The basic idea of the algorithm is to divide the contribution of a simulation cell (which is assumed to be spherical) over a number of grid-cells in the uniform grid.

For each sim-cell we find a patch in the uniform grid which fully encloses the sim-cell. The contribution of the sim-cell is divided over the grid-cells in the patch. For grid-cells which are only partially covered by the sim-cells (at the edge of sim-cell), we give a lower contribution from the sim-cell, based on the number grid-cell vertices which are within the sim-cell region.

This is how it is implemented:

This part gives, for each sim-cell, the indices of the grid cell in which it is centered.

indX=ceil((coords(1,:)./boxSide+0.5).\*Ngrid);

indY=ceil((coords(2,:)./boxSide+0.5).\*Ngrid);

indZ=ceil((coords(3,:)./boxSide+0.5).\*Ngrid);

we ignore sim-cells whose center is found beyond the grid:

skipFlag= indX > Ngrid | indX < 1 | indY>Ngrid | indY<1 | indZ>Ngrid | indZ<1;

Next, for each sim-cell we want to find the region of the grid which covers the sim-cell completely.

We translate the cell diameter to grid-units :

gcl=ceil(cellSize./(boxSide/Ngrid));

gcl=gcl+(1-mod(gcl,2));

gind=(gcl+1)/2

And using this we can now find the minimal and maximal grid-cell index, in each direction, which corresponds to a patch of the grid which fully encloses each sim-cell.

indXlo=indX-gind;indXhi=indX+gind;

indYlo=indY-gind;indYhi=indY+gind;

indZlo=indZ-gind;indZhi=indZ+gind;

We make sure the patch does not extend beyond the actual size of the grid:

indXlo(indXlo<1)=1;

indYlo(indYlo<1)=1;

indZlo(indZlo<1)=1;

indXhi(indXhi>Ngrid)=Ngrid;

indYhi(indYhi>Ngrid)=Ngrid;

indZhi(indZhi>Ngrid)=Ngrid;

Now we generate physical positions for the vertices of the grid:

gl=boxSide/Ngrid;

xg=-0.5\*boxSide:gl:0.5\*boxSide;

yg=xg;zg=xg;

[vertY,vertX,vertZ]=meshgrid(xg,yg,zg);

Next, we loop over the sim-cells, and for every cell we do the following:

These are the indices of the grid-cells in the patch - we use them to assign the values of the contributions to the relevant grid cells:

xi=indXlo(i):indXhi(i);

yi=indYlo(i):indYhi(i);

zi=indZlo(i):indZhi(i);

These are the indices of the vertices of the grid-cells in the patch:

xvi=indXlo(i):indXhi(i)+1;

yvi=indYlo(i):indYhi(i)+1;

zvi=indZlo(i):indZhi(i)+1;

The actual positions of the vertices for the grid-cells in the patch are:

vertX(xvi,yvi,zvi), vertY(xvi,yvi,zvi), vertZ(xvi,yvi,zvi)

*These should be in the same units as the physical positions of the centers of the sim-cells!*

We now find the distance of all these vertices from the center of the sim-cell (coords = sim-cell center):

vertRad=sqrt((vertX(xvi,yvi,zvi)-coords(1,i)).^2 +...

(vertY(xvi,yvi,zvi)-coords(2,i)).^2 + ...

(vertZ(xvi,yvi,zvi)-coords(3,i)).^2);

Only vertices which are within the sim-cell radius should be counted so each vertex is given a value of 1 or zero accordingly:

vertMask=vertRad<=cellSize(i)/2;

We create an array for the grid-cells in the patch that counts, for each grid-call, how many of its vertices are found within the sim-cell radius:

vw=zeros(length(xi),length(yi),length(zi));

for ii=0:1

for jj=0:1

for kk=0:1

vw=vw+vertMask(1+ii:end-1+ii,...

1+jj:end-1+jj,...

1+kk:end-1+kk);

end

end

end

vw=vw./8; % we divide by the maximal number of vertices per grid-cell.

If a sim-cell is totally enclosed by a single grid-cell it may be ignored by this method so we always set the 'weight' of the grid-cell which contains the center of the sim-cell to be 1 :

mid=floor(size(vw)./2)+1;

vw(mid(1),mid(2),mid(3))=1;

We define a total 'vertex-volume'

volTot=sum(sum(sum(vw)));

* For the extensive case, e.g. adding up mass, the contribution to the relevant grid-cells is:

cube(xi,yi,zi)=cube(xi,yi,zi)+vals(i).\*vw./volTot;

Once the cell-loop is done then that’s it!

* For the intensive case, e.g. Temperature, where we may have some sort of weighted-mean:

wtRho=wt(i).\*vw/volTot;

cube(xi,yi,zi)=cube(xi,yi,zi)+vals(i).\*wtRho;

wts(xi,yi,zi)=wts(xi,yi,zi)+wtRho;

And after the cell-looping is completed, normalize by the weights:

cube=cube./wts;

cube(wts==0)=0;