Chapter 9
Gradient Computation

Abstract As was shown in the previous chapter, the discretization of the gradients of ϕ at cell centroids and faces is fundamental to constructing the discretized sets of diffusion equations and, as will be revealed in later chapters, of equations involving the convection term. In addition, the evaluation of gradients is needed for the evaluation of various operators. For example, pressure derivatives are directly needed in the discretized momentum equations, while velocity gradients are required to compute the production term in turbulence models, and the strain rate in non-Newtonian viscosity models. This chapter describes several techniques for evaluating gradients on a general mesh topology. The chapter starts with a description of the techniques for computing the gradient on cartesian structured grids and proceeds with gradient evaluation on unstructured grids. The presented methods follow either the Green-Gauss or the least square approach. Methods to interpolate the gradient to element faces are also presented.

9.1 Computing Gradients in Cartesian Grids

For the one-dimensional problem shown in Fig. 9.1 discretized using a uniform grid, a linear profile assumption for the variation of ϕ between cell centroids results in the following expression for the derivative at cell face e:

$$\left(\frac{d\phi}{dx}\right)_e = \frac{\phi_E - \phi_C}{x_E - x_C} = \frac{\phi_E - \phi_C}{\delta x_e} \tag{9.1}$$

In the same way, the derivative at the cell centroid C (Fig. 9.1) can be written using the values at the two adjacent cells as

F. Moukalled et al., *The Finite Volume Method in Computational Fluid Dynamics*, Fluid Mechanics and Its Applications 113, DOI 10.1007/978-3-319-16874-6_9

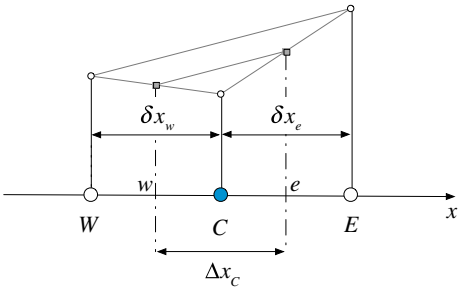

Fig. 9.1 Computing the gradient on a uniform one dimensional grid

$$\left(\frac{d\phi}{dx}\right)_C = \frac{\phi_e - \phi_w}{x_e - x_w} = \frac{\left(\frac{\phi_E+\phi_C}{2}\right) - \left(\frac{\phi_C+\phi_W}{2}\right)}{\Delta x_C} = \frac{\phi_E - \phi_W}{2\Delta x_C} \tag{9.2}$$

This expression is usually referred to as the "central difference" approximation of the first derivative.

For Cartesian grids in multiple dimensions, the derivatives can be computed by applying the same principle along the respective coordinate directions. Consider, for example, the two dimensional grid shown in Fig. 9.2, using the central difference approximation introduced earlier, the partial derivatives in the x and y directions are obtained as

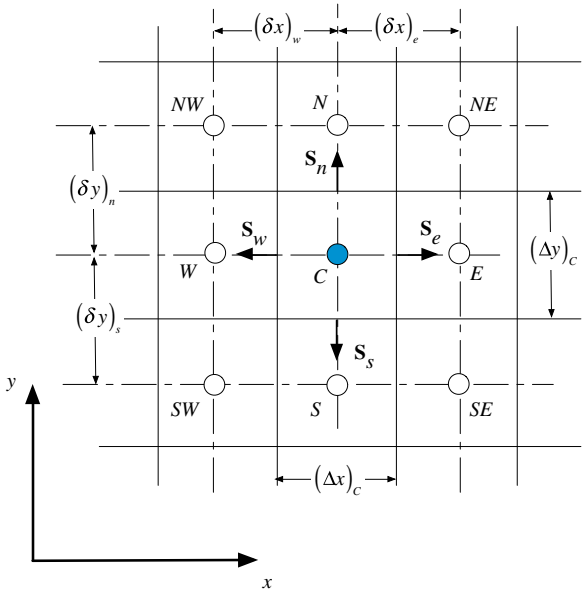

Fig. 9.2 Computing the gradient on a two dimensional Cartesian grid

$$\left(\frac{\partial \phi}{\partial x}\right)_C = \frac{\phi_E - \phi_W}{x_E - x_W} \quad \left(\frac{\partial \phi}{\partial y}\right)_C = \frac{\phi_N - \phi_S}{x_N - x_S} \tag{9.3}$$

A similar equation holds in the z direction for three dimensional grids.

When dealing with unstructured grids the computation of the gradient using the above method becomes unpractical and leads to the use of more general methods, some of these are now presented.

9.2 Green-Gauss Gradient

This is one of the most widely used methods for computing the gradient. It was introduced in the previous chapter and will not be repeated here. Rather the final form of the equation will be given and some additional methods to compute the face values will be introduced.

As derived in the previous chapter, the gradient at the centroid of an element C with volume V_C (Fig. 8.8) is computed as

$$\nabla \phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_f \mathbf{S}_f \tag{9.4}$$

where f refers to a face and $\mathbf{S}_f$ to its surface vector. The face values ϕ_f still need to be defined before the formula can be used. Two approaches are presented for computing ϕ_f. One is face-based with a generally compact stencil involving face neighbors, and the second is vertex-based with a larger stencil involving vertex neighbors. The number of cells in this extended stencil is about twice the compact one.

The use of a compact stencil is attractive with implicit methods because it leads to more compact Jacobian matrices. However, the enlarged stencil brings more information into the reconstruction, and is therefore expected to be more accurate.

Method 1: Compact Stencil [1]

For the two and three dimensional grid system shown in Fig. 9.3a, b, respectively, a simple approximation for calculating ϕ_f is to use the average values of the two cells sharing the face. In this case the value of ϕ_f is calculated as

$$\phi_f = g_C \phi_C + (1 - g_C)\phi_F \tag{9.5}$$

where g_C is a geometric weighting factor equal to

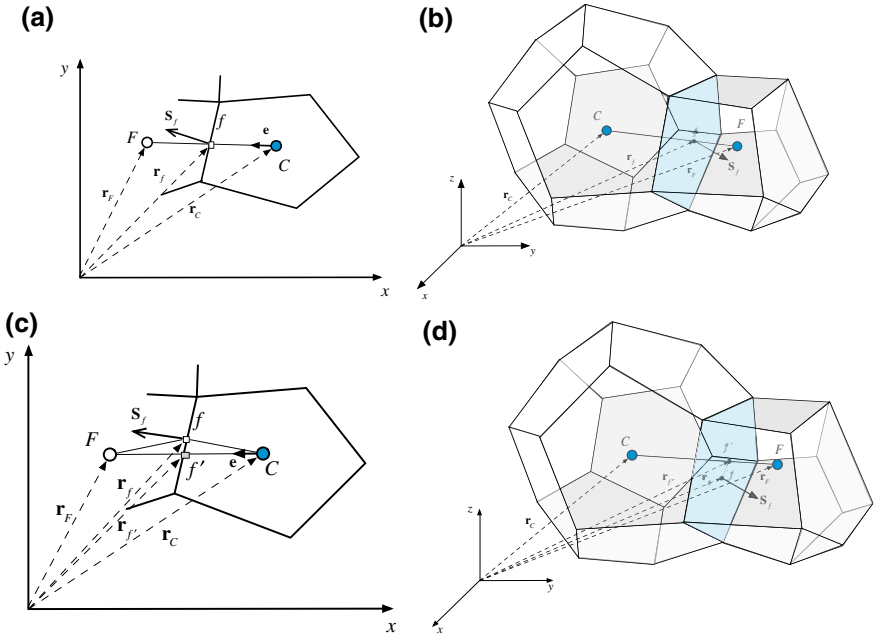

Fig. 9.3 Conjunctional face in **a** a two dimensional and **b** a three dimensional configuration; non-conjunctional face in **c** a two dimensional and **d** a three dimensional configuration

$$g_C = \frac{\| \mathbf{r}_F - \mathbf{r}_f \|}{\| \mathbf{r}_F - \mathbf{r}_C \|} = \frac{d_{Ff}}{d_{FC}} \tag{9.6}$$

where $\mathbf{r}$ designates the position vector and d the distance between two points. When the face is situated halfway between the two cell centers, ϕ_f is found to be

$$\phi_f = \frac{\phi_C + \phi_F}{2} \tag{9.7}$$

This approach is simple to implement in two and three dimensional situations and all operations involved are face-based, not requiring any additional grid connectivity. Accuracy-wise, the above relation leads to a second order approximation of ϕ_f only when the segment $[CF]$ and face S_f intersection point coincides with the centroid of the face S_f, i.e., with the Gaussian integration point f. Thus a second order accurate representation of the gradient is generally not achieved except for the above special case.

