The multigrid technique

With standard iteration methods several issues:

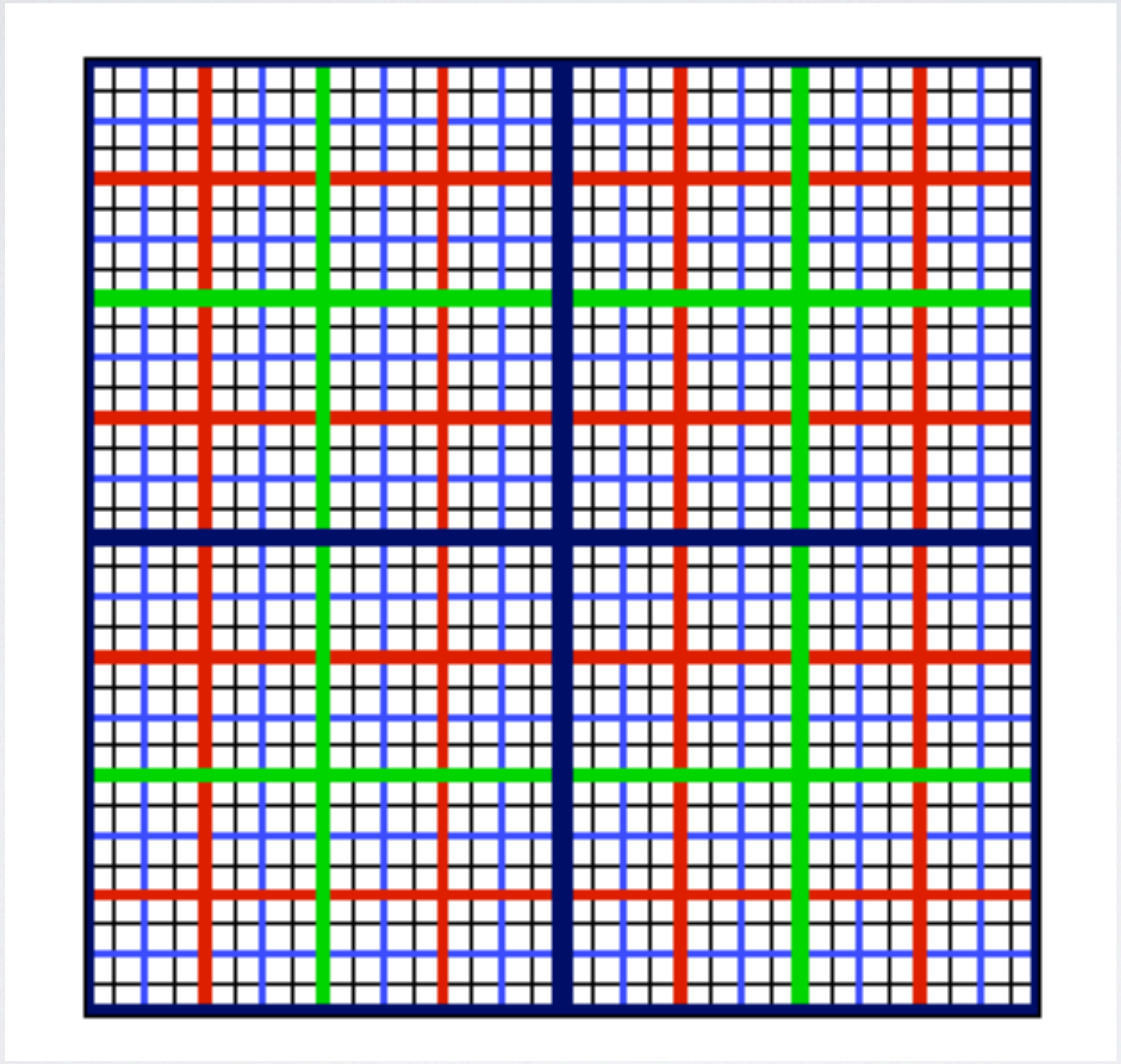
- convergence slow, at best *O(N)*, so computation for N points goes as *O(N^2)*

