Nuclear Modelling using AI from laptops to eXascale (MAtrIX):

1) **Programming CFD/Nuclear Models using AI software** which enables interoperability between GPUs, CPUs and AI computers (energy efficient exascale Cerebras CS2 ~1M node chip) & exploitation of community AI software;

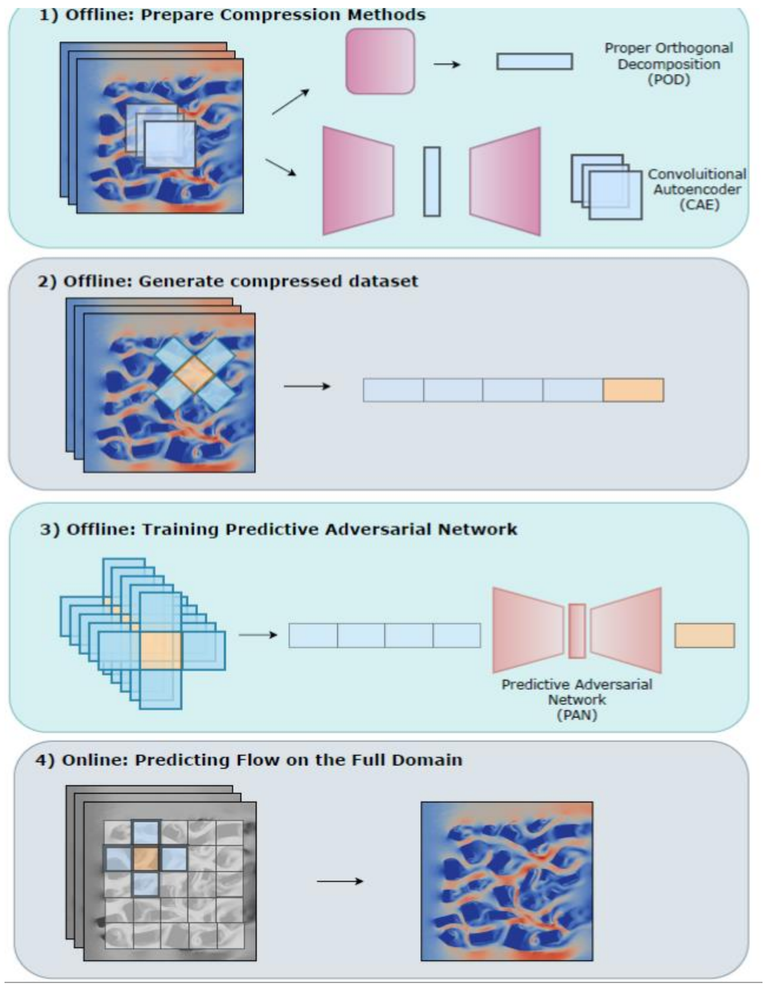
2) **AI-Physics modelling** to reduce the number of unknowns solved for which can be important for some computer architectures as well as speeding up models;

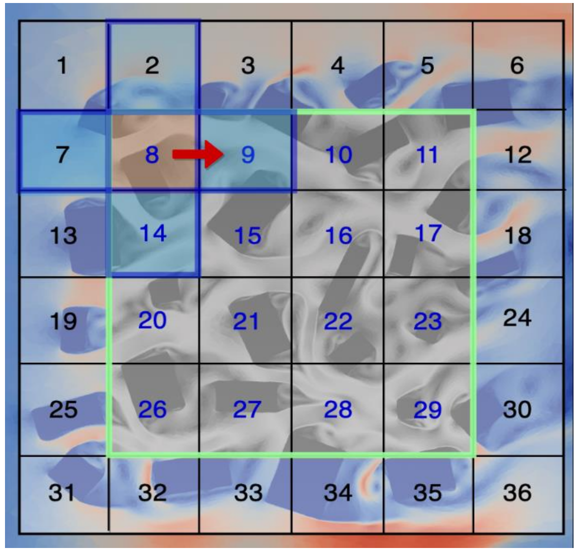
3) **Hybrisation of AI Physics modelling and discretization** of the differential equations in order to construct; new sub-grid-scale (SGS) models, methods that force the equation residuals to zero (RDEIM) or physics informed methods - it’s now accepted that future SGS methods will increasing be based on AI and they may need implicit coupling to CFD using AI programming (1 above);

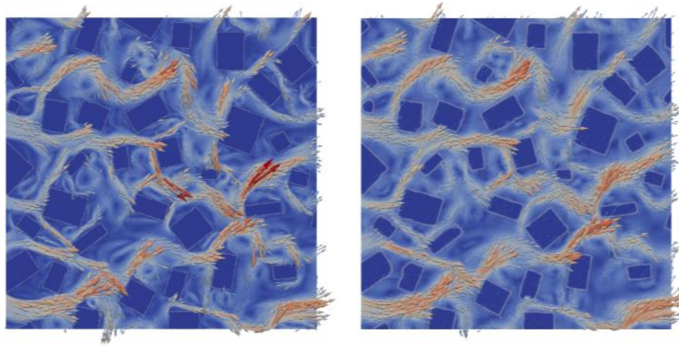
4) **Digital twins** that are able to assimilate data, perform uncertainty quantification and optimization using the optimization engine embedded in all AI software.

Building a detailed flow model of using AI modelling

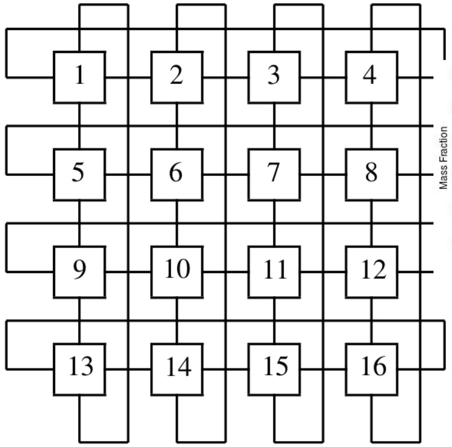
– bottom middle grid architecture of an AI computer e.g. Cerebras CS-2 with 800,000 nodes on a chip

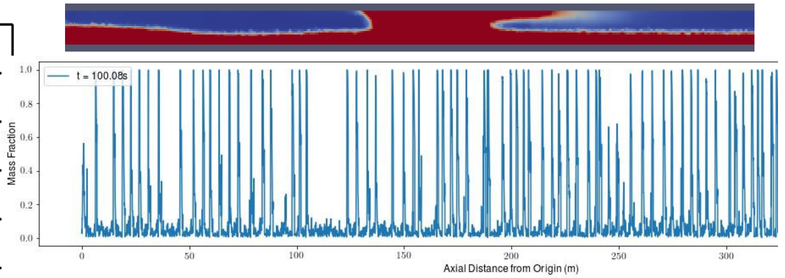

1) Offline: Prepare Compression Methods
Proper Orthogonal Decomposition (POD)
Convoluitional Autoencoder (CAE)
2) Offline: Generate compressed dataset
3) Offline: Training Predictive Adversarial Network
Predictive Adversarial Network (PAN)
4) Online: Predicting Flow on the Full Domain





AI model of flow past buildings velocity vectors. Left: High Fidelity Model. Right: Surrogate

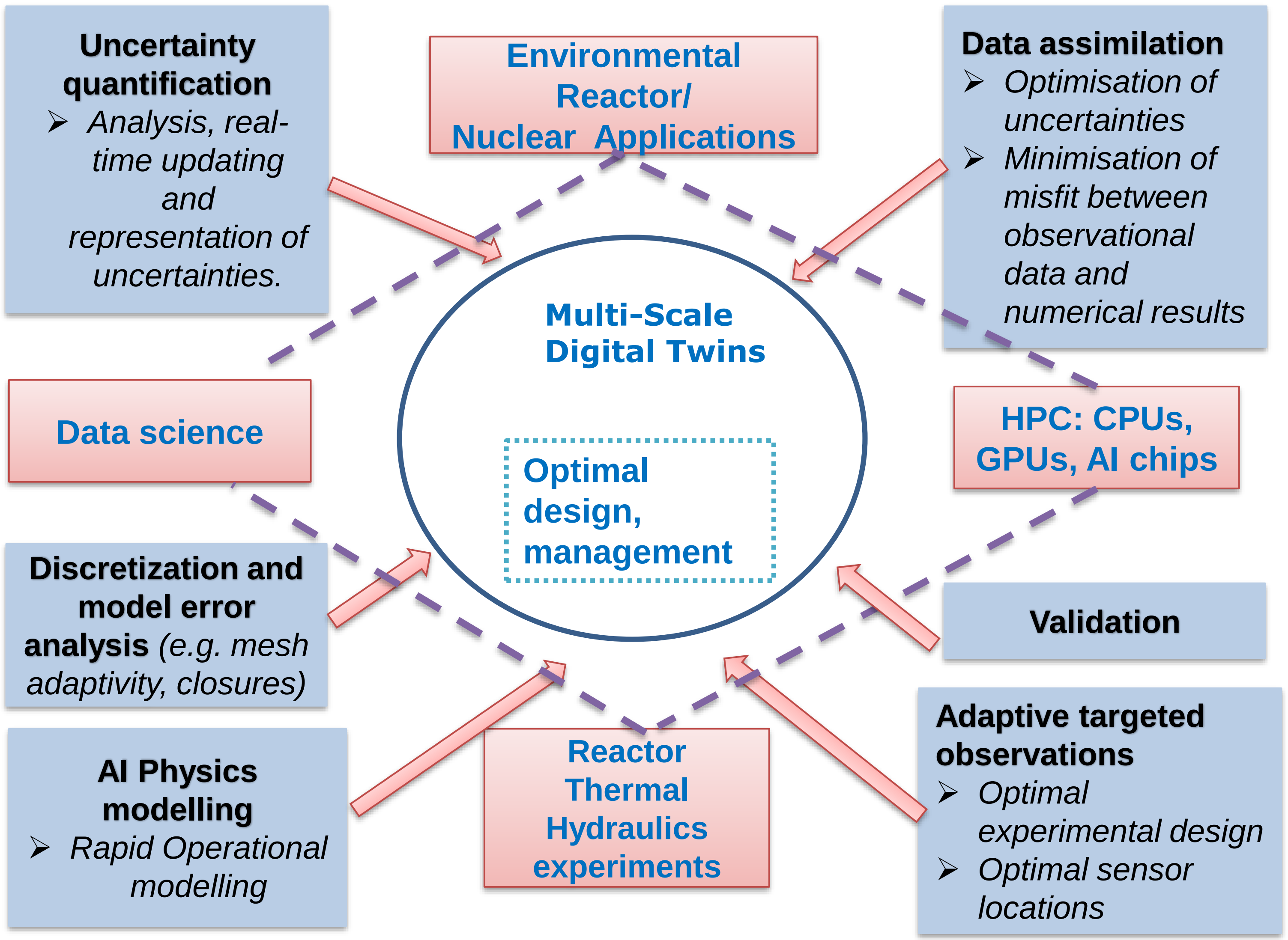




AI model of multiphase slug flow in 1000m pipe from subdomains. Top: cross section of liquid in subdomain. Bottom: liquid along pipe fraction

Nuclear Modelling using AI at eXascale (MATRIX):

- AI-Physics modelling
- AI SGS CFD/TH/RT modelling
- Hierarchical modelling – from CFD/RT to AI Models - multiscale
- AI computers e.g. ~1M node Cerebras chip
- Digital Twins (Right) taking advantage of AI optimization
- Code in AI e.g. PyTorch, SymPy
- Summary speed: AI models 10^3-10^6 faster, AI computer 10^3 faster, Total 10^6-10^9 faster



Uncertainty quantification
- *Analysis, real-time updating and representation of uncertainties.*

Environmental Reactor/ Nuclear Applications

Data assimilation
- *Optimisation of uncertainties*
- *Minimisation of misfit between observational data and numerical results*

Multi-Scale Digital Twins

Optimal design, management

Data science

HPC: CPUs, GPUs, AI chips

Discretization and model error analysis *(e.g. mesh adaptivity, closures)*

Validation

AI Physics modelling
- *Rapid Operational modelling*

Reactor Thermal Hydraulics experiments

Adaptive targeted observations
- *Optimal experimental design*
- *Optimal sensor locations*

Advantages of programming CFD/Neutronics/Solids in AI:

1) **Solvers:** Linear, non-linear multigrid methods using convolutional auto-encoders, Newton Raphson iteration. AI models can also be used as preconditions in CFD.
2) **Hierarchical solvers:** From CFD to fine and coarse AI-Physics models and one catch switch between these locally or globally.
3) **Embedded CFD and AI Physics modelling** in a single neural network – SGS modelling, PINN, RDEIM.
4) **Automatic Adjoint Generation and Digital twins** that are able to assimilate data, perform uncertainty quantification and optimization using the optimization engine embedded in all AI software.
5) **Realize new features in modelling:** reduced arithmetic speed; coupling different physics, finance, economic, language and/or social models; leverage AI code.
6) **More accessible AI programming for modelers** - similar to AI applications.
7) **Optimized code interoperability between computer architectures:** CPU, GPU and AI chips - exascale.
8) **Long term model/code sustainability.**

Strategy Issues for CFD implementation using AI software:

1) Multi-grid solvers – through CNN autoencoder (done)

2) Adjoints – through weights and activation functions (done)

3) Unstructured meshes e.g. Space Filling Curves CNN, MeshCNN (done)

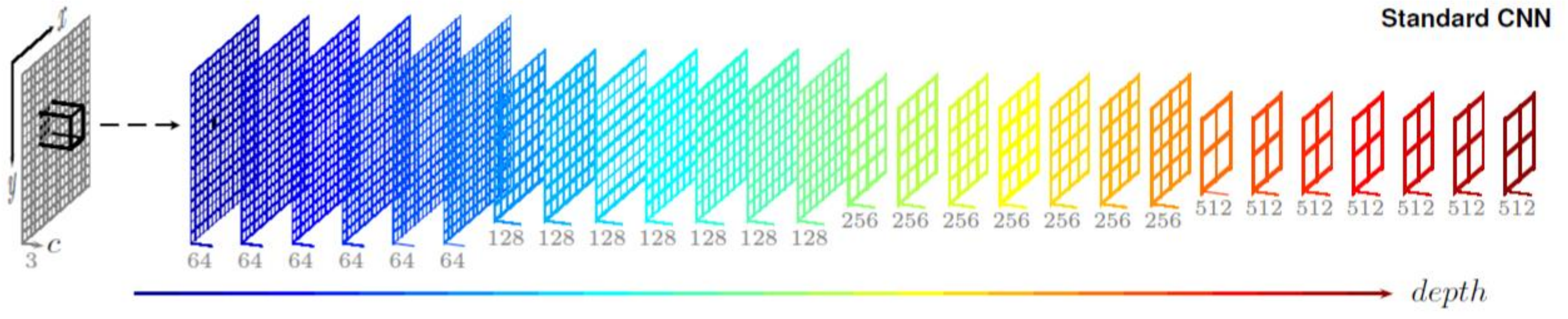
4) **Defining discretization e.g. DEVITO, OpenSBLI, SymPy, Firedrack**

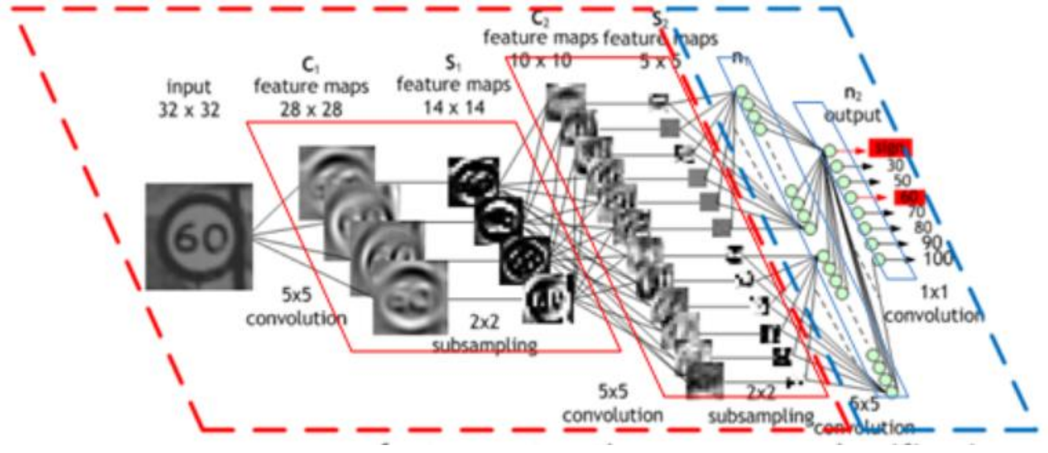
5) **Parallel updating of halos e.g. SciML, Lightning-Horovod, DeepSpeed, transformer methods (dense), MPI may be used for block unstructured**