Such a condition is not usually satisfied with a general structured non-orthogonal or unstructured grid systems as shown for the two and three dimensional grid systems displayed in Fig. 9.3c, d, respectively. Rather, the skewness of the mesh (non-conjunctionality) results in the segment $[CF]$ and the surface S_f intersecting at

a point f', different from the face centroid f. In this case a correction is needed for the interpolated $\phi_{f'}$ value to get ϕ_f. The correction equation is given by

$$\begin{aligned}\phi_f &= \phi_{f'} + correction \\ &= \phi_{f'} + (\nabla\phi)_{f'} \cdot (\mathbf{r}_f - \mathbf{r}_{f'})\end{aligned} \tag{9.8}$$

which may also be written in expanded form as

$$\begin{aligned}\phi_f &= g_C\{\phi_C + (\nabla\phi)_C \cdot (\mathbf{r}_f - \mathbf{r}_C)\} + (1 - g_C)\{\phi_F + (\nabla\phi)_F \cdot (\mathbf{r}_f - \mathbf{r}_F)\} \\ &= \phi_{f'} + \underbrace{g_C(\nabla\phi)_C \cdot (\mathbf{r}_f - \mathbf{r}_C) + (1 - g_C)(\nabla\phi)_F \cdot (\mathbf{r}_f - \mathbf{r}_F)}_{correction}\end{aligned} \tag{9.9}$$

Since g_C depends on f', Eq. (9.9) indicates that improved estimates of the gradient can be obtained iteratively. At each iteration, the average value at the face is computed using the gradients calculated in the pervious iteration. These face values are then used to compute new estimates of the gradients. Doing excessive number of iterations may cause oscillations and usually not more than two iterations are performed.

In this case, the calculation of g_C requires locating the intersection point f' between $[CF]$ and the face S_f. For that purpose three options will be described.

Option 1: In this option f' is taken to be the exact intersection between $[CF]$ and the face S_f of surface vector $\mathbf{S}_f$. With $\mathbf{n}$ denoting the surface unit vector (i.e., $\mathbf{n} = \mathbf{S}_f/\|\mathbf{S}_f\|$) and $\mathbf{e}$ the unit vector along CF (i.e., $\mathbf{e} = \mathbf{CF}/\|\mathbf{CF}\|$), the location of f' (Fig. 9.3c, d) can be found by exploiting the orthogonality condition that exists between $\mathbf{n}$ and the segment ff' (i.e., $\mathbf{n}$ is normal to the face S_f containing the segment ff') to write

$$(\mathbf{r}_f - \mathbf{r}_{f'}) \cdot \mathbf{n} = 0 \tag{9.10}$$

Further, since f' is a point on CF the vector $\boldsymbol{Cf'}$ can be expressed in terms of $\mathbf{e}$ as

$$\boldsymbol{Cf'} = (\mathbf{r}_{f'} - \mathbf{r}_C) = k\mathbf{e} \tag{9.11}$$

where k is a scalar quantity. Combining Eqs. (9.10) and (9.11) yields

$$\mathbf{r}_{f'} = \frac{\mathbf{r}_f \cdot \mathbf{n}}{\mathbf{e} \cdot \mathbf{n}}\mathbf{e} \tag{9.12}$$

with f' located, g_C is computed as

$$g_C = \frac{\|\mathbf{r}_F - \mathbf{r}_{f'}\|}{\|\mathbf{r}_F - \mathbf{r}_C\|} = \frac{d_{Ff'}}{d_{FC}} \tag{9.13}$$

Then, the calculation procedure involves the following steps:

During the first iteration, calculate the gradient field over the entire domain as follows:

1. Calculate $\phi_{f'}$ using $\phi_{f'} = g_C \phi_C + (1 - g_C)\phi_F$
2. Calculate $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_{f'} \mathbf{S}_f$

From the second iteration onward, correct the gradient field according to the following procedure:

3. Update ϕ_f using $\phi_f = \phi_{f'} + g_C(\nabla\phi)_C \cdot (\mathbf{r}_f - \mathbf{r}_C) + (1 - g_C)(\nabla\phi)_F \cdot (\mathbf{r}_f - \mathbf{r}_F)$
4. Update $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_f \mathbf{S}_f$
5. Go back to step 3 and repeat.

Option 2: For f' chosen to be at the centre of segment $[CF]$ [Fig. 9.4a in two dimensions and Fig. 9.4b in three dimensions] the equations become simpler and the calculation of the gradient field over the domain proceeds as follows:

During the first iteration, calculate the gradient field over the entire domain as follows:

1. Calculate $\phi_{f'}$ using $\phi_{f'} = \frac{\phi_C + \phi_F}{2}$
2. Calculate $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_{f'} \mathbf{S}_f$

From the second iteration onward, correct the gradient field according to the following procedure:

3. Update ϕ_f using $\phi_f = \phi_{f'} + 0.5 * \left[(\nabla\phi)_C + (\nabla\phi)_F\right] \cdot \left[\mathbf{r}_f - 0.5 * (\mathbf{r}_C + \mathbf{r}_F)\right]$
4. Update $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_f \mathbf{S}_f$
5. Go back to step 3 and repeat

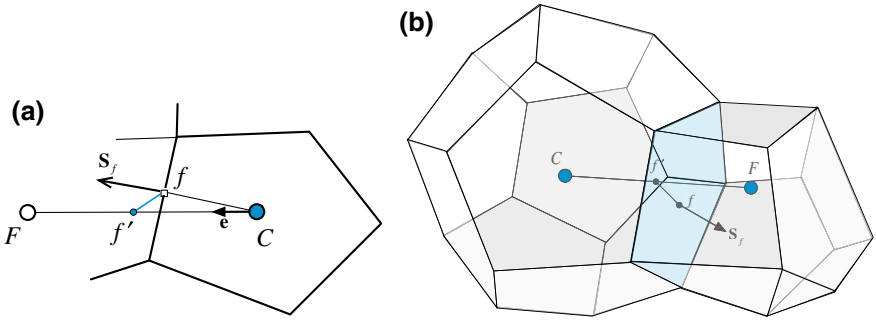

Fig. 9.4 Correction to Non-Conjunctionality using the midpoint approach: **a** two dimensional configuration; **b** three dimensional configuration

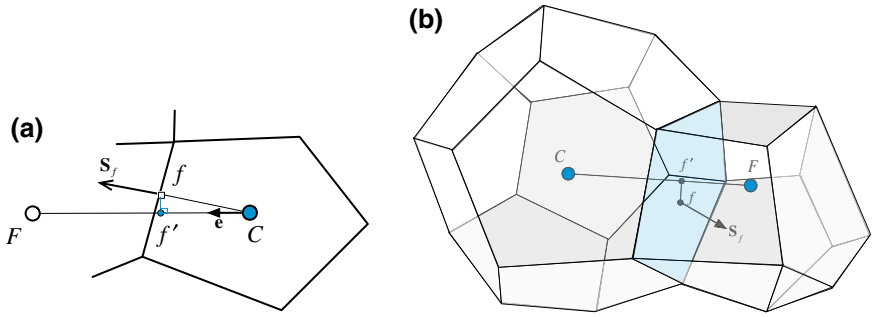

Fig. 9.5 Correction to Non-Conjunctionality using the minimum distance approach: **a** two dimensional configuration; **b** three dimensional configuration

Option 3: The position of f' can be chosen such that the distance ff' is the shortest possible [Fig. 9.5a in two dimensions and Fig. 9.5b in three dimensions], i.e., $[ff']$ perpendicular to $[CF]$. This leads to a more accurate computation of the gradient during the first iteration. In this case f' is computed by minimizing the distance between f and f'. In general $\mathbf{r}_{f'}$ is described by

$$\mathbf{r}_{f'} = \mathbf{r}_C + q(\mathbf{r}_C - \mathbf{r}_F) \tag{9.14}$$

where $0 < q < 1$.

Denoting the distance $f'f$ by d, then its square is obtained as

$$\begin{aligned} d^2 &= (\mathbf{r}_f - \mathbf{r}_{f'}) \cdot (\mathbf{r}_f - \mathbf{r}_{f'}) \\ &= [\mathbf{r}_f - \mathbf{r}_C - q(\mathbf{r}_C - \mathbf{r}_F)] \cdot [\mathbf{r}_f - \mathbf{r}_C - q(\mathbf{r}_C - \mathbf{r}_F)] \\ &= (\mathbf{r}_f - \mathbf{r}_C) \cdot (\mathbf{r}_f - \mathbf{r}_C) - 2q(\mathbf{r}_f - \mathbf{r}_C) \cdot (\mathbf{r}_C - \mathbf{r}_F) + q^2(\mathbf{r}_C - \mathbf{r}_F) \cdot (\mathbf{r}_C - \mathbf{r}_F) \end{aligned} \tag{9.15}$$

Minimizing the function d^2 with respect to q, yields

$$\frac{\partial(d^2)}{\partial q} = 0 \Rightarrow -2(\mathbf{r}_f - \mathbf{r}_C) \cdot (\mathbf{r}_C - \mathbf{r}_F) + 2q(\mathbf{r}_C - \mathbf{r}_F) \cdot (\mathbf{r}_C - \mathbf{r}_F) = 0 \tag{9.16}$$

