Notes on Domain Decomposition

Scott Aiton

November 1, 2017

1 Formulation of Schur Complement Matrix

If there is a γ for each point on the interface, how to we determine the values?

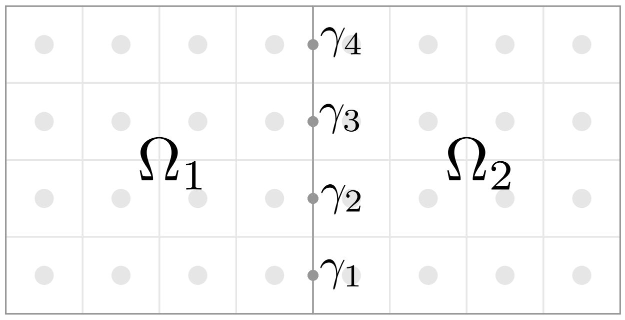


Figure 1: Two domain example

We want the gamma value to equal to the average of the solution on both sides:

$$\gamma = \frac{East(\Omega_1) + West(\Omega_2)}{2} \tag{1}$$

Where $East$ and $West$ are subroutines that return the solution values along the east side or west side of a domain. So now we have a function of γ

$$F(\gamma) = 2\gamma - (East(\Omega_1) + West(\Omega_2)) \tag{2}$$

that we want to find the zero of. In subroutine form, this would look like:

Algorithm 1 Two-Domain Function

```
1: procedure F($\gamma$)
2:     $\Omega_1.solveWithInterface(\gamma)$
3:     $result \leftarrow \gamma - \Omega_1.getSolutionEdge(East)$
4:     $\Omega_2.solveWithInterface(\gamma)$
5:     $result \leftarrow result + \gamma - \Omega_2.getSolutionEdge(West)$
6:     return $result$
7: end procedure
```

Do you involve the RHS at all?

When there are multiple interface points, then we have a system of linear equations

$$F(\gamma) = A\gamma - b \tag{3}$$

The b vector is found by

$$b = -F(0) \tag{4}$$

and each column of the matrix is found by

$$A(:, i) = F(e_i) + b \tag{5}$$

we can then determine the γ vector by solving

$$A\gamma = b \tag{6}$$

1.1 Generalized Function

Given some arbitrary mesh, we want to be able to index the interface values in the γ vector. One way that we can do this is have the interface values on each interface be consecutively indexed, and also give a unique index to each of the interface. If our domains each have $n \times n$ values, the interface with index 1 will have the first n values, a interface with index 2 will have the second n values, and so on.

Indexing of Interfaces Consider the mesh given in figure 2. In order to give a unique index to each interface, we can think of the mesh as a graph. Where each domain is a vertex, and an edge represents domains sharing an interface. We can then perform a breadth-first traversal of the graph and label each edge with a unique index as is shown in figure 3. An algorithm for indexing the interfaces is shown in algorithm 2.