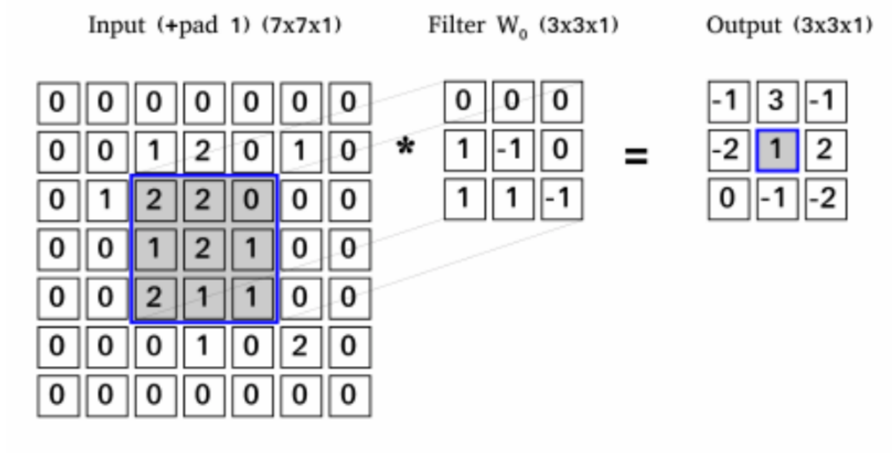
6) Large matrix/solution free methods e.g. radiation transport – use single CNN filter for semi-structured discretization in each block (done)

7) GUIs linkage – keep flexible

Convolutional ANNs with linear activation functions and weights from discretization stencil can be Multi-Grid Solvers:

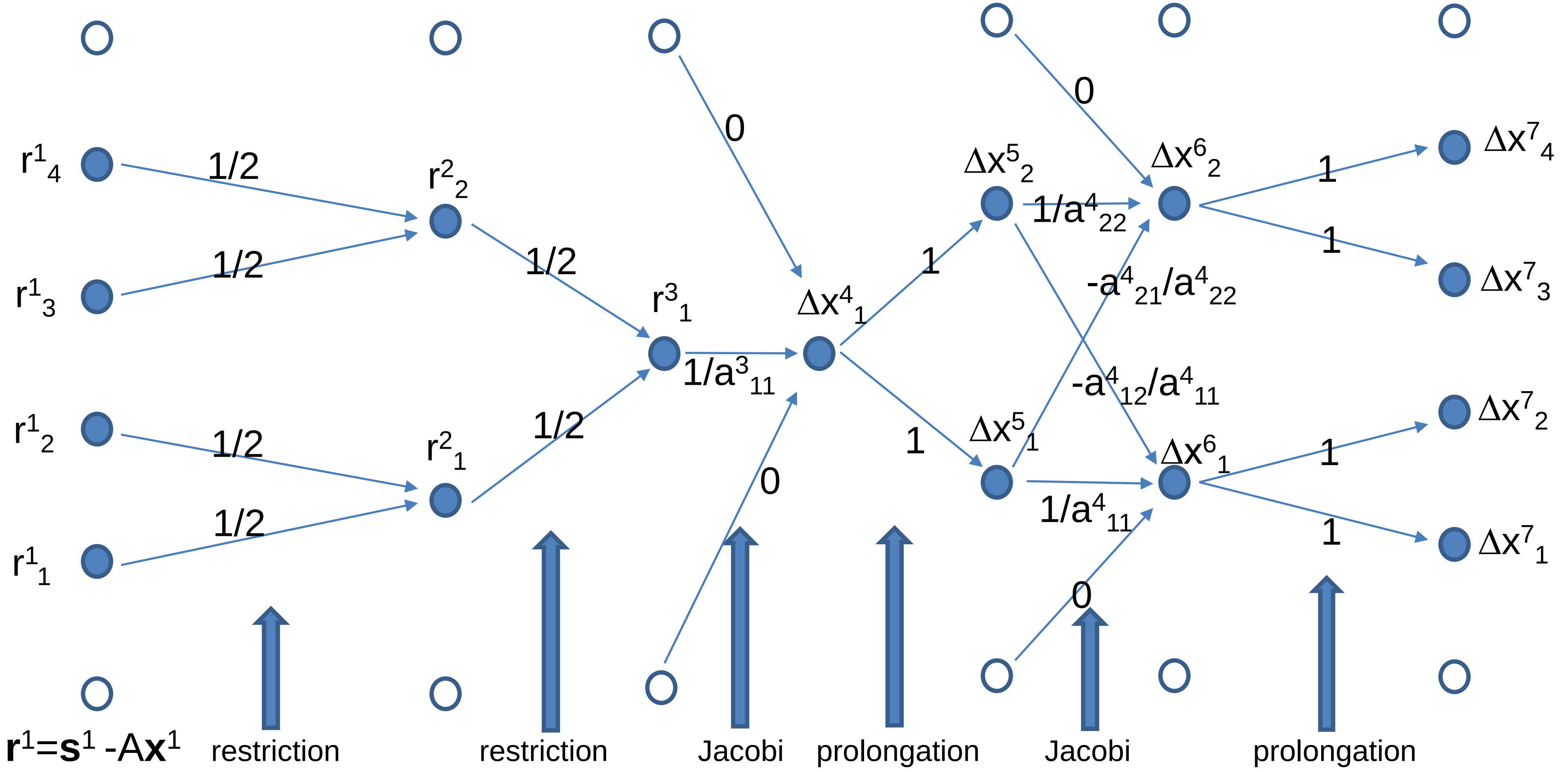






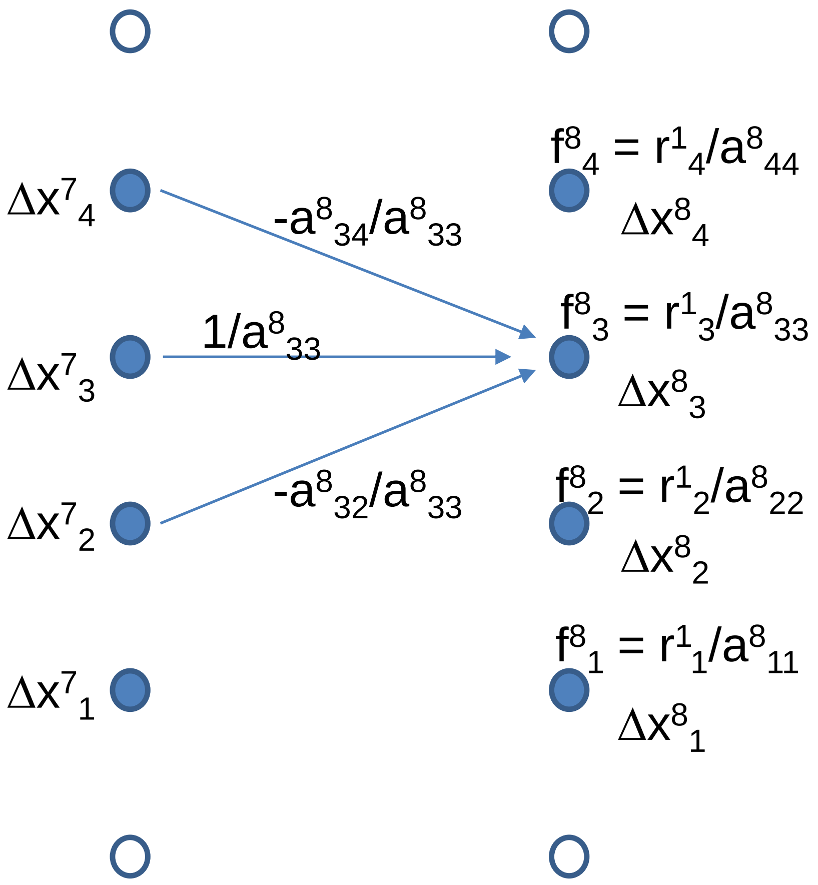
Solve Ax=s using Simplest F-cycle Multi-Grid Method and Convolutional ANN in 1D – neural network shown:

Superscript ANN level, subscript ANN neuron; Jacobi smoother



Final Layer – Jacobi iteration with bias

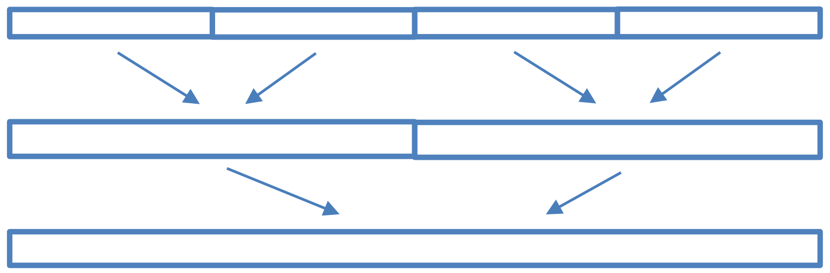
f is the bias



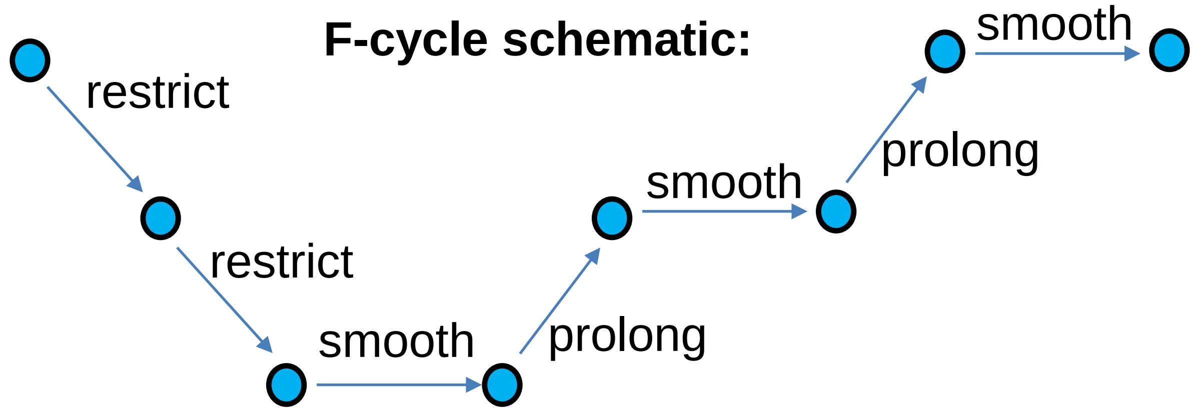
$\mathbf{x}^1 \rightarrow \mathbf{x}^1 + \Delta\mathbf{x}^8$ then repeat F cycle

For 1D linear DG: Have 2 channels. One for each DG node and use the inverse of the 2x2 matrix rather than $1/a^8{}_{33}$ etc. **3D linear DG** has 8 channels for each DG node.

Cells/nodes/neurons:



F-cycle schematic:



Jacobi relaxation when solving Ax=s:

Jacobi iteration: $x^{new}_i = (1/a_{ii}) (-\sum_{j\neq i} a_{ij}x^{old}_j + s_i)$

Suppose $\alpha \in (0,1]$ is the relaxation coefficient (e.g. $\alpha=0.5$) then

$x_i = (1- \alpha)\, x^{old}_i + \alpha\, (1/a_{ii}) (-\sum_{j\neq i} a_{ij}x^{old}_i + s_i)$.

Thus $x_i = \sum_j b_{ij}x^{old}_j + c_i$

in which $b_{ij} = -\alpha\, (1/a_{ii})\, a_{ij}$ if $i \neq j$

and $b_{ii} = 1$

and $c_i = \alpha\, (1/a_{ii})\, s_i$ (is the bias)

Thus $w_{ij} = b_{ij}$ are the weights of the convolutional filter

Time stepping of $\partial T/\partial t + u\partial T/\partial x + v\partial T/\partial y + \sigma T - \mu \nabla . \nabla\ T = s$ (advection velocity (u,v)):

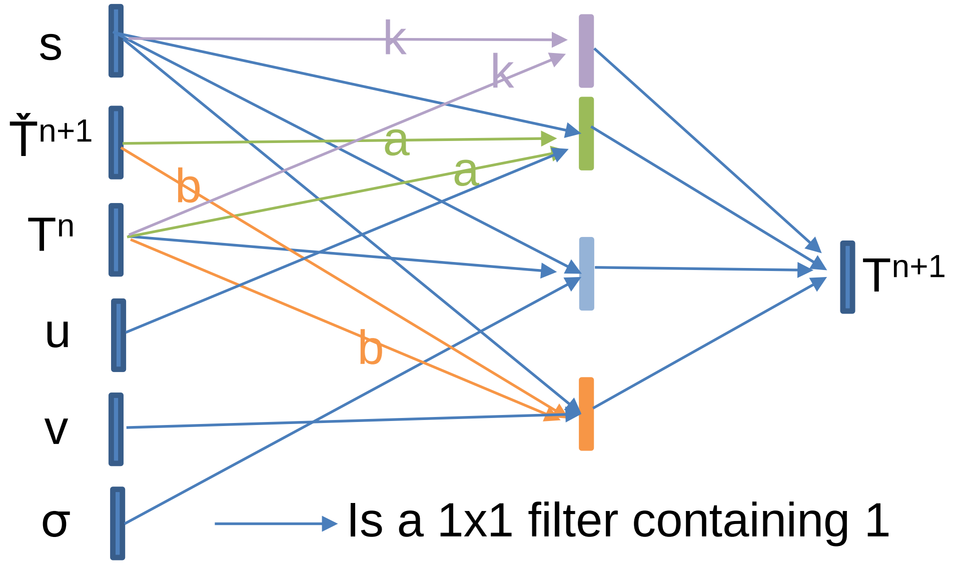
Two filters are needed a_{ij} and b_{ij}. The half in the bellow is used to obtain 2nd order accuracy in time, $\check{T}^{n+1}$ is the best guess for T^{n+1} which might be T^n. Δt = time step size. ả, β are advection/diffusion stencils/discretizations. Discretization becomes:

$$(T^{n+1}_i - T^n_i)/\Delta t + \sigma_i T^{n+1}_i + u_i \textstyle\sum_j ả_{ij} (1/2) (\check{T}^{n+1}_j + T^n_j) + v_i \sum_j \beta_{ij} (1/2) (\check{T}^{n+1}_j + T^n_j) + \sum_j k\mu_{ij} (1/2) (\check{T}^{n+1}_j + T^n_j) = s_i$$

in which $a_{ij} = (1/2)\ _j ả_{ij} \Delta t$, $b_{ij} = (1/2)\ \beta_{ij} \Delta t$ and $k_{ij} = (1/2)\ \mu_{ij} \Delta t$.

$$T^{n+1}_i = (1/(1+ \Delta t\ \sigma_i\ ))\ T^n_i + s_i + (u_i /(1+ \Delta t\ \sigma_i\ )) \textstyle\sum_j a_{ij} (\check{T}^{n+1}_j + T^n_j) + (v_i /(1+ \Delta t\ \sigma_i\ )) \sum_j b_{ij} (\check{T}^{n+1}_j + T^n_j) + \sum_j k_{ij} (\check{T}^{n+1}_j + T^n_j) \quad (1)$$

This can be turned into a CNN as follows:

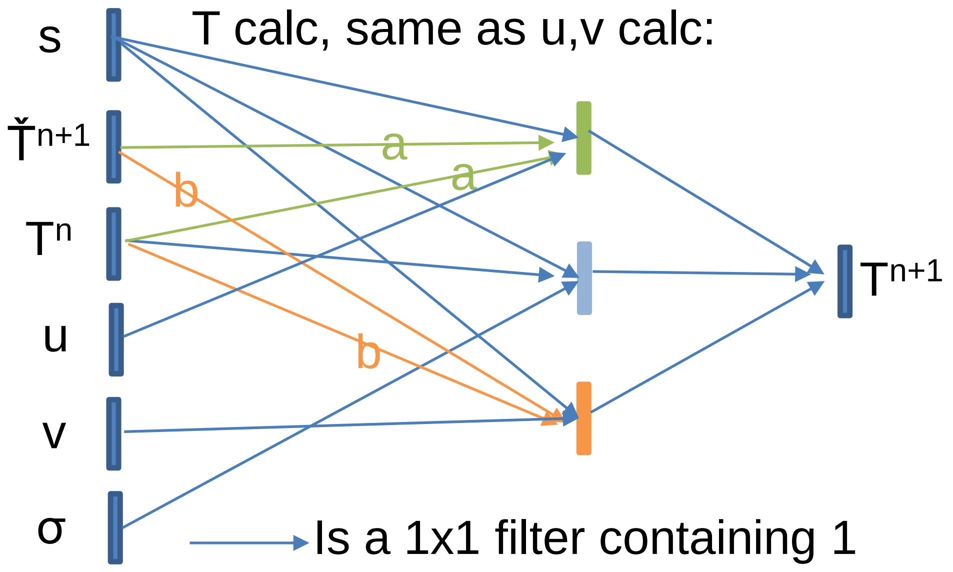


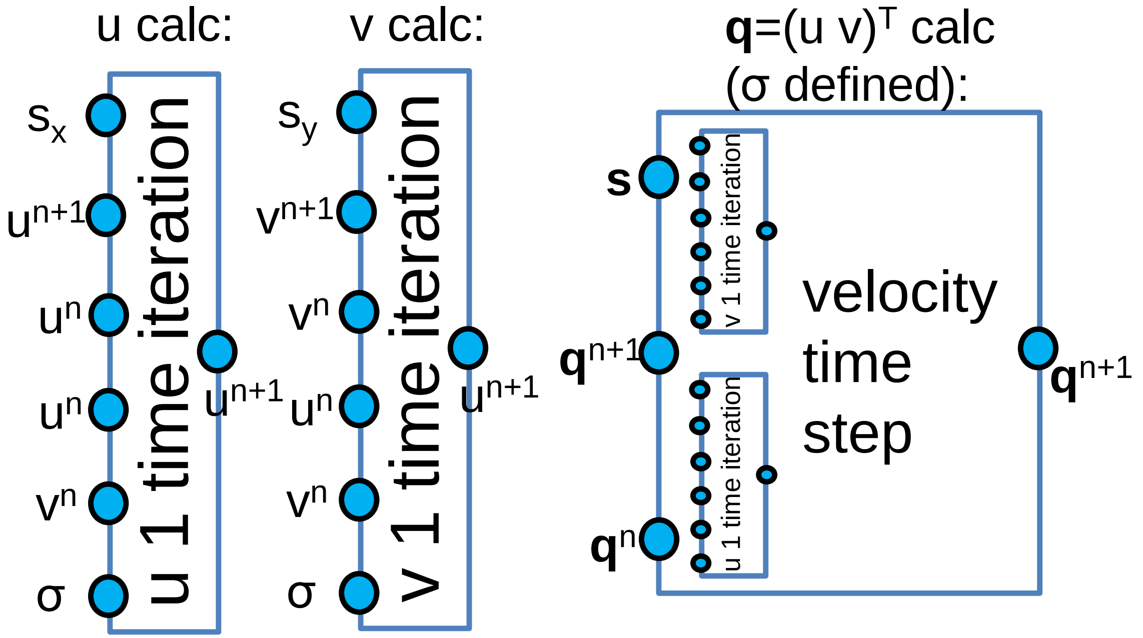
Change the activation functions to implement equation (1) above. The green and orange CNN layers have these tailored activation functions and the purple filter is for constant diffusion.

Use two steps of this CNN gives 2nd order in time.

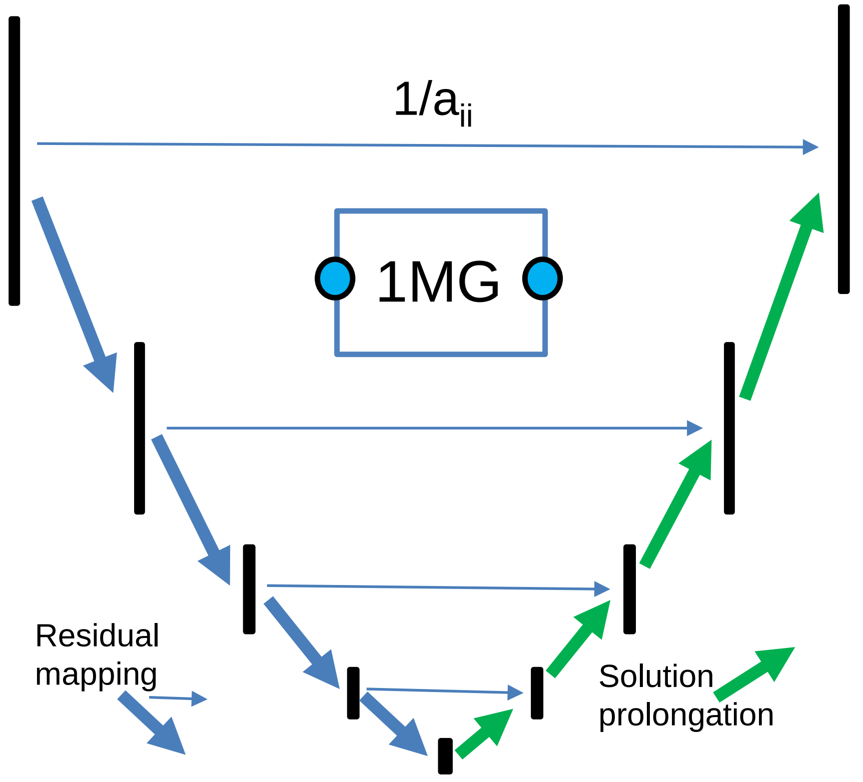
Boundary conditions are applied through padding e.g. Dirichlet - put a value in the halo. For zero derivative apply same value or nearest neighbour padding

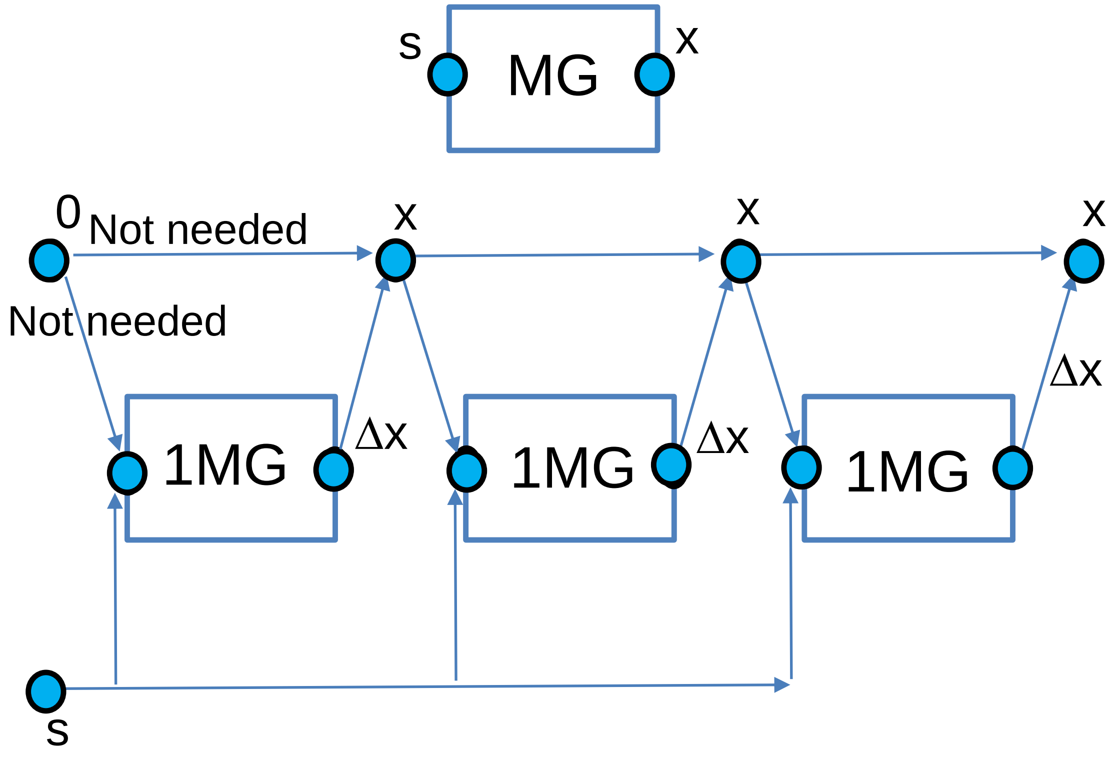
Time stepping of $\partial\mathbf{q}/\partial t + u\partial\mathbf{q}/\partial x + v\partial\mathbf{q}/\partial y + \sigma\mathbf{q} - \mu\nabla.\nabla\mathbf{q} = \mathbf{s}$ (advection velocity $\mathbf{q}=(u\ v)^T$ & source $\mathbf{s}=(s_x\ s_y)^T$):





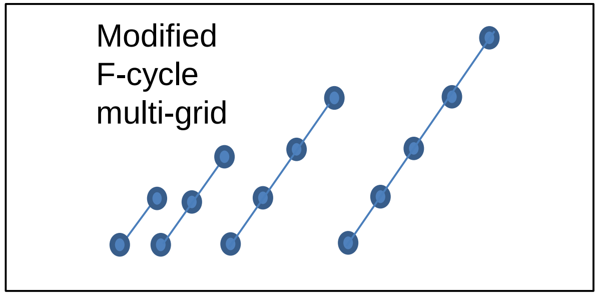
Left: Multi-Grid (MG) CNN skip layers rather than use biases (1MG cycle – similar to U-net); Right: Multi-Grid (3 iterations – cycles) solving Ax= s, starting from 0:

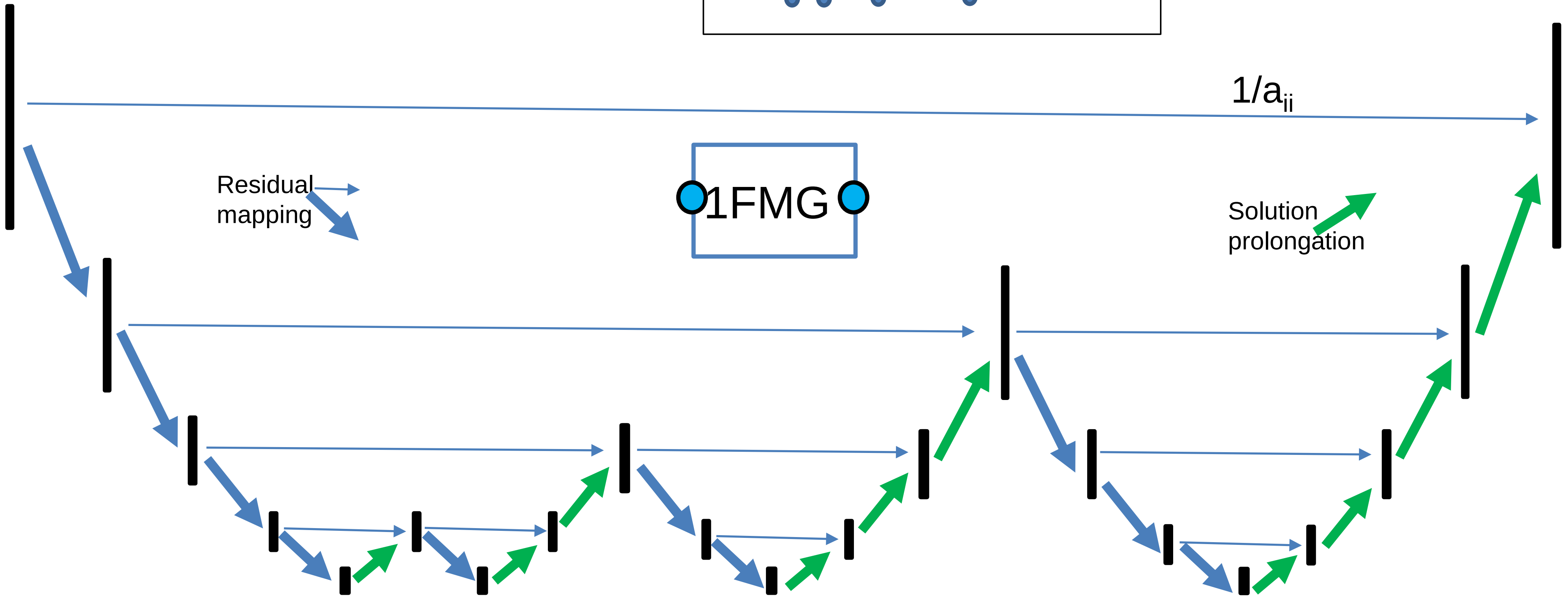




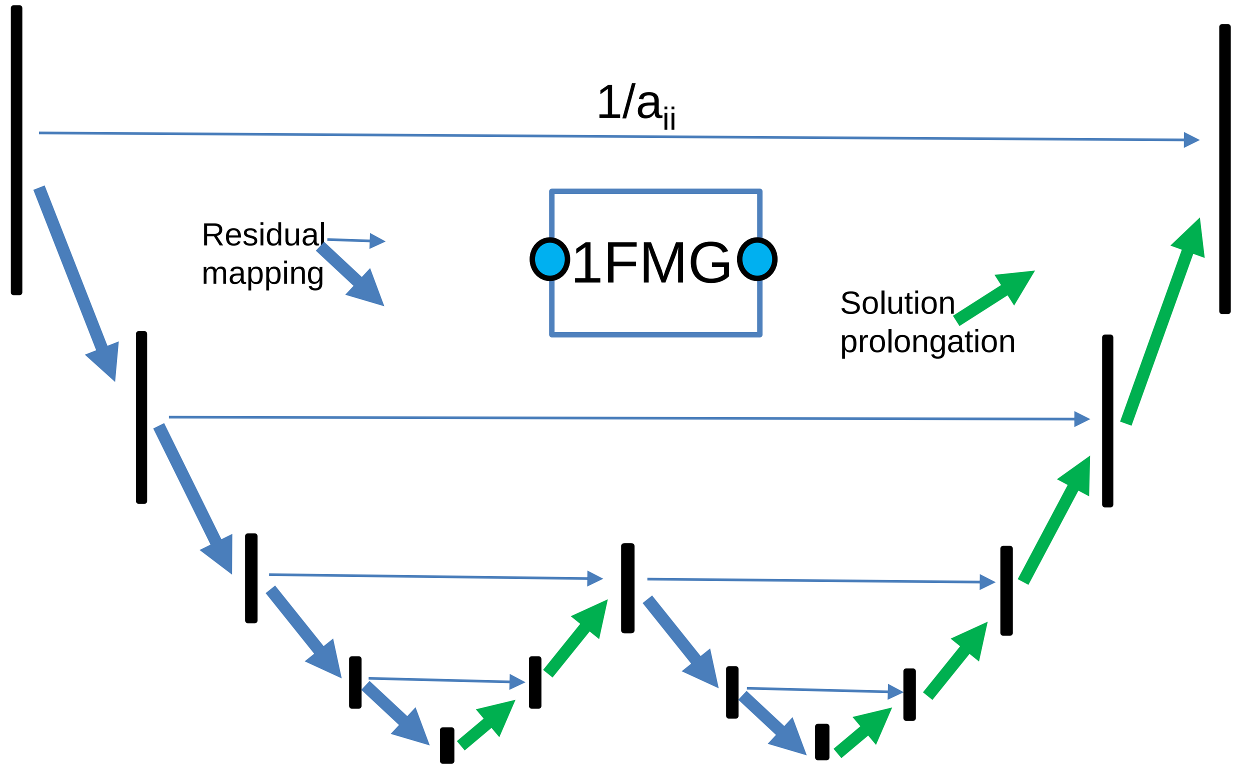
B-net an extension of U-net architecture for Multi-Grid (MG) CNN skip layers (1FMG cycle):

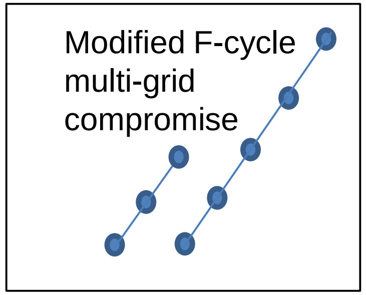
B-net for UNstructured meshES (BUNES-Net) or BunnyNet