Solving, q is obtained as

$$q = -\frac{\mathbf{r}_{Cf} \cdot \mathbf{r}_{CF}}{\mathbf{r}_{CF} \cdot \mathbf{r}_{CF}} \tag{9.17}$$

Knowing the q values, the gradient calculation over the domain proceeds as follows:

During the first iteration, calculate the gradient field over the entire domain as follows:

1. Calculate $\mathbf{r}_{f'}$ using $\quad \mathbf{r}_{f'} = \mathbf{r}_C - \dfrac{\mathbf{r}_{Cf} \cdot \mathbf{r}_{CF}}{\mathbf{r}_{CF} \cdot \mathbf{r}_{CF}} (\mathbf{r}_C - \mathbf{r}_F)$
2. Calculate g_C using $\quad g_C = \|\mathbf{r}_F - \mathbf{r}_{f'}\| / \|\mathbf{r}_F - \mathbf{r}_C\|$

3. Calculate $\phi_{f'}$ using $\phi_{f'} = g_C\phi_C + (1 - g_C)\phi_F$
4. Calculate $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C}\sum_{f\sim nb(C)} \phi_{f'}\mathbf{S}_f$

From the second iteration onward, correct the gradient field according to the following procedure:

5. Calculate $\nabla\phi_{f'}$ using $\nabla\phi_{f'} = g_C\nabla\phi_C + (1 - g_C)\nabla\phi_F$
6. Update ϕ_f using $\phi_f = \phi_{f'} + \nabla\phi_{f'} \cdot (\mathbf{r}_f - \mathbf{r}_{f'})$
7. Update $\nabla\phi_C$ using $\nabla\phi_C = \frac{1}{V_C}\sum_{f\sim nb(C)} \phi_f\mathbf{S}_f$
8. Go back to step 5 and repeat

Example 1

For the mesh shown in Fig. 9.6, the coordinates of the grid point C and its neighbors F_1 through F_6 are given by

$$C(13, 11) \quad F_1(4.5, 9.5) \quad F_2(8, 3) \quad F_3(17, 3.5)$$
$$F_4(22, 10) \quad F_5(16, 20) \quad F_6(7, 18)$$

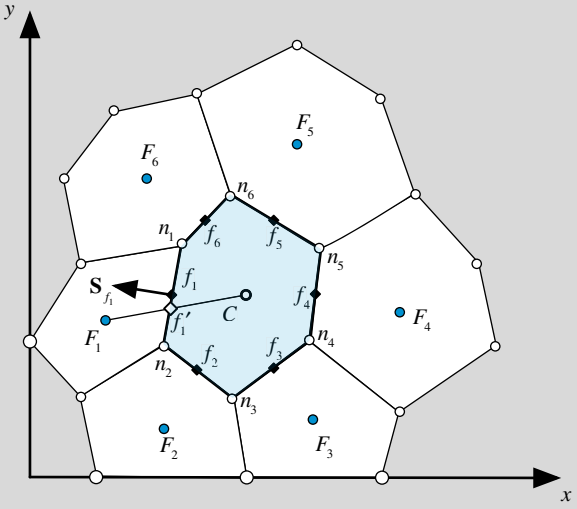

Fig. 9.6 Grid layout for Examples 1 and 2

while the nodes n_1 through n_6 are located at

$$n_1(9, 14) \quad n_2(8, 8) \quad n_3(12, 5)$$
$$n_4(17, 9) \quad n_5(17.5, 14) \quad n_6(12, 17)$$

If the values of the dependent variable ϕ at the centroids are known to be

$$\begin{aligned} &\phi_C = 167 \\ &\phi_{F_1} = 56.75 \quad \phi_{F_2} = 35 \quad \phi_{F_3} = 80 \\ &\phi_{F_4} = 252 \quad \phi_{F_5} = 356 \quad \phi_{F_6} = 151 \end{aligned}$$

and the values of the gradient of ϕ, $(\nabla\phi)$, at all neighboring elements to C are given by

$$\begin{aligned} &\nabla\phi_{F_1} = 10.5\mathbf{i} + 5.5\mathbf{j} \quad \nabla\phi_{F_2} = 4\mathbf{i} + 9\mathbf{j} \quad \nabla\phi_{F_3} = 4.5\mathbf{i} + 18\mathbf{j} \\ &\nabla\phi_{F_4} = 11\mathbf{i} + 23\mathbf{j} \quad \nabla\phi_{F_5} = 21\mathbf{i} + 17\mathbf{j} \quad \nabla\phi_{F_6} = 19\mathbf{i} + 8\mathbf{j} \end{aligned}$$

If the volume of cell C is V_C = 76, find the gradient $\nabla\phi_C$ using

a. *The Green-Gauss method with no correction*
b. *The Green-Gauss method alongside correction to skewness with f′ chosen to be at the centre of segment* [*CF*].

Solution The Green-Gauss method with no correction

a. The interpolation factors are needed to perform the calculations. This, in turn, necessitates computing the coordinates of the face centroids. The coordinates of the centroid f_1 are found as

$$\left.\begin{aligned} x_{f_1} &= 0.5 * (x_{n_1} + x_{n_2}) = 0.5 * (9 + 8) = 8.5 \\ y_{f_1} &= 0.5 * (y_{n_1} + y_{n_2}) = 0.5 * (14 + 8) = 11 \end{aligned}\right\} \Rightarrow f_1(8.5, 11)$$

In a similar way, the coordinates of other face centroids are found to be

$$\begin{aligned} &f_2(10, 6.5) \quad f_3(14.5, 7) \\ &f_4(17.25, 11.5) \quad f_5(14.75, 15.5) \quad f_6(10.5, 15.5) \end{aligned}$$

The surface vectors are calculated as

$$\begin{aligned} &\mathbf{S}_{f_1} = -6\mathbf{i} + \mathbf{j} \quad \mathbf{S}_{f_2} = -3\mathbf{i} - 4\mathbf{j} \quad \mathbf{S}_{f_3} = 4\mathbf{i} - 5\mathbf{j} \\ &\mathbf{S}_{f_4} = 5\mathbf{i} - 0.5\mathbf{j} \quad \mathbf{S}_{f_5} = 3\mathbf{i} + 5.5\mathbf{j} \quad \mathbf{S}_{f_6} = -3\mathbf{i} + 3\mathbf{j} \end{aligned}$$

The interpolation factor $(g_C)_1$ is computed using

$$\left.\begin{aligned}(g_C)_1 &= \frac{F_1 f_1}{F_1 f_1 + Cf_1}\\ F_1 f_1 &= \sqrt{(4.5-8.5)^2+(9.5-11)^2} = 4.272\\ Cf_1 &= \sqrt{(13-8.5)^2+(11-11)^2} = 4.5\end{aligned}\right\} \Rightarrow (g_C)_1 = 0.487$$

In a similar way, the other interpolation factors are found to be

$$(g_C)_2 = 0.427\ (g_C)_3 = 0.502\ (g_C)_4 = 0.538\ (g_C)_5 = 0.492\ (g_C)_6 = 0.455$$

Using Eq. (9.5) the ϕ_f values are computed as

$$\phi_{f_1} = 110.442 \quad \phi_{f_2} = 91.364 \quad \phi_{f_3} = 123.674$$
$$\phi_{f_4} = 206.27 \quad \phi_{f_5} = 263.012 \quad \phi_{f_6} = 158.28$$

Using Eq. (9.4), $\nabla\phi_C$ is calculated as

$$\nabla\phi_C = \frac{1}{76}\left\{\begin{bmatrix}110.442\times(-6)+91.364\times(-3)+123.674\times 4+\\ 206.27\times 5+263.012\times 3+158.28\times(-3)\end{bmatrix}\mathbf{i}+\right.$$
$$\left.\begin{bmatrix}110.442\times(1)+91.364\times(-4)+123.674\times(-5)+\\ 206.27\times(-0.5)+263.012\times 5.5+158.28\times(3)\end{bmatrix}\mathbf{j}\right\}$$
$$= 11.889\mathbf{i} + 12.433\mathbf{j}$$

b. The Green-Gauss method alongside correction to skewness with f' chosen to be at the centre of segment $[CF]$.