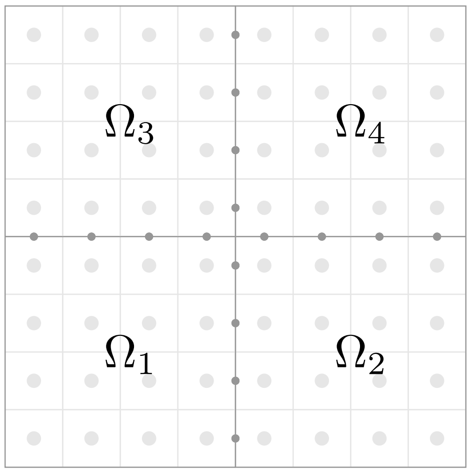


Figure 2: Four domain mesh

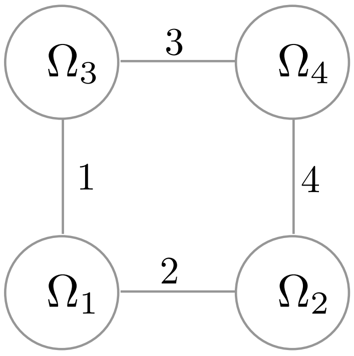


Figure 3: Indexing of interfaces

Algorithm 2 Interface Indexing

```
1: procedure IndexInterfacesBFS($\Omega_{start}$)
2:     $queue()$                                   ▷ queue of domains to visit next
3:     $queue.pushBack(\Omega_{start})$            ▷ insert starting domain
4:     $visited \leftarrow \emptyset$              ▷ set of visited domains
5:     $i \leftarrow 1$
6:     while $!queue.empty()$ do                   ▷ Breadth-First traversal of our mesh
7:         $\Omega \leftarrow queue.popFront()$
8:         for $side \in \{North, East, South, West\}$ do
9:             if $\Omega.hasNeighbor(side)$ and $\Omega.getNeighbor(side) \notin visited$ then
10:                $\Omega_{nbr} \leftarrow \Omega.getNeighbor(side)$
11:                if $\Omega_{nbr} \notin queue$ then
12:                    $queue.pushBack(\Omega_{nbr})$
13:                end if
14:                $\Omega.ifaceIndex(side) \leftarrow i$                        ▷ Set index for interface
15:                $\Omega_{nbr}.ifaceIndex(Opposite(side)) \leftarrow i$        ▷ Also set index for neighbor
16:                $i \leftarrow i + 1$
17:            end if
18:        end for
19:        $visited.insert(\Omega)$
20:    end while
21: end procedure
```

Consecutive Indexing of γ Values We can index the values on each interface in the following way:

- If the interface is on the east or west side of the patch, the γ values will be indexed consecutively from the bottom up.
- If the interface is on the north or south side of the patch, the γ values will be indexed consecutively from left to right.

Since the γ values on each interface indexed consecutively, the Schur complement matrix takes on a block structure, where each block is $n \times n$.

Generalized Function Once each interface vlaue has an index, we can use algorithm 3 for the function that we want to find the zero of.

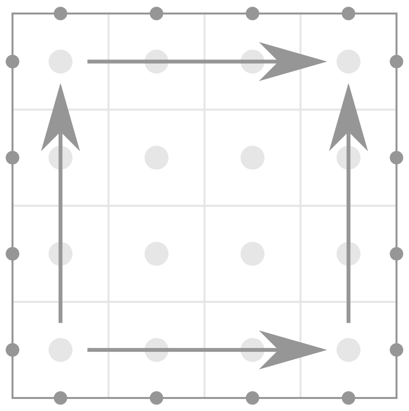


Figure 4: Graph of four domain mesh

Algorithm 3 Generalized Function

```
procedure F($\gamma, Domains$)
    $result(\gamma.size())$                                   ▷ Allocate new vector
    for $\Omega \in Domains$ do
        $\Omega.solveWithInterface(\gamma)$
        for $side \in \{North, East, South, West\}$ do
            if $\Omega.hasNeighbor(side)$ then
                $i \leftarrow \Omega.ifaceIndex(side)$
                $start \leftarrow i * (n-1) + 1$
                $stop \leftarrow i * n$
                $result(start : stop) \leftarrow \gamma(start : stop) - \Omega.getSolutionEdge(side)$
            end if
        end for
    end for
    return $result$
end procedure
```

2 Quick Formulation of the Schur Complement Matrix

In this section, we can use the process described in equation (5) to derive an algorithm that allow us to quickly form the Schur compliment matrix.

The matrix does not depent on the RHS on the domains First, let's reconsider equations (4) and (5). If we are solving on a system where the solution zero, the b vector for the Schur complement matrix will be 0, since 0 is the correct solution for the interfaces. So equation (5) turns into

$$A(:,i) = F_{zero}(e_i) \tag{7}$$

where F_{zero} is the same as equation 2, but instead we are solving a system where the solution is zero. So now we use can use this to reduce the amount of work needed to form the Schur complement matrix.

Let's consider what happens when we solve for $F_{zero}(e_i)$. If a single interface value is set to 1, only the two adjacent domains will have non-zero dirichlet boundary conditions, meaning that only the two adjacent domains will have non-zero solutions. This means that when we are solving for $F_{zero}(e_i)$, we can assume that solution on any domain that is not adjacent to the interface with the 1 is zero. In other words, we only have to do a solve on the two adjacent domains, rather than solving for all the domains.