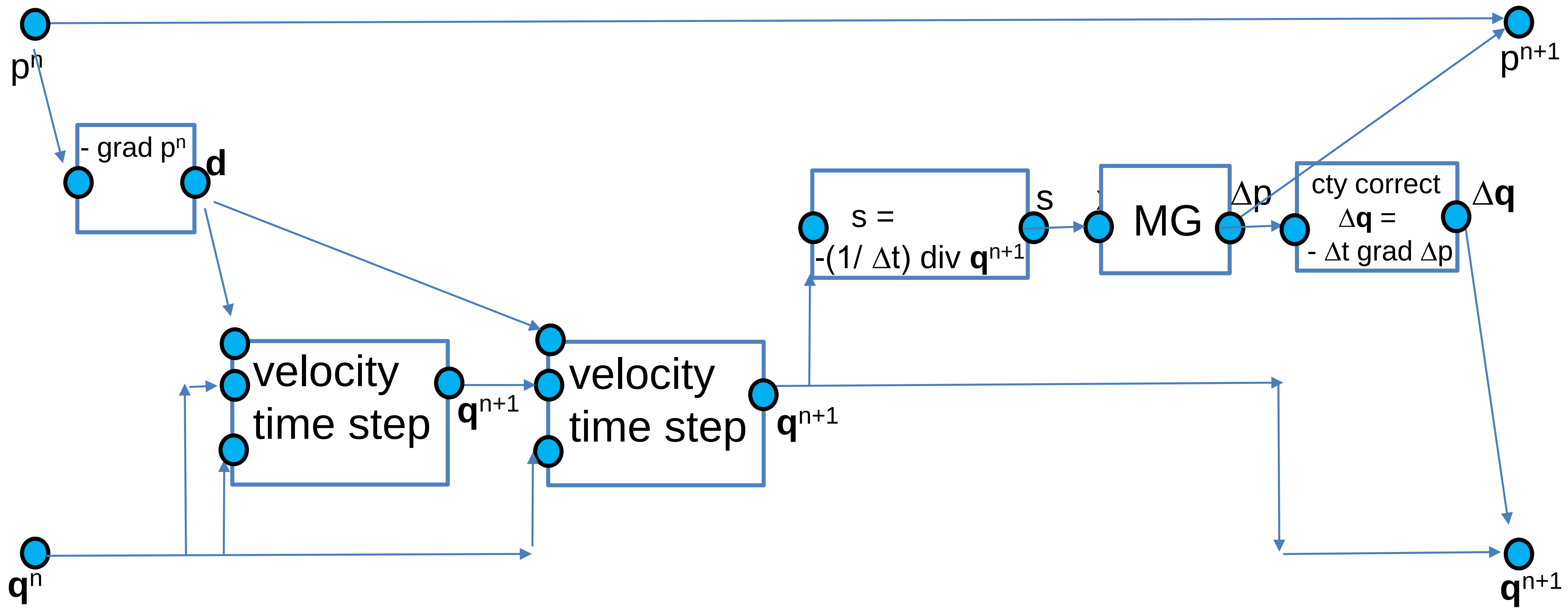


B-net an extension of U-net architecture for Multi-Grid (MG) CNN skip layers (1FMG cycle). Lots of options can be used that for example use more or less relaxations on the coarsest grid e.g. repeating the upward sweeps through the grids gives you more. A method that gives you less is a compromise between U-net and B-net that spends less time on the coarsest grids:





CFD time step:



Governing equation for projection base CFD:

Governing eqns ($\mathbf{q}=(u \;\; v \;\; w)^T$)

$$D\mathbf{q}/Dt + \sigma\,\mathbf{q} - \mu\nabla.\nabla\mathbf{q} = -\nabla p,$$

$$\mathrm{div}\,\mathbf{q} = 0.$$

Derived eqns – solution steps:

1) $D\mathbf{q}/Dt + \sigma\,\mathbf{q} - \mu\nabla.\nabla\mathbf{q} = -\nabla p,$

2) $\nabla\,(1/(1+\Delta t\,\sigma))\,\nabla\Delta p = -\,(1/\Delta t)\,\mathrm{div}\,\mathbf{q},$

or for simplicity $\nabla\,.\,\nabla\Delta p = -\,(1/\Delta t)\,\mathrm{div}\,\mathbf{q}$

3) $\Delta\mathbf{q} = -\,(\Delta t\,/(1+\Delta t\,\sigma))\,\nabla\Delta p,$

or for simplicity $\Delta\mathbf{q} = -\,\Delta t\,\nabla\Delta p$

4) $p \rightarrow p + \Delta p$

5) $\mathbf{q} \rightarrow \mathbf{q} + \Delta\mathbf{q}$

Governing equation for projection based interface tracking multi-phase flow CFD (blue difference from single phase):

Governing eqns ($\mathbf{q}=(u\ \ v\ \ w)^T$)

$\rho\ (\partial\mathbf{q}/\partial t+\mathbf{q}.\nabla\mathbf{q})\ +\sigma\ \mathbf{q}\ -\nabla.\mu\nabla\mathbf{q} = -\nabla p+\rho\ \mathbf{g}$ &

$\partial C/\partial t+\nabla.\mathbf{q}C+\sigma_c\ C-\nabla.\ k_c\nabla C = 0$ with $\rho = C\ \rho_l + (1\text{-}C)\ \rho_g$

$\nabla.\mathbf{q} = 0$

Derived eqns – solution steps:

0) $\rho = C\ \rho_l + (1\text{-}C)\ \rho_g$

1a) $\rho\ (\partial\mathbf{q}/\partial t+\mathbf{q}.\nabla\mathbf{q}) + \sigma\ \mathbf{q} - \nabla.\mu\nabla\mathbf{q}= -\nabla p+\rho\ \mathbf{g}$

1b) $\partial C/\partial t+\nabla.\mathbf{q}C +\sigma_c C - \nabla.k_c\nabla C = 0$

2) $\nabla.(1/\rho)\nabla\Delta p = -(1/\Delta t)\ \nabla.\mathbf{q}$ & use a harmonic average of $1/\rho$ in multi-grid

3) $\Delta\mathbf{q} = -(\Delta t/\rho)\ \nabla\Delta p$

4) $p \rightarrow p + \Delta p$

5) $\mathbf{q} \rightarrow \mathbf{q} + \Delta\mathbf{q}$

CFD Turbulence Modelling using Petrov-Galerkin Dissipation

Define $c_{xi} = (1/m_i)\,(A_x\, C)|_i$, $c_{yi} = (1/m_i)\,(A_y\, C)|_i$, $c_{zi} = (1/m_i)\,(A_z\, C)|_i$,
$r_i=(1/m_i)(1/3)(|\Delta x_i u_i| + |\Delta y_i v_i| + |\Delta z_i w_i|)|A_{D2}C|_i\,|$ or more accurately
$r_i=(1/m_i)(1/3)(|\Delta x_i u_i(A_{D2x}C)|_i| + |\Delta y_i v_i(A_{D2y}C)|_i| + |\Delta z_i w_i(A_{D2z}C)|_i|)$ in which A_{D2} is the filter for ∇^2, A_{D2x} is the filter curvature in the x-direction, and $m_i = \Delta xi \Delta yi \Delta z_i$ is the mass associated with FEM node i; A_x, A_y, A_z are the filters for the derivative in the x, y and z directions.
$P_i=\min\{1/(\sigma+\epsilon), \hat{P}_i\}$, in which ϵ is a small number and in which
$\hat{P}_i = (1/4)\,|(u_i, v_i, w_i) \bullet (c_{xi}\, , c_{yi}\, , c_{zi}\,)|\;\; (1/3)(\mathrm{I}(2/\Delta x_i)c_{xi}\mathrm{I} + \mathrm{I}(2/\Delta y_i)c_{yi}\mathrm{I} + \mathrm{I}(2/\Delta z_i)c_{zi}\mathrm{I})$ and I.I represents the absolute value.

Thus, the diffusion coefficient is

$k_i = r_i^2\, P_i\, /\max\{\epsilon_k, c_{xi}^2 + c_{yi}^2 + c_{zi}^2\}$,
ϵ_k needs to be sufficiently large so that k_i does not get too big. The final diffusion matrix Multiplication is given by ½($k_i(A_{D2}C)|_i + (A_{D2}(KC))|_i - c_i(A_{D2}K)|_i$) – see variable diffusion coefficient slide. Thus, an ANN channel is needed that stores k_i and A_{D2} is a discretisation (e.g. FEM) of the Laplacian.

Calculating diffusion/gradient ANN filters for $-\nabla.k\nabla c$, ∇c on non-uniform but not distorted grids with variable diffusion

From 1D we can see that assuming $\Delta x=1$ then the diffusion 3 point discretization is

$-½(k_i+k_{i+1})(c_{i+1}-c_i)+½(k_{i-1}+k_i)(c_i - c_{i-1}) = -½(k_i+k_{i-1})c_{i-1} + (½k_{i-1}+k_i+½k_{i+1})c_i - ½(k_i+k_{i+1})c_{i+1}$ —(D1)

In multi-D this becomes in the x-direction diffusion:

$A_{D2x}C\,|_i = ½\,(1/\Delta x_i)\,(k_i\,(\hat{A}_{D2x}C)|_i + (\hat{A}_{D2x}(KC))|_i - c_i(\hat{A}_{D2x}K)|_i)$ in which $KC = K \odot C$ is a vector.

In differential form this is the identity: $2\nabla.k\nabla c= k\nabla^2 c + \nabla^2(kc) - c\nabla^2 k$.

In 1D one can see this is equations eqn (D1) when $\hat{A}_{D2x}=(-1, 2, -1)$.

Here A_{2Dx} is normalised to obtain A_{2Dx} with $\Delta x_i=\Delta y_i=\Delta z_i=1$ and A_{D2x} is obtained from FEM or FD discretisations.

For non-uniform grid spacing use $k_i = \tilde{k}_i/(2\Delta x_i)$ in which $\tilde{k}_i$ is the diffusion coefficient of node/cell i. This k_i uses the approx. for 3 point FD stencils:

$1/(½\Delta x_i+½\Delta x_{i+1})\approx 1/(2\Delta x_i)+1/(2\Delta x_{i+1})$ with an error of 5% when ratio of cell sizes is 1.5.

In the y-direction: $A_{D2y}C\,|_i =½\,(1/\Delta y_i)\,(k_i\,(\hat{A}_{D2y}C)|_i + (\hat{A}_{D2y}(KC))|_i - c_i\,(\hat{A}_{D2y}K)|_i)$, similarly for A_{D2z}.

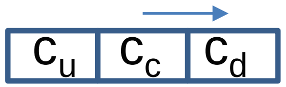
The overall diffusion filter operating on C becomes:

$A_{D2}C = A_{D2x}C + A_{D2y}C + A_{D2z}C$ and is a discretization of $\nabla.k\nabla c$

For advection filter A_x this becomes $A_x = (2/(½\Delta x_{i-1}+\Delta x_i + ½\Delta x_{i+1}))\,\hat{A}_x$ and similarly for A_y, A_z.

Interface tracking with compressive advection

Using the approach of ‘Pavlidis 2016 Compressive…’ eqn 33 we define a –ve diffusion coefficient to maintain a sharp interface and in each direction k_{xi}^-, k_{yi}^-, k_{zi}^-.
Using $(u_i^*, v_i^*, w_i^*)= ((u_i c_{xi}+ v_i c_{yi}+ w_i c_{zi})/\max\{\epsilon_x, c_{xi}^2+ c_{yi}^2+ c_{zi}^2\})\,(c_{xi}, c_{yi}, c_{zi})$ with $\epsilon_x=1^{-10}$ and
$\mu_i^-=-\beta\,(u_i^2+v_i^2+w_i^2)/(u_i^{*2}+v_i^{*2}+w_i^{*2})^{\frac{1}{2}}$ with $\beta=1/4$ and
$k_{xi}^- = \mu_i^-/\Delta x_i$, $k_{yi}^- = \mu_i^-/\Delta y_i$, $k_{zi}^- = \mu_i^-/\Delta z_i$, and the overall diffusion coefficients are:
$k_{xi}=w_{xi}\, k_{xi}^- +(1- w_{xi})\, k_i^+$, $k_{yi}=w_{yi}\, k_{yi}^- +(1- w_{yi})\, k_i^+$, $k_{zi}=w_{zi}\, k_{zi}^- +(1- w_{zi})\, k_i^+$
in which the k_i^+ is positive isotropic diffusion from the previous Petrov-Galerkin method.

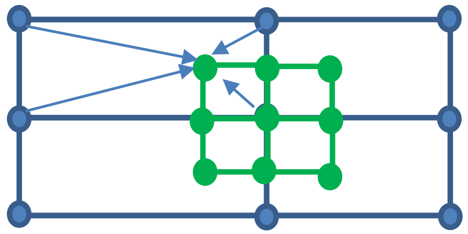
Suppose the cells are arranged 

in which the cells are arranged in the direction of the flow from cell c.

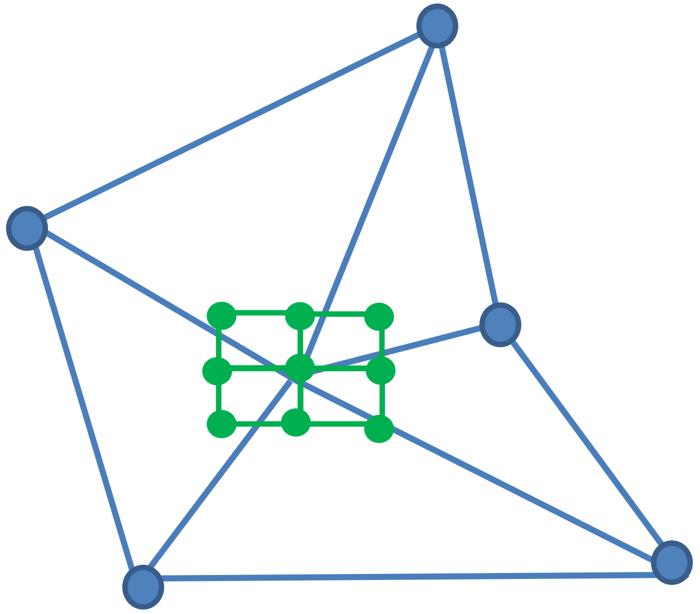
Define the face value of c as $\check{c}_f = \varphi_f\,(c_d-c_u)+c_u$ and
$\varphi_f = f_{limit}(\psi_f, \psi_c)$ if $\psi_c \in [0,1]$ else $\varphi_f = \psi_c$ with $f_{limit}(\psi_f, \psi_c) =\min\{ \max\{1/(3\mathbb{C}_{xi}),2\}\, \psi_c, \max\{0, \psi_f\} \}$
and the Courant number is $\mathbb{C}_{xi}= \Delta t u_i/\Delta x_i$.
Also define $\psi_f=(c_f-c_u)/(c_d-c_u)$, $\psi_c=(c_c-c_u)/(c_d-c_u)$
in which the face value using the trapezium rule $c_f=½\,(c_d+c_c)$.
Also $w_{xi}=(\varphi_f-\psi_f)/(\psi_c-\psi_f)$ and similarly for w_{yi}, w_{zi}.

Structured-unstructured mesh discretisation for rapid/simple discretisation and on the fly filter formation with graph ANNs

One could have used interpolation to extend/diffusion filters to grids with non-uniform spacing. Imagine interpolating from the larger variables Δx, Δy spacing grid to the smaller uniform grid. On the right interpolate from blue nodes to green nodes. The FEM solution variables are the blue nodes.

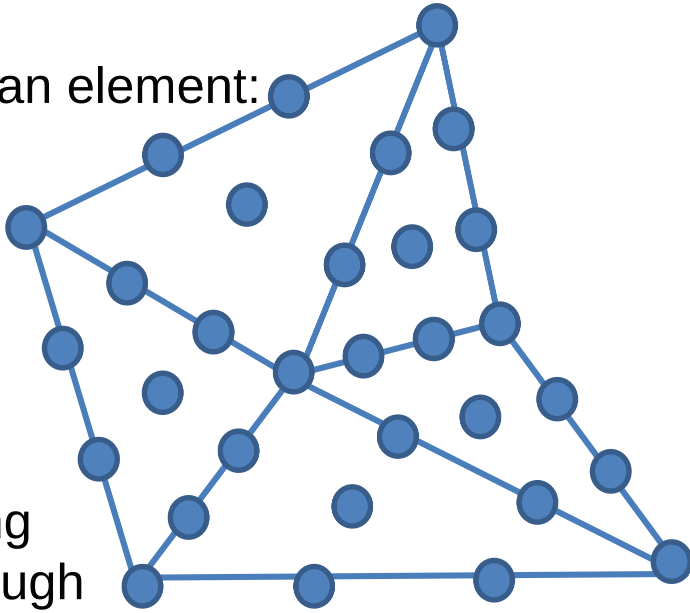


On unstructured grid shrink to small element. The unknowns are on the unstructured grid. If high order elements are used then one might have to interpolate by mapping to certain polynomials. One can also mix discretisation types. e.g. map to rectangular DG structured grid discretisation from a continuous grid of triangles. Care must be taken to avoid singularities in the resulting matrix because values of the unstructured nodes are not defined.



Unstructured mesh discretisation for rapid/simple discretisation and on the fly filter formation with graph and classical ANNs

As an alternative to the structured-unstructured approach one could use the FEM representation of the differential equation or, often equivalently, using mass lumping by having quadrature points at the FEM nodes. This is particularly suitable for high order elements as it makes implementation rapid and simple and allows high order derivatives in the differential equation to be used directly (see right picture).