The values at the f' locations are computed as half the sum of the values at the nodes straddling the face and are given by

$$\phi_{f_1} = 111.875 \quad \phi_{f_2} = 101 \quad \phi_{f_3} = 123.5 \quad \phi_{f_4} = 209.5 \quad \phi_{f_5} = 261.5 \quad \phi_{f_6} = 158.5$$

Using these values, the first estimate for the gradient is obtained using Eq. (9.4) as

$$\nabla\phi_C = 11.53\mathbf{i} + 11.826\mathbf{j}$$

Defining $\mathbf{d_f} = \mathbf{r_f} - 0.5 \times (\mathbf{r_C} + \mathbf{r_F})$, the various values are found to be

$$\mathbf{d}_{f_1} = -0.25\mathbf{i} + 0.75\mathbf{j} \quad \mathbf{d}_{f_2} = -0.5\mathbf{i} - 0.5\mathbf{j} \quad \mathbf{d}_{f_3} = -0.5\mathbf{i} - 0.25\mathbf{j}$$
$$\mathbf{d}_{f_4} = -0.25\mathbf{i} + \mathbf{j} \quad \mathbf{d}_{f_5} = 0.25\mathbf{i} \quad \mathbf{d}_{f_6} = 0.5\mathbf{i} + \mathbf{j}$$

Using $\phi_f = \phi_{f'} + 0.5 * [(\nabla\phi)_C + (\nabla\phi)_F] \cdot [\mathbf{r}_f - 0.5 * (\mathbf{r}_C + \mathbf{r}_F)]$ the updated values at the faces are obtained as

$$\phi_{f_1} = 115.619 \quad \phi_{f_2} = 91.911 \quad \phi_{f_3} = 115.764$$
$$\phi_{f_4} = 224.097 \quad \phi_{f_5} = 265.566 \quad \phi_{f_6} = 176.046$$

These values are used in Eq. (9.4) yielding the updated value of the gradient as

$$\nabla\phi_C = 11.614\mathbf{i} + 13.761\mathbf{j}$$

Method 2: Extended Stencil [2]

The value of ϕ_f at the surface centroid f can be computed as the mean of the values at the vertices defining the surface. This necessitates the estimation of the properties at the vertices. The properties at a vertex node are calculated using the weighted average of the properties within the cells surrounding that node. Figure 9.7 shows the cells that are considered for the weighted average of the properties at the vertex nodes n_1 and n_2. The weight is taken as the inverse of the distance of the vertex from the cell centre [3]. The resulting equation for the properties at the vertices are written as,

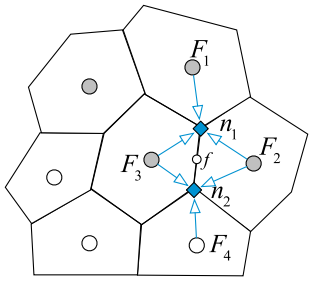

Fig. 9.7 Cells Contributing to node n for the Weighted Average

$$\phi_n = \frac{\sum_{k=1}^{NB(n)} \frac{\phi_{F_k}}{\|\mathbf{r}_n - \mathbf{r}_{F_k}\|}}{\sum_{k=1}^{NB(n)} \frac{1}{\|\mathbf{r}_n - \mathbf{r}_{F_k}\|}} \tag{9.18}$$

where n refers to the vertex node, F_k to the neighboring cell node, $NB(n)$ the total number of cell nodes surrounding the vertex node n, and $\|\mathbf{r}_n - \mathbf{r}_{F_k}\|$ the distance from the vertex node to the centroid of the neighboring cells.

Once the values ϕ_n at the vertices are found, the values ϕ_f at the surface centroids are calculated followed by the gradients at the control volume centroids. In two dimensional situations, ϕ_f is computed as

$$\phi_f = \frac{\phi_{n_1} + \phi_{n_2}}{2} \tag{9.19}$$

Then the gradient at C is found using

$$\nabla\phi_C = \frac{1}{V_C} \sum_{f=nb(C)} \phi_f \mathbf{S}_f = \frac{1}{V_C} \sum_{f \sim nb(C)} \left(\frac{\phi_{n_1} + \phi_{n_2}}{2} \right)_f \mathbf{S}_f \tag{9.20}$$

In three dimensional situations, the calculations are a little more involved as the number of a face vertices depends on the element type. The value of ϕ_f is found from the values at the vertices using

$$\phi_f = \frac{\sum_{k=1}^{nb(f)} \frac{\phi_{n_k}}{\|\mathbf{r}_{n_k} - \mathbf{r}_f\|}}{\sum_{k=1}^{nb(f)} \frac{1}{\|\mathbf{r}_{n_k} - \mathbf{r}_f\|}} \tag{9.21}$$

where n represents the number of vertices of face f. Once the values ϕ_f are calculated, the gradient at C is computed using Eq. (9.4).

One of the disadvantages of this approach is that information from the wrong side of the cell face also contributes to the weighted average values of the conserved variables. This can be overcome by using upwind biased gradients as discussed by Cabello [4]. The higher order calculations based on the upwind biased gradients, however, have both higher memory overheads required to store the information about the cells used for the upwind biased gradient calculation and increased coding complexity.

Example 2
Using the data of example I, calculate ϕ_{f_1} *using the extended stencil approach* via *Eq.* (9.16).

Solution First the distances have to be calculated and are given by

$$\|\mathbf{r}_{n_1} - \mathbf{r}_{F_6}\| = \sqrt{(9-7)^2+(14-18)^2} = 4.472$$
$$\|\mathbf{r}_{n_1} - \mathbf{r}_{F_1}\| = \sqrt{(9-4.5)^2+(14-9.5)^2} = 6.364$$
$$\|\mathbf{r}_{n_1} - \mathbf{r}_{C}\| = \sqrt{(9-13)^2+(14-11)^2} = 5$$
$$\|\mathbf{r}_{n_2} - \mathbf{r}_{F_1}\| = \sqrt{(8-4.5)^2+(8-9.5)^2} = 3.808$$
$$\|\mathbf{r}_{n_2} - \mathbf{r}_{F_2}\| = \sqrt{(8-8)^2+(8-3)^2} = 5$$
$$\|\mathbf{r}_{n_2} - \mathbf{r}_{C}\| = \sqrt{(8-13)^2+(8-11)^2} = 5.831$$

The values at nodes n_1 and n_2 are computed using Eq. (9.15) as

$$\phi_{n_1} = \frac{\frac{151}{4.472}+\frac{56.75}{6.364}+\frac{167}{5}}{\frac{1}{4.472}+\frac{1}{6.364}+\frac{1}{5}} = 131.009 \quad \phi_{n_2} = \frac{\frac{56.75}{3.808}+\frac{35}{5}+\frac{167}{5.831}}{\frac{1}{3.808}+\frac{1}{5}+\frac{1}{5.831}} = 79.708$$

The value of ϕ_{f_1} is found to be

$$\phi_{f_1} = 0.5(131.009 + 79.708) = 105.3585$$

9.3 Least-Square Gradient

Using least-square methods to compute the gradients [5] offers more flexibility with regard to the order of accuracy achieved [6] and the stencil used [7]. In the least-square method, the divergence-based gradient can be recovered as a special case. This flexibility comes at a cost, as proper weighting is needed for the stencil terms, and computation of the weights adds to the computational cost. The method is described next.

Considering a control volume and its immediate neighbors (Fig. 9.8), the change in centroid values between C and F is given by $(\phi_F - \phi_C)$, if the cell gradient $(\nabla\phi_C)$ is exact, then the difference can also be computed as

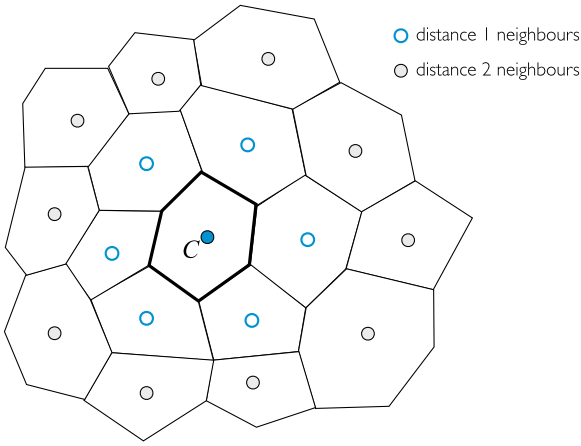

Fig. 9.8 A control volume with its immediate neighbors

$$\phi_F = \phi_C + (\nabla\phi)_C \cdot \underbrace{(\mathbf{r}_F - \mathbf{r}_C)}_{\mathbf{r}_{CF}} \tag{9.22}$$

However unless the solution field is linear the cell gradient cannot be exact because C has more neighbors than the gradient vector has components. In the least square methods, a gradient is computed by an optimization procedure that finds the minimum of the function G_C defined as

$$\begin{aligned} G_C &= \sum_{k=1}^{NB(C)} \left\{ w_k \left[\phi_{F_k} - (\phi_C + \nabla\phi_C \cdot \mathbf{r}_{CF_k}) \right]^2 \right\} \\ &= \sum_{k=1}^{NB(C)} \left\{ w_k \left[\Delta\phi_k - \left(\Delta x_k \left(\frac{\partial\phi}{\partial x} \right)_C + \Delta y_k \left(\frac{\partial\phi}{\partial y} \right)_C + \Delta z_k \left(\frac{\partial\phi}{\partial z} \right)_C \right) \right]^2 \right\} \end{aligned} \tag{9.23}$$

where w_k is some weighting factor. The various terms in the above equation represent

$$\begin{aligned} \Delta\phi_k &= \phi_{F_k} - \phi_C \\ \Delta x_k &= \mathbf{r}_{CF_k} \cdot \mathbf{i} \\ \Delta y_k &= \mathbf{r}_{CF_k} \cdot \mathbf{j} \\ \Delta z_k &= \mathbf{r}_{CF_k} \cdot \mathbf{k} \end{aligned} \tag{9.24}$$