Sparsity Consider the example grid given in Figure 5. When a single 1 is set on the interface i_{main}, only 7 interfaces will end up having non-zero values: the main interface, i_{main}, and the 6 auxiliary interfaces, $i_{\text{left north}}$, $i_{\text{left south}}$, $i_{\text{left west}}$, $i_{\text{right north}}$, $i_{\text{right east}}$, and $i_{\text{right south}}$.

For each 1 that is set along the main interface, the same 7 interface will return non-zero values. This means that for each interface, there will be maximum of 7 blocks of size $n \times n$.

$$A(:,j) = \begin{bmatrix} \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \vdots \end{bmatrix} \qquad A(:,j:(j+n)) = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

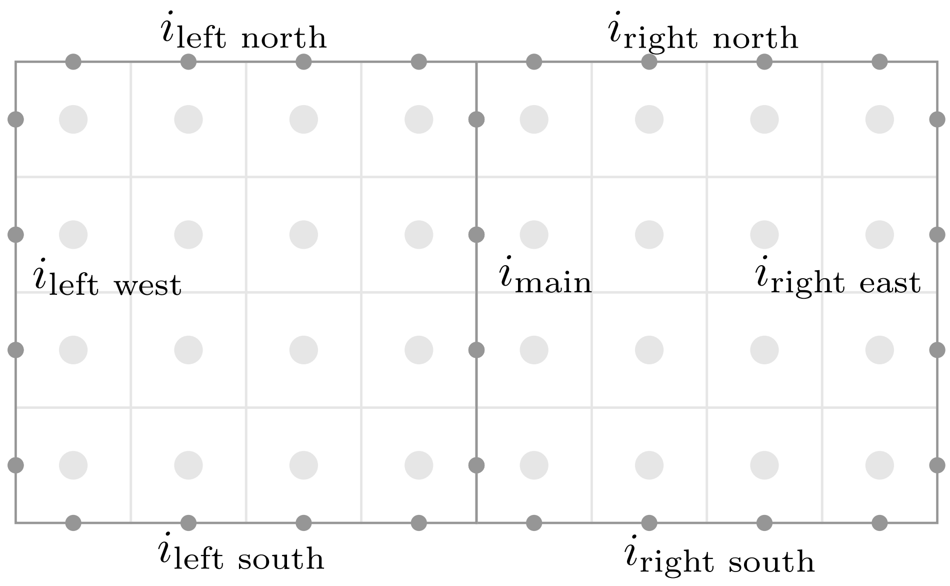


Figure 5: Two domain example

Rotation We can get the coefficients for the blocks in the Schur complement matrix by setting each γ value along one side of the domain to 1, and then reuse these coefficients for the interfaces on the other sides.

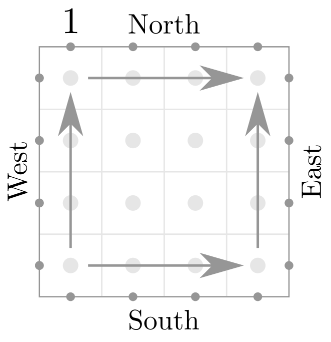
Let the north side be our canonical side. If we set each γ value along the north interface to 1, we can save the resulting coefficients into four blocks. This is done in algorithm 4. We can reuse these blocks for the other interfaces, but the order of the columns and the rows in each block may be incorrect.

First, let's consider the order of the columns. In part (a) of figure 6, the first γ value on the north interface is set to 1. If we rotate this domain clockwise by 90 degrees, that is equivalent to setting the last γ value on the east interface to 1. This means that the order of the columns of our canonical blocks will be in the wrong order if we want to reuse those blocks for the east interface. This is also true for the south interface. A table for whether or not to reverse the order of the columns of the canonical blocks is given in table 1

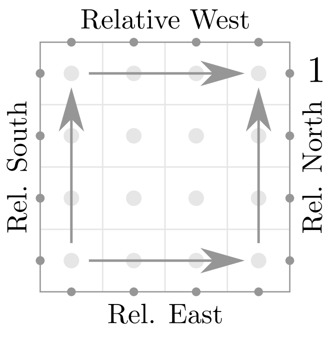
North	East	South	West
False	True	True	False