The 1st order derivative of a field can be discretised FEM over an element:

$\int_E N_i\,(\partial C/\partial x)\,dV = -\int_E (\partial N_i/\partial x) C\,dV + \int_{\Gamma in} n_x N_i\, C_{bc}\, d\Gamma + \int_{\Gamma out} n_x N_i\, C\, d\Gamma$

Using:

$\int_E N_i\,(\partial C/\partial x)\,dV = -\int_E (\partial N_i/\partial x) C\,dV + \int_{\Gamma in} n_x N_i\, C\, d\Gamma + \int_{\Gamma out} n_x N_i\, C\, d\Gamma$

then

$= \int_E N_i\,(\partial C/\partial x)\,dV + \int_{\Gamma in} n_x N_i\,(C_{bc} - C)\,d\Gamma$

and C_{bc} is from the neighbouring element and Γ_{in} , Γ_{out} are the boundary of the element associated with incoming and outgoing information respectively. Using mass lumping and dividing through by the lumped mass: $\partial C/\partial x \approx \partial C/\partial x|_i + (N_p n_x/\Delta x)(C_{bc}-C)|_i$ & N_p=no nodes normal to Γ_{in} .

Unstructured mesh discretisation for rapid/simple discretisation and on the fly filter formation with ANNs (cont)

For second order terms this becomes:

$\int_E N_i (\partial^2 C/\partial x^2)\, dV = - \int_E (\partial N_i/\partial x)(\partial C/\partial x) dV + \int_\Gamma n_x N_i (\partial C/\partial x)|_{bc}\, d\Gamma.$

Using mass lumping and dividing through

by the lumped mass:

$(\partial^2 C/\partial x^2)|_i \approx (\partial^2 C/\partial x^2)|i + (4 N_p n_x/(\Delta x_i + \Delta x_j))(C_{bci} - C_i) - \alpha 2 n_x /(\Delta x_i)(\partial C/\partial x)|_i$

in which α =1 applies a zero derivative b.c. on the element boundary and α=0 does not.

For continuous FEM the same approach can be applied but one may form for each element a value of the derivatives like $(\partial^2 C/\partial x^2)|_i$ at node i and then the final value of $(\partial^2 C/\partial x^2)|_i$ is calculated by taking an area/volume weighting of these derivatives.

Unstructured mesh discretisation for rapid/simple discretisation and on the fly filter formation with ANNs – on the fly shape function derivatives:

To form ∂C/∂x at a node i we use: $(\partial C/\partial x)|_i=\sum_j (N_{xj})|_i C_j$ in which we form $(N_{xj})|_i$ using (Fortran code):

```
do  GI=1,NGI
     AGI=0.
     BGI=0.
     CGI=0.
     DGI=0.

     do  L=1,NLOC
       AGI=AGI+NLX(GI,1,L)*x_loc(1,L)
       BGI=BGI+NLX(GI,1,L)*x_loc(2,L)
       CGI=CGI+NLX(GI,2,L)*x_loc(1,L)
       DGI=DGI+NLX(GI,2,L)*x_loc(2,L)
     end do

     DETJ= AGI*DGI-BGI*CGI
     ! For coefficient in the inverse mat of the Jacobian.
     A11= DGI /DETJ
     A21=-BGI /DETJ
     A12=-CGI /DETJ
     A22= AGI /DETJ

     do  L=1,NLOC
       NX(GI,1,L)= A11*NLX(GI,1,L)+A12*NLX(GI,2,L)
       NX(GI,2,L)= A21*NLX(GI,1,L)+A22*NLX(GI,2,L)
     end do
   end do
```

Thus in 2D one stores, at each FEM node, the 2x2 inverse of the FEM Jacobian matrix which becomes a 3x3 matrix in 3D. Then form on the fly $(N_{xj})|_i$ when forming the discretisation for each element and to form the discretisation of terms like $(\partial C/\partial x)|_i=\sum_j (N_{xj})|_i C_j$. This makes the method fast and memory efficient. In FEM form this is:

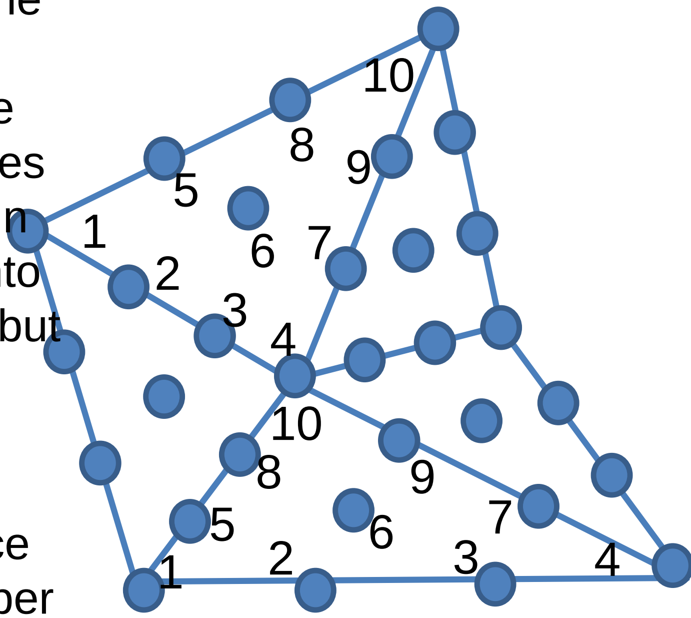
$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = \mathbf{J}^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} \quad ; \qquad \mathbf{J} = \begin{bmatrix} \sum \dfrac{\partial N_i}{\partial \xi} x_i, & \sum \dfrac{\partial N_i}{\partial \xi} y_i, & \sum \dfrac{\partial N_i}{\partial \xi} z_i \\ \sum \dfrac{\partial N_i}{\partial \eta} x_i, & \sum \dfrac{\partial N_i}{\partial \eta} y_i, & \sum \dfrac{\partial N_i}{\partial \eta} z_i \\ \sum \dfrac{\partial N_i}{\partial \zeta} x_i, & \sum \dfrac{\partial N_i}{\partial \zeta} y_i, & \sum \dfrac{\partial N_i}{\partial \zeta} z_i \end{bmatrix}.$$

DG Unstructured mesh discretisation data structure:

To form an efficient data structure for the neural network one may split the data structure into two 1D CNNs. Assuming the nodes are ordered consecutively local to each element then **one filter is structured** with lopsided (not centred) filters and a filter specific for each local element. Then each of these filters has a stride of N_{loc} which is the number of local nodes per element (on the right its 10 for 2D cubic elements (see right pic) and 20 for 3D tetrahedral elements).

The second set of filters are formed from this 1D vector but with **graph neural networks but with limited connectivity** – just connecting to the single node on the other side of the face (see previous section on lumping). However, there may be a number of faces a node is opposite ~5 at corner nodes, 2 at edge nodes, 1 at face nodes, 0 in interior nodes in 3D (mean 2.4 graph connections per cubic DG node) and max of 2 in 2D. Even more efficient data structures are possible which only take into account the connectivity of the elements in the graph neural networks but these are more complex.

To solve the equations efficiently it is suggested that one would perform lots of Jacobi style iterations on the finest grid (possibly only updating the nodes within an element for efficiency) then use the Space Filling Curve (SFC) multi-grid applied to the collapsed to one variable per element. The SFC must go through all the element of the mesh.



Stokes Flow Solution Method in AI-CFD

Governing eqns ($\mathbf{q}=(u \ \ v)^T$)

Derived eqns – solution steps:

1) $\sigma \, \mathbf{q} - \mu \nabla . \nabla \mathbf{q} + \nabla p = 0$,
2) $\nabla . \mathbf{q} + (\mu/\Delta x^2) p - (\Delta x/L) \, \mu \, \nabla . \nabla \, p = (\mu/\Delta x^2) p$,

in which the term just before = avoids pressure modes and the terms $(\mu/\Delta x^2)p$ are to make it easier to apply block Jacobi smoothing.

Use Harmonic average of σ in multi-grid.

Also in multi-grid method form a block Jacobi relaxation which couples u,v,w,p and thus the ANN channel weights for this are the inverse of the associated block.

CFD DG CNN filters – Pressure/Diffusion operator with interior penalty function method

For the diffusion operator ∇^2 p which we might solve for pressure, say, have a 3x3 but with a stride of 2. Need for of these in 2D rectangular elements and 8 in 3D.

For multi-grid collapse to element the 4 variables in each element in 2D. Thus getting the same stencil used for the continuous FEM.

Apply each filter by missing out every other node in each direction (stride =2).

The green square, in the figure, shows the stencil associated with local node 1 – this has a 3x3 stencil. Diffusion discretisation:

$\int_{Ve} N_i \nabla^2 p \, dV = -\int_{Ve} (\nabla N_i) . (\nabla p) \, dV + \int_{\Gamma e} N_i \, n . (\nabla p) \, d\Gamma$

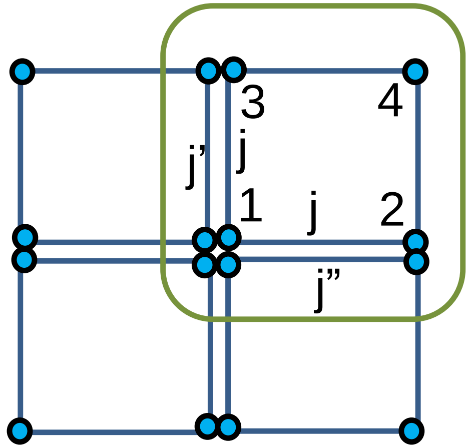
In which n is the normal to the element. The filter then becomes for local node 1 using the interior penalty function method:

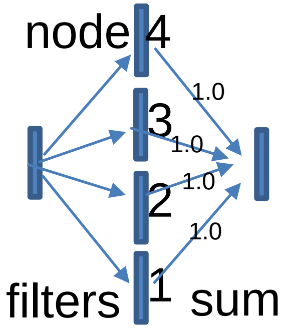
$A_{filtD1\,j} = -\int_{Ve} (\nabla N_1) . (\nabla N_j) \, dV - \int_{\Gamma e} N_1 \lambda N_j \, d\Gamma$;

$A_{filtD1\,j'} = \int_{\Gamma e} N_1 \lambda N_{j'} \, d\Gamma$; $A_{filtD1\,j''} = \int_{\Gamma e} N_1 \lambda N_{j''} \, d\Gamma$

in which $\lambda = 2/\Delta x$.

This is repeated for the 4 local nodes as shown in diagram at bottom to get all filters A_{filtD1} , A_{filtD2} , A_{filtD3} , A_{filtD4} .





CFD DG CNN filters – For advection discretization u . ∇C or for non-linear momentum

Use two versions of each component of the derivative ∇ – one for positive and one for negative direction (e.g. u_1) of advection $\mathbf{u}=(u\ v)^T$. in which subscript 1 indicates its for node 1. The basic idea is to form the filter centred around node 1, say, depending on if u_1 is positive or negative and then pre-multiply the resulting filter by u_1 for form a discretisation of $u_1\ \partial C/\partial x$. That is:

$u_1 A_{filtu1} = u_1(H(u_1)\, A_{filtu1}^{right} + H(-u_1)\, A_{filtu1}^{left})$ in which A_{filtu1}^{right} , A_{filtu1}^{left} are filter associated with up and down advection (they contain the same volume integrals) and H (H(a)=1 if a≥0 otherwise 0) is the heavy side function.

Similarly for v1:

$v_1 A_{filtv1} = v_1(H(v_1)\, A_{filtv1}^{up} + H(-v_1)\, A_{filtv1}^{down})$

The discretisation for the horizontal derivative is:

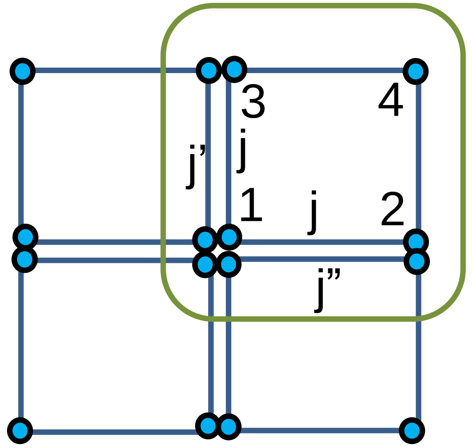
$\int_{Ve} N_1\, \mathbf{u} . (\nabla C)\, dV \approx u_1 \int_{Ve} N_1\, \partial C/\partial x\, dV + v_1 \int_{Ve} N_1\, \partial C/\partial y\, dV$

Thus $\int_{Ve} N_1\, \partial C/\partial x\, dV$:

$A_{filtu1j}^{right} = \int_{Ve} \partial N_1/\partial x\, N_j\, dV$; $A_{filtu1j'}^{right} = \int_{Ve} N_1\, n_x\, N_{j'}\, dV$ with $n_x=-1$;

$A_{filtu1j}^{left} = \int_{Ve} \partial N_1/\partial x\, N_j\, dV + \int_{Ve} N_1\, n_x\, N_j\, dV$

and similarly for the vertical to form A_{filtv1}^{up} & A_{filtv1}^{down} and for all the other local nodes 2,3,4 to produce A_{filtu2}, A_{filtv2}, A_{filtu3}, A_{filtv3}, A_{filtu4}, A_{filtv4}.

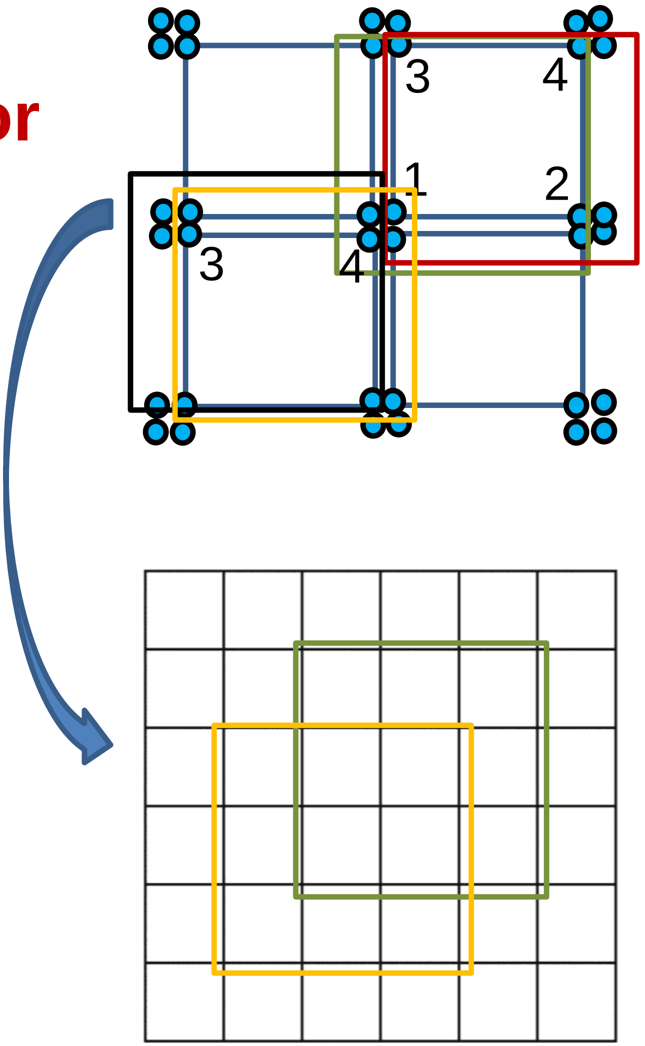


DG CNN filters – How to use stride=2 for the discretization of advection and diffusion – need 4 channels or filters in 2D one for each local node

Nodes inside green box shows filter for local nodes 1, red for nodes 2, black 3, yellow 4.

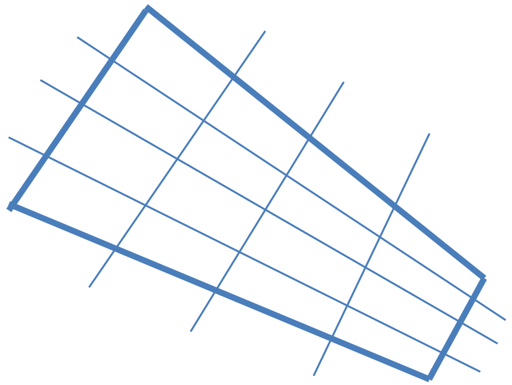
Bottom diagram shows how the 4 DG elements are mapped to a regular grid where a 3x3 filter is placed over the domain (an example is shown in green which corresponds to the green filter shown in the top diagram) in order to form a CNN that can do DG discretization on a structured grid.

