method obtains the final converged solutions by iterations

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots$$

Iterative Method
反復法

- Stationary Method
  - Only x (solution vector) changes during iterations.
  - SOR, Gauss-Seidel, Jacobi
  - Generally slow, impractical

$$\mathbf{Ax} = \mathbf{b} \Rightarrow$$
$$\mathbf{x}^{(k+1)} = \mathbf{Mx}^{(k)} + \mathbf{Nb}$$

- Non-Stationary Method
  - With restriction/optimization conditions
  - Krylov-Subspace
  - CG: Conjugate Gradient
  - BiCGSTAB: Bi-Conjugate Gradient Stabilized
  - GMRES: Generalized Minimal Residual

Iterative Method (cont.)

Good

- Less expensive than direct methods, especially in memory.
- Suitable for parallel and vector computing.

Bad

- Convergence strongly depends on problems, boundary conditions (condition number etc.)
- Preconditioning is required : Key Technology for Real-World Applications in Scientific Computing (FEM, FVM, FDM etc)

Non-Stationary/Krylov Subspace Method (1/2)

非定常法・クリロフ部分空間法

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}$$

Compute $\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}$ by the following iterative procedures:

$$\begin{aligned}\mathbf{x}_k &= \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}_{k-1} \\ &= (\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) + \mathbf{x}_{k-1} \\ &= \mathbf{r}_{k-1} + \mathbf{x}_{k-1}\end{aligned} \qquad \textit{where } \mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k \textbf{ : residual}$$

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i$$

$$\begin{aligned}\mathbf{r}_k &= \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}(\mathbf{r}_{k-1} + \mathbf{x}_{k-1}) \\ &= (\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) - \mathbf{A}\mathbf{r}_{k-1} = \mathbf{r}_{k-1} - \mathbf{A}\mathbf{r}_{k-1} = (\mathbf{I} - \mathbf{A})\mathbf{r}_{k-1}\end{aligned}$$

Non-Stationary/Krylov Subspace Method (2/2)

非定常法・クリロフ部分空間法

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=0}^{k-2} (\mathbf{I} - \mathbf{A})\mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \mathbf{r}_0$$

$$\mathbf{z}_k = \mathbf{r}_0 + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \mathbf{r}_0 = \left[\mathbf{I} + \sum_{i=1}^{k-1} (\mathbf{I} - \mathbf{A})^i \right] \mathbf{r}_0$$

$\mathbf{z_k}$ is a vector which belongs to k^{th} Krylov Subspace（クリロフ部分空間）, approximate solution vector $\mathbf{x_k}$ is derived by the Krylov Subspace:

$$\left[\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\right]$$

Conjugate Gradient Method
共役勾配法

- Conjugate Gradient: CG
  - Most popular “non-stationary” iterative method
- for Symmetric Positive Definite (SPD) Matrices
  - 対称正定
  - $\{x\}^{\mathrm{T}}[A]\{x\}>0$ for arbitrary $\{x\}$
  - All of diagonal components, eigenvaules and leading principal minors > 0（主小行列式･首座行列式）
  - Matrices of Galerkin-based FEM & FVM: heat conduction, Poisson, static linear elastic problems
- Algorithm
  - “Steepest Descent Method”
  - $\boldsymbol{x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}}$
    - $x^{(i)}$: solution, $p^{(i)}$: search direction, α_i: coefficient
  - Solution $\{x\}$ minimizes $\boldsymbol{\{x-y\}^{\mathrm{T}}[A]\{x-y\}}$, where $\{y\}$ is exact solution.

$$\det\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}$$

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY (Double Precision: a{X} + {Y})

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) – αiq(i)
    check convergence |r|
end
```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

Compute $r^{(0)} = b - [A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)}\, z^{(i-1)}$
  if i=1
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1}\, p^{(i-1)}$
  endif
  $q^{(i)} = [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
end

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

- Mat-Vec. Multiplication
- Dot Products
- DAXPY
  - Double
  - {y}= a{x} + {y}

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

Compute $r^{(0)} = b - [A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)}\, z^{(i-1)}$
  if i=1
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1}\, p^{(i-1)}$
  endif
  $q^{(i)} = [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
end

$x^{(i)}$: Vector

α_i : Scalar

Derivation of CG Algorithm (1/5)

Solution x minimizes the following equation if y is the exact solution ($Ay=b$)

$$(x-y)^T[A](x-y)$$

$$(x-y)^T[A](x-y)=(x,Ax)-(y,Ax)-(x,Ay)+(y,Ay)$$
$$=(x,Ax)-2(x,Ay)+(y,Ay)=(x,Ax)-2(x,b)+\underline{(y,b)}$$ Const.

Therefore, the solution x minimizes the following $f(x)$:

$$f(x)=\frac{1}{2}(x,Ax)-(x,b)$$

$$f(x+h)=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$ Arbitrary vector h

$$f(x)=\frac{1}{2}(x,Ax)-(x,b)$$

$$f(x+h)=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$

Arbitrary vector h

$$f(x+h)=\frac{1}{2}(x+h,A(x+h))-(x+h,b)$$

$$=\frac{1}{2}(x+h,Ax)+\frac{1}{2}(x+h,Ah)-(x,b)-(h,b)$$

$$=\frac{1}{2}(x,Ax)+\frac{1}{2}(h,Ax)+\frac{1}{2}(x,Ah)+\frac{1}{2}(h,Ah)-(x,b)-(h,b)$$

$$=\frac{1}{2}(x,Ax)-(x,b)+(h,Ax)-(h,b)+\frac{1}{2}(h,Ah)$$

$$=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$

Derivation of CG Algorithm (2/5)

CG method minimizes $f(x)$ at each iteration.
Start from initial solution $x^{(0)}$ and assume that approximate solution: $x^{(k)}$, and search direction vector $p^{(k)}$ is defined at k-th iter.

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$$

Minimization of $f(x^{(k+1)})$ is done as follows:

$$f\left(x^{(k)} + \alpha_k p^{(k)}\right) = \frac{1}{2}\alpha_k{}^2\left(p^{(k)}, Ap^{(k)}\right) - \alpha_k\left(p^{(k)}, b - Ax^{(k)}\right) + f\left(x^{(k)}\right)$$

$$\frac{\partial f\left(x^{(k)} + \alpha_k p^{(k)}\right)}{\partial \alpha_k} = 0 \Rightarrow \alpha_k = \frac{\left(p^{(k)}, b - Ax^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} \quad \text{(1)}$$

$r^{(k)} = b - Ax^{(k)}$ residual vector

Derivation of CG Algorithm (3/5)

Residual vector at $(k+1)$-th iteration:

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)} \quad \textbf{(2)}$$

$$r^{(k+1)} = b - Ax^{(k+1)},\ r^{(k)} = b - Ax^{(k)}$$

$$r^{(k+1)} - r^{(k)} = -Ax^{(k+1)} + Ax^{(k)} = -\alpha_k A p^{(k)}$$

Search direction vector p is defined by the following recurrence formula:

$$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)},\ r^{(0)} = p^{(0)} \quad \textbf{(3)}$$

It's lucky if we can get exact solution y at $(k+1)$-th iteration:

$$y = x^{(k+1)} + \alpha_{k+1} p^{(k+1)}$$

Derivation of CG Algorithm (4/5)

BTW, we have the following (convenient) orthogonality relation:

$$\left(Ap^{(k)}, y - x^{(k+1)}\right) = 0$$

$$\begin{aligned}
&\left(Ap^{(k)}, y - x^{(k+1)}\right) = \left(p^{(k)}, Ay - Ax^{(k+1)}\right) = \left(p^{(k)}, b - Ax^{(k+1)}\right) \\
&= \left(p^{(k)}, b - A\left[x^{(k)} + \alpha_k p^{(k)}\right]\right) = \left(p^{(k)}, b - Ax^{(k)} - \alpha_k Ap^{(k)}\right) \\
&= \left(p^{(k)}, r^{(k)} - \alpha_k Ap^{(k)}\right) = \left(p^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, Ap^{(k)}\right) = 0
\end{aligned}$$

$$\because \alpha_k = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

Thus, following relation is obtained:

$$\left(Ap^{(k)}, y - x^{(k+1)}\right) = \left(Ap^{(k)}, \alpha_{k+1} p^{(k+1)}\right) = 0 \Rightarrow \left(p^{(k+1)}, Ap^{(k)}\right) = 0$$

Derivation of CG Algorithm (5/5)

$$\left(p^{(k+1)}, Ap^{(k)}\right)=\left(r^{(k+1)}+\beta_k p^{(k)}, Ap^{(k)}\right)=\left(r^{(k+1)}, Ap^{(k)}\right)+\beta_k\left(p^{(k)}, Ap^{(k)}\right)=0$$

$$\Rightarrow \beta_k=\frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} \quad (4)$$

$\left(p^{(k+1)}, Ap^{(k)}\right)=0$

$p^{(k)}$ & $p^{(k+1)}$ are **"conjugate（共役）"** for matrix A

$p^{(k)}$: search direction vector, "gradient" vector

```
Compute p(0)=r(0)= b-[A]x(0)
for i= 1, 2, …
    calc. α(i-1)
    x(i)= x(i-1) + α(i-1)p(i-1)
    r(i)= r(i-1) - α(i-1)[A]p(i-1)

    check convergence |r|
    (if not converged)
    calc. β(i-1)
    p(i)= r(i) + β(i-1) p(i-1)
end
```

$$\alpha_{i-1}=\frac{\left(p^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

$$\beta_{i-1}=\frac{-\left(r^{(i)}, Ap^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

Properties of CG Algorithm

Following “conjugate（共役）” relationship is obtained for arbitrary (i,j):

$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \left(i \neq j\right)$$

Following relationships are also obtained for $p^{(k)}$ and $r^{(k)}$:

$$\left(r^{(i)}, r^{(j)}\right) = 0 \left(i \neq j\right), \quad \left(p^{(k)}, r^{(k)}\right) = \left(r^{(k)}, r^{(k)}\right)$$

In N-dimensional space, only N sets of orthogonal and linearly independent residual vector $r^{(k)}$. This means CG method converges after N iterations if number of unknowns is N. Actually, round-off error sometimes affects convergence.

Proof (1/3)

Mathematical Induction

数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \left(i \neq j\right)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \left(i \neq j\right)$$

(1) $$\alpha_k = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

(2) $$r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$$

(3) $$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}, \; r^{(0)} = p^{(0)}$$

(4) $$\beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

Proof (2/3)

Mathematical Induction
数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \ (i \neq j)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \ (i \neq j) \qquad (*)$$

$(*)$ is satisfied for $i \leq k,\ j \leq k$ where $i \neq j$

if $i < k$

$$\left(r^{(k+1)}, r^{(i)}\right) = \left(r^{(i)}, r^{(k+1)}\right) \overset{(2)}{=} \left(r^{(i)}, r^{(k)} - \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} -\alpha_k \left(r^{(i)}, Ap^{(k)}\right) \overset{(3)}{=} -\alpha_k \left(p^{(i)} - \beta_{i-1} p^{(i-1)}, Ap^{(k)}\right)$$
$$= -\alpha_k \left(p^{(i)}, Ap^{(k)}\right) + \alpha_k \beta_{i-1} \left(p^{(i-1)}, Ap^{(k)}\right) \overset{(*)}{=} 0$$

if $i = k$

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, Ap^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)}\right)$$
$$= -\beta_{k-1} \left(p^{(k-1)}, r^{(k)}\right) \overset{(2)}{=} -\beta_{k-1} \left(p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} Ap^{(k-1)}\right)$$
$$= -\beta_{k-1} \left\{\left(p^{(k-1)}, r^{(k-1)}\right) - \alpha_{k-1} \left(p^{(k-1)}, Ap^{(k-1)}\right)\right\} \overset{(1)}{=} 0$$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

Proof (3/3)

Mathematical Induction
数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \quad (i \neq j)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \quad (i \neq j) \qquad (*)$$

$(*)$ is satisfied for $i \leq k,\ j \leq k$ where $i \neq j$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

if $i < k$

$$\left(p^{(k+1)}, Ap^{(i)}\right) \overset{(3)}{=} \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}\right)$$
$$\overset{(*)}{=} \left(r^{(k+1)}, Ap^{(i)}\right)$$
$$\overset{(2)}{=} \frac{1}{\alpha_i}\left(r^{(k+1)}, r^{(i)} - r^{(i-1)}\right) = 0$$

if $i = k$

$$\left(p^{(k+1)}, Ap^{(k)}\right) \overset{(3)}{=} \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k \left(p^{(k)}, Ap^{(k)}\right)$$
$$\overset{(4)}{=} 0$$

$$\left(r^{(k+1)}, r^{(k)}\right) = 0$$

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k A p^{(k)}\right)$$

$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k A p^{(k)}\right)$$

$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, A p^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right) = 0$$

$$\therefore \left(r^{(k)}, r^{(k)}\right) = \left(p^{(k)}, r^{(k)}\right)$$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, A p^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, A p^{(k)}\right)}{\left(p^{(k)}, A p^{(k)}\right)}$

$\alpha_k,\ \beta_k$

Usually, we use simpler definitions of $\alpha_k,\ \beta_k$ as follows:

$$\alpha_k = \frac{\left(p^{(k)}, b - Ax^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(r^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

$$\because \left(p^{(k)}, r^{(k)}\right) = \left(r^{(k)}, r^{(k)}\right)$$

$$\beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(r^{(k+1)}, r^{(k+1)}\right)}{\left(r^{(k)}, r^{(k)}\right)}$$

$$\because \left(r^{(k+1)}, Ap^{(k)}\right) = \frac{\left(r^{(k+1)}, r^{(k)} - r^{(k+1)}\right)}{\alpha_k} = -\frac{\left(r^{(k+1)}, r^{(k+1)}\right)}{\alpha_k}$$

Procedures of Conjugate Gradient

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

$x^{(i)}$: Vector

α_i : Scalar

$$\beta_{i-1} = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(r^{(i-2)}, r^{(i-2)}\right)} \quad \left(= \rho_{i-1}\right) \quad \left(= \rho_{i-2}\right)$$

$$\alpha_i = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i)}, Ap^{(i)}\right)} \quad \left(= \rho_{i-1}\right)$$

Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix A.
  - Eigenvalue distribution is small, eigenvalues are close to 1
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if A is symmetric) is large（条件数）.
- A preconditioner M (whose properties are similar to those of A) transforms the linear system into one with more favorable spectral properties （前処理）
  - M transforms Ax=b into A'x=b' where $A'=M^{-1}A$, $b'=M^{-1}b$
  - If M~A, $M^{-1}A$ is close to identity matrix.
  - If $M^{-1}=A^{-1}$, this is the best preconditioner (Gaussian Elim.)
  - Generally, A'x'=b' where $A'=M_L^{-1}AM_R^{-1}$, $b'=M_L^{-1}b$, $x'=M_Rx$
  - M_L/M_R: Left/Right Preconditioning（左／右前処理）

Preconditioned CG Solver

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

  solve $[M]z^{(i-1)}= r^{(i-1)}$

  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$

  if i=1

    $p^{(1)}= z^{(0)}$

  else

    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$

    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$

  endif

  $q^{(i)}= [A]p^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$

  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence $|r|$

end

$$[M]= [M_1][M_2]$$

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

$$[A']=[M_1]^{-1}[A][M_2]^{-1}$$

$$x'=[M_2]x,\ b'=[M_1]^{-1}b$$

$$p'=>[M_2]p,\ r'=>[M_1]^{-1}r$$

$$p'^{(i)}_{x}= r'^{(i-1)} + \beta'_{i-1}\ p'^{(i-1)}$$

$$[M_2]p^{(i)}= [M_1]^{-1}r^{(i-1)} + \beta'_{i-1}\ [M_2]p^{(i-1)}$$

$$p^{(i)}= [M_2]^{-1}[M_1]^{-1}r^{(i-1)} + \beta'_{i-1}\ p^{(i-1)}$$

$$p^{(i)}= [M]^{-1}r^{(i-1)} + \beta'_{i-1}\ p^{(i-1)}$$

$$\beta'_{i-1}= ([M]^{-1}r^{(i-1)}, r^{(i-1)})/([M]^{-1}r^{(i-2)}, r^{(i-2)})$$

$$\alpha'_{i-1}= ([M]^{-1}r^{(i-1)}, r^{(i-1)})/(p^{(i-1)}, [A]p^{(i-1)})$$

In CG method, preconditioner usually satisfies $[M_2]=[M_1]^T$, such as Incomplete Cholesky/Incomplete Modified Cholesky Factorizations. In this problem, let us define $[M_1]$ and $[M_2]$ as follows:

$$[M_1]=[X]^T,[M_2]=[X],[M]=[M_1][M_2]$$

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

$$[A']=[M_1]^{-1}[A][M_2]^{-1}=[[X]^T]^{-1}[A][X]^{-1}=[X]^{-T}[A][X]^{-1}$$

$$x'=[X]x,\quad b'=[X]^{-T}b,\quad r'=[X]^{-T}r$$

$$\alpha'_{i-1}=\frac{\left(r'^{(i-1)},r'^{(i-1)}\right)}{\left(p'^{(i-1)},A'p'^{(i-1)}\right)}=\frac{\left([X]^{-T}r^{(i-1)},[X]^{-T}r^{(i-1)}\right)}{\left([X]p^{(i-1)},[X]^{-T}[A][X]^{-1}[X]p^{(i-1)}\right)}$$

$$=\frac{\left(\left([X]^{-T}r^{(i-1)}\right)^T,[X]^{-T}r^{(i-1)}\right)}{\left(\left([X]p^{(i-1)}\right)^T,[X]^{-T}[A]p^{(i-1)}\right)}=\frac{\left(\left(r^{(i-1)}\right)^T[X]^{-1},\left[X^T\right]^{-1}r^{(i-1)}\right)}{\left(\left(p^{(i-1)}\right)^T[X]^T,[X]^{-T}[A]p^{(i-1)}\right)}$$

$$=\frac{\left(r^{(i-1)},\left[\left[X^T\right][X]\right]^{-1}r^{(i-1)}\right)}{\left(p^{(i-1)},[A]p^{(i-1)}\right)}=\frac{\left(r^{(i-1)},[M]^{-1}r^{(i-1)}\right)}{\left(p^{(i-1)},[A]p^{(i-1)}\right)}=\frac{\left(r^{(i-1)},z^{(i-1)}\right)}{\left(p^{(i-1)},[A]p^{(i-1)}\right)}$$

$$\beta'_{i-1} = \frac{\left(r'^{(i-1)}, r'^{(i-1)}\right)}{\left(r'^{(i-2)}, r'^{(i-2)}\right)} == \frac{\left(\left[\mathrm{X}\right]^{-T} r^{(i-1)}, \left[\mathrm{X}\right]^{-T} r^{(i-1)}\right)}{\left(\left[\mathrm{X}\right]^{-T} r^{(i-2)}, \left[\mathrm{X}\right]^{-T} r^{(i-2)}\right)}$$

$$= \frac{\left(\left(\left[\mathrm{X}\right]^{-T} r^{(i-1)}\right)^{T}, \left[\mathrm{X}\right]^{-T} r^{(i-1)}\right)}{\left(\left(\left[\mathrm{X}\right]^{-T} r^{(i-2)}\right)^{T}, \left[\mathrm{X}\right]^{-T} r^{(i-2)}\right)} = \frac{\left(\left(r^{(i-1)}\right)^{T}\left[\mathrm{X}\right]^{-1}, \left[\mathrm{X}^{T}\right]^{-1} r^{(i-1)}\right)}{\left(\left(r^{(i-2)}\right)^{T}\left[\mathrm{X}\right]^{-1}, \left[\mathrm{X}^{T}\right]^{-1} r^{(i-2)}\right)}$$

$$= \frac{\left(r^{(i-1)}, \left[\left[\mathrm{X}^{T}\right]\left[\mathrm{X}\right]\right]^{-1} r^{(i-1)}\right)}{\left(r^{(i-2)}, \left[\left[\mathrm{X}^{T}\right]\left[\mathrm{X}\right]\right]^{-1} r^{(i-2)}\right)} = \frac{\left(r^{(i-1)}, \left[\mathrm{M}\right]^{-1} r^{(i-1)}\right)}{\left(r^{(i-2)}, \left[\mathrm{M}\right]^{-1} r^{(i-2)}\right)} = \frac{\left(r^{(i-1)}, z^{(i-1)}\right)}{\left(r^{(i-2)}, z^{(i-2)}\right)}$$

Preconditioned Conjugate Gradient Method（PCG）

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) – αiq(i)
    check convergence |r|
end
```