The function G_C is minimized by enforcing the conditions

$$\frac{\partial G_C}{\partial\left(\frac{\partial\phi}{\partial x}\right)} = \frac{\partial G_C}{\partial\left(\frac{\partial\phi}{\partial y}\right)} = \frac{\partial G_C}{\partial\left(\frac{\partial\phi}{\partial z}\right)} = 0 \tag{9.25}$$

to yield the following set of three equations in three unknowns:

$$\begin{aligned}
&\sum_{k=1}^{NB(C)} \left\{ 2w_k \Delta x_k \left[-\Delta\phi_k + \Delta x_k \left(\frac{\partial\phi}{\partial x}\right)_C + \Delta y_k \left(\frac{\partial\phi}{\partial y}\right)_C + \Delta z_k \left(\frac{\partial\phi}{\partial z}\right)_C \right] \right\} = 0 \\
&\sum_{k=1}^{NB(C)} \left\{ 2w_k \Delta y_k \left[-\Delta\phi_k + \Delta x_k \left(\frac{\partial\phi}{\partial x}\right)_C + \Delta y_k \left(\frac{\partial\phi}{\partial y}\right)_C + \Delta z_k \left(\frac{\partial\phi}{\partial z}\right)_C \right] \right\} = 0 \\
&\sum_{k=1}^{NB(C)} \left\{ 2w_k \Delta z_k \left[-\Delta\phi_k + \Delta x_k \left(\frac{\partial\phi}{\partial x}\right)_C + \Delta y_k \left(\frac{\partial\phi}{\partial y}\right)_C + \Delta z_k \left(\frac{\partial\phi}{\partial z}\right)_C \right] \right\} = 0
\end{aligned} \tag{9.26}$$

which can be written in matrix form as

$$\begin{aligned}
&\begin{bmatrix}
\sum_{k=1}^{NB(C)} w_k \Delta x_k \Delta x_k & \sum_{k=1}^{NB(C)} w_k \Delta x_k \Delta y_k & \sum_{k=1}^{NB(C)} w_k \Delta x_k \Delta z_k \\
\sum_{k=1}^{NB(C)} w_k \Delta y_k \Delta x_k & \sum_{k=1}^{NB(C)} w_k \Delta y_k \Delta y_k & \sum_{k=1}^{NB(C)} w_k \Delta y_k \Delta z_k \\
\sum_{k=1}^{NB(C)} w_k \Delta z_k \Delta x_k & \sum_{k=1}^{NB(C)} w_k \Delta z_k \Delta y_k & \sum_{k=1}^{NB(C)} w_k \Delta z_k \Delta z_k
\end{bmatrix}
\begin{bmatrix}
\left(\frac{\partial\phi}{\partial x}\right)_C \\
\left(\frac{\partial\phi}{\partial y}\right)_C \\
\left(\frac{\partial\phi}{\partial z}\right)_C
\end{bmatrix} \\
&= \begin{bmatrix}
\sum_{k=1}^{NB(C)} w_k \Delta x_k \Delta\phi_k \\
\sum_{k=1}^{NB(C)} w_k \Delta y_k \Delta\phi_k \\
\sum_{k=1}^{NB(C)} w_k \Delta z_k \Delta\phi_k
\end{bmatrix}
\end{aligned} \tag{9.27}$$

The solution to the above set of equations yields the gradient $(\nabla\phi)_C$. A solution exists provided that the matrix on the left hand side is not singular. Moreover, the choice of the w_k is important in determining the properties of the gradient. For example if w_k is chosen to be 1 for all neighbors of C, then all neighboring points will have the same weight in the computation of the gradient irrespective of whether they are near or far from point C. Actually points that are farther from C will have a more important influence as the error function will be more affected by their error.

Another choice for w_k, which was used earlier with the extended stencil method, is the inverse distance between C and F given by

$$w_k = \frac{1}{|\mathbf{r}_{F_k} - \mathbf{r}_C|} = \frac{1}{\sqrt{\Delta x_{F_k}^2 + \Delta y_{F_k}^2 + \Delta z_{F_k}^2}} \tag{9.28}$$

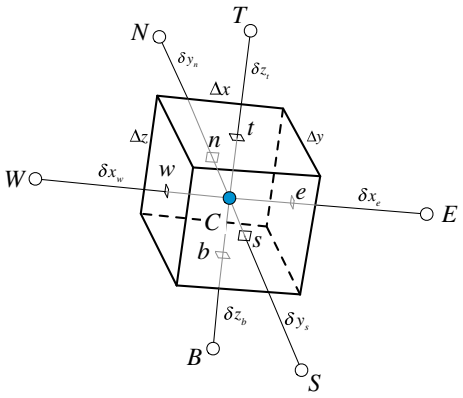

Fig. 9.9 A three dimensional Cartesian control volume

Other options that can be pursued include the inverse distance raised to any power n such that

$$w_k = \frac{1}{\left|\mathbf{r}_{F_k} - \mathbf{r}_C\right|^n} \tag{9.29}$$

where n can be set to 1, 2, 3, etc.

As mentioned above, the divergence based gradient is a special case of the least-square formulation. This can be shown for a Cartesian grid (see Fig. 9.9) where substituting the geometric quantities into Eq. (9.27) would yield the following set of three equations.

$$\begin{bmatrix} x_E - x_W & 0 & 0 \\ 0 & y_N - y_S & 0 \\ 0 & 0 & z_T - z_B \end{bmatrix} \begin{bmatrix} (\partial\phi/\partial x)_C \\ (\partial\phi/\partial y)_C \\ (\partial\phi/\partial z)_C \end{bmatrix} = \begin{bmatrix} \phi_E - \phi_W \\ \phi_N - \phi_S \\ \phi_T - \phi_B \end{bmatrix} \tag{9.30}$$

Solving the above equation, the derivatives in the various directions are found to be

$$\left(\frac{\partial\phi}{\partial x}\right)_C = \frac{\phi_E - \phi_W}{x_E - x_W} \quad \left(\frac{\partial\phi}{\partial y}\right)_C = \frac{\phi_N - \phi_S}{y_N - y_S} \quad \left(\frac{\partial\phi}{\partial z}\right)_C = \frac{\phi_T - \phi_B}{z_T - z_B} \tag{9.31}$$

The obtained values are exactly the ones given in Eq. (9.3), demonstrating that the divergence-based gradient is a special case of the least-square method. Finally it can easily be demonstrated that the accuracy of the resulting gradient is at least first order. Indeed, the Taylor series expansion of the ϕ value around node C can be written as

$$\phi(\mathbf{r}) - \phi(\mathbf{r}_C) = (\nabla\phi)_C \cdot (\mathbf{r} - \mathbf{r}_C) + O\left(\mathbf{r}^2\right) \tag{9.32}$$

which when solved for $(\nabla\phi)_C$ results in $O(\mathbf{r})$.

9.4 Interpolating Gradients to Faces

It was shown in Chap. 8 that the discretization of the diffusion term in non-orthogonal grids requires the use of correction terms involving gradients at control volume faces. Thus in this situation the gradients need to be interpolated from the control volume centroids where they were computed to the control volume faces where they will be used. Figure 9.10a shows the stencil used in the Gauss gradient computation for two neighboring control volumes. The gradient at the face will have exactly the same stencil, while ideally it should be similar to that of Fig. 9.10b.

A better insight is gained by considering the configuration in Fig. 9.11a, which shows the gradients $\nabla\phi_C$ and $\nabla\phi_F$ of the variable ϕ at the two nodes C and F, respectively. The interpolated gradient at the face, $\overline{\nabla\phi}_f$, is obtained by averaging the values at nodes C and F, as shown in Fig. 9.11b. It is important for the stencil of the gradient at the face to be heavily based on the nodes straddling the face, which is not guaranteed by this simple averaging practice. As schematically displayed in Fig. 9.11c, this can be accomplished by forcing the face gradient along the CF direction to be equal to the local gradient defined by the values of ϕ at C and F. Mathematically this can be written as

$$\nabla\phi_f = \overline{\nabla\phi}_f + \underbrace{\left[\frac{\phi_F - \phi_C}{d_{CF}} - \left(\overline{\nabla\phi}_f \cdot \mathbf{e}_{CF}\right)\right]\mathbf{e}_{CF}}_{\textit{Correction interpolated face gradient}} \tag{9.33}$$

where

$$\overline{\nabla\phi}_f = g_C\nabla\phi_C + g_F\nabla\phi_F, \quad \mathbf{e}_{CF} = \frac{\mathbf{d}_{CF}}{d_{CF}}, \quad \mathbf{d}_{CF} = \mathbf{r}_F - \mathbf{r}_C \tag{9.34}$$