For advection-diffusion the filter for node 1 (green) is:
$A_{filt1} = kA_{filtD1} + u_1 A_{filtu1} + v_1 A_{filtv1}$ in which k is diffusion and similarly to produce all filters A_{filt1} , A_{filt2} , A_{filt3} , A_{filt4} .



CFD DG CNN filters – Distorted blocks or unstructured meshes

For local node 1, say, need a 2 filters (for advection to define incoming and outgoing information) and 1 filter for diffusion for each face touched by local node 1. That is 2 faces in 2D and 3 faces in 3D. For advection need 2 filters in 2D (associated with each advection direction) in 3D for the volume of each element and 3 filters in 3D. Repeat for each of the local nodes.

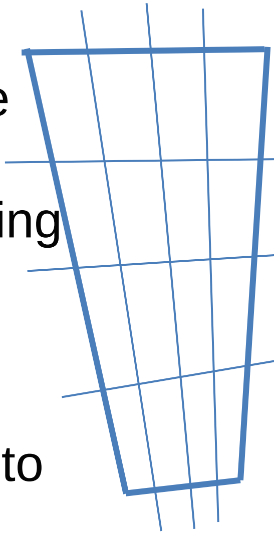


CFD DG CNN filters – Conservative advection ∇ . uC rather than u . ∇C

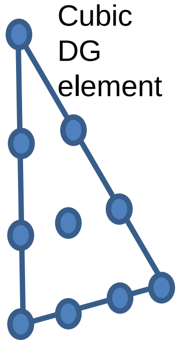
This is similar to the previous non-conservative approach (using the same filters) to discretising but a rapid way to do this is simply operate on the product **u**C, directly, rather than C. When one needs to decide if we have to use incoming or outgoing direction information on surfaces associated with node i, say, then use the average velocity 0.5($\mathbf{u}_i$+$\mathbf{u}_{i'}$) to determine this in which i' is the node on the other side of the face and use the normal **n** to the normal to the face to help determine if we have incoming or outgoing information and also could advect with this velocity 0.5($\mathbf{u}_i$+$\mathbf{u}_{i'}$) across faces.

High order DG CNN filters with or without distorted blocks or unstructured meshes using 1D CNNs or 1D structured weights

One can order the nodes along a line element wise so that the element nodes are all grouped together. This means that a N filters with a structured compact representation can be applied to all the elements simultaneously in which N is the number of local nodes. This means that it will be just as fast as the 1D CNN approach. The coupling between the elements is then where the indirect addressing occurs and a graph neural network is needed for this coupling. However, mass lumping of the surface integrals can be used so that the sparsity of this coupling could be very limited.
In addition, further reductions in the indirect addressing may be possible taking into account the structure of the surface integrals across neighbouring elements.

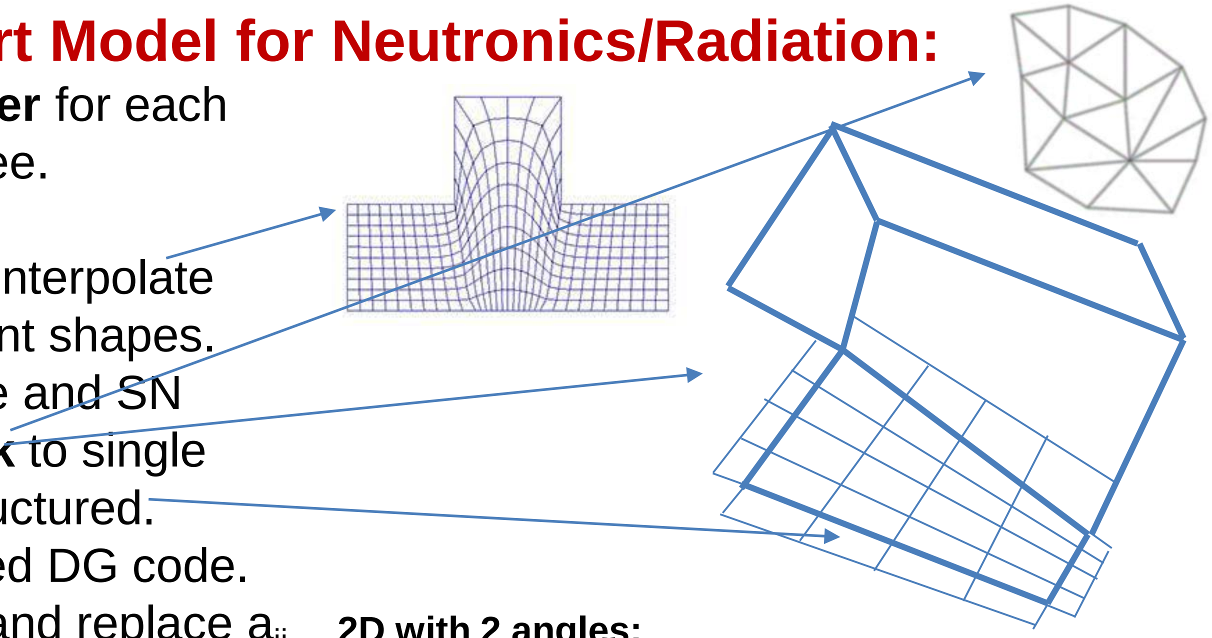


For the DG multi-grid method one can use a Space Filling Curve (SFC) within each element and collapse the nodes, for algebraic multi-grid method, in pairs along the SFC. When one finally has one variables within each element then one needs to resort to a fully unstructured representation (for an unstructured mesh) but an SFC multi-grid method can then be applied at this level to achieve good solver scaling.

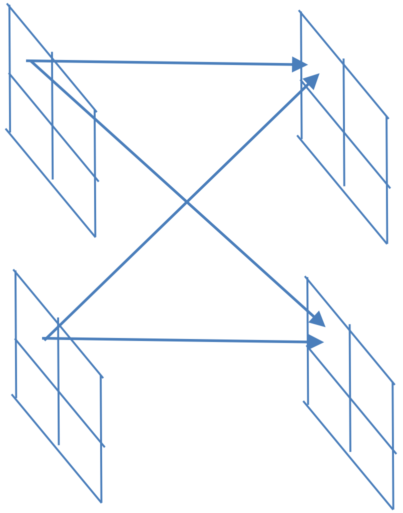


Matrix Free Boltzmann Transport Model for Neutronics/Radiation:

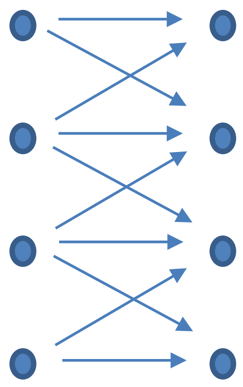
1) **Have a single filter for advection layer** for each SN direction to make method matrix free.
2) **Each SN direction has a channel.**
3) **For distorted blocks** have filters that interpolate between different orientations & element shapes.
4) **Use Multi-Grid** in DG Cartesian Space and SN angle. Collapse each **structured block** to single cell/node. Halo cells/DG nodes still structured.
5) **Use block unstructured** e.g. structured DG code.
6) **For linear DG** in 1D have 2 channels and replace a_{ij} with the inverse of the element block diagonal matrix. Similarly 3D have 8 channels.
7) **Start simple** with 1 block and 1 direction.
8) **For scattering removal**. Could have a filter for each material in layer or have different weights of the connections – see pics bottom right.
9) **If massive number of angles** can efficiently have a single element in a structured block.



2D with 2 angles:



CNN Layers in 1D:
Advection/absorption



Boltzmann Details: Formation of SN DG Filters:

1) For structured block that is not distorted then a single filter is enough to represent $\Omega \cdot \nabla \Phi$ for each SN direction as each sub-element is the same.
2) For distorted blocks have form filters from spatial tables for each element and face of elements.
3) The filters for 3D DG 8 node bi-linear elements have 3 inner element contributions for each component of ∇. These are multiplied by Ω to achieve a discretization of $\Omega \cdot \nabla \Phi$ internal to an element. Each of these 3 filters has 8x8 values operating on the nodes of the element.
4) For the 3D DG 8 node element there for each face there is an incoming and an outgoing contribution filter set. Each of these has 3 components (filters) and the filters are each of size 4x4. There are thus 8 of these two surface filters (one for each element face) and thus there are 8 times 2 times 3 surface (=48) filters to represent the surface integrals.
5) For SN in angle multi-grid formation and using CNN then choose SN directions so as to map to a square grid.

Boltzmann Details: Solution Free Boltzmann transport – the solution vector too large to fit on computer e.g. for some GPU or AI computers say

The coarse grid in angle equation is:

$A\,\Phi \;+\; B\,\Psi \;= s_{\Phi}$

The fine grid equation is:

$C\,\Phi \;+\; D\,\Psi \;= s_{\Psi}$

In which Φ is the coarse grid in angle solution vector (e.g. scalar flux or low SN order angular flux) and Ψ is the fine SN angular vector. D is the discretized transport operator. The coarse grid in angle equation could be from diffusion theory (DSA) or low order SN.

Combining these the coarse grid equation becomes:

$(A \;- B\,D^{-1}\,C)\,\Phi \;= s_{\Phi} - \;BD^{-1}\,s_{\Psi}$

The basic idea is only to use the coarse (in angle solution) vector Φ and never actually form the whole solution vector Ψ or in the above $D^{-1}\,C\,\Phi$. Instead each direction is solved for with D^{-1} and is discarded after operating on B.

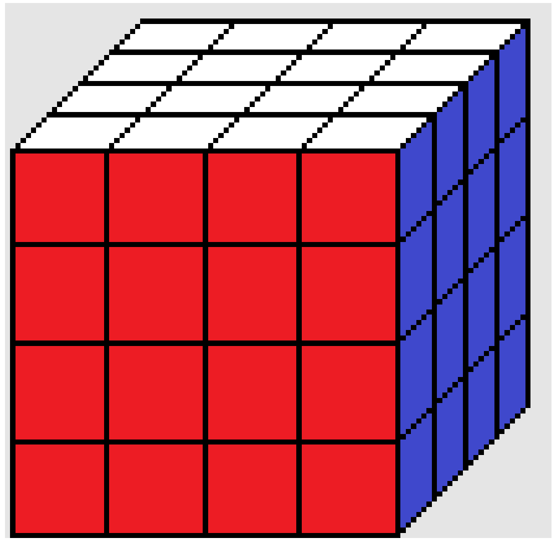
A multi-grid in angle and space can be applied to the SN in angle discretized system.

For time dependent one can only resolve the low order SN solution in time. For SN only include transport and total cross section terms in matrix D put other terms into the sources.

Boltzmann Details: Discretization in SN Angle and Cartesian Space on Structured Grids

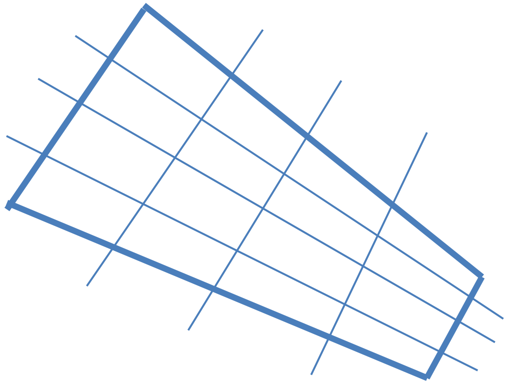
Discretization in SN Angle:

1) Use hexahedra on unit sphere and then put a rectangular grid across each square. See figure.
2) Have a power of 2 of the number of cells across on this rectangle e.g. 4 in the figure.
3) Each cell then represents an SN direction.



Discretization in Cartesian space:

1) Use DG in space with NxN elements and N is a power of 2 – needed for multi-grid method. See figure bottom right in which N=4.
2) For 4-node DG linear 2D element have 4 convolutional channels. One for each node.



Boltzmann Details: Convolutional Filters in SN Angle and Cartesian Space on Structured Grids

1) Assuming 1D in space and in angle (similar in multi-dimensions) have a different filter for each SN direction but same across space. See figure.

2) Add discretization filters together to form coarser filters.

3) In angle just add them together to form coarser SN discrete equations.

4) In Cartesian space use the prolongation and restriction matrices to form coarse discrete equations.

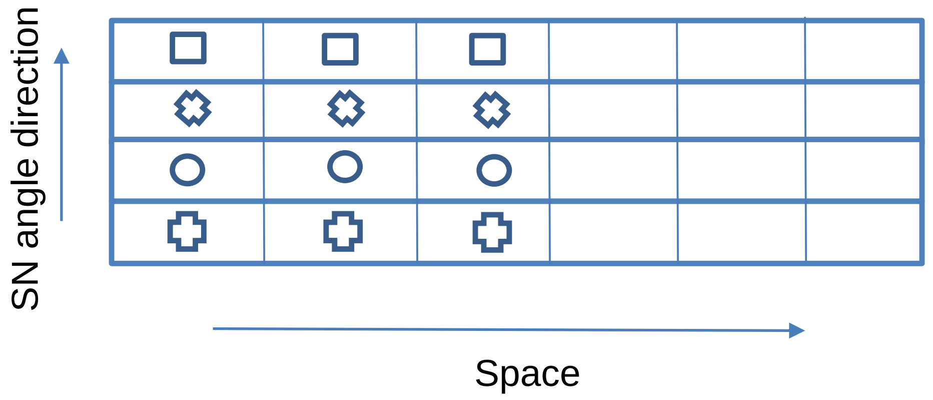


Figure: Space - SN angle CNN filters

Boltzmann Details: DG Prolongation and Restriction Matrices

Fine to coarse grid equation is:

$\int N_i^c \,(\psi_c - \psi_f)\, dV$; $\quad \psi_c = \sum N_j^c \,\psi_{cj}$; $\quad \psi_f = \sum N_j^f \,\psi_{fj}$;

$M^c \,\psi_c = N^{cf} \,\psi_f$;

Thus: $\psi_c = (M^c)^{-1} N^{cf} \,\psi_f = P^{cf} \,\psi_f$.

Course to fine grid:

$\int N_i^f \,(\psi_f - \psi_c)\, dV$;

$M^f \,\psi_f = N^{fc} \,\psi_c$;

$\psi_f = (M^f)^{-1} N^{fc} \,\psi_c = P^{fc} \,\psi_c$.

Therefore the fine grid matrix equation is: $A^f \,\psi_f = s_f$.

The coarse matrix: $P^{cf} A^f P^{fc} \,\psi_c = P^{cf} s_f = s_c$.

Or: $A^c \,\psi_c = s_c$.

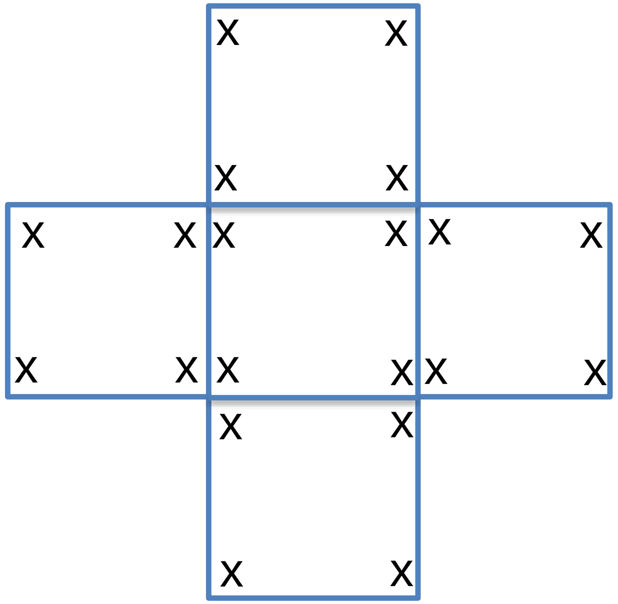
The filters are similarly operated on by these prolongation and restriction matrices: P^{fc}, P^{cf} .

Boltzmann Details: Structured mesh linear DG CNN filters in 2D

Suppose ψ^p is the SN direction angular flux with angle Ω^p and $\psi^p = \sum N_j \psi^p_j$.