Table 1: Reversal of columns

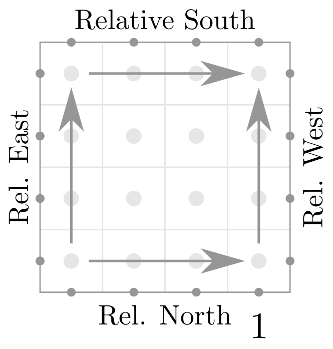
Next, let's consider the order of the rows. If we compare part (a) to part (b) in figure 6, we can see that the consecutive ordering of the interface values (the arrows in the figures) are pointing the opposite direction. This means that when we get each column of coefficients



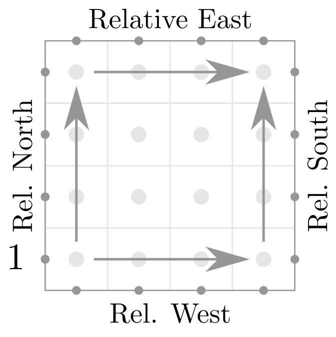
(a) North



(b) East



(c) South



(d) West

Figure 6: Rotation

for the north canonical block, the order of the rows of coefficients will be wrong when we use them in the other interfaces. A completely worked out logic table for whether or not we need to reverse the order of columns in each of the canonical blocks is given in table 2.

	North	East	South	West
North Block	False	True	True	False
East Block	False	False	True	True
South Block	False	True	True	False
West Block	False	False	True	True

Table 2: Reversal of rows

Algorithm 4

```
 1: procedure GETBLOCKS(n)
 2:     $\Omega()$                                        ▷ Create empty domain
 3:     $\Omega.rhs \leftarrow 0$
 4:     $\Omega.northBoundary \leftarrow 0$
 5:     $\Omega.eastBoundary \leftarrow 0$
 6:     $\Omega.southBoundary \leftarrow 0$
 7:     $\Omega.westBoundary \leftarrow 0$
 8:     $NorthBlock(n * n)$                           ▷ Allocate blocks of size n*n
 9:     $EastBlock(n * n)$
10:     $SouthBlock(n * n)$
11:     $WestBlock(n * n)$
12:     for $i \leftarrow 1, n$ do
13:         $\Omega.northBoundary(i) \leftarrow 1$
14:         $\Omega.$SOLVE()
15:         $NorthBlock(:, i) \leftarrow \Omega.northBoundary -$ NORTH($\Omega$)
16:         $EastBlock(:, i) \leftarrow -$EAST($\Omega$)
17:         $SouthBlock(:, i) \leftarrow -$SOUTH($\Omega$)
18:         $WestBlock(:, i) \leftarrow -$WEST($\Omega$)
19:         $\Omega.northBoundary(i) \leftarrow 0$
20:     end for
21:     return $NorthBlock, EastBlock, SouthBlock, WestBlock$
22: end procedure
```

Algorithm The algorithm for quick matrix formulation will have three steps:

1. Enumerate a set of interface structs

2. Get coefficients for cannonical blocks

3. For each interface struct, insert the blocks for that interface into the matrix

Algorithm 5 shows how to enumerate a set of inteface structs. Each struct has *side*, *northIndex*, *southIndex*, and *westIndex* fields. The *side* field is which side of the domain the inteface is on, and each index is relative to that side (see figure 6).

Algorithm 5

```
1: procedure EnumerateIfaceStructs(Domains)
2:     $ifaces \leftarrow \emptyset$                                        ▷ Set of interfaces to be processed
3:     for all $\Omega \in Domains$ do
4:         for $side \in \{North, East, South, West\}$ do
5:             if $\Omega.hasNeighbor(side)$ then
6:                 $iface()$                                            ▷ New iface object
7:                 $iface.side \leftarrow side$
8:                 $iface.northIndex \leftarrow \Omega.getIndex(side)$
9:                 $iface.eastIndex \leftarrow \Omega.getIndex(\text{Rotate}(side, 90))$
10:                $iface.southIndex \leftarrow \Omega.getIndex(\text{Rotate}(side, 180))$
11:                $iface.westIndex \leftarrow \Omega.getIndex(\text{Rotate}(side, 260))$
12:                $ifaces.insert(iface)$
13:            end if
14:        end for
15:    end for
16:    return $ifaces$
17: end procedure
```