Solving the following equation:

$$\{z\}=[M]^{-1}\{r\}$$

"Approximate Inverse Matrix"

$$[M]^{-1}\approx[A]^{-1},\quad [M]\approx[A]$$

Ultimate Preconditioning: Inverse Matrix

$$[M]^{-1}=[A]^{-1},\quad [M]=[A]$$

Diagonal Scaling: Simple but weak

$$[M]^{-1}=[D]^{-1},\quad [M]=[D]$$

Diagonal Scaling, Point-Jacobi

$$[M]=\begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **`solve [M]z(i-1)= r(i-1)`** is very easy.
- Provides fast convergence for simple problems.

ILU(0), IC(0)

- Widely used Preconditioners for Sparse Matrices
  - Incomplete LU Factorization
  - Incomplete Cholesky Factorization (for Symmetric Matrices)
- Incomplete Direct Method
  - Even if original matrix is sparse, inverse matrix is not necessarily sparse.
  - **fill-in**
  - ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices
- **More details are shown in the later stage of this class**

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **PCG Solver for Poisson's Equations**
  - **How to run**
    - **Data Structure**
  - Program
    - Initialization
    - Coefficient Matrices
    - PCG

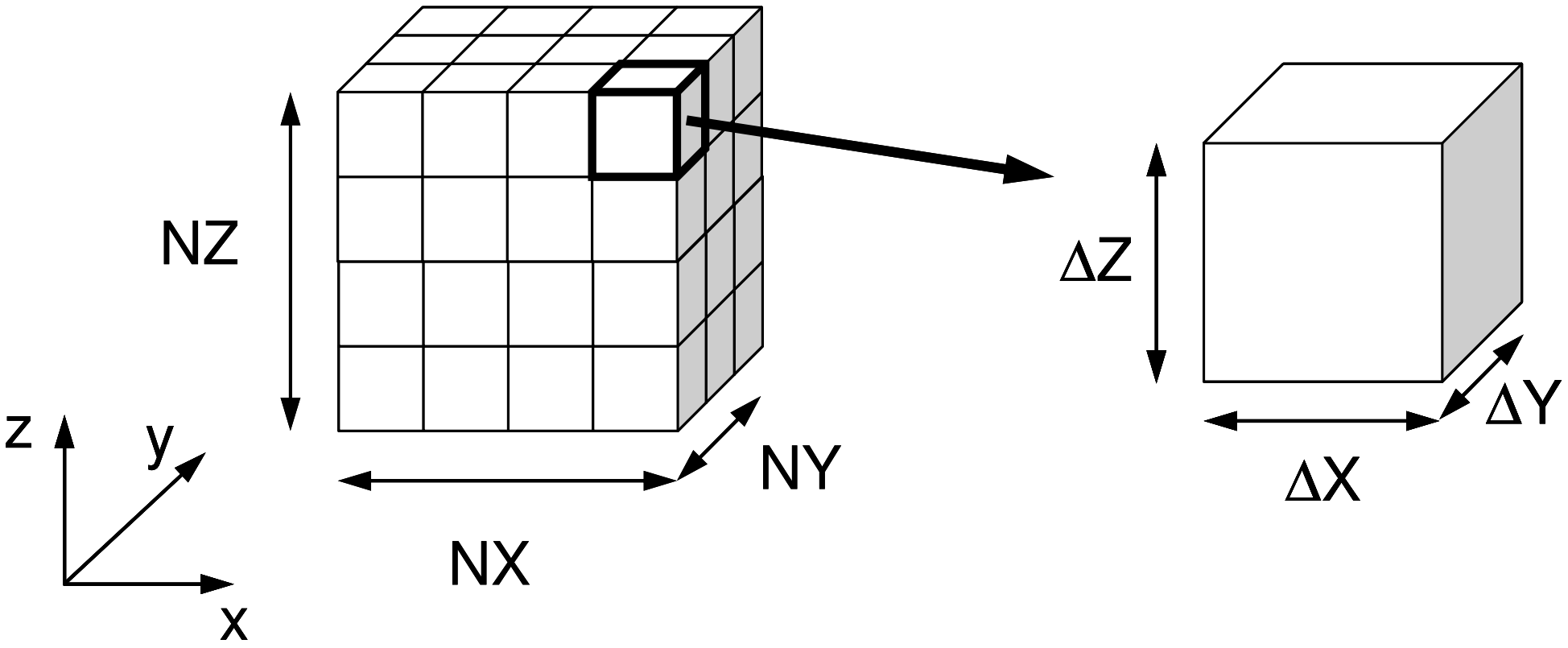
Target Application

- 3D Poisson Equation/Poisson's Equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

- Finite Volume Method (FVM)
  - Arbitrary Shape Meshes, Cell-Centered
  - "Direct" Finite Difference Method
- Boundary Conditions (B.C.) etc.
  - Dirichlet B.C., Volume Flux
- Preconditioned Iterative Solvers
  - Conjugate Gradient + Preconditioner

3D Structured Mesh
Internal data structure is “unstructured”



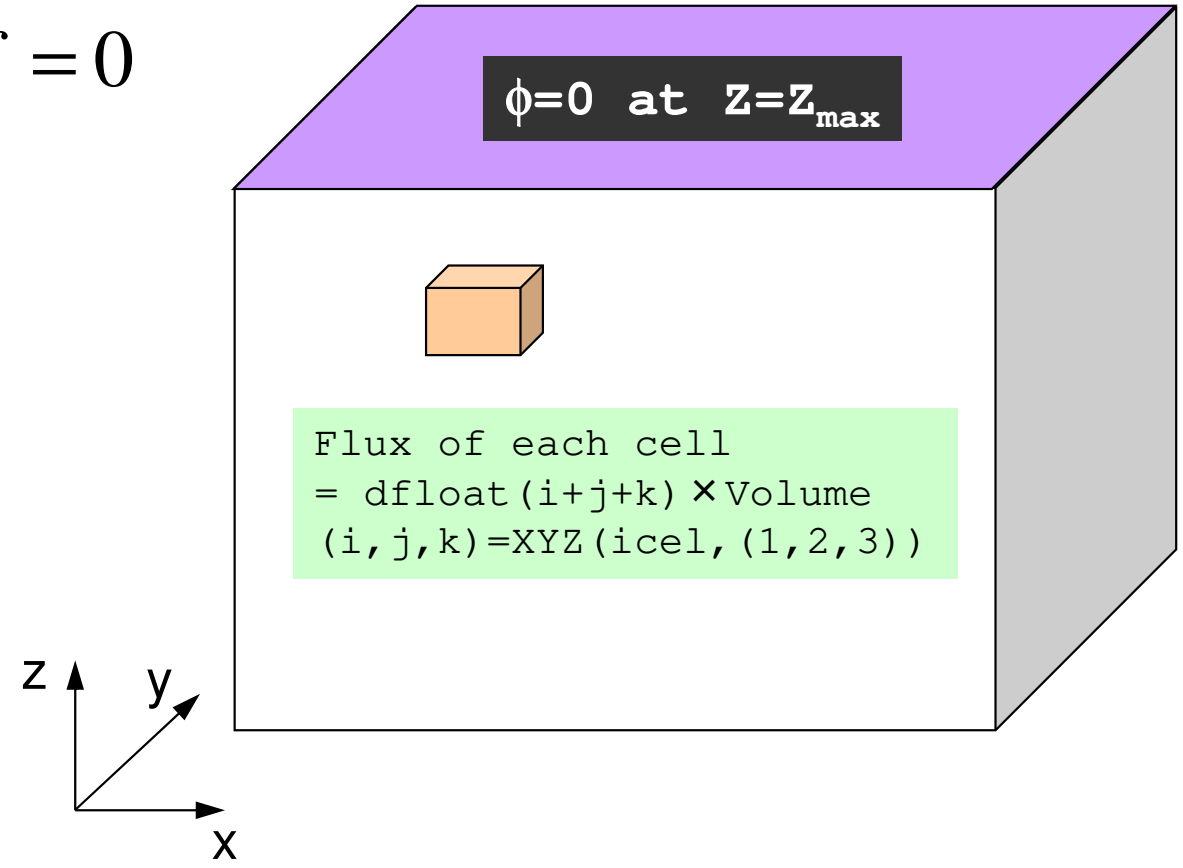
Target Problem: Variables are defined at cell-center's

Poisson Equation/ Poisson's Equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

Boundary Conditions (B.C.) etc.

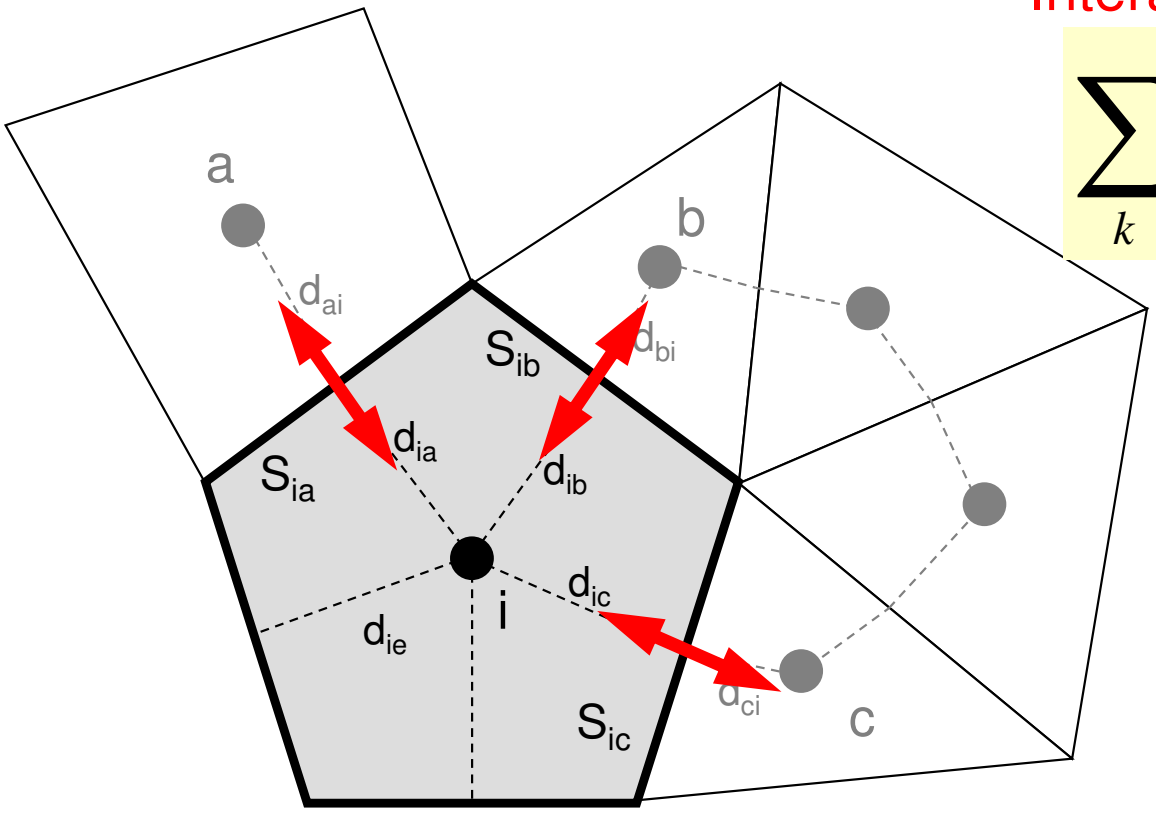
- Volume Flux
- $\phi = 0 @ Z = Z_{max}$



Poisson Equation by Finite Volume Method (FVM)

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

Conservation of Fluxes through Surfaces



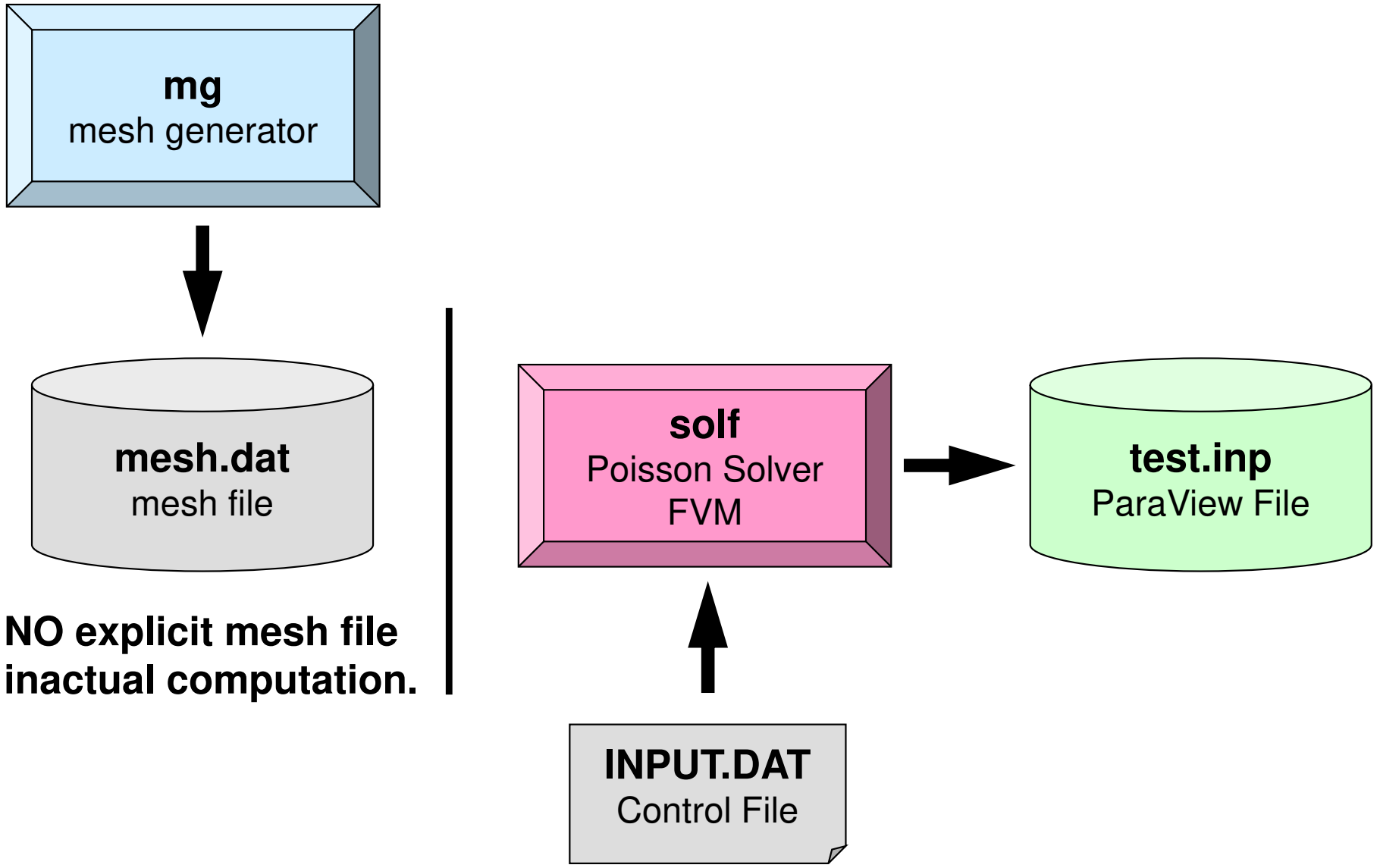
Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux

V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Running the Program: <$P-FVM>/run