- convergence depends on "wavelength" (slower for long wavelengths on the mesh, namely for contribution to the potential from most distant mesh points)

- at every iteration only nearby cells communicate (eg in red black ordering scheme), but fast convergence requires information on updates communicated efficiently through the whole domain

Idea: go to a coarser mesh to achieve faster convergence (or solve the problem exactly) and produce guess value for iteration on finer mesh (eg if coarser mesh has 2x less resolution convergence speed ups by ~ 1/8 order of magnitude since there would be ~ 1/8 mesh points at which to compute potential)

For the remainder imagine to always have defined and laid down 2 or more grids with different grid spacing (= resolution) covering the same domain



Interpolation between coarse and fine mesh

Coarse-to-fine

Assume two meshes that span the same computational domain, one with resolution twice as the other one (= mesh spacing *2h* rather than *h*).

Def. *prolongation operator* $\mathbf{I}_{2h}^{h}$ as a linear interpolation operator that maps a vector $x^{(h)}$ defined on the fine mesh to a vector $x^{(2h)}$ on the coarse mesh:

$$\mathbf{I}_{2h}^{h}\mathbf{x}^{(2h)} = \mathbf{x}^{(h)}.$$

A simple realization of such operator in 1D is the following:

$$\mathbf{I}_{2h}^{h}: \quad \begin{array}{l} x_{2i}^{(h)} = x_i^{(2h)} \\ \\ x_{2i+1}^{(h)} = \frac{1}{2}(x_i^{(2h)} + x_{i+1}^{(2h)}) \end{array} \qquad \text{for } 0 \le i < \tfrac{N}{2}.$$

With the definition given above every second point on the fine mesh is directly injected from the coarse mesh while the intermediate points are linearly *interpolated* from the neighboring points, which here corresponds to a simple arithmetic average

Fine-to-coarse

The converse mapping represents a smoothing operation, called *restriction.* The corresponding operator can be naturally defined as:

$$\mathbf{I}_h^{2h}\mathbf{x}^{(h)} = \mathbf{x}^{(2h)}$$

Again in 1D a simple representation of such operator is:

$$\mathbf{I}_h^{2h}: \; x_i^{(2h)} = \frac{x_{2i-1}^{(h)} + 2x_{2i}^{(h)} + x_{2i+1}^{(h)}}{4} \quad \text{for } 0 \le i < \tfrac{N}{2}$$

3-point stencil

The two operators, prolongation and restriction, are chosen such that the transpose of one is proportional to the other:

$$\mathbf{I}_h^{2h} = c\,[\mathbf{I}_{2h}^h]^{\mathrm{T}}$$

A compact notation, that highlights the weighs of different points in both prolongation and restriction, is the following:

1D-prolongation, $\mathbf{I}_{2h}^h :\ \left]\frac{1}{2}\ 1\ \frac{1}{2}\right[$, $\mathbf{I}_h^{2h} :\ \left[\frac{1}{4}\ \frac{1}{2}\ \frac{1}{4}\right]$ 1D restriction

The notation above means **(a)** that each coarse grid point is mappd to three points of the fine grid with the indicated weighs (the fine grid points with weigh 1/2 get contribution from two coarse points, see above) and **(b)** that every coarse grid point is the weighted sum of three fine points

In 2D the weighs are:

$$\text{2D-prolongation, } \mathbf{I}_{2h}^{h} : \left] \begin{matrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{matrix} \right[$$

$$\text{2D-restriction, } \mathbf{I}_{h}^{2h} : \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

The multigrid V-cycle

One we have defined prolongation and restriction we must decide how we proceed in practice to compute a relevant quantity, eg the potential , by interpolating between the two grids. Recall the goal is to have a faster algorithm than Jacobi/Gauss-Seidel, namely the iteration should converge faster than when perfirmed on the fine mesh.

The general idea is to use the fine grid to “correct” the solution obtained faster on the coarse grid.