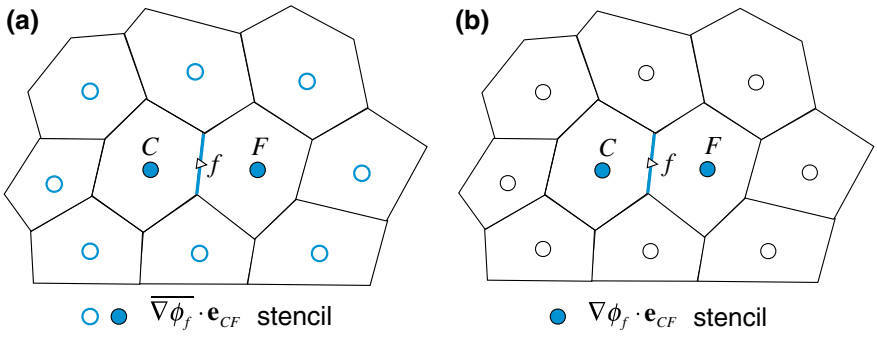

Fig. 9.10 **a** Interpolated and **b** corrected gradient at a control volume face

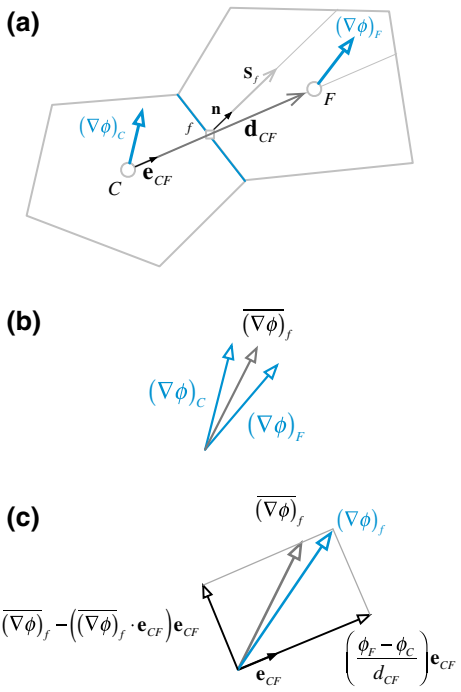

Fig. 9.11 **a** Schematics of the gradients at the two nodes C and F straddling face f; **b** computing $\overline{\nabla\phi}_f$ as a simple average of $\nabla\phi_C$ and $\nabla\phi_F$ using Eq. (9.31); **c** the gradient at the face with its value heavily based on the nodes straddling the face

where $\mathbf{r}_F$ and $\mathbf{r}_C$ are position vectors, as displayed in Fig. 9.3. This approach is applicable to both structured and unstructured grids. For unstructured grids, while the stencil used for the face gradient might not be decreased, the gradient across a face will still be based on the nodes straddling that face.

9.5 Computational Pointers

9.5.1 uFVM

In uFVM, the functions cfdComputeGradientGauss0 and cfdComputeGradientNodal are used to compute the element gradients. The Gauss gradient routine (Listing 9.1) takes as input the phi element array and returns the element gradient array. The values are interpolated to the faces using a weighted factor with no correction for non-conjunctionality, hence the 0 digit in the function name.

```
function phiGrad = cfdComputeGradientGauss0(phi,theMesh)
%=================================================
%  written by the CFD Group @ AUB, Fall 2006
%=================================================
%
if(nargin<2)
    theMesh = cfdGetMesh;
end

theSize = size(phi);
theNumberOfComponents = theSize(2);
if(theNumberOfComponents > 3)
   echo('********* ERROR **********');
   exit;
end
%-------------------------------------------------------
% INTERIOR FACES contribution to gradient
%-------------------------------------------------------
iFaces = 1:theMesh.numberOfInteriorFaces;
iBFaces = theMesh.numberOfInteriorFaces+1:theMesh.numberOfFaces;
iElements = 1:theMesh.numberOfElements;
iBElements        =        theMesh.numberOfElements+1:theMesh.numberOfElements
+theMesh.numberOfBFaces;

iOwners = [theMesh.faces(iFaces).iOwner]';
iNeighbours = [theMesh.faces(iFaces).iNeighbour]';

Sf = [theMesh.faces(iFaces).Sf]';
gf = [theMesh.faces(iFaces).gf]';
%-------------------------------------------------------
% Initialize phiGrad Array
%-------------------------------------------------------

phiGrad       =       zeros(theMesh.numberOfElements+theMesh.numberOfBElements,
3,theNumberOfComponents);

for iComponent=1:theNumberOfComponents
    phi_f = gf.*phi(iNeighbours,iComponent) + (1-gf).*phi(iOwners,iComponent);
    %
    for iFace=iFaces
        phiGrad(iOwners(iFace),:,iComponent)                  =
phiGrad(iOwners(iFace),:,iComponent)     + phi_f(iFace)*Sf(iFace,:);
        phiGrad(iNeighbours(iFace),:,iComponent)     =
phiGrad(iNeighbours(iFace),:,iComponent) - phi_f(iFace)*Sf(iFace,:);
    end
end

%-------------------------------------------------------
% BOUNDARY FACES contribution to gradient
%-------------------------------------------------------
iBOwners = [theMesh.faces(iBFaces).iOwner]';
phi_b = phi(iBElements,iComponent);
Sb = [theMesh.faces(iBFaces).Sf]';
for iComponent=1:theNumberOfComponents
    %
    for k=1:theMesh.numberOfBFaces
        phiGrad(iBOwners(k),:,iComponent)                   =
phiGrad(iBOwners(k),:,iComponent)     + phi_b(k)*Sb(k,:);
    end
end
```

Listing 9.1 Function used in uFVM to compute the gradient field at the centroids of elements following the Green-Gauss approach with phi values at the face computed using simple a weighted average interpolation technique with no correction to non-conjunctionality

```
%-----------------------------------------------------
% Get Average Gradient by dividing with element volume
%-----------------------------------------------------
volumes = [theMesh.elements(iElements).volume]';
for iComponent=1:theNumberOfComponents
    for iElement =1:theMesh.numberOfElements
            phiGrad(iElement,:,iComponent)  =  phiGrad(iElement,:,iComponent)/
volumes(iElement);
    end
end
%-----------------------------------------------------
% Set boundary Gradient equal to associated element
% Gradient
%-----------------------------------------------------
phiGrad(iBElements,:,:) = phiGrad(iBOwners,:,:);


end
```

Listing 9.1 (continued)

The nodal gradient routine, listed below (Listing 9.2), is very similar to the Gauss gradient routine except that it uses the functions cfdInterpolateFrom-ElementsToNodes and cfdInterpolateFromNodesToFaces to interpolate phi values to the face that are subsequently used in the Gauss algorithm.

```
%
%=====================================================
% INTERIOR Gradients
%=====================================================
theNumberOfElements = theMesh.numberOfElements;
theNumberOfBElements = theMesh.numberOfBElements;
theNumberOfInteriorFaces = theMesh.numberOfInteriorFaces;
%
phiNodes = cfdInterpolateFromElementsToNodes(phi);
phi_f = cfdInterpolateFromNodesToFaces(phiNodes);
%
phiGrad = zeros(3,theNumberOfElements+theNumberOfBElements);
%-----------------------------------------------------
% INTERIOR FACES contribution to gradient
%-----------------------------------------------------
fvmFaces = theMesh.faces;

% interpolate phi to faces
%
for iFace=1:theNumberOfInteriorFaces
   %
   theFace = fvmFaces(iFace);
   %
   iElement1 = theFace.iOwner;
   iElement2 = theFace.iNeighbour;
   %
```

Listing 9.2 Function used in uFVM to compute the gradient field at the centroids of elements following the Green-Gauss approach with phi values at the face computed using nodal values, i.e., the extended stencil approach

```
   Sf = theFace.Sf;
   %
   %
   phiGrad(:,iElement1) = phiGrad(:,iElement1) + phi_f(iFace)*Sf;
   phiGrad(:,iElement2) = phiGrad(:,iElement2) - phi_f(iFace)*Sf;

end


%======================================================
% BOUNDARY FACES contribution to gradient
%======================================================
for iBPatch=1:theNumberOfBElements
    %
    iBFace = theNumberOfInteriorFaces+iBPatch;
    iBElement = theNumberOfElements+iBPatch;
    theFace = fvmFaces(iBFace);
    %
    iElement1 = theFace.iOwner;
    %
    Sb = theFace.Sf;
    phi_b = phi(iBElement);
    %
    phiGrad(:,iElement1) = phiGrad(:,iElement1) + phi_b*Sf;

end
%------------------------------------------------------
% Get Average Gradient by dividing with element volume
%------------------------------------------------------
for iElement =1:theNumberOfElements
   theElement = fvmElements(iElement);
   phiGrad(:,iElement) = phiGrad(:,iElement)/theElement.volume;
end


%------------------------------------------------------
% Set boundary Gradient equal to associated element
% Gradient
%------------------------------------------------------
for iBPatch = 1:theNumberOfBElements
    iBElement = iBPatch+theNumberOfElements;
    iBFace = iBPatch+theNumberOfInteriorFaces;
    theBFace = fvmFaces(iBFace);
    iOwner = theBFace.iOwner;
    phiGrad(:,iBElement) = phiGrad(:,iOwner);
end
```