Running the Program

Compiling

```
$> cd <$P-FVM>/run

$> cc -O mg.c -o mg
$> ls mg
     mg
```

Mesh Generator: **mg**

```
$> cd ../src-f
$> make
$> ls ../run/solf
     solf
```

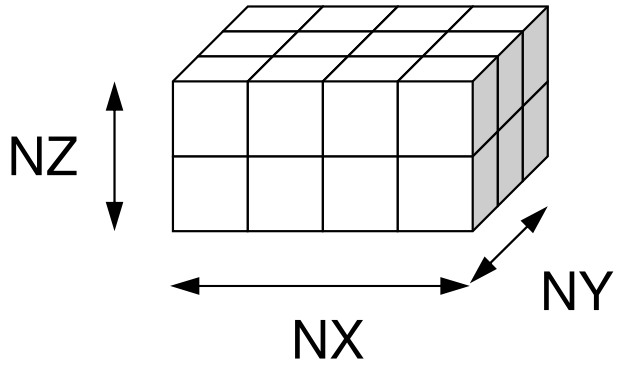
Poisson Solver (FVM): **solf**

Running the Program

Mesh Generation

```
$> cd ../run
$> ./mg
   4 3 2
$> ls mesh.dat
     mesh.dat
```

NX, NY, NZ



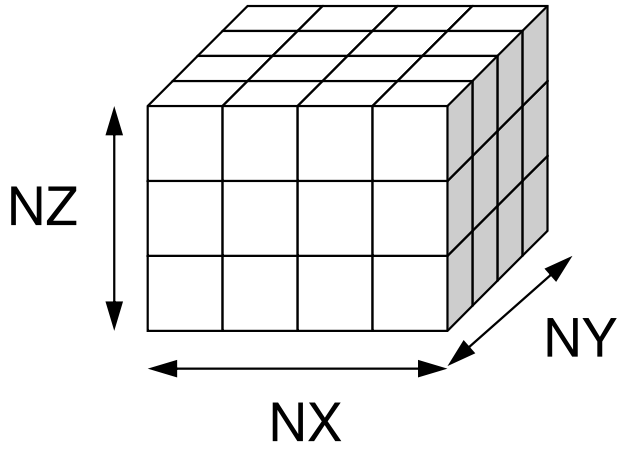
mesh.dat (1/5)

```
  4   3   2
 24
  1   0   2   0   5   0  13   1   1   1
  2   1   3   0   6   0  14   2   1   1
  3   2   4   0   7   0  15   3   1   1
  4   3   0   0   8   0  16   4   1   1
  5   0   6   1   9   0  17   1   2   1
  6   5   7   2  10   0  18   2   2   1
  7   6   8   3  11   0  19   3   2   1
  8   7   0   4  12   0  20   4   2   1
  9   0  10   5   0   0  21   1   3   1
 10   9  11   6   0   0  22   2   3   1
 11  10  12   7   0   0  23   3   3   1
 12  11   0   8   0   0  24   4   3   1
 13   0  14   0  17   1   0   1   1   2
 14  13  15   0  18   2   0   2   1   2
 15  14  16   0  19   3   0   3   1   2
 16  15   0   0  20   4   0   4   1   2
 17   0  18  13  21   5   0   1   2   2
 18  17  19  14  22   6   0   2   2   2
 19  18  20  15  23   7   0   3   2   2
 20  19   0  16  24   8   0   4   2   2
 21   0  22  17   0   9   0   1   3   2
 22  21  23  18   0  10   0   2   3   2
 23  22  24  19   0  11   0   3   3   2
 24  23   0  20   0  12   0   4   3   2
```

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (2/5)

```
 4   3   2
24
 1   0   2   0   5   0  13   1   1   1
 2   1   3   0   6   0  14   2   1   1
 3   2   4   0   7   0  15   3   1   1
 4   3   0   0   8   0  16   4   1   1
 5   0   6   1   9   0  17   1   2   1
 6   5   7   2  10   0  18   2   2   1
 7   6   8   3  11   0  19   3   2   1
 8   7   0   4  12   0  20   4   2   1
 9   0  10   5   0   0  21   1   3   1
10   9  11   6   0   0  22   2   3   1
11  10  12   7   0   0  23   3   3   1
12  11   0   8   0   0  24   4   3   1
13   0  14   0  17   1   0   1   1   2
14  13  15   0  18   2   0   2   1   2
15  14  16   0  19   3   0   3   1   2
16  15   0   0  20   4   0   4   1   2
17   0  18  13  21   5   0   1   2   2
18  17  19  14  22   6   0   2   2   2
19  18  20  15  23   7   0   3   2   2
20  19   0  16  24   8   0   4   2   2
21   0  22  17   0   9   0   1   3   2
22  21  23  18   0  10   0   2   3   2
23  22  24  19   0  11   0   3   3   2
24  23   0  20   0  12   0   4   3   2
```



Number of meshes in X/Y/Z directions

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (3/5)

Number of Meshes (Cells)
= NX x NY x NZ

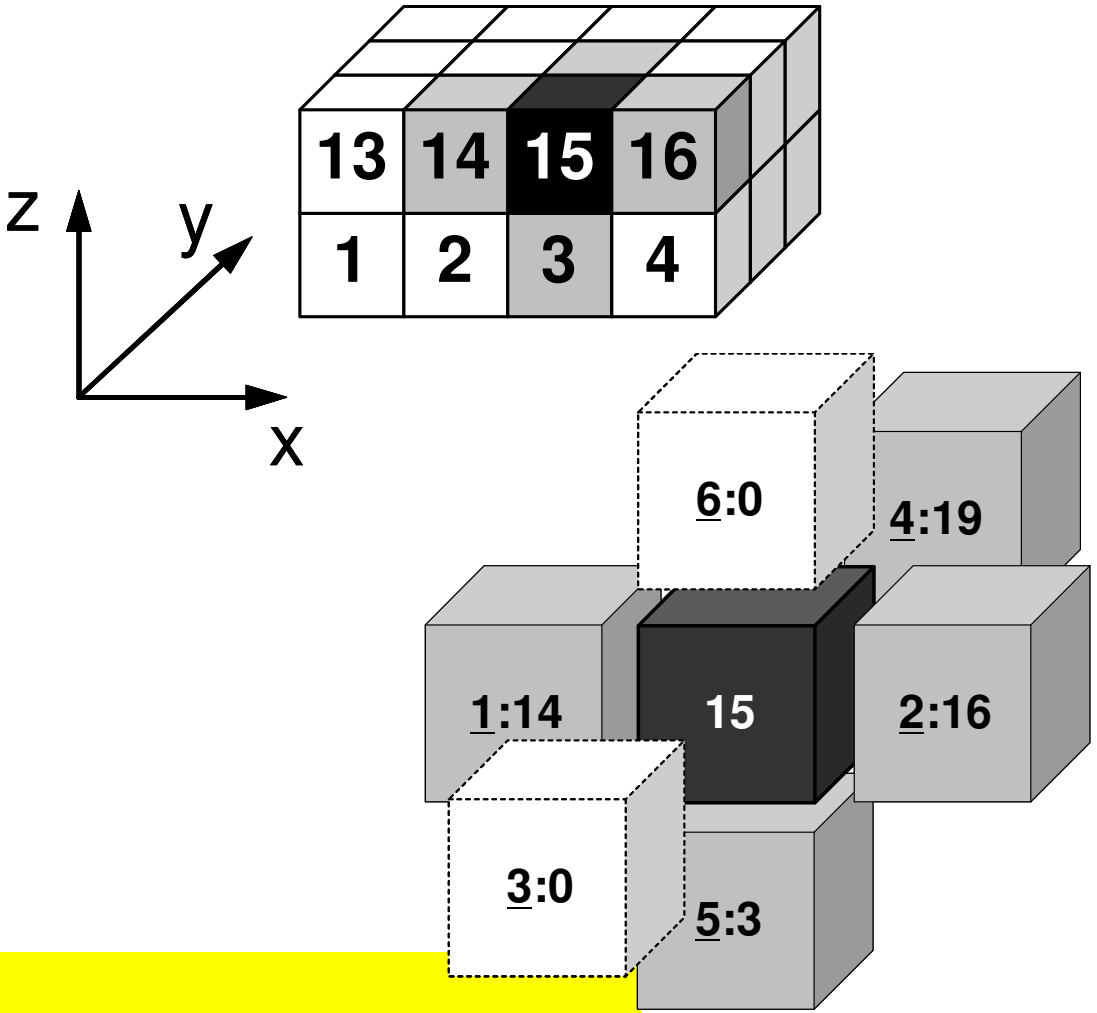
```
  4   3   2
24
  1   0   2   0   5   0  13   1   1   1
  2   1   3   0   6   0  14   2   1   1
  3   2   4   0   7   0  15   3   1   1
  4   3   0   0   8   0  16   4   1   1
  5   0   6   1   9   0  17   1   2   1
  6   5   7   2  10   0  18   2   2   1
  7   6   8   3  11   0  19   3   2   1
  8   7   0   4  12   0  20   4   2   1
  9   0  10   5   0   0  21   1   3   1
 10   9  11   6   0   0  22   2   3   1
 11  10  12   7   0   0  23   3   3   1
 12  11   0   8   0   0  24   4   3   1
 13   0  14   0  17   1   0   1   1   2
 14  13  15   0  18   2   0   2   1   2
 15  14  16   0  19   3   0   3   1   2
 16  15   0   0  20   4   0   4   1   2
 17   0  18  13  21   5   0   1   2   2
 18  17  19  14  22   6   0   2   2   2
 19  18  20  15  23   7   0   3   2   2
 20  19   0  16  24   8   0   4   2   2
 21   0  22  17   0   9   0   1   3   2
 22  21  23  18   0  10   0   2   3   2
 23  22  24  19   0  11   0   3   3   2
 24  23   0  20   0  12   0   4   3   2
```

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (4/5)

```
  4    3    2
 24
  1    0    2    0    5    0   13    1    1    1
  2    1    3    0    6    0   14    2    1    1
  3    2    4    0    7    0   15    3    1    1
  4    3    0    0    8    0   16    4    1    1
  5    0    6    1    9    0   17    1    2    1
  6    5    7    2   10    0   18    2    2    1
  7    6    8    3   11    0   19    3    2    1
  8    7    0    4   12    0   20    4    2    1
  9    0   10    5    0    0   21    1    3    1
 10    9   11    6    0    0   22    2    3    1
 11   10   12    7    0    0   23    3    3    1
 12   11    0    8    0    0   24    4    3    1
 13    0   14    0   17    1    0    1    1    2
 14   13   15    0   18    2    0    2    1    2
 15   14   16    0   19    3    0    3    1    2
 16   15    0    0   20    4    0    4    1    2
 17    0   18   13   21    5    0    1    2    2
 18   17   19   14   22    6    0    2    2    2
 19   18   20   15   23    7    0    3    2    2
 20   19    0   16   24    8    0    4    2    2
 21    0   22   17    0    9    0    1    3    2
 22   21   23   18    0   10    0    2    3    2
 23   22   24   19    0   11    0    3    3    2
 24   23    0   20    0   12    0    4    3    2
```

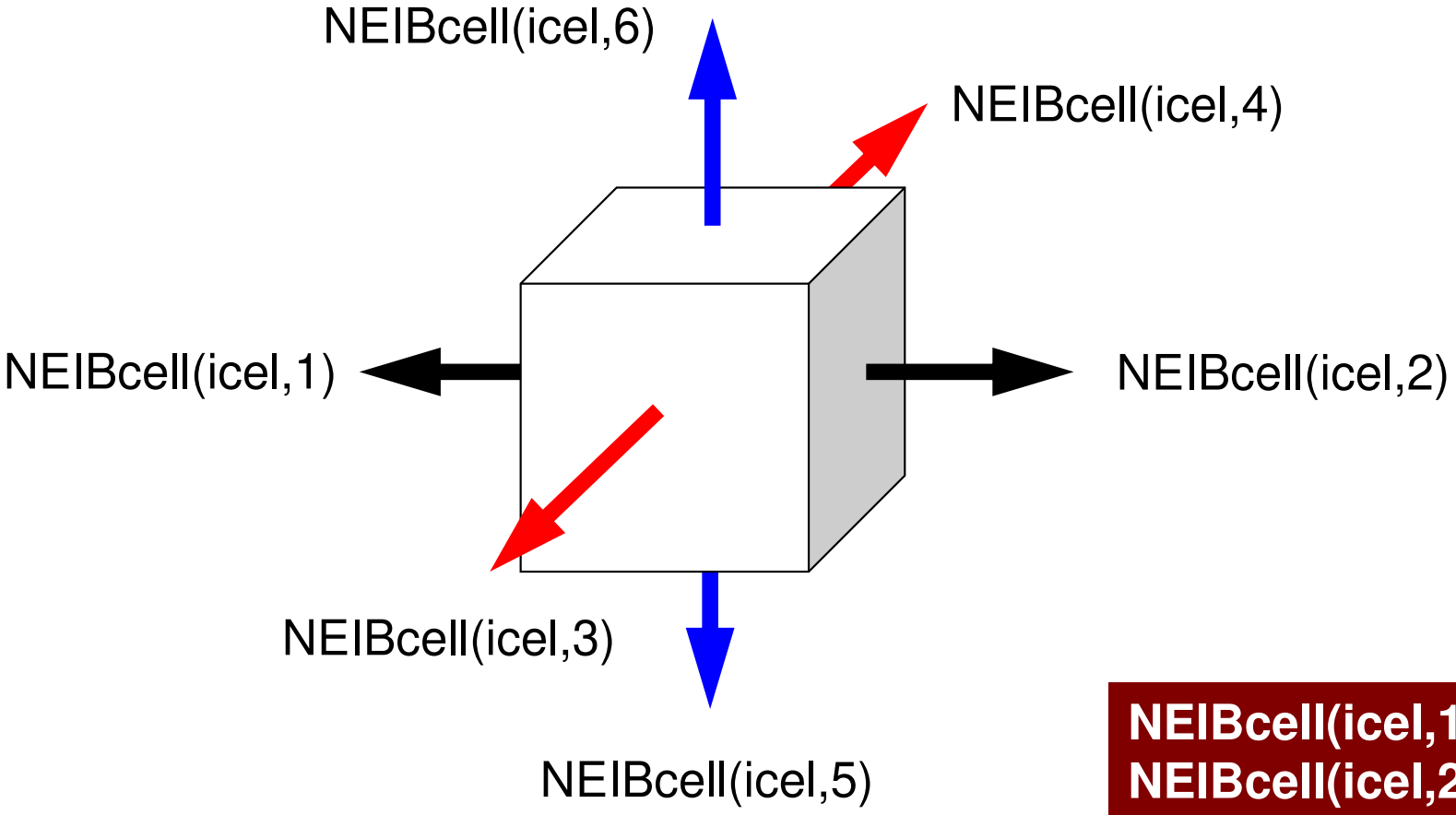
Neighboring Cells: NEIBcell(i,k)




```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

1st Col.: Global ID of the Cell

NEIBcell: ID of Neighboring Mesh/Cell =0: for Boundary Surface

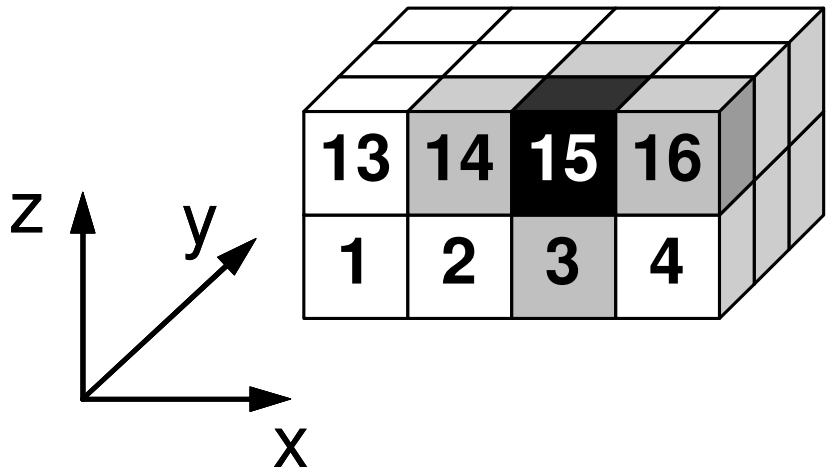


NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

mesh.dat (5/5)

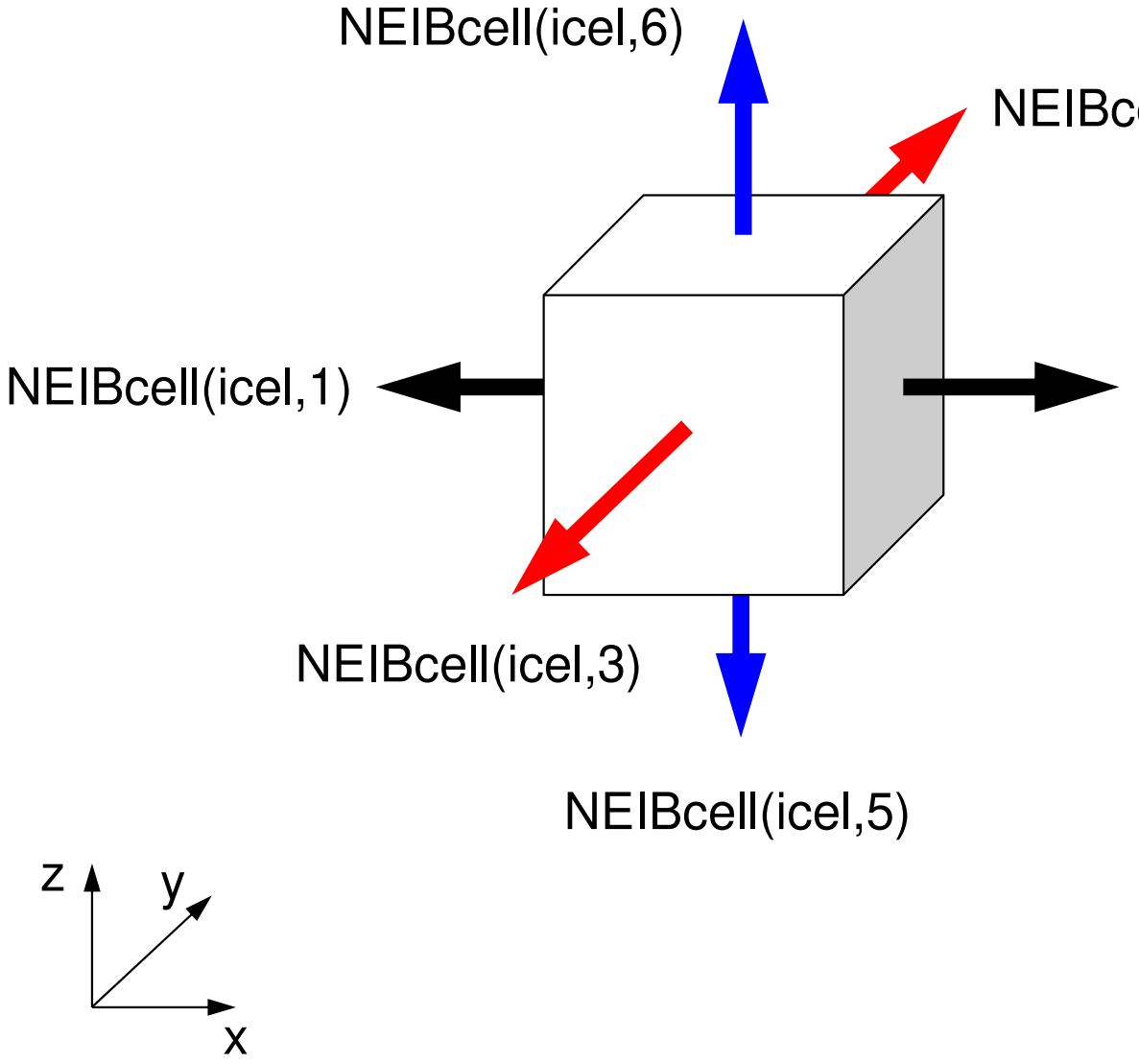
Location in X,Y,Z-directions: XYZ(i,j)