In order to perform such "correction" we first need to define an *error vector* and a *residual vector.* Recall we want to solve a linear equation of the type **Ax=b**

$$\mathbf{e} \equiv \mathbf{x}_{\text{exact}} - \tilde{\mathbf{x}},$$

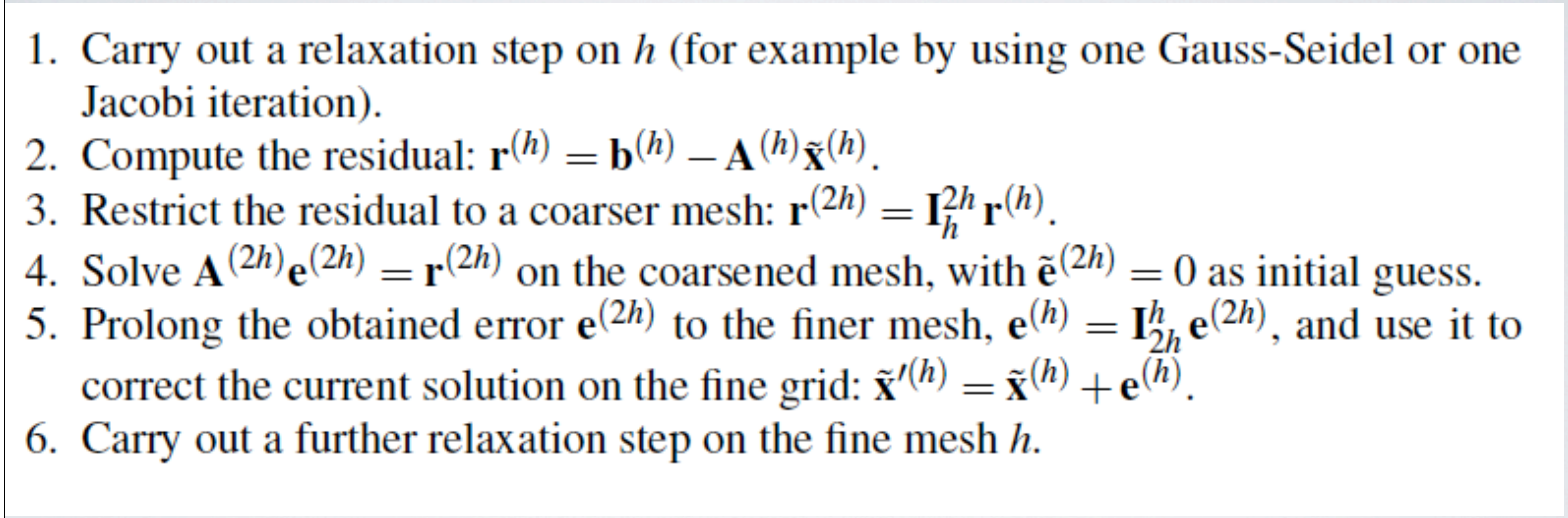
$$\mathbf{r} \equiv \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}.$$

Note that the pair of error and residual solutions of the original linear system given the way we defined them, i.e. we have

$$\mathbf{A}\mathbf{e} = \mathbf{r}.$$

One can exploit this by finding the error on a coarser grid using the above linear equation and use that to correct the solution on the finer grid.

The *coarse grid correction* is supposed to return an improved solution $\tilde{\mathbf{x}}'^{(h)}$ at the fine grid level "h" based on correcting an initial guess $\tilde{\mathbf{x}}^{(h)}$ by solving the problem on the coarser mesh and interpolating back. There are 5 steps in the procedure/algorithm:



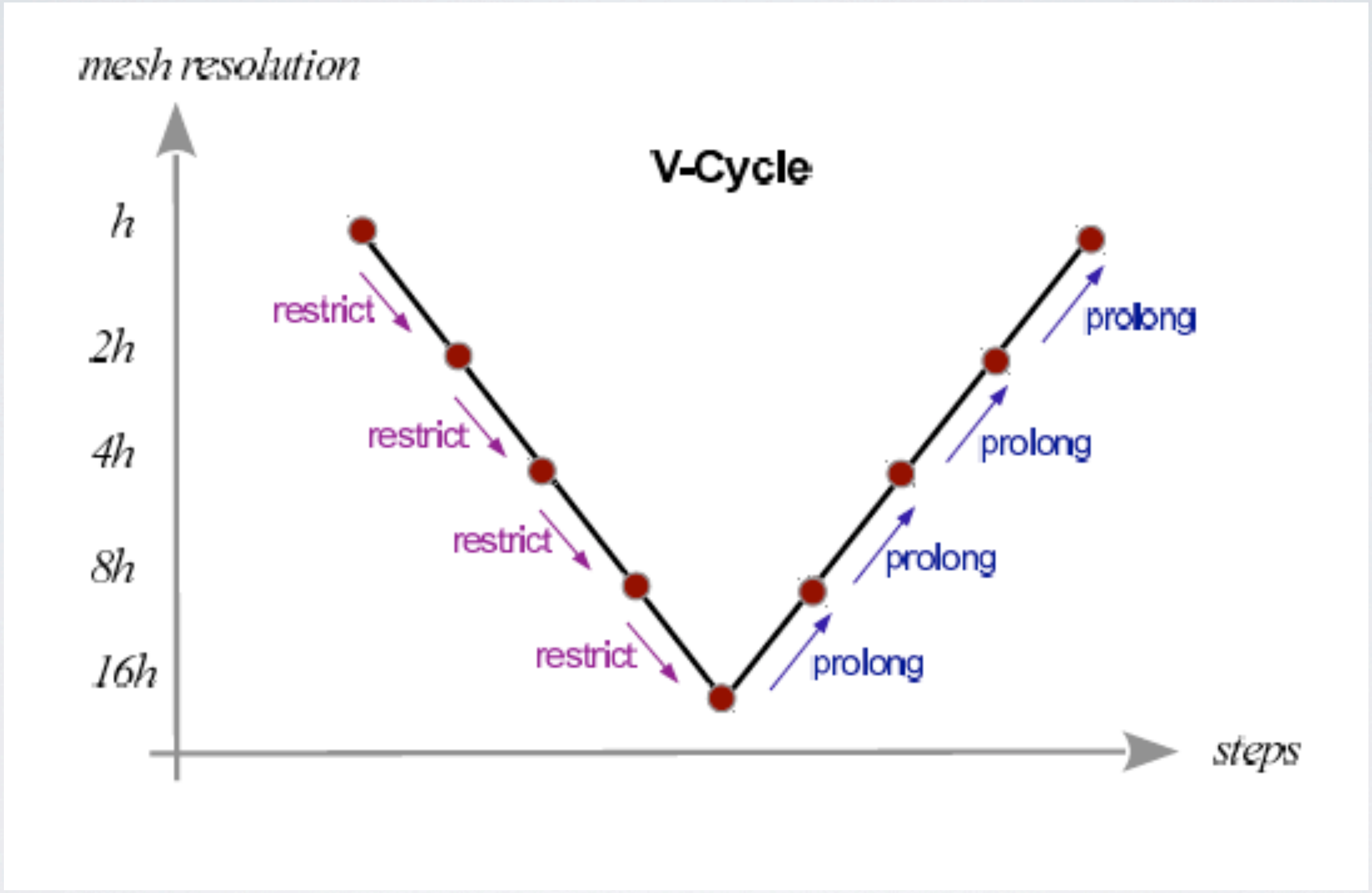
1. Carry out a relaxation step on h (for example by using one Gauss-Seidel or one Jacobi iteration).
2. Compute the residual: $\mathbf{r}^{(h)} = \mathbf{b}^{(h)} - \mathbf{A}^{(h)}\tilde{\mathbf{x}}^{(h)}$.
3. Restrict the residual to a coarser mesh: $\mathbf{r}^{(2h)} = \mathbf{I}_h^{2h}\mathbf{r}^{(h)}$.
4. Solve $\mathbf{A}^{(2h)}\mathbf{e}^{(2h)} = \mathbf{r}^{(2h)}$ on the coarsened mesh, with $\tilde{\mathbf{e}}^{(2h)} = 0$ as initial guess.
5. Prolong the obtained error $\mathbf{e}^{(2h)}$ to the finer mesh, $\mathbf{e}^{(h)} = \mathbf{I}_{2h}^{h}\mathbf{e}^{(2h)}$, and use it to correct the current solution on the fine grid: $\tilde{\mathbf{x}}'^{(h)} = \tilde{\mathbf{x}}^{(h)} + \mathbf{e}^{(h)}$.
6. Carry out a further relaxation step on the fine mesh h.