Listing 9.2 (continued)

For face interpolation of gradients, a variety of interpolation options are allowed in the function **cfdInterpolateGradientsFromElementsToInteriorFaces**. The input to the function is **theInterpolationScheme**, the element gradient **grad**, the element array **phi** and **mdot** for cases where an upwind or downwind scheme is used. The function is shown in Listing 9.3.

```
function grad_f=
cfdInterpolateGradientsFromElementsToInteriorFaces(theInterpolationScheme,grad
,phi,mdot_f)
%=================================================

%  written by the CFD Group @ AUB, Fall 2006
%=================================================

theMesh= cfdGetMesh;

numberOfInteriorFaces = theMesh.numberOfInteriorFaces;

iOwners = [theMesh.faces(1:numberOfInteriorFaces).iOwner]';
iNeighbours = [theMesh.faces(1:numberOfInteriorFaces).iNeighbour]';
gf = [theMesh.faces(1:numberOfInteriorFaces).gf]';

if(strcmp(theInterpolationScheme,'Average')==1)
    grad_f(:,1) = (1-gf).*grad(iNeighbours,1) + gf.*grad(iOwners,1);
    grad_f(:,2) = (1-gf).*grad(iNeighbours,2) + gf.*grad(iOwners,2);
    grad_f(:,3) = (1-gf).*grad(iNeighbours,3) + gf.*grad(iOwners,3);

elseif(strcmp(theInterpolationScheme,'Upwind')==1)
     pos = zeros(size(mdot_f));
     pos((mdot_f>0))=1;
     %
     grad_f(:,1) = pos.*grad(iNeighbours,1) + (1-pos).*grad(iOwners,1);
     grad_f(:,2) = pos.*grad(iNeighbours,2) + (1-pos).*grad(iOwners,2);
     grad_f(:,3) = pos.*grad(iNeighbours,3) + (1-pos).*grad(iOwners,3);
elseif(strcmp(theInterpolationScheme,'Downwind')==1)
     pos = zeros(size(mdot_f));
     pos((mdot_f>0))=1;
     %
     grad_f(:,1) = (1-pos).*grad(iNeighbours,1) + pos.*grad(iOwners,1);
     grad_f(:,2) = (1-pos).*grad(iNeighbours,2) + pos.*grad(iOwners,2);
     grad_f(:,3) = (1-pos).*grad(iNeighbours,3) + pos.*grad(iOwners,3);

elseif(strcmp(theInterpolationScheme,'Average:Corrected')==1)
    grad_f(:,1) = (1-gf).*grad(iNeighbours,1) + gf.*grad(iOwners,1);
    grad_f(:,2) = (1-gf).*grad(iNeighbours,2) + gf.*grad(iOwners,2);
    grad_f(:,3) = (1-gf).*grad(iNeighbours,3) + gf.*grad(iOwners,3);

    d_CF     =     [theMesh.elements(iNeighbours).centroid]'     -
[theMesh.elements(iOwners).centroid]';
    dmag=cfdMagnitude(d_CF);

    e_CF(:,1)=d_CF(:,1)./dmag;
    e_CF(:,2)=d_CF(:,2)./dmag;
    e_CF(:,3)=d_CF(:,3)./dmag;

    local_grad_mag_f = (phi(iNeighbours)-phi(iOwners))./dmag;
    local_grad(:,1) = local_grad_mag_f.*e_CF(:,1);
    local_grad(:,2) = local_grad_mag_f.*e_CF(:,2);
    local_grad(:,3) = local_grad_mag_f.*e_CF(:,3);

    local_avg_grad_mag = dot(grad_f',e_CF')';
    local_avg_grad(:,1)=local_avg_grad_mag.*e_CF(:,1);
    local_avg_grad(:,2)=local_avg_grad_mag.*e_CF(:,2);
    local_avg_grad(:,3)=local_avg_grad_mag.*e_CF(:,3);

    grad_f = grad_f - local_avg_grad + local_grad;
else
    theInterpolationScheme
    grad,
    phi,mdot
    exit;
end
```

Listing 9.3 Function used to interpolation the element gradient field to the faces

It should be noted that **theInterpolationScheme** "*Average:Corrected*" implements the face gradient correction technique used to get a more accurate gradient representation along the *CF* direction, i.e., Eq. (9.33).

9.5.2 OpenFOAM®

In OpenFOAM® [8] several types of gradient evaluation techniques are defined: the standard Green-Gauss method, the second order least square method, and the fourth order least square method. Attention will be focussed here on the Green-Gauss method. Nonetheless the discretization in all cases is performed explicitly and is thus part of the fvc operator namespace. The Green-Gauss gradient is defined at the cell centre, as in Eq. (9.4), and its source code in OpenFOAM is located in the directory "src/finiteVolume/finiteVolume/gradSchemes/gaussGrad".

Implementation-wise, the gradient evaluation is performed in the following two steps:

- Face interpolation of the variable
- Green-Gauss formula evaluation

The interpolation to the face is in the **calcGrad** routine listed below (Listing 9.4).

```
Foam::fv::gaussGrad<Type>::calcGrad
(
    const GeometricField<Type, fvPatchField, volMesh>& vsf,
    const word& name
) const
{
    typedef typename outerProduct<vector, Type>::type GradType;
    tmp<GeometricField<GradType, fvPatchField, volMesh> > tgGrad
    (

        gradf(tinterpScheme_().interpolate(vsf), name)
    );
    GeometricField<GradType, fvPatchField, volMesh>& gGrad = tgGrad();
    correctBoundaryConditions(vsf, gGrad);
    return tgGrad;
}
```

Listing 9.4 Script to interpolate variable values to cell faces

As a first step, values are interpolated to the faces and stored in the generic field "vsf" (the interpolation class will be described with more details in Chap. 11). Then the gradient is computed based on the Green-Gauss formula in the gradf routine shown in Listing 9.5.

```
Foam::fv::gaussGrad<Type>::gradf
(
    const GeometricField<Type, fvsPatchField, surfaceMesh>& ssf,
    const word& name
)
{
//     Info << "Calculating gradient for " << name  << endl;
    typedef typename outerProduct<vector, Type>::type GradType;

    const fvMesh& mesh = ssf.mesh();
    tmp<GeometricField<GradType, fvPatchField, volMesh> > tgGrad
    (
        new GeometricField<GradType, fvPatchField, volMesh>
        (
            IOobject
            (
                name,
                ssf.instance(),
                mesh,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            mesh,
            dimensioned<GradType>
            (
                "0",
                ssf.dimensions()/dimLength,
                pTraits<GradType>::zero
            ),
            zeroGradientFvPatchField<GradType>::typeName
        )
    );
    GeometricField<GradType, fvPatchField, volMesh>& gGrad = tgGrad();

    const labelUList& owner = mesh.owner();
    const labelUList& neighbour = mesh.neighbour();
    const vectorField& Sf = mesh.Sf();

    Field<GradType>& igGrad = gGrad;
    const Field<Type>& issf = ssf;

    forAll(owner, facei)
    {
        GradType Sfssf = Sf[facei]*issf[facei];
        igGrad[owner[facei]] += Sfssf;
        igGrad[neighbour[facei]] -= Sfssf;
    }

    forAll(mesh.boundary(), patchi)
    {
        const labelUList& pFaceCells =
            mesh.boundary()[patchi].faceCells();
        const vectorField& pSf = mesh.Sf().boundaryField()[patchi];
        const fvsPatchField<Type>& pssf = ssf.boundaryField()[patchi];
        forAll(mesh.boundary()[patchi], facei)
        {
            igGrad[pFaceCells[facei]] += pSf[facei]*pssf[facei];
        }
    }
    igGrad /= mesh.V();
    gGrad.correctBoundaryConditions();
    return tgGrad;
}
```

Listing 9.5 Routine used to compute the gradient using the Green-Gauss method

The sum over cell faces is performed using the LDU addressing. As such the "for" loop that evaluates the sum over the faces of the cell is based only on the global face numbering and uses the upper and lower addressing vectors to add or subtract (Listing 9.6) the flux to cell values. After all fluxes have been processed, the net value is divided by the volume to yield the gradient, as in Eq. 9.4.

```
        igGrad[owner[facei]] += Sfssf;
        igGrad[neighbour[facei]] -= Sfssf;
```

Listing 9.6 Adding or subtracting fluxes to cell values

The type of gradient is defined in fvSchemes shown in Listing 9.7.

```
gradSchemes
{
    default         none;
    grad(phi)   Gauss;
}
```

Listing 9.7 Defining the gradient calculation method

The face interpolation scheme is defined as displayed in Listing 9.8.

```
interpolationSchemes
{
    interpolate(phi) linear;
}
```

Listing 9.8 Defining the interpolation method used in calculating face values

The idea of evaluating the gradient based on a generic interpolation scheme that has to be defined by dictionary, allows computing the gradient with the Green-Gauss formula in several ways by just changing the interpolation scheme.