```
  4    3    2
 24
  1    0    2    0    5    0   13    1    1    1
  2    1    3    0    6    0   14    2    1    1
  3    2    4    0    7    0   15    3    1    1
  4    3    0    0    8    0   16    4    1    1
  5    0    6    1    9    0   17    1    2    1
  6    5    7    2   10    0   18    2    2    1
  7    6    8    3   11    0   19    3    2    1
  8    7    0    4   12    0   20    4    2    1
  9    0   10    5    0    0   21    1    3    1
 10    9   11    6    0    0   22    2    3    1
 11   10   12    7    0    0   23    3    3    1
 12   11    0    8    0    0   24    4    3    1
 13    0   14    0   17    1    0    1    1    2
 14   13   15    0   18    2    0    2    1    2
 15   14   16    0   19    3    0    3    1    2
 16   15    0    0   20    4    0    4    1    2
 17    0   18   13   21    5    0    1    2    2
 18   17   19   14   22    6    0    2    2    2
 19   18   20   15   23    7    0    3    2    2
 20   19    0   16   24    8    0    4    2    2
 21    0   22   17    0    9    0    1    3    2
 22   21   23   18    0   10    0    2    3    2
 23   22   24   19    0   11    0    3    3    2
 24   23    0   20    0   12    0    4    3    2
```




```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

NEIBcell: ID of Neighboring Mesh/Cell =0: for Boundary Surface



i= XYZ(icel,1)
j= XYZ(icel,2), k= XYZ(icel,3)
icel= (k-1)*NX*NY + (j-1)*NX + i

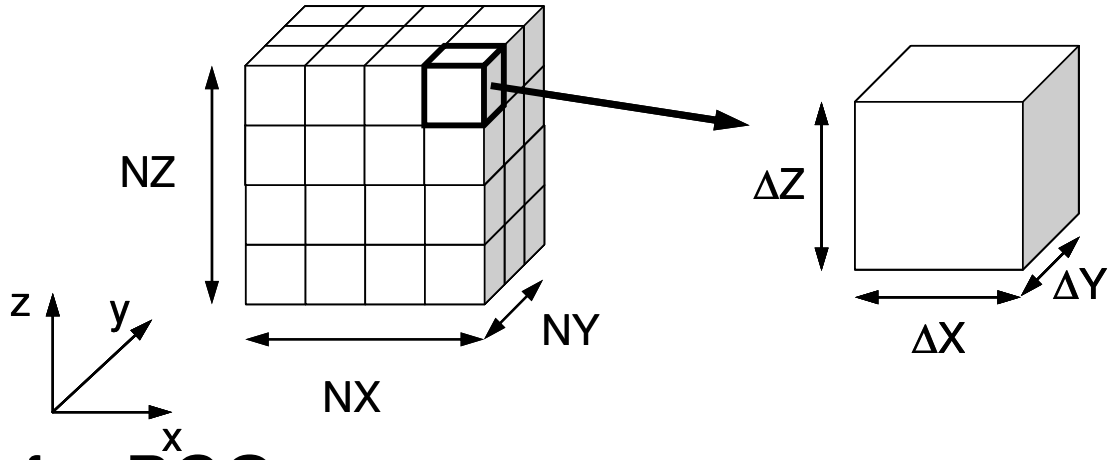
NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

Running the Program

Control Data: <$P-FVM>/run/INPUT.DAT

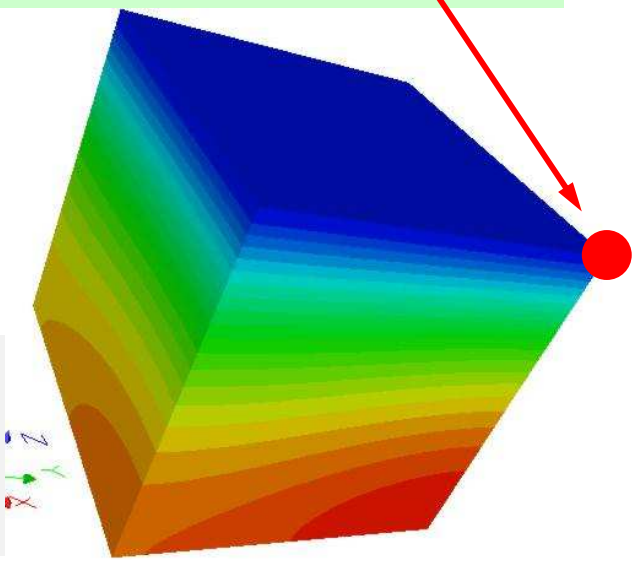
```
32 32 32                                    NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00                  DX/DY/DZ
1.0e-08                                     EPSICCG
```

- NX, NY, NZ
  - Number of meshes in X/Y/Z dir.
- DX, DY, DZ
  - Size of meshes
- EPSICCG
  - Convergence Criteria for PCG



Running the Program

```
$> cd <$P-FVM>/run
$> ./solf

    32   32   32          NX,NY,NZ
     1   4.409359E+00    Residual at the 1st Iteration
   101   1.807571E-02
   201   2.194680E-08
   208   9.354536E-09    Residual at convergence (<10^-8)

##ANSWER      32768     9.297409E+02   Result at ●-point

$> ls test.inp
   test.inp
```



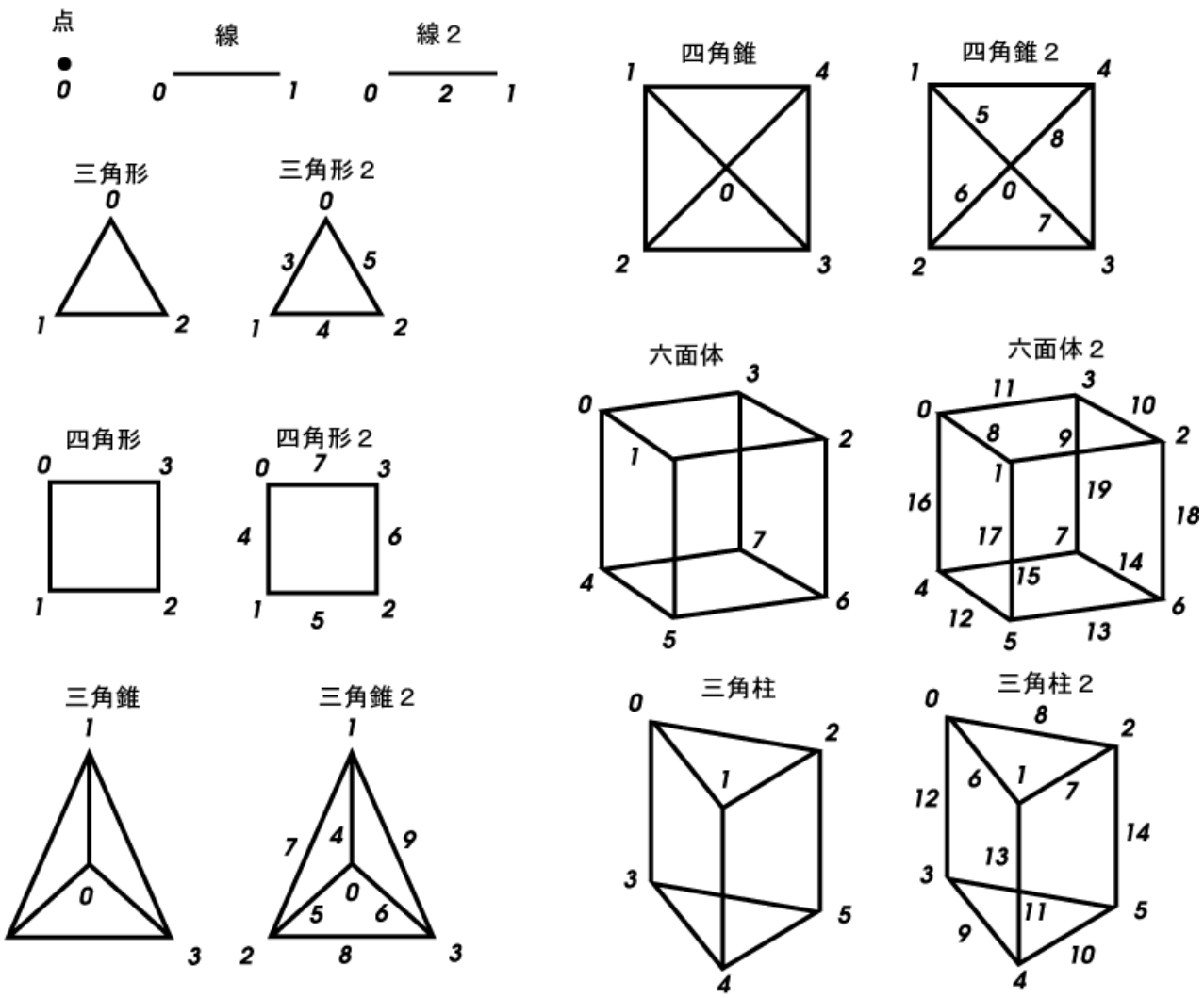
ParaView

http://www.paraview.org/

http://nkl.cc.u-tokyo.ac.jp/22s/ParaView.pdf

UCD Format (1/2)
Unstructured Cell Data

要素の種類	キーワード
点	pt
線	line
三角形	tri
四角形	quad
四面体	tet
角錐	pyr
三角柱	prism
六面体	hex
二次要素	
線2	line2
三角形2	tri2
四角形2	quad2
四面体2	tet2
角錐2	pyr2
三角柱2	prism2
六面体2	hex2



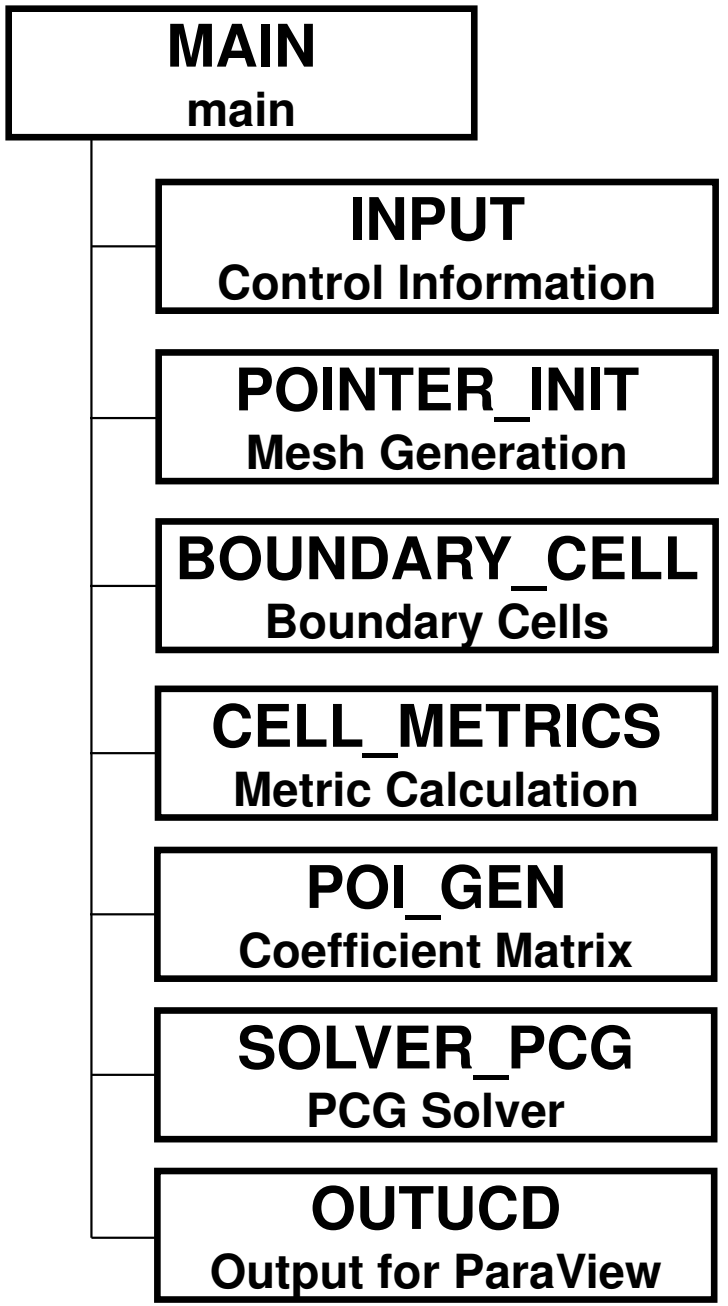
UCD Format (2/2)

- Originally for AVS, microAVS
- Extension of the UCD file is “inp”
- There are two types of formats. Only old type can be read by ParaView.

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **PCG Solver for Poisson's Equations**
  - How to run
    - Data Structure
  - **Program**
    - **Initialization**
    - **Coefficient Matrices**
    - PCG

Structure of the Program

```
program MAIN

use STRUCT
use PCG
use solver_PCG

implicit REAL*8 (A-H,O-Z)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
PHI=  0.d0

call solve_PCG  (…)

call OUTUCD

stop
end
```



struct.f: Variables/Arrays for FVM

```
 module STRUCT

      include 'precision.inc'
!C
!C-- METRICs & FLUX
      integer (kind=kint) :: ICELTOT, ICELTOTp, N
      integer (kind=kint) :: NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT
      integer (kind=kint) :: NXc, NYc, NZc

      real (kind=kreal) ::                                                &
     &      DX, DY, DZ, XAREA, YAREA, ZAREA, RDX, RDY, RDZ,              &
     &      RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ

      real (kind=kreal), dimension(:), allocatable ::                     &
     &      VOLCEL, VOLNOD, RVC, RVN

      integer (kind=kint), dimension(:,:), allocatable ::                 &
     &      XYZ, NEIBcell
!C
!C-- BOUNDARYs
      integer (kind=kint) :: ZmaxCELtot
      integer (kind=kint), dimension(:), allocatable :: BC_INDEX, BC_NOD
      integer (kind=kint), dimension(:), allocatable :: ZmaxCEL
!C
!C-- WORK
      integer (kind=kint), dimension(:,:), allocatable :: IWKX
      real(kind=kreal),    dimension(:,:), allocatable :: FCV

      end module STRUCT
```

struct.f: Variables/Arrays for FVM

Name	Type	Size	Content
NX, NY, NZ	I		Number of meshes in x/y/z directions
ICELTOT	I		Total number of meshes (NX x NY x NZ)
N	I		Total number of nodes (for visualization)
NXP1,NYP1, NZP1	I		Number of nodes in x/y/z directions (for visualization)
IBNODTOT	I		NXP1 × NYP1
XYZ	I	(ICELTOT,3)	Location of meshes
NEIBcell	I	(ICELTOT,6)	Neighboring meshes

pcg.f: Variables/Arrays for Sparse Matrix

```
module PCG

integer :: N2
integer :: NLUmax, NLU, METHOD
integer :: NPLU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AMAT

integer, dimension(:), allocatable :: INLU
integer, dimension(:), allocatable :: indexLU, itemLU

end module PCG
```