The crucial step is #4, which requires to solve the problem on the coarser grid. This a the step that should be made as efficient as possible so to reduce the cost of iteration.

The solution can be found *recursively* by keep going to increasingly coarser grids, until we reach a minimum number of cells (a threshold that should be decided based on how accurate we want to be, often driven by empirical considerations). At this "maximally coarser grid" one solves the problem exactly or using an iterative method (Gauss-Seidel/Jacobi) and skips steps 2 to 5.

This recursive method is the so-called *V-cycle*. One can show that the algorithm reduces the cost to $O(N\log N)$ to reach machine truncation/ round-off error (each V-cycle has a cost of $O(N)$, more than one is needed to converge).

V-cycle graph



One technical point that remains is how to find, in general, the operator $\mathbf{A}^{(2h)}$ on the coarse mesh given $\mathbf{A}^{(h)}$ on the fine mesh. There are two generic approaches:

(a) *Direct coarse grid approximation*. In this case one discretizes the equation (eg Poisson) directly on the coarse grid, just scaled by the resolution (which we measure as multiples of grid spacing *h* the finer grid). The stencil of the matrix does not change

(b) *Galerkin coarse grid approximation*. In this case one does not apply a new discretization, rather defines the coarse grid operator as $\mathbf{A}^{(2h)} = \mathbf{I}_h^{2h} \mathbf{A}^{(h)} \mathbf{I}_{2h}^{h}$
Formally better but computationally more expensive (de facto larger stencil to compute)

Final note: in V-cycle one performs one iteration per level (step 4). One could perform 2 or more (more than 2 is typically too expensive, called "W-cycle").

The full multi-grid method

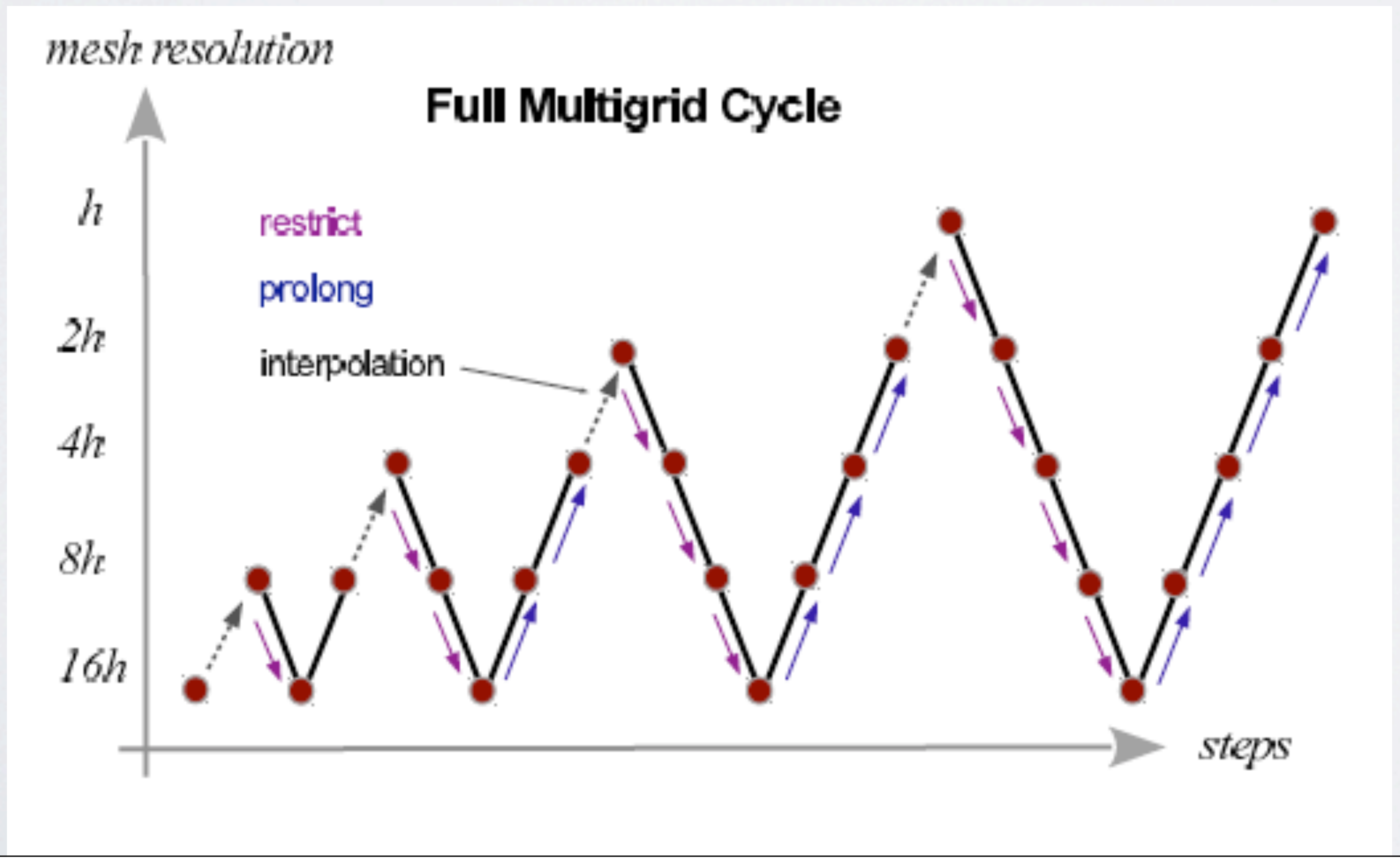
In the V-cycle the initial guess cannot be informed by any previous procedure. Depending on how good is the guess, namely how large or small is the residual to the exact solution the V-cycle can be cheap or computationally expensive.