If skew correction, as defined in Eq. (9.8), is required then the above interpolation scheme definition should be replaced by (Listing 9.9)

```
interpolationSchemes
{
    interpolate(phi) skewCorrected linear;
}
```

Listing 9.9 Defining the interpolation scheme to calculate face values with skew correction

A more compact syntax can be used for defining the gradient in which the interpolation type is specified directly under gradSchemes shown in Listing 9.10.

```
gradSchemes
{
    default         none;
    grad(phi)   Gauss linear;
}
```

Listing 9.10 Defining the gradient calculation method: compact syntax

In this case the interpolation method is defined directly in the gradient dictionary. The choice of the syntax type is up to the user keeping in mind that a separate definition of the interpolation scheme helps clarifying the various steps of the gradient calculation.

9.6 Closure

This chapter presented the discretization details of two methods for computing the gradient at the centroids of control volume meshes in general non-orthogonal grid systems. One method is based on the Green-Gauss theorem and the second on the Least-Square reconstruction approach. Chapter 10 will be devoted to methods used for solving systems of algebraic equations.

9.7 Exercises

Exercise 1
In the configuration depicted in Fig. 9.22, the values of the variable ϕ and its gradient $\nabla\phi$ at C are known. Using these known values estimate the value of ϕ at F.

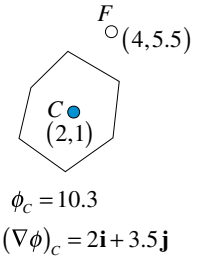

Fig. 9.22 A two dimensional element of centroid C and neighbor F

Exercise 2

For the two cells shown in Fig. 9.23 the value of some scalar ϕ at their centroids C and F can be calculated from

$$\phi(x, y) = 100\left(x^2 y + y^2 x\right)$$

The gradient at C and F are also computed to be the exact gradient of the function at these points such that

$$\nabla\phi(x, y) = 100\left(2xy + y^2\right)\mathbf{i} + 100\left(x^2 + 2yx\right)\mathbf{j}$$

With C and F located at (0.32, −0.1) and (0.52, 0.31), respectively, find the value of the gradient at face f (0.43, 0.1) numerically (not from the expression for the gradient) using

a. A simple averaging between C and F.
b. A corrected averaging between C and F.
c. Compare the two computed values with the exact gradient at point f.

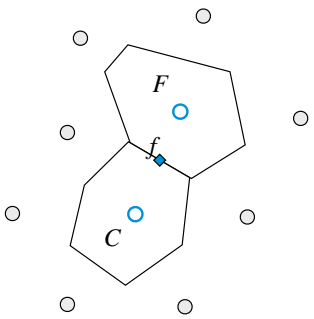

Fig. 9.23 Two elements in a two dimensional plane with their centroids ***C*** and ***F***

Exercise 3

Consider the mesh composed of equilateral triangular elements shown in Fig. 9.24. The coordinates of the mesh vertices are as shown. The temperatures at the cell centroids are given by

$$T(x, y, z) = 100\left(x^2 + y^2 + 1\right)$$

a. Compute the gradient at 1 using the Green-Gauss method.
b. Compute the gradient at 1 using the least squares approach with a limited stencil (i.e., nodes 2, 3 and 4) and with an extended stencil (i.e., nodes 2–13) used for the reconstruction.
c. Compare the least squares gradient with the limited and extended stencil and the gauss gradient to the exact gradient.

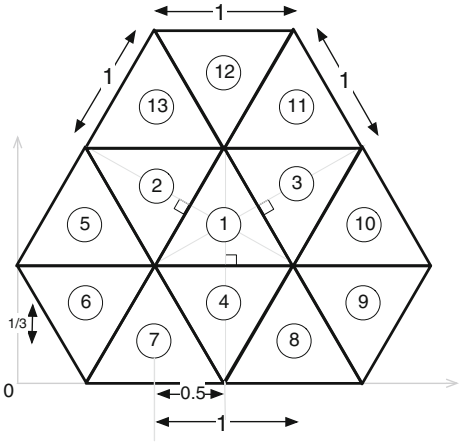

Fig. 9.24 A domain discretized using triangular elements

Exercise 4

Consider the uniform mesh shown in Fig. 9.25. The coordinates of the mesh vertices are as shown. The temperatures at the cell nodes are given by

$$T(x, y) = 100\left(x^3 + y^3 + xy + 1\right)$$

compute the gradient at the centroid of cell 1 using the gauss gradient with and without face corrections and compare to the exact gradient given by

$$\nabla T(x, y) = 100\left(3x^2 + y\right)\mathbf{i} + 100\left(3y^2 + x\right)\mathbf{j}$$

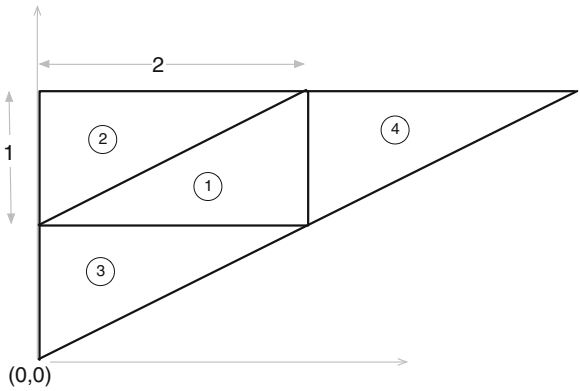

Fig. 9.25 Mesh system for Exercise 4

Exercise 5 (uFVM, OpenFOAM®)
Using uFVM and then OpenFOAM® write a program that

a. reads an OpenFOAM® mesh and sets a scalar field with values equal to $\phi(x, y, z) = 10x^2y^2 + 3y$
b. computes the gradients using the second order least square method,
c. computes the gradients using the gauss gradient method with linear interpolation,
d. computes the gradients using the gauss gradient method with vertex interpolation,
e. and computes the root mean square error between each of the computed gradients and the exact gradient.

Exercise 6 (uFVM, OpenFOAM®)
Using uFVM and then OpenFOAM® write a program that

a. reads an OpenFOAM® mesh and sets a scalar field with values equal to $\phi(x, y, z) = 10x^2y^2 + 3x$

Use the exact gradients at the element nodes to compute the gradients at the interior faces using

b. simple linear interpolation,
c. corrected linear interpolation,
d. and exact gradient formulation at the face centroids.
e. Compute the root mean square error between each of the computed gradients and the exact gradient.

Exercise 7
Starting with Eq. (9.8) derive Eq. (9.9).
Hint: $\mathbf{r}_f - \mathbf{r}_{f'} = \mathbf{r}_f - \mathbf{r}_C - \mathbf{r}_{cf'} = \mathbf{r}_f - \mathbf{r}_f - \mathbf{r}_{Ff'}$.

Exercise 8 (OpenFOAM®)

a. List all possible gradient type definitions available in OpenFOAM® by modifying the gradSchemes in the fvSchemes dictionary (Hint: just mistype a scheme, e.g., banana, and launch any solver or application, i.e., gradSchemes {default banana;}).
b. Define in the dictionary file fvSchemes the option to evaluate the explicit gradient with the least square algorithm.
c. Use the Doxygen documentation [9] to analyze the member function correctBoundaryConditions and explain its operations and aim.
d. Define in the dictionary file fvSchemes the option to use a limited gradient (face and cell) (Gauss limited 0.8;).

References

1. Ferziger JH, Perić M (2002) Computational Methods for Fluid Dynamics, 3rd edn. Springer, Berlin
2. Soni B (1998) Hybrid techniques in computational fluid dynamics. Technical Report no. MSSU-COE-ERC-98-3, Engineering Research Center for Computational Field Simulation, Mississippi State University
3. Frink NT, Parikh P, Pirzadeh S (1991) Aerodynamic Analysis of Complex Configurations Using Unstructured Grids. AIAA 91-3292
4. Cabello J, Morgon K, Lohner R (1994) A Comparison of Higher Order Schemes Used in a Finite Volume Solver For Unstructured Grids. AIAA 94-2293. Presented at the 25th AIAA Plasmadynamics and Lasers Conference, Colorado Springs, CO
5. Musaferija I, Gosman D (1997) Finite-Volume CFD procedure and adaptive error control strategy for grids of arbitrary topology. J Comp Phys 138:766–787
6. Barth TJ, Jespersen DC (1989) The design and application of upwind schemes on unstructured meshes. AIAA 89-0366
7. Ollivier-Gooch C, Van Altena M (2002) A high-order-accurate unstructured mesh finite-volume scheme for the advection–diffusion equation. J Comp Phys 181:729–752
8. OpenFOAM (2015) Version 2.3.x. http://www.openfoam.org
9. OpenFOAM Doxygen (2015) Version 2.3.x. http://www.openfoam.org/docs/cpp/