pcg.f: Variables/Arrays for Sparse Matrix

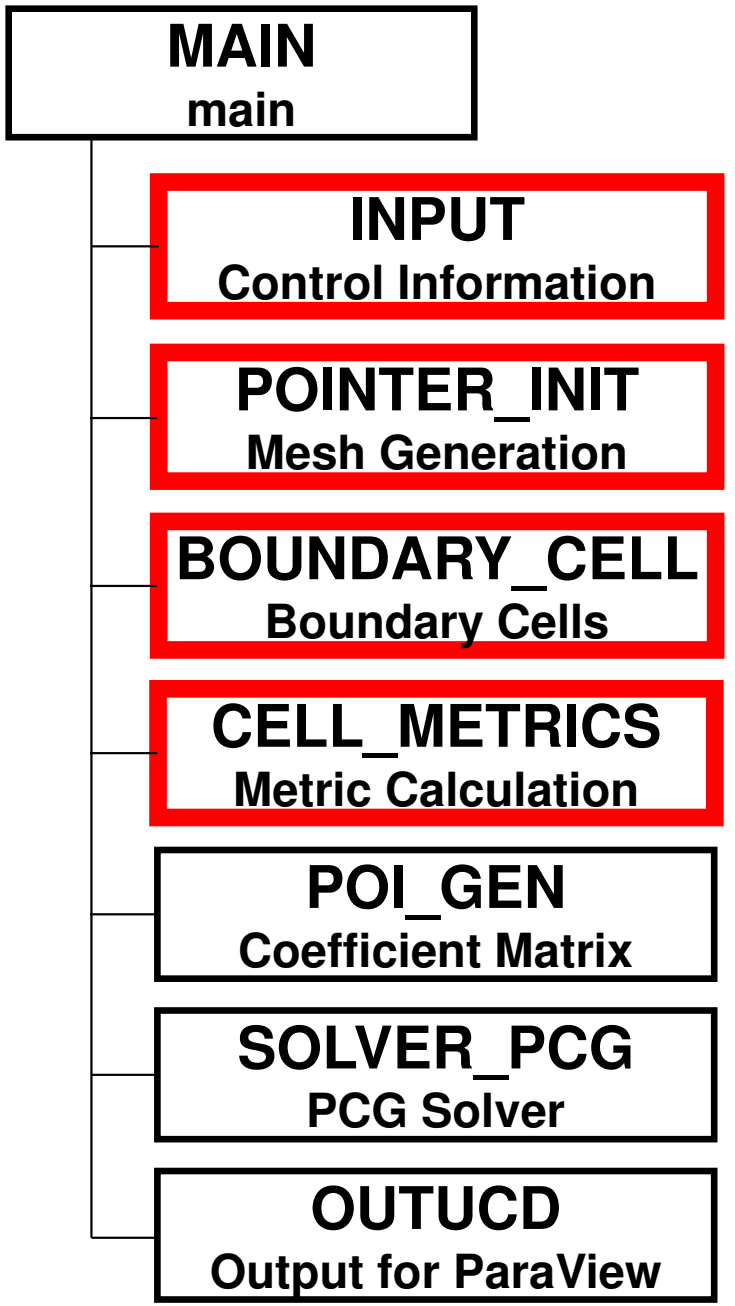
Name	Type	Size	Content
NLU	I		Maximum number of neighbors for each mesh (=6)
EPSICCG	R		Convergence Criteria for PCG
NPLU	I		Total number of non-zero off-diagonal components (CRS)
indexLU	I	(0:ICELTOT)	Number of non-zero off-diagonal components (CRS)
itemLU	I	(NPLU)	Column ID of non-zero off-diagonal components (CRS)
PHI	R	(ICELTOT)	Unknown vector
BFORCE	R	(ICELTOT)	RHS vector
D	R	(ICELTOT)	Diagonal components of the matrix
AMAT	R	(NPLU)	Non-zero off-diagonal components of the matrix (CRS)
INLU	I	(ICELTOT)	Number of non-zero off-diagonal components for each mesh, to be used for calculation of NPLU

Mat-Vec Multiplication: {q}=[A]{p}

```
do i= 1, N

  VAL= D(i)*p(i)

  do k= indexLU(i-1)+1, indexLU(i)
    VAL= VAL + AMAT(k)*p(itemLU(k))
  enddo

  q(i)= VAL

enddo
```

Structure of the Program

```
program MAIN

use STRUCT
use PCG
use solver_PCG

implicit REAL*8 (A-H,O-Z)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
PHI=  0.d0

call solve_PCG  (…)

call OUTUCD

stop
end
```



input: reading “INPUT.DAT”

```
      subroutine INPUT
      use STRUCT
      use PCG

      implicit REAL*8 (A-H,O-Z)

      character*80 CNTFIL

      open   (11, file='INPUT.DAT', status='unknown')
        read (11,*) NX, NY, NZ
        read (11,*) DX, DY, DZ
        read (11,*) EPSICCG
      close (11)

      return
      end
```

```
32 32 32                                NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

pointer_init (1/3): “mesh.dat”

```
!C
!C***
!C*** POINTER_INIT
!C***
!C
      subroutine POINTER_INIT

      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)

!C
!C +----------------------+
!C | Generating MESH info. |
!C +----------------------+
!C===
      ICELTOT= NX  * NY  * NZ

      NXP1= NX + 1
      NYP1= NY + 1
      NZP1= NZ + 1

      allocate (NEIBcell(ICELTOT,6), XYZ(ICELTOT,3))
      NEIBcell= 0
```

NX,NY,NZ:

Number of meshes in x/y/z directions

NXP1,NYP1,NZP1:

Number of nodes in x/y/z directions (for visualization)

ICELTOT:

Number of meshes (NX x NY x NZ)

XYZ(ICELTOT,3):

Location of meshes

NEIBcell(ICELTOT,6):

Neighboring meshesc

allocate, deallocate (1/2)

Same interface with FORTRAN

```
#include <stdio.h>
#include <stdlib.h>

void* allocate_vector(int size,int m)
{
  void *a;
  if ( ( a=(void * )malloc( m * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in vector ¥n");
     exit(1);
  }
  return a;
}

void deallocate_vector(void *a)
{
  free( a );
}
```

```
VOLCEL = (double *)allocate_vector(sizeof(double),ICELTOT);
indexLU= (int *)allocate_vector(sizeof(int),ICELTOT+1);
```

allocate, deallocate (2/2)

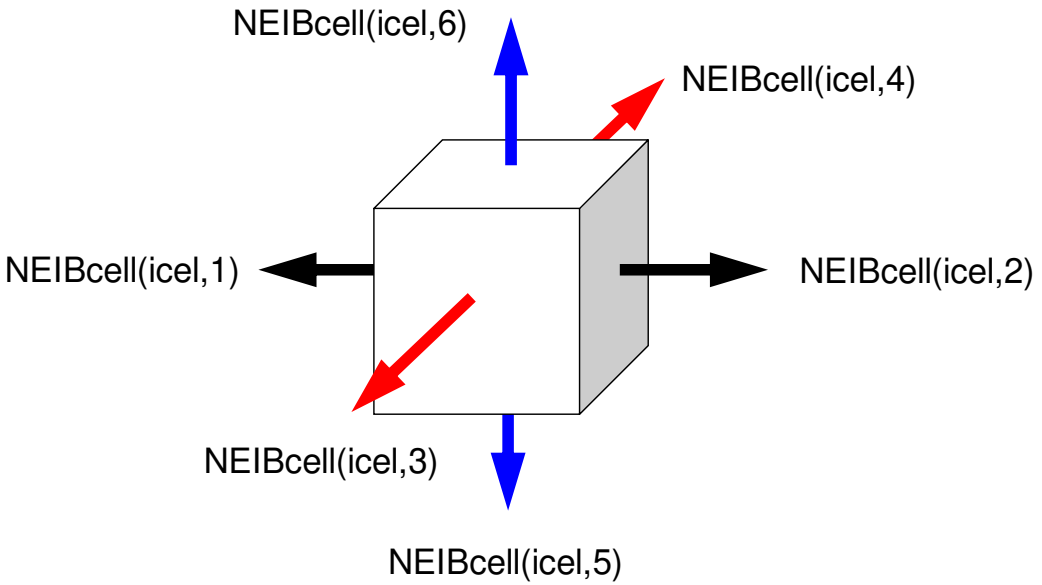
Same interface with FORTRAN

```
void** allocate_matrix(int size,int m,int n)
{
  void **aa;
  int i;
  if ( ( aa=(void ** )malloc( m * sizeof(void*) ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! aa in matrix ¥n");
     exit(1);
  }
  if ( ( aa[0]=(void * )malloc( m * n * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in matrix ¥n");
     exit(1);
  }
        for(i=1;i<m;i++) aa[i]=(char*)aa[i-1]+size*n;
  return aa;
}

void deallocate_matrix(void **aa)
{
        free( aa );
}
```

```
NEIBcell = (int **)allocate_matrix(sizeof(int),ICELTOT,6);
XYZ      = (int **)allocate_matrix(sizeof(int),ICELTOT,3);
```

pointer_init (2/3): “mesh.dat”

```
      do k= 1, NZ
        do j= 1, NY
          do i= 1, NX
            icel= (k-1)*NX*NY + (j-1)*NX + i
            NEIBcell(icel,1)= icel - 1
            NEIBcell(icel,2)= icel + 1
            NEIBcell(icel,3)= icel - NX
            NEIBcell(icel,4)= icel + NX
            NEIBcell(icel,5)= icel - NX*NY
            NEIBcell(icel,6)= icel + NX*NY
            if (i.eq. 1) NEIBcell(icel,1)= 0
            if (i.eq.NX) NEIBcell(icel,2)= 0
            if (j.eq. 1) NEIBcell(icel,3)= 0
            if (j.eq.NY) NEIBcell(icel,4)= 0
            if (k.eq. 1) NEIBcell(icel,5)= 0
            if (k.eq.NZ) NEIBcell(icel,6)= 0

            XYZ(icel,1)= i
            XYZ(icel,2)= j
            XYZ(icel,3)= k

          enddo
        enddo
      enddo
!C===
```



i= XYZ(icel,1)
j= XYZ(icel,2), k= XYZ(icel,3)
icel= (k-1)*NX*NY + (j-1)*NX + i

NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

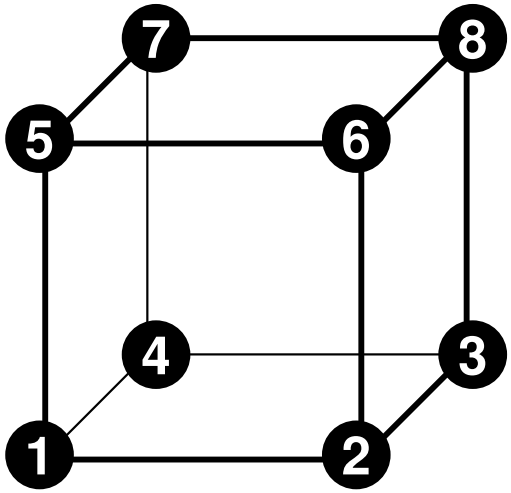
pointer_init (3/3): “mesh.dat”

```
!C
!C +------------+
!C | Parameters |
!C +------------+
!C===
      if (DX.le.0.0e0) then
        DX= 1.d0 / dfloat(NX)
        DY= 1.d0 / dfloat(NY)
        DZ= 1.d0 / dfloat(NZ)
      endif

      NXP1= NX + 1
      NYP1= NY + 1
      NZP1= NZ + 1

      IBNODTOT= NXP1 * NYP1
      N       = NXP1 * NYP1 * NZP1
!C===
      return
      end
```

if DX is no larger than 0.0

pointer_init (3/3): “mesh.dat”

```
!C
!C +------------+
!C | Parameters |
!C +------------+
!C===
      if (DX.le.0.0e0) then
        DX= 1.d0 / dfloat(NX)
        DY= 1.d0 / dfloat(NY)
        DZ= 1.d0 / dfloat(NZ)
      endif

      NXP1= NX + 1
      NYP1= NY + 1
      NZP1= NZ + 1

      IBNODTOT= NXP1 * NYP1
      N       = NXP1 * NYP1 * NZP1
!C===
      return
      end
```



NXP1,NYP1,NZP1:

Number of nodes in x/y/z directions

IBNODTOT:

= NXP1 x NYP1

N:

Number of modes meshes (for visualization)

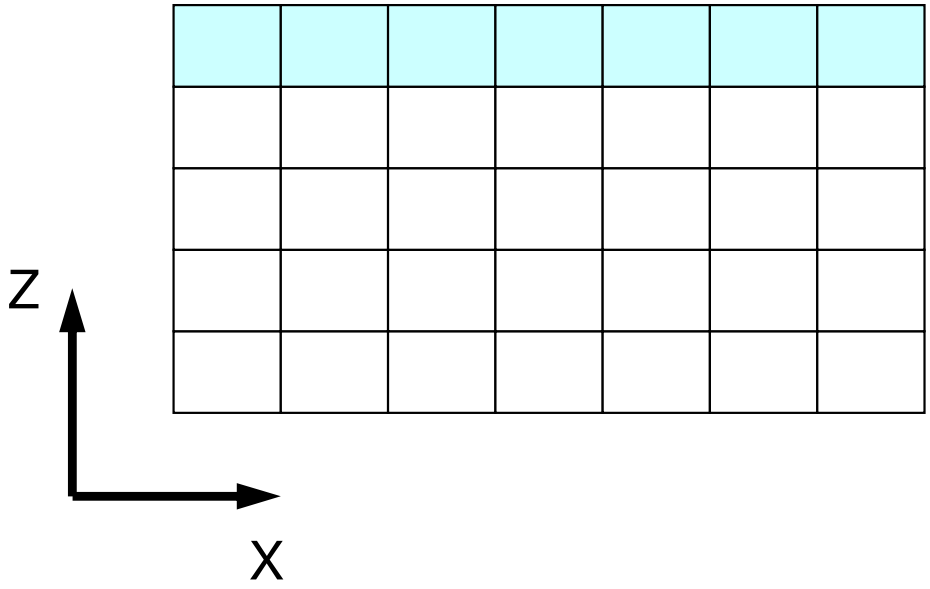
boundary_cell

```
!C
!C***
!C*** BOUNDARY_CELL
!C***
!C
      subroutine BOUNDARY_CELL
      use STRUCT

      implicit REAL*8 (A-H,O-Z)

!C
!C +------+
!C | Zmax |
!C +------+
!C===
      IFACTOT= NX * NY
      ZmaxCELtot= IFACTOT

      allocate (ZmaxCEL(ZmaxCELtot))

      icou= 0
      k   = NZ
      do j= 1, NY
      do i= 1, NX
        icel= (k-1)*IFACTOT + (j-1)*NX + i
        icou= icou + 1
        ZmaxCEL(icou)= icel
      enddo
      enddo
!C===
      return
      end
```

Meshes @ $Z=Z_{max}$

Number: ZmaxCELtot

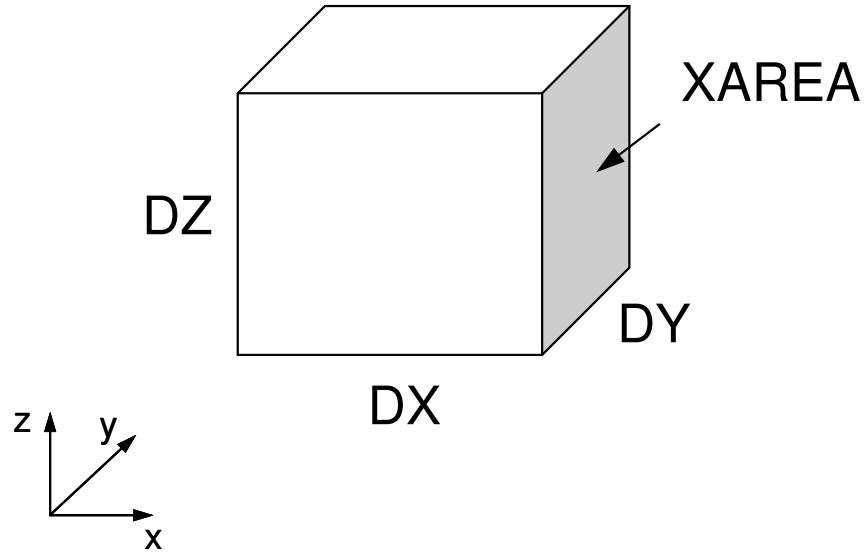
Mesh ID: ZmaxCEL(:)



cell_metrics

```
!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))
!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1.d0 / DX
      RDY= 1.d0 / DY
      RDZ= 1.d0 / DZ

      RDX2= 1.d0 / (DX**2)
      RDY2= 1.d0 / (DY**2)
      RDZ2= 1.d0 / (DZ**2)
      R2DX= 1.d0 / (0.50d0*DX)
      R2DY= 1.d0 / (0.50d0*DY)
      R2DZ= 1.d0 / (0.50d0*DZ)

       VO= DX * DY * DZ
      RVO= 1.d0/VO
      VOLCEL=  VO
      RVC   = RVO

      return
      end
```

Parameters for Computations



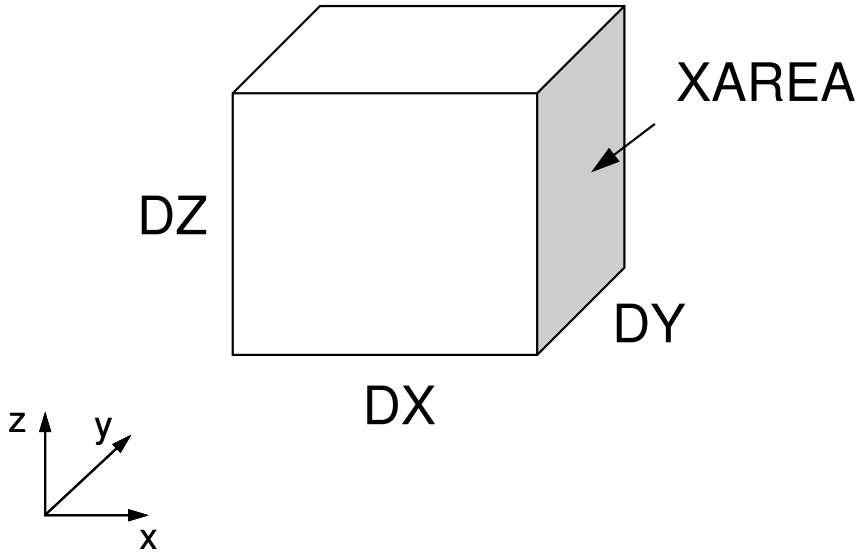
$$XAREA = \Delta Y \times \Delta Z, \quad YAREA = \Delta Z \times \Delta X,$$
$$ZAREA = \Delta X \times \Delta Y$$

$$RDX = \frac{1}{\Delta X}, \quad RDY = \frac{1}{\Delta Y}, \quad RDZ = \frac{1}{\Delta Z}$$

cell_metrics

```
!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))
!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1.d0 / DX
      RDY= 1.d0 / DY
      RDZ= 1.d0 / DZ