The *full multi-grid method* solves naturally this problem of how to initiate step 1. It gets the guess by solving the problem on a coarse grid first, then interpolates to the fine grid.
(For subsequent times can use previous timestep's solution as first guess)

Of course also this procedure can be applied recursively, namely finding recursively an even coarser grid on which to solve the problem (iteratively or exactly) and recursively do interpolations on finer grids.

With a set of grids with varying resolution (grid spacing) then one can define a sequence of cycles (*restrict, interpolate, prolong*) with increasing depth. The algorithm is the following:

1. Initialize the right hand side on all grid levels, $\mathbf{b}^{(h)}, \mathbf{b}^{(2h)}, \mathbf{b}^{(4h)}, \ldots, \mathbf{b}^{(H)}$, down to some coarsest level H.
2. Solve the problem (exactly) on the coarsest level H.
3. Given a solution on level i with spacing $2h$, map it to the next level $i+1$ with spacing h and obtain the initial guess $\tilde{\mathbf{x}}^{(h)} = I_{2h}^{h}\mathbf{x}^{(2h)}$.
4. Use this starting guess to solve the problem on the level $i+1$ with one V-cycle.
5. Repeat Step 3 until the finest level is reached.



Final step: mapping force from mesh back to particles.

$$\mathbf{F}(\mathbf{x}) = m \sum_{\mathbf{p}} \mathbf{a}_{\mathbf{p}} W(\mathbf{x} - \mathbf{x}_{\mathbf{p}}),$$

Key notion; by (1) using the same kernel used for the initial mapping of the particle distribution to the density function on the mesh and (2) ensuring that the adopted kernel is symmetric one makes sure that:

(i) two interacting particles feel equal but opposite forces

(ii) there is no self-force

This also automatically ensures momentum conservation

To see this: write the acceleration at mesh point **p** as a convolution between the density and an antisymmetric vector operator (= Green's function) based on the superposition principle and third Newton's law

$$\mathbf{a}_{\mathbf{p}} = \sum_{\mathbf{p}'} \underbrace{\mathbf{d}(\mathbf{p}, \mathbf{p}')}_{\text{Green's function}} \underbrace{h^3 \rho_{\mathbf{p}'}}_{\text{mass in cell } \mathbf{p}}$$