The filter is the entire discretization centred on the inner element in the figure in which DG nodes are represented by x.
The filter is defined through the discretization for element e at the centre of figure:

$-\int_{V_e} (\nabla N_i) \cdot \Omega^p \psi^p \, dV + \int_{\Gamma_{e(\Omega.n<0)}} N_i \, \Omega^p \cdot n \, \psi^p_{bc} \, d\Gamma + \int_{\Gamma_{e(\Omega.n>0)}} N_i \, \Omega^p \cdot n \, \psi^p \, d\Gamma$
$+ \int_{V_e} N_i \, \sigma^p \psi^p \, dV = s^p$

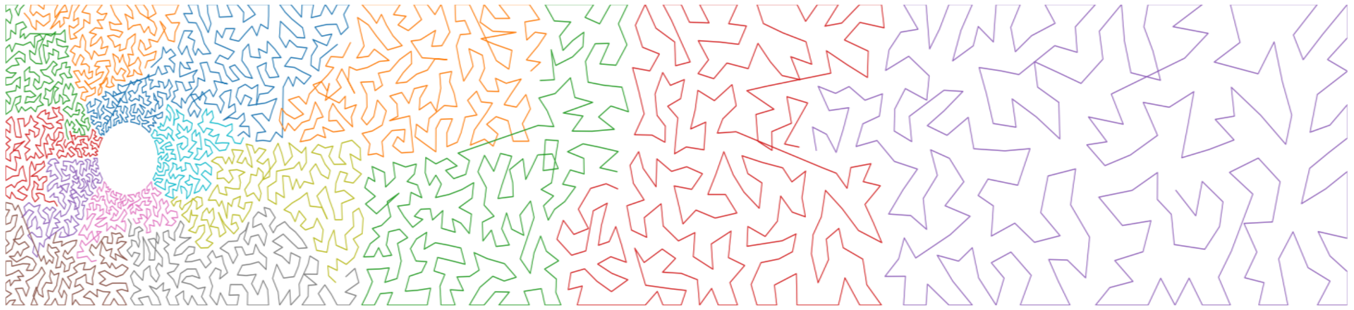


Test cases to demonstrate approach – simple to complex:

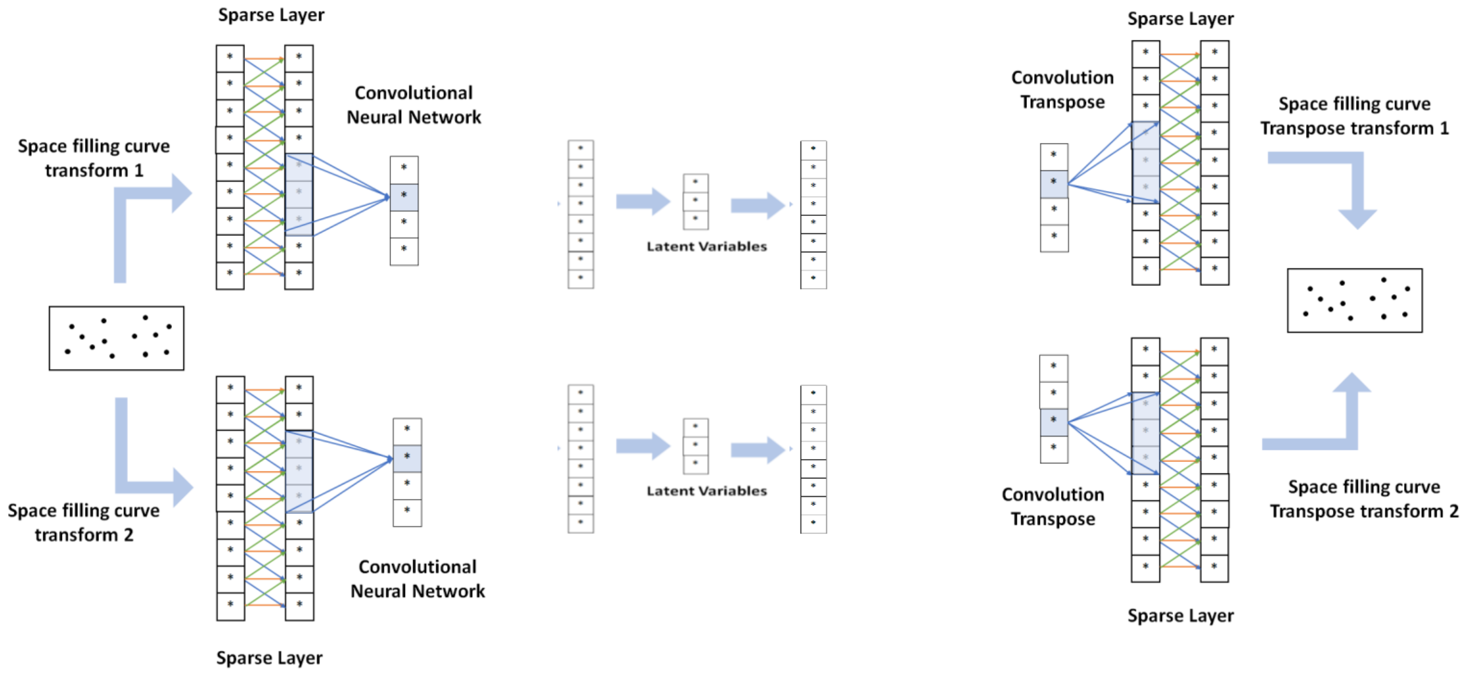
1) Time dependent 2D advection diffusion equation on structured grid with central difference discretization. (done)
2) 2D Burgers equation discretization and solution with non-linearity built into network. (done)
3) Multi-grid on 2D advection diffusion equation. (done)
4) Discretize fluids equation with central difference and projection method Poisson equation for pressure so can use an A grid. (done)
5) 2D Burgers equation to demonstrate adjoint.
6) Differentiate fluids equation to find adjoint by defining activation functions.
7) Discretize Boltzmann Transport equation with 4D multi-grid on a structured grid – start with diamond differencing then DG in 2D and with SN.
8) Solve OPS/2 test cases: convection test case.
9) Parallel for structured grid advection-diffusion, fluids and AI model.
10) Unstructured and semi-structured mesh multi-grid based on space-filling curves or Earth mover distance optimization (see unstructured mesh slides).

Unstructured mesh CNN - Multi-dimensional convolutional methods – extension of 1D Space Filling Curve (SFC) method

Space Filling Curve going through all the nodes of an unstructured mesh (right)



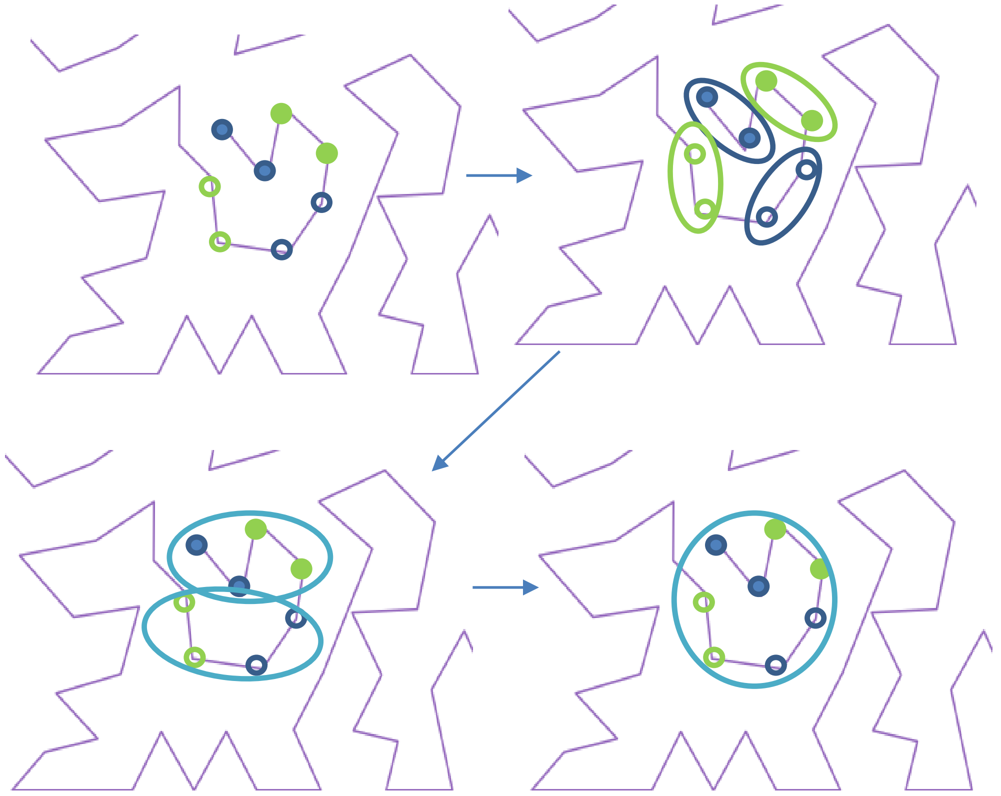
2 Space Filling Curve multi-grid methods (right)


Sparse Layer
Convolutional Neural Network
Space filling curve transform 1
Space filling curve transform 2
Latent Variables
Latent Variables
Convolutional Neural Network
Sparse Layer
Sparse Layer
Convolution Transpose
Space filling curve Transpose transform 1
Convolution Transpose
Space filling curve Transpose transform 2
Sparse Layer

Unstructured mesh CNN - Multi-dimensional convolutional methods – extension of 1D Space Filling Curve (SFC) method

In the multi-grid method one first collapses the nodes with the same color to reduce the number of nodes by a factor of 2 on the course grid (see picture on the right). Then repeat to reduce by a factor of two and so on until we get to one node.

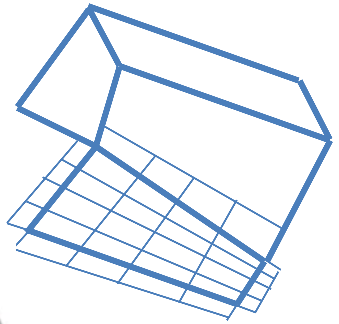
The matrices are formed on each grid level simply by summing the edge values from the finest grid.

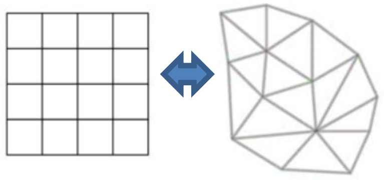


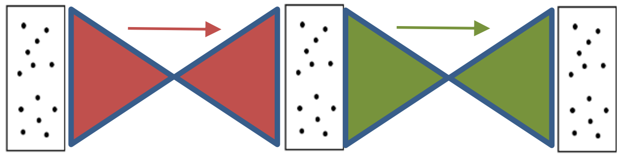
Algebraic node coarsening along the SFC – the circles/ellipses represent coarsened nodes

Unstructured meshes and parallel

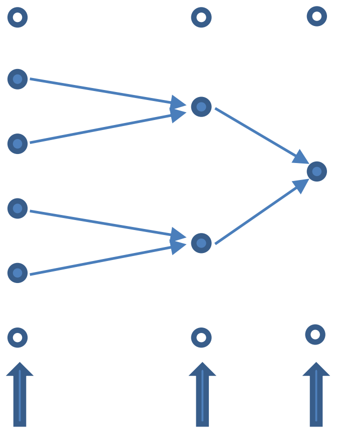
1) Use semi-structured or block approach with unstructured grid representing the interaction between the blocks to start with. See figure.
2) Use optimal mappings between unstructured and structured meshes (interaction between structured blocks or elements) based on the earth mover’s distance (the Wasserstein metric) using a NN for: a) the mapping of the subdomains to the AI computer cores taking its architecture; b) mapping subdomains to a 2D or 3D structured grid on which the multi-grid CNN can be applied.
3) For multi-grid solver either use SFC approach on unstructured representation or structured CNN formed from earth movement distance optimization.
4) For fully unstructured meshes (not nec. block structured) use a graph structure for the CNN and have different weights on all the edges of this graph.
5) For parallel have a halo update on each coarsening level of the multi-grid CNN method.







2 SFC MG CNN unstructured



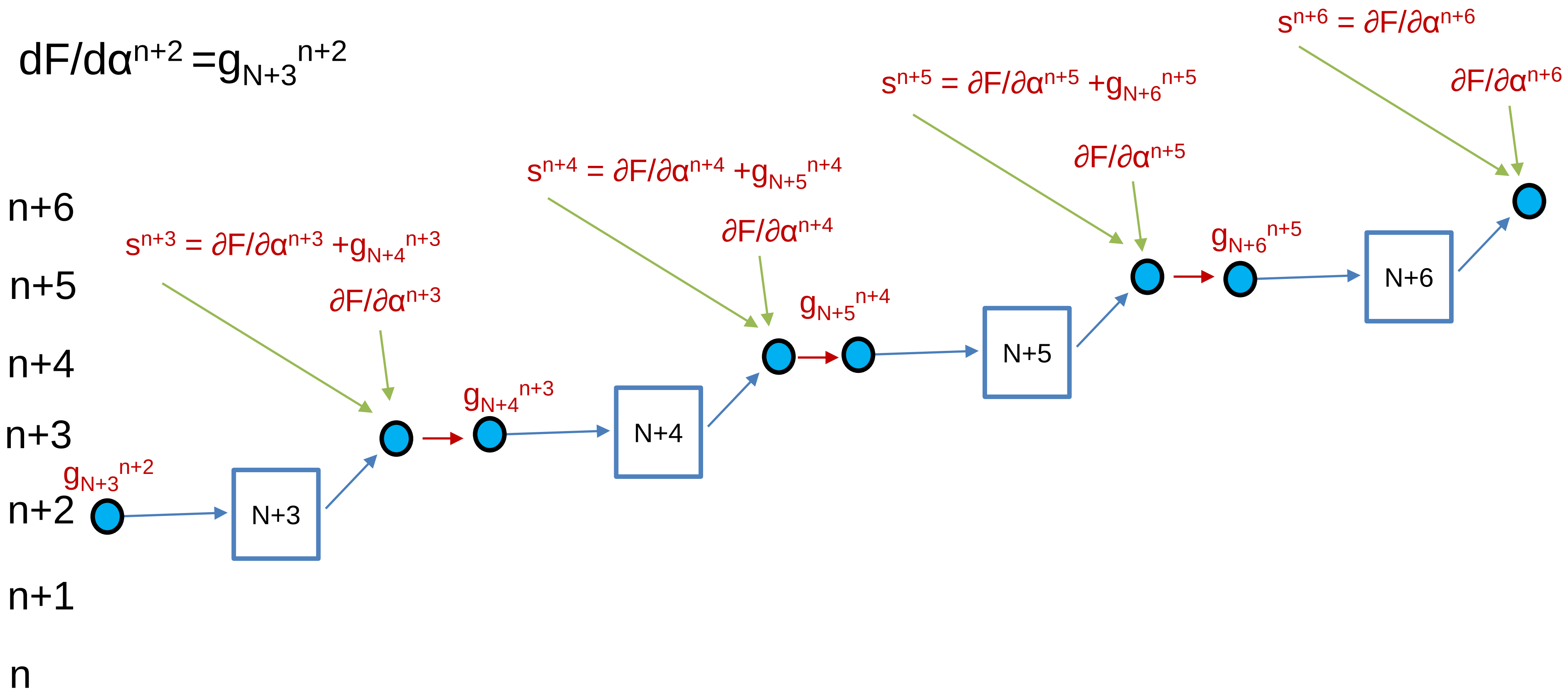
MG CNN halo update

Forming
(i) sensitivities,
(ii) data assimilation,
(iii) uncertainty quantification,
(iii) control
(iv) and optimisation
using AI-HFM & AI-Physics modelling

Two time level stepping method: F is the data mismatch functional in data assimilation
s^{n+3} is the source that's used in the backpropergation algorithm for neural network N+3
g_{N+3}^{n+2} is the gradient contribution from neural network N+3 with a source s^{n+3}

The final sensitivities are:

$$dF/d\alpha^{n+2} = g_{N+3}^{n+2}$$



$s^{n+6} = \partial F/\partial\alpha^{n+6}$
$\partial F/\partial\alpha^{n+6}$
$s^{n+5} = \partial F/\partial\alpha^{n+5} + g_{N+6}^{n+5}$
$\partial F/\partial\alpha^{n+5}$
g_{N+6}^{n+5}
N+6
$s^{n+4} = \partial F/\partial\alpha^{n+4} + g_{N+5}^{n+4}$
$\partial F/\partial\alpha^{n+4}$
g_{N+5}^{n+4}
N+5
$s^{n+3} = \partial F/\partial\alpha^{n+3} + g_{N+4}^{n+3}$
$\partial F/\partial\alpha^{n+3}$
g_{N+4}^{n+3}
N+4
g_{N+3}^{n+2}
N+3
n+6
n+5
n+4
n+3
n+2
n+1
n