      RDX2= 1.d0 / (DX**2)
      RDY2= 1.d0 / (DY**2)
      RDZ2= 1.d0 / (DZ**2)
      R2DX= 1.d0 / (0.50d0*DX)
      R2DY= 1.d0 / (0.50d0*DY)
      R2DZ= 1.d0 / (0.50d0*DZ)

       VO= DX * DY * DZ
      RVO= 1.d0/VO
      VOLCEL=  VO
      RVC   = RVO

      return
      end
```

Parameters for Computations



$$RDX2 = \frac{1}{\Delta X^2}, \quad RDY2 = \frac{1}{\Delta Y^2}, \quad RDZ2 = \frac{1}{\Delta Z^2}$$

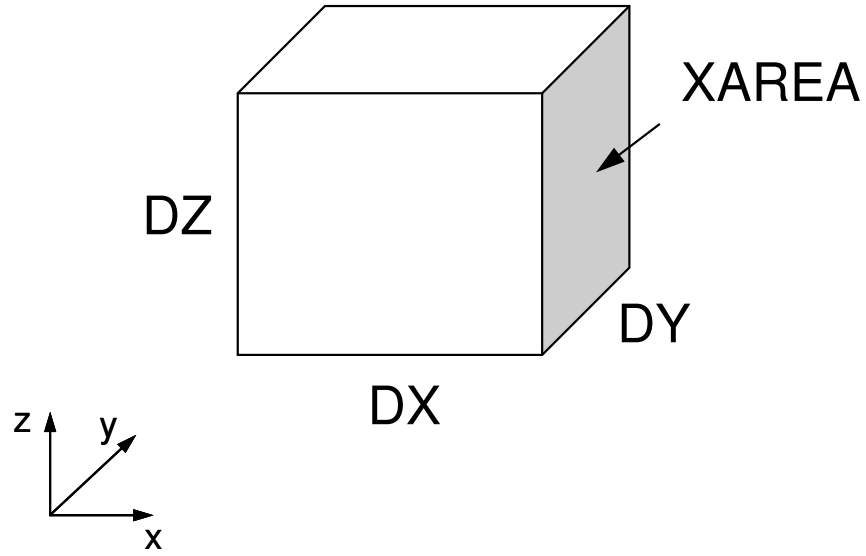
$$R2DX = \frac{1}{0.5 \times \Delta X}, \quad R2DY = \frac{1}{0.5 \times \Delta Y},$$

$$R2DZ = \frac{1}{0.5 \times \Delta Z}$$

cell_metrics

```
!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))
!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1.d0 / DX
      RDY= 1.d0 / DY
      RDZ= 1.d0 / DZ

      RDX2= 1.d0 / (DX**2)
      RDY2= 1.d0 / (DY**2)
      RDZ2= 1.d0 / (DZ**2)
      R2DX= 1.d0 / (0.50d0*DX)
      R2DY= 1.d0 / (0.50d0*DY)
      R2DZ= 1.d0 / (0.50d0*DZ)

       V0= DX * DY * DZ
      RV0= 1.d0/V0
      VOLCEL=  V0
      RVC   = RV0

      return
      end
```

Parameters for Computations

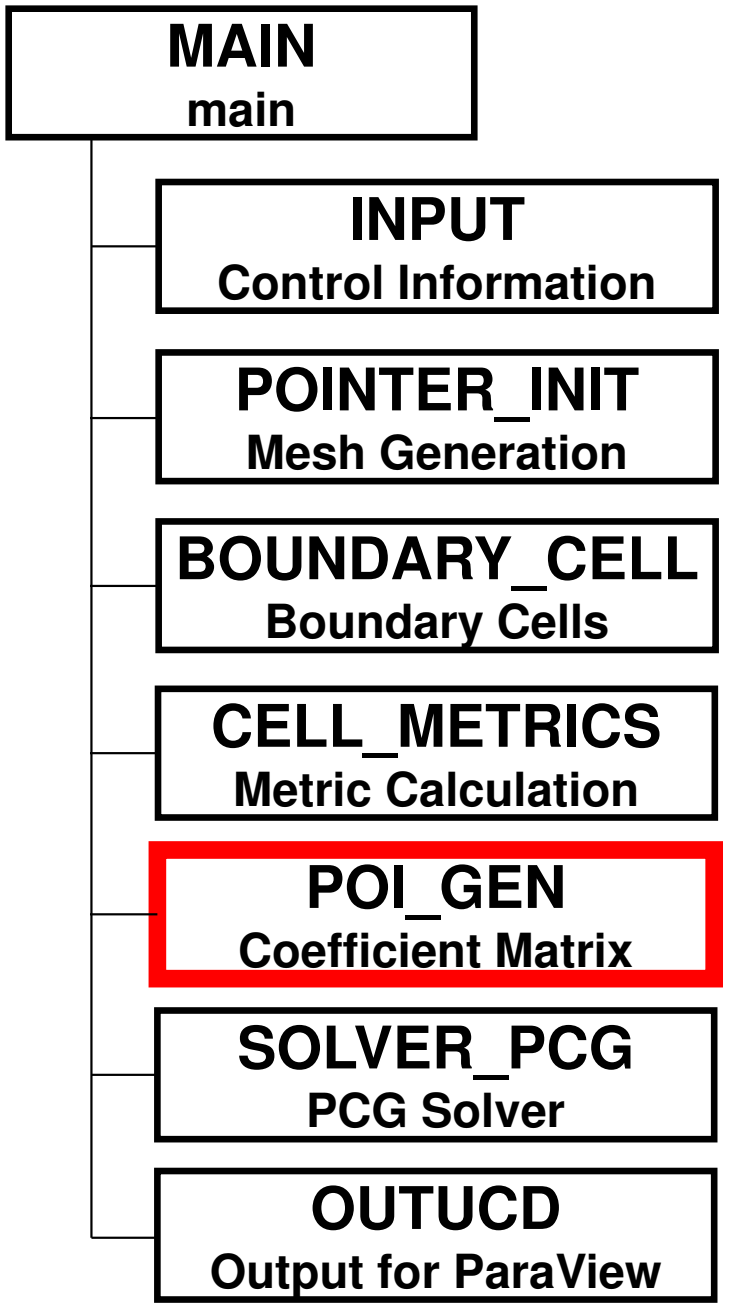


$$VOLCEL = V0 = \Delta X \times \Delta Y \times \Delta Z$$

$$RV0 = RVC = \frac{1}{VOLCEL}$$

Structure of the Program

```
program MAIN

use STRUCT
use PCG
use solver_PCG

implicit REAL*8 (A-H,O-Z)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
PHI=  0.d0

call solve_PCG  (…)

call OUTUCD

stop
end
```



poi_gen (1/6)

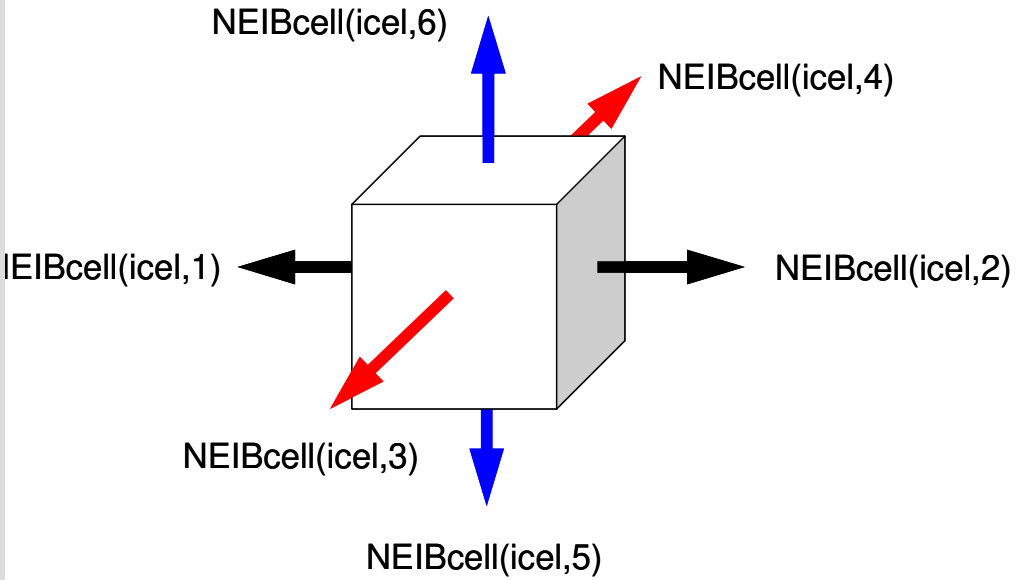
```
      use STRUCT
      use PCG

      implicit REAL*8 (A-H,O-Z)

!C
!C-- INIT.
      nn = ICELTOT

      NLU= 6

      allocate (BFORCE(nn), D(nn), PHI(nn))
      allocate (INLU(nn))

      PHI   = 0.d0
      BFORCE= 0.d0
        D   = 0.d0

      INLU= 0
```

pcg.f: Variables/Arrays for Sparse Matrix

Name	Type	Size	Content
NLU	I		Maximum number of neighbors for each mesh (=6)
EPSICCG	R		Convergence Criteria for PCG
NPLU	I		Total number of non-zero off-diagonal components (CRS)
indexLU	I	(0:ICELTOT)	Number of non-zero off-diagonal components (CRS)
itemLU	I	(NPLU)	Column ID of non-zero off-diagonal components (CRS)
PHI	R	(ICELTOT)	Unknown vector
BFORCE	R	(ICELTOT)	RHS vector
D	R	(ICELTOT)	Diagonal components of the matrix
AMAT	R	(NPLU)	Non-zero off-diagonal components of the matrix (CRS)
INLU	I	(ICELTOT)	Number of non-zero off-diagonal components for each mesh, to be used for calculation of NPLU

```
      do icel= 1, ICELTOT
        icN1= NEIBcell(icel,1)
        icN2= NEIBcell(icel,2)
        icN3= NEIBcell(icel,3)
        icN4= NEIBcell(icel,4)
        icN5= NEIBcell(icel,5)
        icN6= NEIBcell(icel,6)

        if (icN5.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif

        if (icN3.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif

        if (icN1.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif

        if (icN2.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif

        if (icN4.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif

        if (icN6.ne.0) then
          INLU(icel)= INLU(icel) + 1
        endif
```

poi_gen(2/6)



Lower Triangular Components

NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,1)= icel – 1

Upper Triangular Components

NEIBcell(icel,2)= icel + 1
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,6)= icel + NX*NY

poi_gen (3/6)

```
!C
!C-- 1D array
      allocate (indexLU(0:nn))
      indexLU= 0

      do icel= 1, ICELTOT
        indexLU(icel)= INLU(icel)
      enddo

      do icel= 1, ICELTOT
        indexLU(icel)= indexLU(icel) + indexLU(icel-1)
      enddo

      NPLU= indexLU(ICELTOT)

      allocate (itemLU(NPLU), AMAT(NPLU))
      itemLU= 0
      AMAT  = 0.d0
```

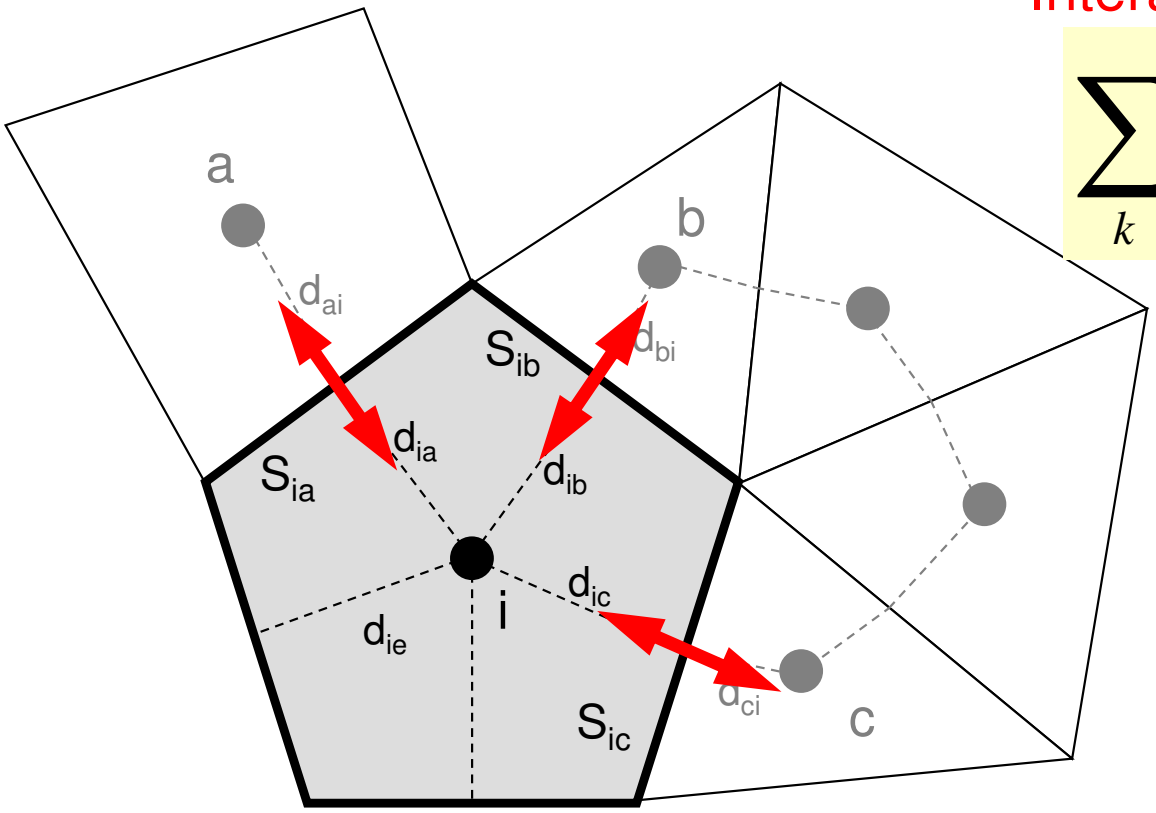
```
do i= 1, N
  VAL= D(i)*p(i)
  do k= indexLU(i-1)+1, indexLU(i)
    VAL= VAL + AMAT(k)*p(itemLU(k))
  enddo
  q(i)= VAL
enddo
```

Name	Type	Size	Content
NLU	I		Maximum number of neighbors for each mesh (=6)
EPSICCG	R		Convergence Criteria for PCG
NPLU	I		Total number of non-zero off-diagonal components (CRS)
indexLU	I	(0:ICELTOT)	Number of non-zero off-diagonal components (CRS)
itemLU	I	(NPLU)	Column ID of non-zero off-diagonal components (CRS)
PHI	R	(ICELTOT)	Unknown vector
BFORCE	R	(ICELTOT)	RHS vector
D	R	(ICELTOT)	Diagonal components of the matrix
AMAT	R	(NPLU)	Non-zero off-diagonal components of the matrix (CRS)
INLU	I	(ICELTOT)	Number of non-zero off-diagonal components for each mesh, to be used for calculation of NPLU

Poisson Equation by Finite Volume Method (FVM)

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

Conservation of Fluxes through Surfaces



Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux

V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Constructing Coefficient Matrix

Conservation for i-th mesh

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

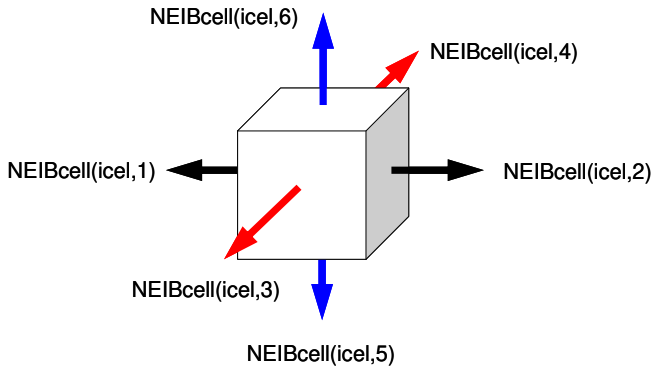
$$+\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_i=-V_i\dot{Q}_i$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AMAT (off-diag.)** **BFORCE (RHS)**

poi_gen(4/6)