So for example for self-force:

$$
\begin{aligned}
\mathbf{F}_{\text{self}}(\mathbf{x}_i) &= m_i \mathbf{a}_i(\mathbf{x}_i) = m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \mathbf{a}_{\mathbf{p}} \\
&= m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p},\mathbf{p}') h^3 \rho_{\mathbf{p}'} \\
&= m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p},\mathbf{p}') m_i W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}'}) \\
&= m_i^2 \sum_{\mathbf{p},\mathbf{p}'} \mathbf{d}(\mathbf{p},\mathbf{p}') W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}'}) \\
&= 0.
\end{aligned}
$$

Double sum = 0 because of antisymmetry of **d(p,p')** plus symmetry of (same) *W*

Exploiting the same mathematical properties of the force interpolation expression one can show $F_{12}(\mathbf{x_1}) = - F_{21}(\mathbf{x_2})$ where particle 1 is at $\mathbf{x_1}$ and particle 2 at $\mathbf{x_2}$ (see Springel's lecture notes pag. 27-28)

Treecode

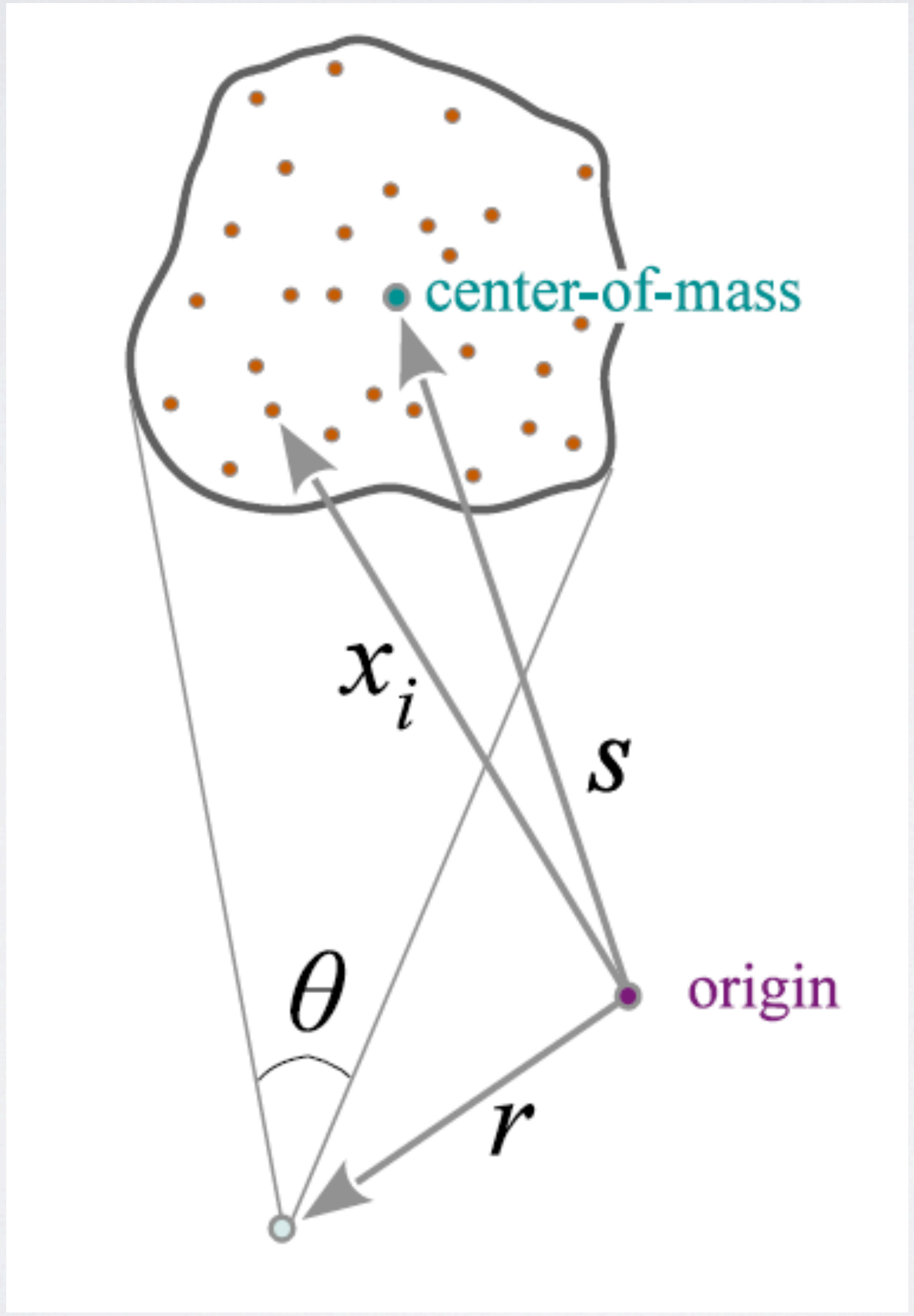
A multipole expansion for potential Φ *plus hierarchical tree-like structure to group particles. No underlying mesh hence geometrically very flexible and naturally adaptive to particle distribution*