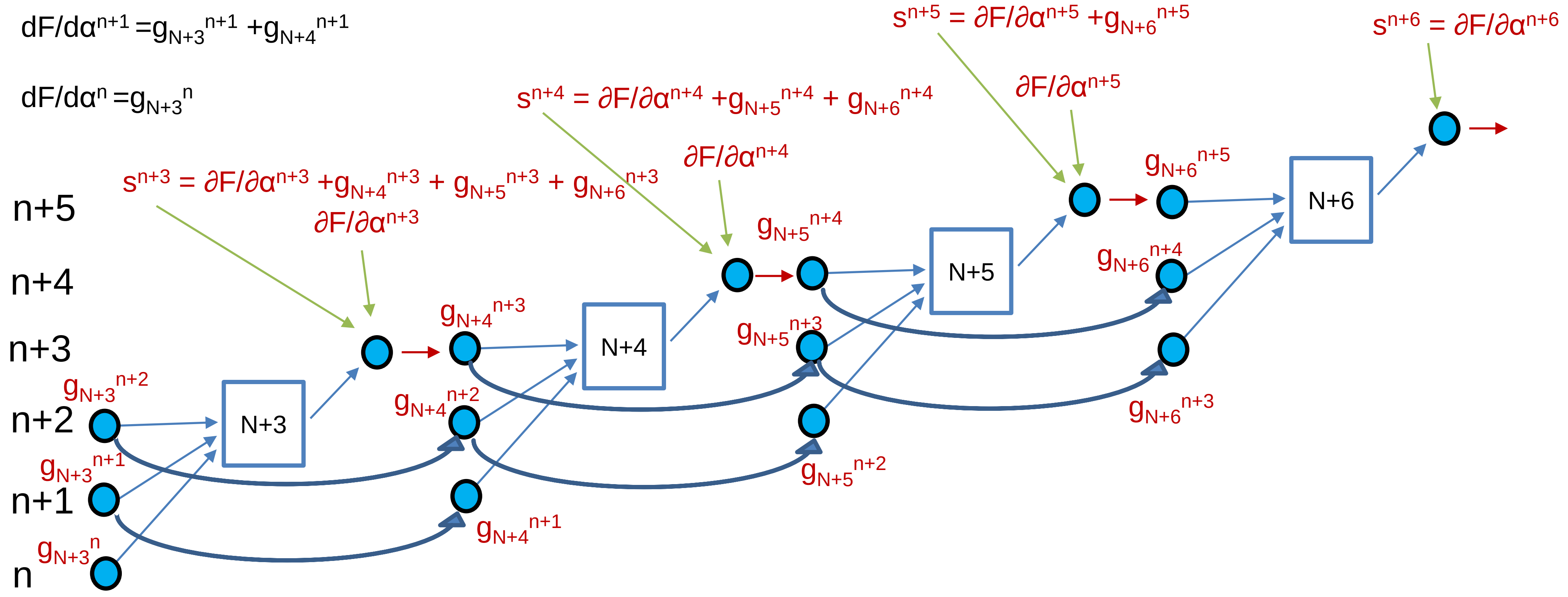
Multi time level stepping method: F is the data mismatch functional in data assimilation s^{n+3} is the source that's used in the backpropergation algorithm for neural network N+3 g_{N+3}^{n+2} is the gradient contribution from neural network N+3 with a source s^{n+3}

The final sensitivities are:

$dF/d\alpha^{n+2} = g_{N+3}^{n+2} + g_{N+4}^{n+2} + g_{N+5}^{n+2}$

$dF/d\alpha^{n+1} = g_{N+3}^{n+1} + g_{N+4}^{n+1}$

$dF/d\alpha^{n} = g_{N+3}^{n}$


$s^{n+5} = \partial F/\partial\alpha^{n+5} + g_{N+6}^{n+5}$
$s^{n+6} = \partial F/\partial\alpha^{n+6}$
$\partial F/\partial\alpha^{n+5}$
$s^{n+4} = \partial F/\partial\alpha^{n+4} + g_{N+5}^{n+4} + g_{N+6}^{n+4}$
$\partial F/\partial\alpha^{n+4}$
g_{N+6}^{n+5}
$s^{n+3} = \partial F/\partial\alpha^{n+3} + g_{N+4}^{n+3} + g_{N+5}^{n+3} + g_{N+6}^{n+3}$
$\partial F/\partial\alpha^{n+3}$
n+5
n+4
n+3
n+2
n+1
n
N+3
N+4
N+5
N+6
g_{N+5}^{n+4}
g_{N+6}^{n+4}
g_{N+4}^{n+3}
g_{N+5}^{n+3}
g_{N+3}^{n+2}
g_{N+4}^{n+2}
g_{N+6}^{n+3}
g_{N+3}^{n+1}
g_{N+5}^{n+2}
g_{N+4}^{n+1}
g_{N+3}^{n}

Data Assimilation and Control using Adversarial Autoencoders (AAE) with sensor outputs:

Suppose $\hat{s}_i^n$ are the desired values at the sensors (ith sensor and nth time level). $\hat{s}_i^n$ could be the values of the measurements at the sensors for the data assimilation (DA) or for control the desired value of the variable e.g. $\hat{s}_i^n$ =21 Deg C for temperature comfort of the room or $\hat{s}_i^n$ =400 for CO2 – the background CO2 level or $\hat{s}_i^n$ =0 for viral load and for controls c_i^n =0 could be the desired air velocity of a Dyson fan (desired to minimize energy consumption) and $\hat{c}_i^n$ =background temperature of the Dyson fan again to minimize its energy use. Everything we have said about the sensor values $\hat{s}_i^n$ also applies to the controls $\hat{c}_i^n$.

Relaxation of the AAE sensor or control inputs

Once can decide that a desired value for control or in DA is more or less importânt with a relaxation coefficient β say, with $\beta \epsilon [0,100]$ in which $\beta = 0$ corresponds to no importance and in which case one uses the value that the time stepping method predicts $\tilde{s}_i^n$, say, otherwise ône uses the value:

$s_i^n = \beta \hat{s}_i^n + (1- \beta) \tilde{s}_i^n$

($\hat{s}_i^n$ desired value and $\tilde{s}_i^n$ the best prediction e.g. from s_i^{n-1} or from the output of the AAE).

Notice that $\beta=1$ corresponds to a ‘standard’ weighting and $\beta>1$ puts more importance on achieving the desired output $\hat{s}_i^n$. Similarly for the controls c_i^n.

Adding the error Δs_i^n in the AAE output-difference:

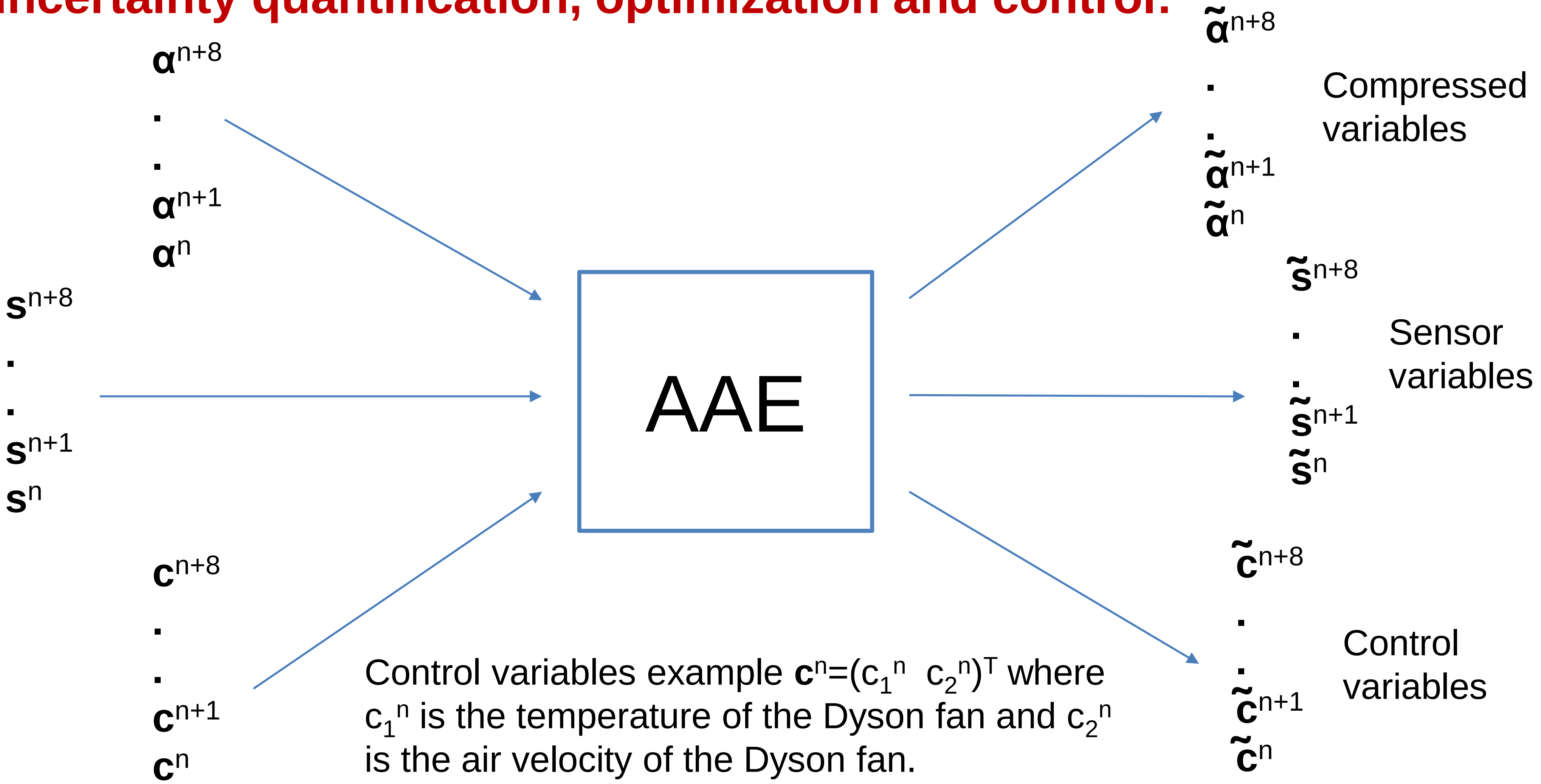
$\Delta s_i^n = \tilde{s}_i^n - \hat{s}_i^n$ in which s_i^n is the AAE output.

Keeping on accumulating the difference:

$\Delta\Delta s_i^n \rightarrow \Delta\Delta s_i^n + \Delta s_i^n$

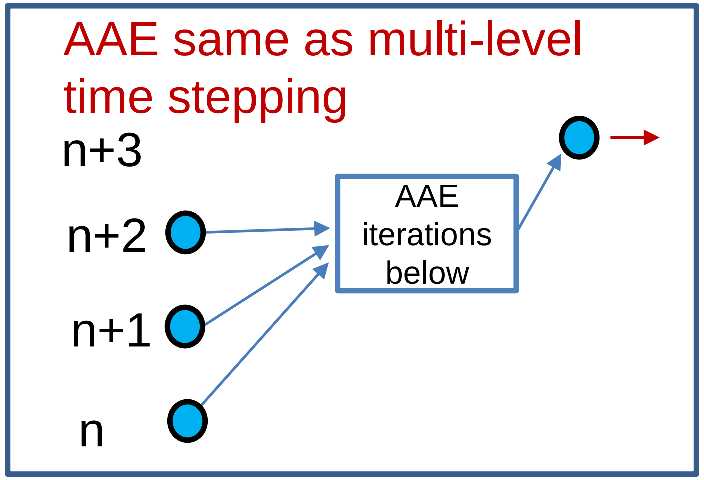
Starting from zero (0.0) and add that into the input. Similarly for the controls c_i^n.

Adversarial Autoencoder (AAE) for time-stepping, data assimilation, uncertainty quantification, optimization and control:


$\boldsymbol{\alpha}^{n+8}$ . . $\boldsymbol{\alpha}^{n+1}$ $\boldsymbol{\alpha}^{n}$
$\mathbf{s}^{n+8}$ . . $\mathbf{s}^{n+1}$ $\mathbf{s}^{n}$
$\mathbf{c}^{n+8}$ . . $\mathbf{c}^{n+1}$ $\mathbf{c}^{n}$
AAE
$\tilde{\boldsymbol{\alpha}}^{n+8}$ . . $\tilde{\boldsymbol{\alpha}}^{n+1}$ $\tilde{\boldsymbol{\alpha}}^{n}$
Compressed variables
$\tilde{\mathbf{s}}^{n+8}$ . . $\tilde{\mathbf{s}}^{n+1}$ $\tilde{\mathbf{s}}^{n}$
Sensor variables
$\tilde{\mathbf{c}}^{n+8}$ . . $\tilde{\mathbf{c}}^{n+1}$ $\tilde{\mathbf{c}}^{n}$
Control variables

Control variables example $\mathbf{c}^n=(c_1^n \;\; c_2^n)^T$ where c_1^n is the temperature of the Dyson fan and c_2^n is the air velocity of the Dyson fan.

AAE sensitivities with multi time level stepping – $(.)^l$ corresponds to the lth AAE iteration – 3 AAE iterations below

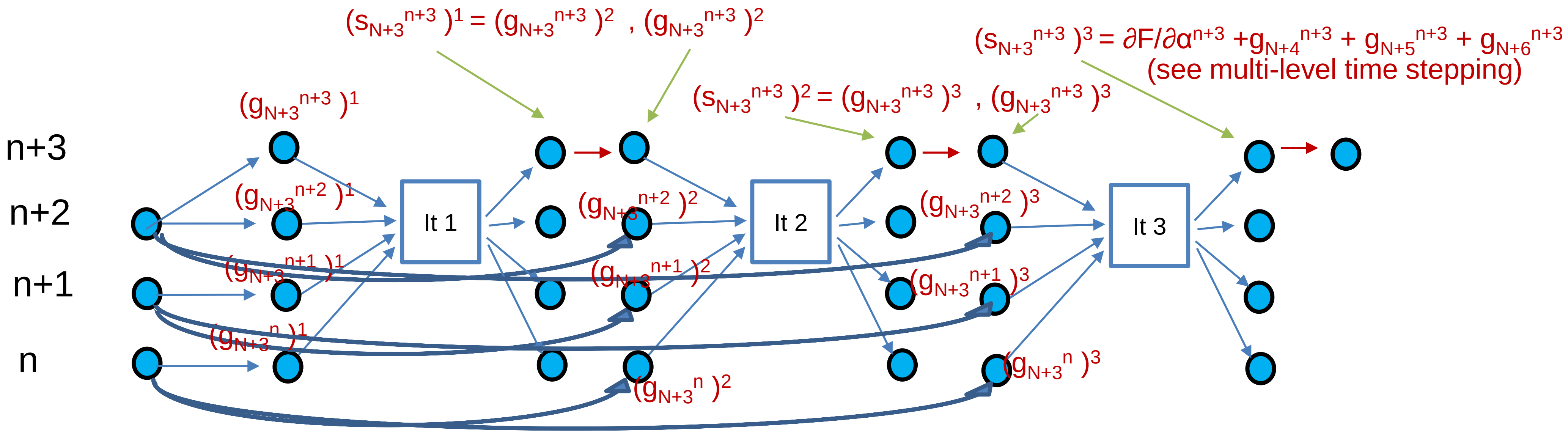


The final sensitivities are:

$$g_{N+3}^{n+2} = dF/d\alpha^{n+2} = (g_{N+3}^{n+3})^1 + (g_{N+3}^{n+2})^1 + (g_{N+3}^{n+2})^2 + (g_{N+3}^{n+2})^3$$

$$g_{N+3}^{n+1} = dF/d\alpha^{n+1} = (g_{N+3}^{n+1})^1 + (g_{N+3}^{n+1})^2 + (g_{N+3}^{n+1})^3$$

$$g_{N+3}^{n} = dF/d\alpha^{n} = (g_{N+3}^{n})^1 + (g_{N+3}^{n})^2 + (g_{N+3}^{n})^3$$


$(s_{N+3}^{n+3})^1 = (g_{N+3}^{n+3})^2$, $(g_{N+3}^{n+3})^2$
$(s_{N+3}^{n+3})^3 = \partial F/\partial\alpha^{n+3} + g_{N+4}^{n+3} + g_{N+5}^{n+3} + g_{N+6}^{n+3}$ (see multi-level time stepping)
$(s_{N+3}^{n+3})^2 = (g_{N+3}^{n+3})^3$, $(g_{N+3}^{n+3})^3$
$(g_{N+3}^{n+3})^1$
n+3
n+2
n+1
n
$(g_{N+3}^{n+2})^1$
$(g_{N+3}^{n+1})^1$
$(g_{N+3}^{n})^1$
It 1
$(g_{N+3}^{n+2})^2$
$(g_{N+3}^{n+1})^2$
$(g_{N+3}^{n})^2$
It 2
$(g_{N+3}^{n+2})^3$
$(g_{N+3}^{n+1})^3$
$(g_{N+3}^{n})^3$
It 3