```
do icel= 1, ICELTOT
  icN1= NEIBcell(icel,1)
  icN2= NEIBcell(icel,2)
  icN3= NEIBcell(icel,3)
  icN4= NEIBcell(icel,4)
  icN5= NEIBcell(icel,5)
  icN6= NEIBcell(icel,6)
  VOL0= VOLCEL(icel)
  icou= 0
  if (icN5.ne.0) then
    coef   =RDZ * ZAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
         k= icou + indexLU(icel-1)
    itemLU(k)= icN5
      AMAT(k)= coef
  endif

  if (icN3.ne.0) then
    coef   =RDY * YAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
         k= icou + indexLU(icel-1)
    itemLU(k)= icN3
      AMAT(k)= coef
  endif
  if (icN1.ne.0) then
    coef   =RDX * xAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
         k= icou + indexLU(icel-1)
    itemLU(k)= icN1
      AMAT(k)= coef
  endif
```

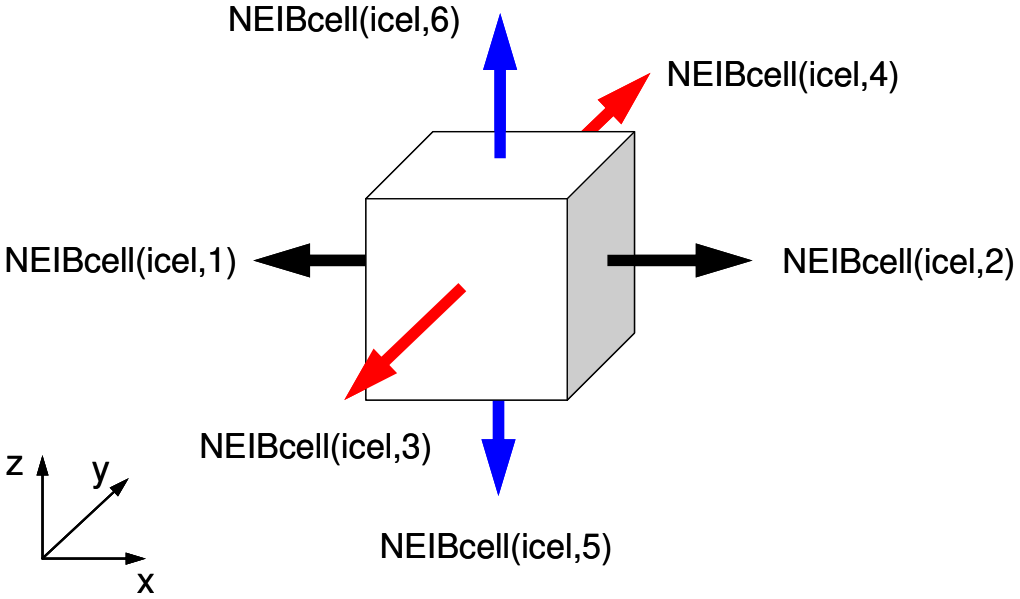


$$\frac{\phi_{neib(icel,1)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib(icel,2)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib(icel,3)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib(icel,4)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib(icel,5)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib(icel,6)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

Calculations of Coefficients



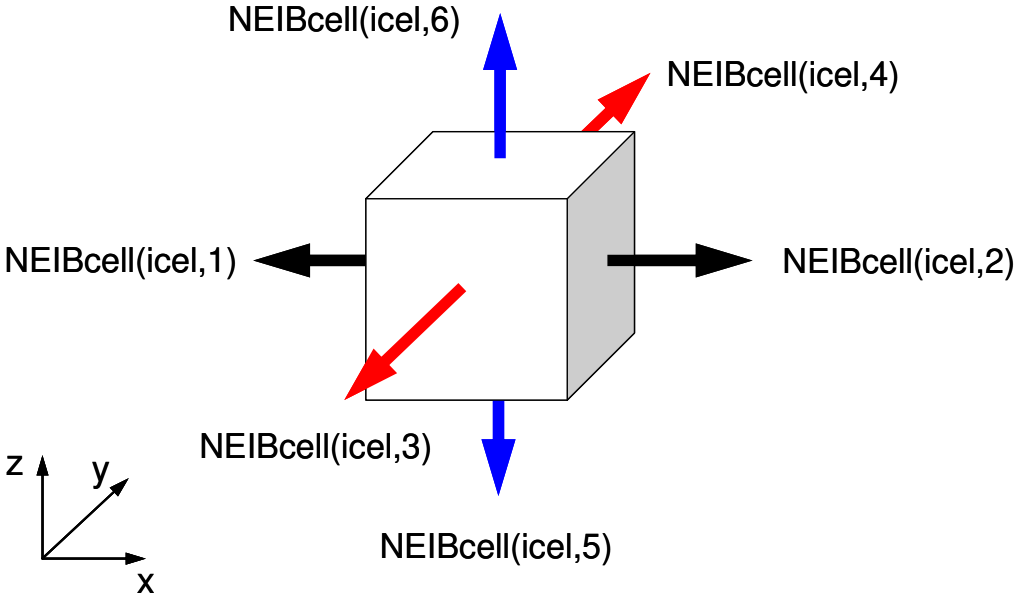

```
if (icN5.ne.0) then
    coef   =RDZ * ZAREA
    D(icel)= D(icel) - coef
    icou= icou + 1
    k= icou + indexLU(icel-1)

    itemLU(k)= icN5
      AMAT(k)= coef
  endif
```

$$\frac{\phi_{neib(icel,1)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib(icel,2)}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib(icel,3)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib(icel,4)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib(icel,5)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib(icel,6)}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

Calculations of Coefficients




```
if (icN5.ne.0) then
    coef   =RDZ * ZAREA
    D(icel)= D(icel) - coef
    icou= icou + 1
    k= icou + indexLU(icel-1)

    itemLU(k)= icN5
      AMAT(k)= coef
  endif
```

$$\frac{\Delta y\Delta z}{\Delta x}\left(\phi_{neib(icel,1)}-\phi_{icel}\right)+\frac{\Delta y\Delta z}{\Delta x}\left(\phi_{neib(icel,2)}-\phi_{icel}\right)+$$

$$\frac{\Delta z\Delta x}{\Delta y}\left(\phi_{neib(icel,3)}-\phi_{icel}\right)+\frac{\Delta z\Delta x}{\Delta y}\left(\phi_{neib(icel,4)}-\phi_{icel}\right)+$$

$$\frac{\Delta x\Delta y}{\Delta z}\left(\phi_{neib(icel,5)}-\phi_{icel}\right)+\frac{\Delta x\Delta y}{\Delta z}\left(\phi_{neib(icel,6)}-\phi_{icel}\right)+f_{icel}\Delta x\Delta y\Delta z=0$$

ZAREA

RDZ

poi_gen(5/6)

```
do icel= 1, ICELTOT
(…)
  if (icN2.ne.0) then
    coef   =RDX * xAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN2
      AMAT(k)= coef
  endif
  if (icN4.ne.0) then
    coef   =RDY * YAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN4
      AMAT(k)= coef
  endif
  if (icN6.ne.0) then
    coef   =RDZ * ZAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN6
      AMAT(k)= coef
  endif

  ii= XYZ(icel,1)
  jj= XYZ(icel,2)
  kk= XYZ(icel,3)

  BFORCE(icel)= -dfloat(ii+jj+kk)*VOLO

enddo
```

```
do icel= 1, ICELTOT
(…)
  if (icN2.ne.0) then
    coef   =RDX * xAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN2
      AMAT(k)= coef
  endif
  if (icN4.ne.0) then
    coef   =RDY * YAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN4
      AMAT(k)= coef
  endif
  if (icN6.ne.0) then
    coef   =RDZ * ZAREA
    D(icel)= D(icel) - coef
       icou= icou + 1
          k= icou + indexLU(icel-1)
    itemLU(k)= icN6
      AMAT(k)= coef
  endif

  ii= XYZ(icel,1)
  jj= XYZ(icel,2)
  kk= XYZ(icel,3)

  BFORCE(icel)= -dfloat(ii+jj+kk)*VOLO

enddo
```

poi_gen(5/6)

Volume Flux

$$f = dfloat(i_0 + j_0 + k_0)$$

$$i_0 = XYZ(icel,1),$$

$$j_0 = XYZ(icel,2),$$

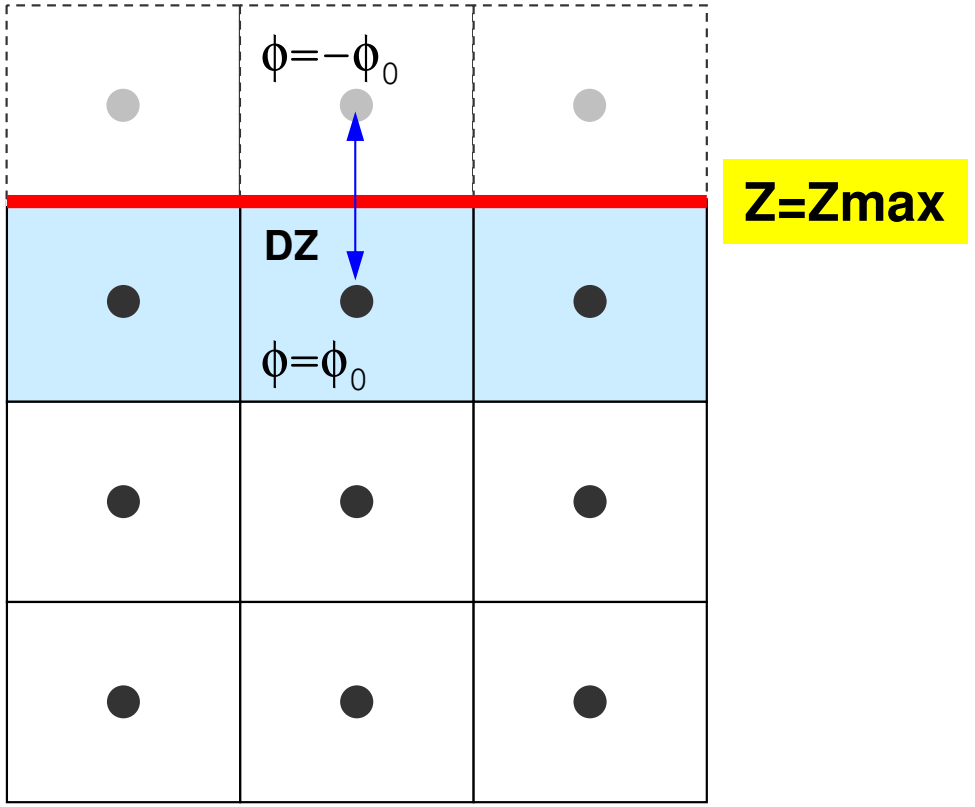
$$k_0 = XYZ(icel,3)$$

$XYZ(icel, k)$(k=1,2,3)
Index for location of finite-difference mesh in X-/Y-/Z-axis.

poi_gen (6/6)

```
!C
!C +-------------------------+
!C | DIRICHLET BOUNDARY CELLs |
!C +-------------------------+
!C   TOP SURFACE
!C===
      do ib= 1, ZmaxCELtot
        icel= ZmaxCEL(ib)
        coef= 2.d0 * RDZ * ZAREA
        D(icel)= D(icel) - coef
      enddo
!C===
      return
      end
```

Calculation of Coefficients on Boundary Surface @ $Z=Z_{max}$



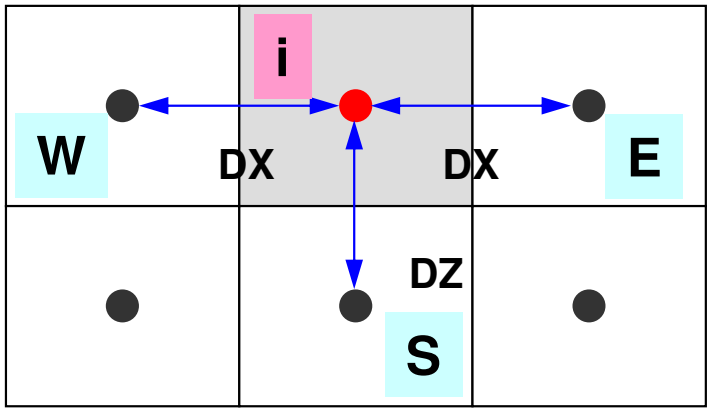
1st Order Approximation:

Mirror Image according to $Z=Z_{max}$ surface.

$\phi=-\phi_0$ at the center of the (virtual) mesh

$\phi=0$@$Z=Z_{max}$ surface

Dirichlet B.C.

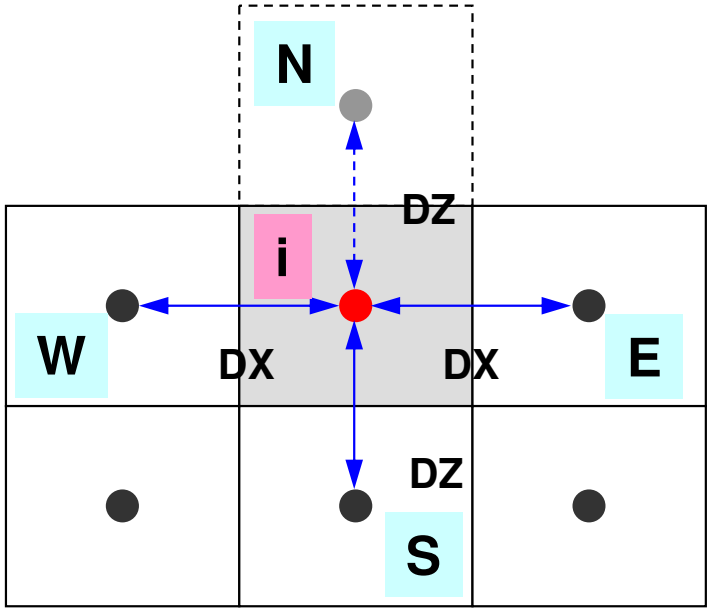


$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AMAT (off-diag.)** **BFORCE (RHS)**

Dirichlet B.C.



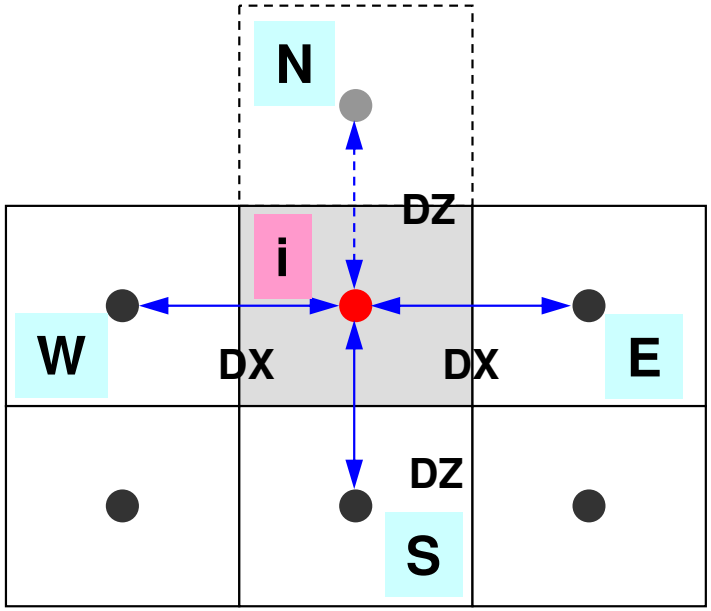
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AMAT (off-diag.)** **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{\phi_N-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i,\quad \phi_N=-\phi_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

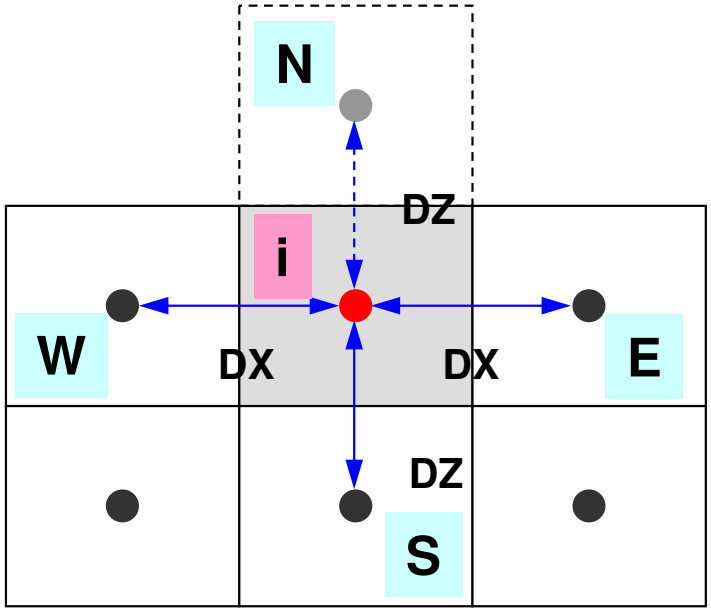
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AMAT (off-diag.)** **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{\phi_N-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i,\quad \phi_N=-\phi_i$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-\phi_i-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i$$

Dirichlet B.C.



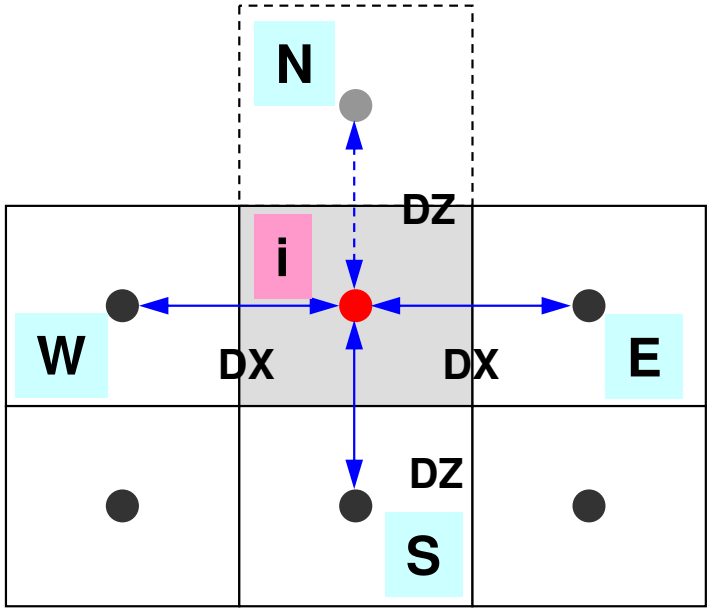
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AMAT (off-diag.)** **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-2\phi_i}{\Delta z}\Delta x\Delta y=+V_i\dot{Q}_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

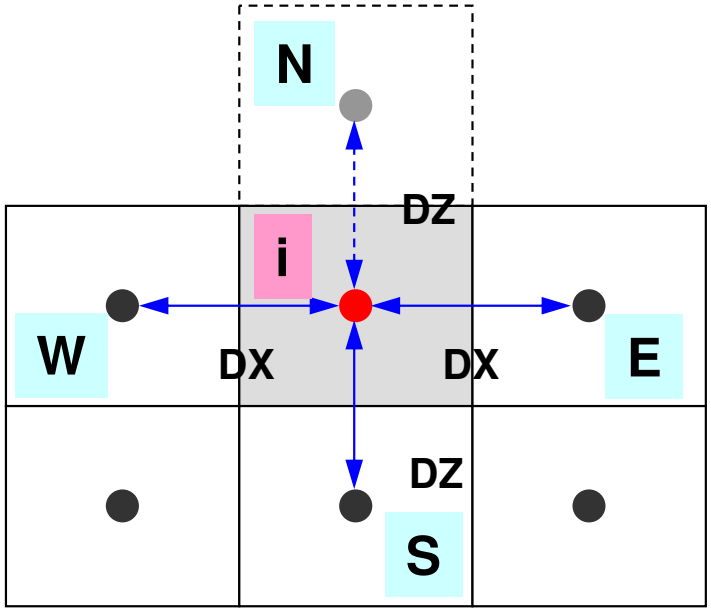
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AMAT (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-2\phi_i}{\Delta z}\Delta x\Delta y=+V_i\dot{Q}_i$$

$$\left[-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}-\frac{2}{\Delta z}\Delta x\Delta y\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+=-V_i\dot{Q}_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

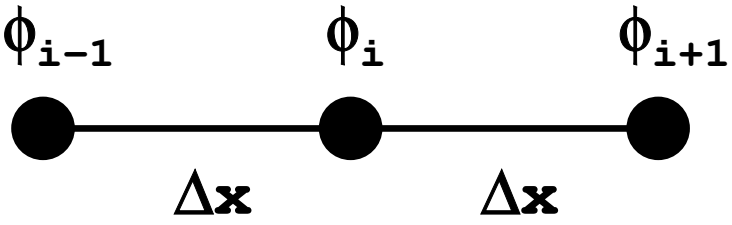
D (diagonal) | **AMAT (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]$$