Algorithm 6 Matrix Formulation

```
procedure FormMatrix(Domains)
    $ifaces \leftarrow$ EnumerateIfaceStructs($Domains$)
    $NorthBlock, EastBlock, SouthBlock, WestBlock \leftarrow$ getBlocks($n$)
    $A()$                                                   ▷ Allocate Matrix
    for all $iface \in ifaces$ do                           ▷ Insert Blocks into Matrix

        $reverseColumns \leftarrow$ RevColTable($iface.side$)
        $reverseRows \leftarrow$ RevRowTable($iface.side, North$)
        $j \leftarrow iface.northIndex$
        $A.insertBlock(NorthBlock, j, j, reverseColumns, reverseRows)$

        if $\Omega.eastIndex \neq null$ then
            $reverseRows \leftarrow$ RevRowTable($iface.side, East$)
            $i \leftarrow iface.eastIndex$
            $A.insertBlock(EastBlock, i, j, reverseColumns, reverseRows)$
        end if

        if $\Omega.southIndex \neq null$ then
            $reverseRows \leftarrow$ RevRowTable($iface.side, South$)
            $i \leftarrow iface.southIndex$
            $A.insertBlock(SouthBlock, i, j, reverseColumns, reverseRows)$
        end if

        if $\Omega.westIndex \neq null$ then
            $reverseRows \leftarrow$ RevRowTable($iface.side, West$)
            $i \leftarrow iface.westIndex$
            $A.insertBlock(WestBlock, i, j, reverseColumns, reverseRows)$
        end if

    end for
    return $A$
end procedure
```

3 Handling Refinement

We want the flux going out of the coarse cell to match the fluxes going into the fine cells.

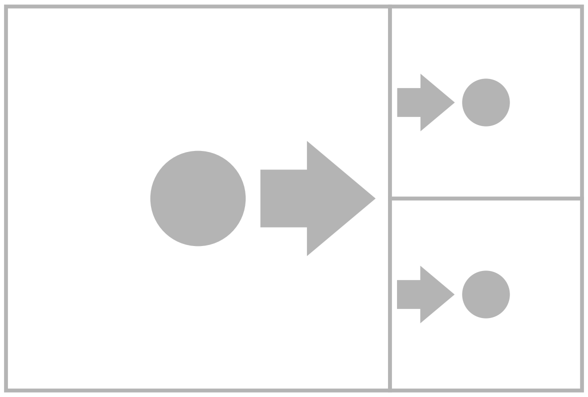


Figure 7: flux

We can represent this with the equation:

$$\Phi_c = \Phi_{f_1} + \Phi_{f_2} \tag{8}$$

Coming up with a stencil

Lets say we want to find the ghost values for the coarse cell, the first fine cell, and the second fine cell. Labeled g_c,g_{f_1},and g_{f_2}, respectively.

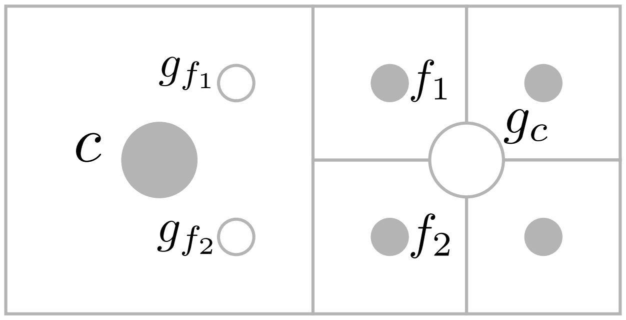


Figure 8: ghost points

We can enforce flux conservation by interpolating to the fine ghost points, and then using equation 8 to find the ghost point for the coarse cell.