Part I - the multipole expansion method

Main notion: use multipole expansion of distant group of particles to describe approximately the gravity exerted by such group on a given target particle, instead on summing up directly the forces from individual particles.

A criterion will be designed such that if the group of particles is *too close* to the target particle then forces will be computed by direct summation.

Example of treecode multipole expansion procedure: compute forces on particle located at distance ***r*** from origin from group of particles whose center of mass distance is ***s*** from the origin



The gravitational potential of the group is given by:

$$\Phi(\mathbf{r}) = -G\sum_i \frac{m_i}{|\mathbf{r}-\mathbf{x}_i|}$$

which can be rewritten as
$$\Phi(\mathbf{r}) = -G\sum_i \frac{m_i}{|\mathbf{r}-\mathbf{s}+\mathbf{s}-\mathbf{x}_i|}$$

Then, by expanding the denominator when the opening angle θ is small enough, one can assume $|\mathbf{x}_i-\mathbf{s}| \ll |\mathbf{r}-\mathbf{s}|$

which leads to
$$\frac{1}{|\mathbf{y}+\mathbf{s}-\mathbf{x}_i|} = \frac{1}{|\mathbf{y}|} - \frac{\mathbf{y}\cdot(\mathbf{s}-\mathbf{x}_i)}{|\mathbf{y}|^3} + \frac{1}{2}\frac{\mathbf{y}^T\left[3(\mathbf{s}-\mathbf{x}_i)(\mathbf{s}-\mathbf{x}_i)^T-(\mathbf{s}-\mathbf{x}_i)^2\right]\mathbf{y}}{|\mathbf{y}|^5} + .$$

above **y= r - s**

The expansion just written is truncated at the quadrupole term but could be extended further.
Since we have carried it out in the center of mass frame the dipole cancels out, so one is left only with monopole and quadrupole terms:

S is the position of the node
k is the position of the child node

monopole: $M = \sum_i m_i,$

$$Q_{ij} = \sum_k m_k \left[3(\mathbf{s}-\mathbf{x}_k)_i(\mathbf{s}-\mathbf{x}_k)_j - \delta_{ij}(\mathbf{s}-\mathbf{x}_k)^2\right]$$

$$\Phi(\mathbf{r}) = -G\left(\frac{M}{|\mathbf{y}|} + \frac{1}{2}\frac{\mathbf{y}^T\mathbf{Q}\mathbf{y}}{|\mathbf{y}|^5}\right), \quad \mathbf{y} = \mathbf{r} - \mathbf{s},$$

To get the force from this, we take its derivative with respecto to y

From the above multipole expansion of the potential the force can be readily obtained through differentiation.
We consider the expansion *accurate enough* if:

$$\theta \simeq \frac{\langle|\mathbf{x}_i - \mathbf{s}|\rangle}{|\mathbf{y}|} \simeq \frac{l}{y} \ll 1$$