```
for(ib=0; ib<ZmaxCELtot; ib++) {
    icel  = ZmaxCEL[ib];
    coef = 2.0 * RDZ * ZAREA;
    D[icel-1] -= coef;
}
```

$$\left[-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}-\frac{2}{\Delta z}\Delta x\Delta y\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+=-V_i\dot{Q}_i$$

Taylor Series Expansion



$$\phi_{i+1} = \phi_i + \Delta x\left(\frac{\partial \phi}{\partial x}\right)_i + \frac{(\Delta x)^2}{2!}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i + \frac{(\Delta x)^3}{3!}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i \cdots$$

$$\phi_{i-1} = \phi_i - \Delta x\left(\frac{\partial \phi}{\partial x}\right)_i + \frac{(\Delta x)^2}{2!}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(\Delta x)^3}{3!}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i \cdots$$

$$\phi_{i-1} + \phi_{i+1} = 2\phi_i + 2\times\frac{(\Delta x)^2}{2!}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i + 2\times\frac{(\Delta x)^4}{4!}\left(\frac{\partial^4 \phi}{\partial x^4}\right)_i \cdots$$

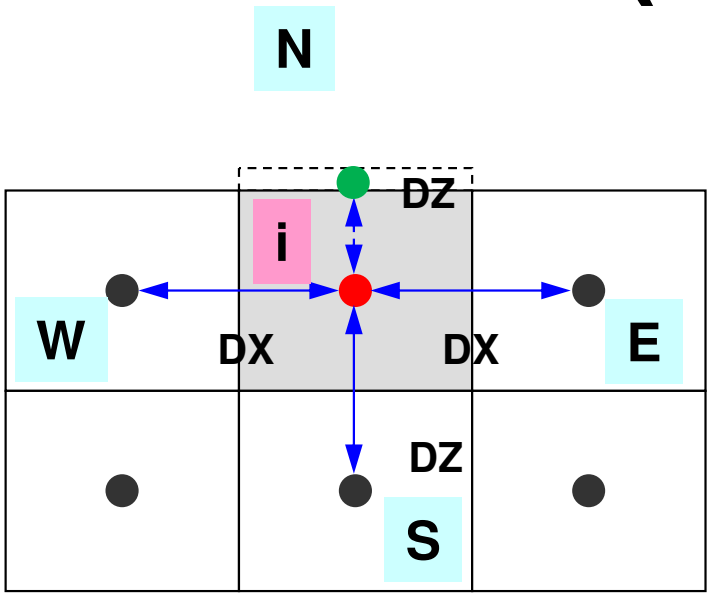
$$\frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{(\Delta x)^2} = \left(\frac{\partial^2 \phi}{\partial x^2}\right)_i + \frac{(\Delta x)^2}{12}\left(\frac{\partial^4 \phi}{\partial x^4}\right)_i \cdots$$

Truncation Err.: 2nd Order

2nd Order Accuracy

If Δx is not uniform: 1st or Lower Order Accuracy

Dirichlet B.C. "N" is very thin (=ε)
1st order (or lower) Accuracy



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

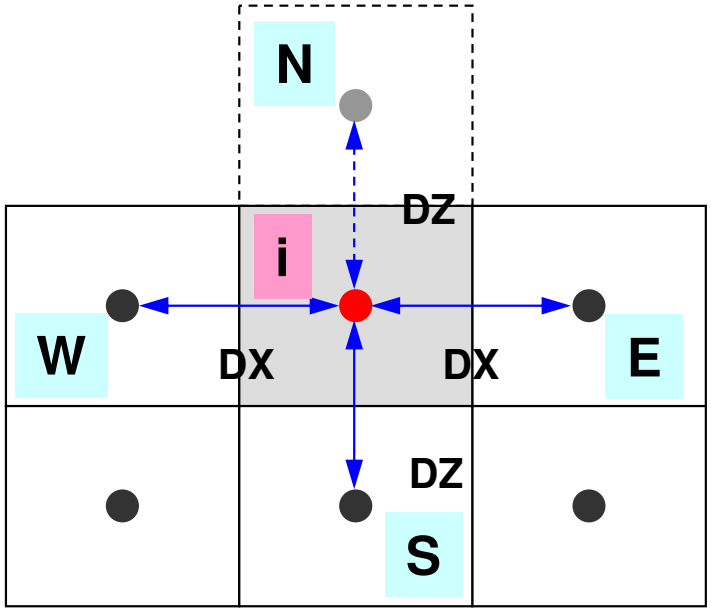
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AMAT (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{\phi_N-\phi_i}{(\frac{\Delta z}{2}+\frac{\varepsilon}{2})}\Delta x\Delta y=-V_i\dot{Q}_i,\quad \phi_N=0,\quad \varepsilon\sim 0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]-\frac{2\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i$$

Dirichlet B.C. using Mirror Image



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

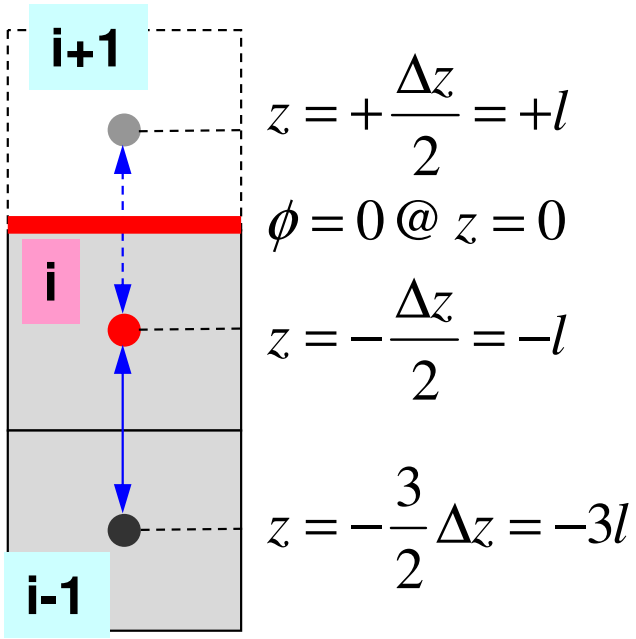
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AMAT (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-2\phi_i}{\Delta z}\Delta x\Delta y=+V_i\dot{Q}_i$$

$$\left[-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}-\frac{2}{\Delta z}\Delta x\Delta y\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+=-V_i\dot{Q}_i$$

Higher Order Approximation for Dirichlet B.C. in 1D Problem



more complicated in 2D/3D cases

$$\phi = az^2 + bz + c$$

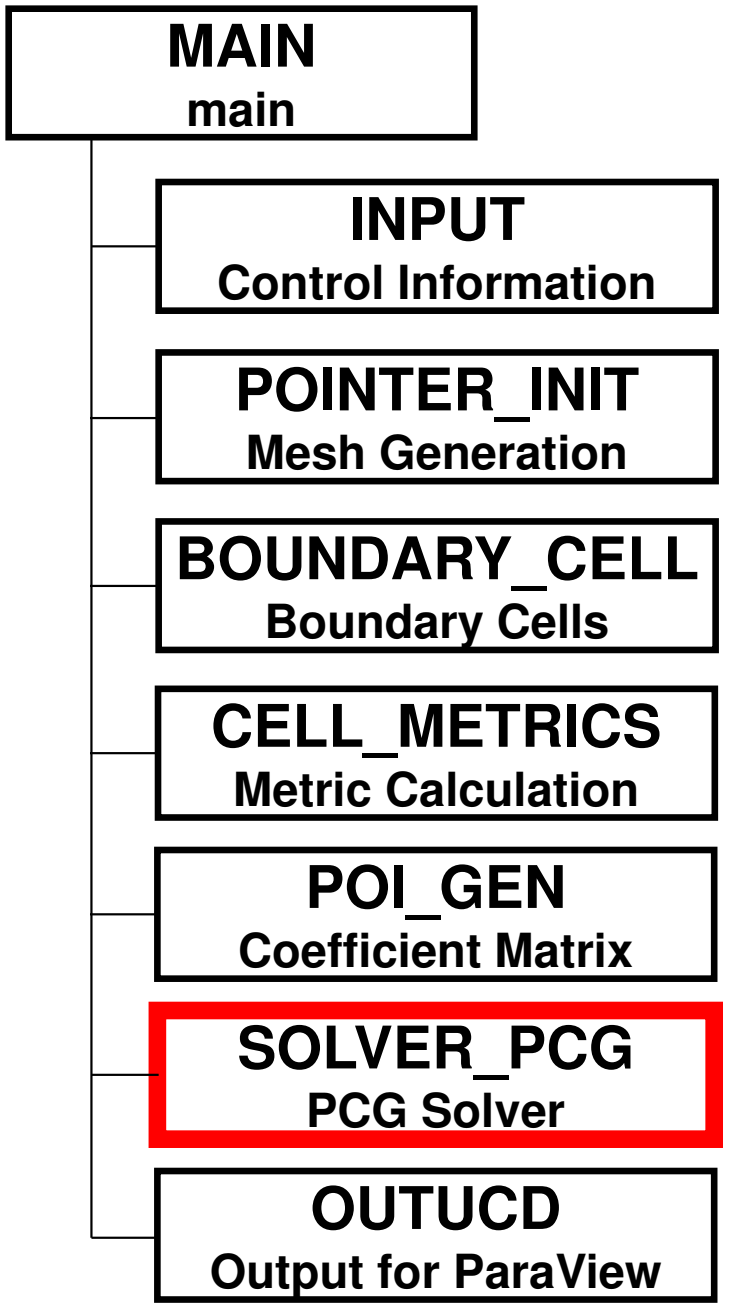
$$\phi(z=0) = c = 0$$

$$\phi_i = al^2 - bl + c = al^2 - bl, \quad \phi_{i-1} = 9al^2 - 3bl + c = 9al^2 - 3bl$$

$$a = \frac{\phi_{i-1} - 3\phi_i}{6l^2}, \quad b = \frac{\phi_{i-1} - 9\phi_i}{6l} \Rightarrow \phi_{i+1} = al^2 + bl = \frac{1}{3}\phi_{i-1} - 2\phi_i$$

Structure of the Program

```
program MAIN

use STRUCT
use PCG
use solver_PCG

implicit REAL*8 (A-H,O-Z)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
PHI=  0.d0

call solve_PCG  (…)

call OUTUCD

stop
end
```



- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **ICCG Solver for Poisson's Equations**
  - How to run
    - Data Structure
  - **Program**
    - Initialization
    - Coefficient Matrices
    - **ICCG**

Solving Linear Equations

- Conjugate Gradient, CG
- Preconditioning
  - Point Jacobi, Diagonal Scaling
- PCG

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

solve_PCG (1/6)

```
 module solver_PCG
 contains

 subroutine solve_PCG                                    &
&          ( N, NPLU, indexLU, itemLU, D, B, X,          &
&            AMAT, EPS, ITR, IER)

 implicit REAL*8 (A-H,O-Z)

 real(kind=8), dimension(N)    :: D
 real(kind=8), dimension(N)    :: B
 real(kind=8), dimension(N)    :: X
 real(kind=8), dimension(NPLU):: AMAT

 integer, dimension(0:N) :: indexLU
 integer, dimension(NPLU):: itemLU
 real(kind=8), dimension(:,:), allocatable :: W

 integer, parameter ::  R= 1
 integer, parameter ::  Z= 2
 integer, parameter ::  Q= 2
 integer, parameter ::  P= 3
 integer, parameter :: DD= 4
```

ICELTOT → N
BFORCE → B
PHI → X
EPSICCG → EPS

W(i,1)= W(i,R) ⇒ {r}
W(i,2)= W(i,Z) ⇒ {z}
W(i,2)= W(i,Q) ⇒ {q}
W(i,3)= W(i,P) ⇒ {p}

W(i,4)= W(i,DD) ⇒ {1/d}

solve_PCG(2/6)

```
!C
!C +------+
!C | INIT |
!C +------+
!C===
      allocate (W(N,4))

      do i= 1, N
        X(i)   = 0.d0
        W(i,1)= 0.0D0
        W(i,2)= 0.0D0
        W(i,3)= 0.0D0
        W(i,4)= 0.0D0
      enddo

      do i= 1, N
        W(i,DD)= 1.d0/D(i)
      enddo
!C===
```

Reciprocal numbers（逆数）of diagonal components are stored in W[DD][i].Computational cost for division is usually expensive.

Although it was said (division):(+,-,*) is 10:1 before, the difference is much smaller now. Generally, multiplying is still faster than division.

solve_PCG(3/6)

```
!C
!C-- {r0}= {b} - [A]{xini}

      do i= 1, N
        VAL= D(i)*X(i)
        do k= indexLU(i-1)+1, indexLU(i)
          VAL= VAL + AMAT(k)*X(itemLU(k))
        enddo
        W(i,R)= B(i) - VAL
      enddo

      BNRM2= 0.0D0
      do i= 1, N
        BNRM2 = BNRM2 + B(i)  **2
      enddo
!C===
```

Compute $\mathbf{r^{(0)}= b-[A]x^{(0)}}$

for i= 1, 2, …

- solve $[M]z^{(i-1)}= r^{(i-1)}$
- $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
- if i=1
  - $p^{(1)}= z^{(0)}$
- else
  - $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
  - $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
- endif
- $q^{(i)}= [A]p^{(i)}$
- $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
- $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
- $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
- check convergence $|r|$

end

BNRM2=$|b|^2$
to be used for convergence evaluation

solve_PCG(4/6)

```
      ITR= N
      do L= 1, ITR

!C-- {z}= {Minv}{r} |
      do i= 1, N
        W(i,Z)= W(i,R)*W(I,DD)
      enddo

!C-- RHO= {r}{z} |
      RHO= 0.d0
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

!C-- {p} = {z} if      ITER=1
!C-- BETA= RHO / RHO1  otherwise

      if ( L.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
        BETA= RHO / RHO1
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

- **solve $[M]z^{(i-1)}= r^{(i-1)}$**
- **$\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$**
- **if i=1**
  - **$p^{(1)}= z^{(0)}$**
- **else**
  - **$\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$**
  - **$p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$**
- **endif**
- $q^{(i)}= [A]p^{(i)}$
- $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
- $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
- $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
- check convergence $|r|$

end

solve_PCG(5/6)

```
!C-- {q}= [A]{p}

  do i= 1, N
    VAL= D(i)*W(i,P)
    do k= indexLU(i-1)+1, indexLU(i)
      VAL= VAL + AMAT(k)*W(itemLU(k),P)
    enddo
    W(i,Q)= VAL
  enddo

!C-- ALPHA= RHO / {p}{q}

  C1= 0.d0
  do i= 1, N
    C1= C1 + W(i,P)*W(i,Q)
  enddo
  ALPHA= RHO / C1

!C--{x}= {x} + ALPHA*{p} |
!C--{r}= {r} - ALPHA*{q} |

  do i= 1, N
    X(i)   = X(i)   + ALPHA * W(i,P)
    W(i,R)= W(i,R) - ALPHA * W(i,Q)
  enddo
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

- solve $[M]z^{(i-1)}= r^{(i-1)}$
- $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
- if i=1
  - $p^{(1)}= z^{(0)}$
- else
  - $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
  - $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
- endif
- **$q^{(i)}= [A]p^{(i)}$**
- **$\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$**
- **$x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$**
- **$r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$**
- check convergence $|r|$

end

solve_PCG(6/6)

```
    do L= 1, ITR
(…)
      DNRM2= 0.d0
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo

      ERR = dsqrt(DNRM2/BNRM2)
      if (ERR .lt. EPS) then
        IER = 0
        goto 900
       else
        RHO1 = RHO
      endif

    enddo
    IER= 1

900 continue
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
   else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_PCG(6/6)

```
      do L= 1, ITR
(…)
        DNRM2= 0.d0
        do i= 1, N
          DNRM2= DNRM2 + W
        enddo

        ERR = dsqrt(DNRM2/BNRM2)
        if (ERR .lt. EPS) then
          IER = 0
          goto 900
        else
          RHO1 = RHO
        endif

      enddo
      IER= 1

900 continue
```

$$ERR = \sqrt{\frac{\mathrm{DNorm2}}{\mathrm{BNorm2}}} = \frac{|r|}{|b|} = \frac{|b - Ax|}{|b|} \leq \mathrm{Eps}$$

r= b-[A]x
DNRM2= $|r|^2$
BNRM2= $|b|^2$

ERR= |r|/|b|

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
      solve [M]z(i-1)= r(i-1)
        else
          βi-1= ρi-1/ρi-2
          p(i)= z(i-1) + βi-1 p(i-1)
        endif
        q(i)= [A]p(i)
        αi  = ρi-1/p(i)q(i)
        x(i)= x(i-1) + αip(i)
        r(i)= r(i-1) - αiq(i)
        check convergence |r|
end
```

$|r|, |b|$: 2 / L2 / Euclidean − norm $\quad (\|r\|_2, \|b\|_2)$

$$Ax = b \Rightarrow \alpha Ax = \alpha b$$

$$r = b - Ax \Rightarrow R = \alpha b - \alpha Ax = \alpha r$$