The fluxes for each cell will be:

$$\Phi_c = g_c - c \tag{9}$$
$$\Phi_{f_1} = f_1 - g_{f_1} \tag{10}$$
$$\Phi_{f_2} = f_2 - g_{f_2} \tag{11}$$

We can then solve for the value of g_c:

$$\Phi_c = \Phi_{f_1} + \Phi_{f_2} \tag{12}$$
$$g_c - c = f_1 - g_{f_1} + f_2 - g_{f_2} \tag{13}$$
$$g_c = c + f_1 - g_{f_1} + f_2 - g_{f_2} \tag{14}$$

Bilinear interpolation

Bilinear interpolation for the fine ghost points works, but error is not continuous.

TODO: Explain this and show an example

Quadratic interpolation

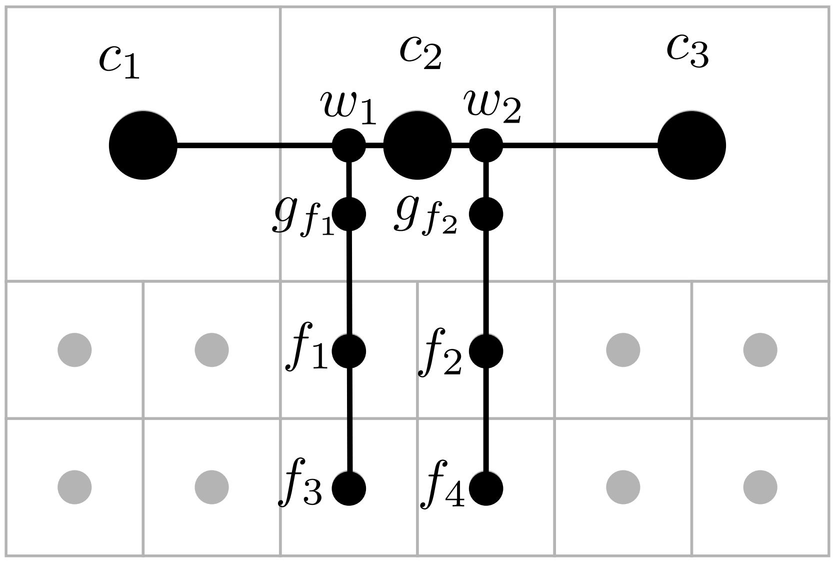


Figure 9: flux

To find the value of g_{f_1}, we first use quadratic interpolation with the points c_1, c_2, and c_3 to interpolate to w_1:

$$w_1 = \frac{5}{32}c_1 + \frac{15}{16}c_2 - \frac{3}{32}c_3$$

We then use quadratic interpolation with the points w_1, f_1, and f_3 to interpolate to g_{f_1}:

$$g_{f_1} = \frac{8}{15}w_1 + \frac{2}{3}f_1 - \frac{1}{5}f_3$$

Plug in the value for w_2, and we get the final equation for g_{f_1}:

$$g_{f_1} = \frac{1}{12}c_1 + \frac{1}{2}c_2 - \frac{1}{20}c_3 + \frac{2}{3}f_1 - \frac{1}{5}f_3$$

The equation for g_{f_2} is similar:

$$g_{f_2} = -\frac{1}{20}c_1 + \frac{1}{2}c_2 + \frac{1}{12}c_3 + \frac{2}{3}f_2 - \frac{1}{5}f_4$$

Now that we have g_{f_1} and g_{f_2}, we can use Eq. 14 to get the value of the ghost point for the coarse cell, g_{c_2}:

$$g_{c_2} = c_2 + f_1 - g_{f_1} + f_2 - g_{f_2}$$
$$g_{c_2} = c_2 + f_1 - \left(\frac{1}{12}c_1 + \frac{1}{2}c_2 - \frac{1}{20}c_3 + \frac{2}{3}f_1 - \frac{1}{5}f_3\right) + f_2 - \left(-\frac{1}{20}c_1 + \frac{1}{2}c_2 + \frac{1}{12}c_3 + \frac{2}{3}f_2 - \frac{1}{5}f_4\right)$$
$$g_{c_2} = -\frac{1}{30}c_1 - \frac{1}{30}c_3 + \frac{1}{3}f_1 + \frac{1}{3}f_2 + \frac{1}{5}f_3 + \frac{1}{5}f_4$$