Another common way of solving the differential equation (26.10) uses finite differences on the grid. It is clear that if a control point coincides with a node then the node value should be set to the control point. At nodes where there are no control points in the vicinity we know that $\nabla^4 z = 0$. We may approximate the biharmonic operator by finite differences. Consider $\frac{\partial z(x, y)}{\partial x}$. At $x_i$ this is the slope of $z$ at $x_i$. This can be approximated by

$$\frac{\partial z(x, y)}{\partial x} \approx \frac{z(x_{i+1}, y) - z(x_i, y)}{\Delta x}$$

$\frac{\partial^2 z}{\partial x^2}$ can then be obtained by applying the first difference twice which gives

$$\frac{\partial^2 z}{\partial x^2} \bigg|_{x,y} \approx \frac{1}{\Delta x^2} \left(z(x_{i+1}, y) - 2z(x_i, y) + z(x_{i-1}, y)\right)$$

The same can be done in the $y$-direction:

$$\frac{\partial^2 z}{\partial y^2} \bigg|_{x,y} \approx \frac{1}{\Delta y^2} \left(z(x_i, y_{j+1}) - 2z(x_i, y_j) + z(x_i, y_{j-1})\right)$$

For simplicity we assume a square lattice ($\Delta x = \Delta y$), then

$$\nabla^2 z \bigg|_{x,y} \approx \frac{1}{\Delta^2} \left(z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1} - 4z_{i,j}\right)$$

(27.1)

The biharmonic is then

$$\nabla^4 z = \nabla^2 \nabla^2 z \approx \frac{1}{\Delta^2} \left(C_{i+1,j} + C_{i-1,j} + C_{i,j+1} + C_{i,j-1} - 4C_{i,j}\right)$$

(27.2)

By letting $i = 0$ and $j = 0$ be the current node, we evaluate and find $\nabla^4 z = 0$ becomes

$$z_{20} + z_{-20} + z_{02} + z_{0-2} + 2(z_{11} + z_{-11} + z_{1-1} + z_{-1-1}) - 8(z_{10} + z_{-10} + z_{01} + z_{0-1}) + 20z_{00} = 0$$

(27.3)

where we have used the notation $z_{i+1,j+1} = z_{i+1,j+1}$. We solve for $z_{00}$ and find

$$z_{00} = \frac{1}{20} \left[8\left(z_{10} + z_{-10} + z_{01} + z_{0-1}\right) - 2\left(z_{11} + z_{-11} + z_{1-1} + z_{-1-1}\right) - z_{20} - z_{-20} - z_{02} - z_{0-2}\right]$$

(27.4)
Fig. 27-1. Zooming in on a node with a nearby data constraint (E).

However, most of the time the data point will not exactly coincide with the nodes, so how can we add the constraints to the process? In the cases where we have a control datum near a node we must develop a separate difference equation that equates \( z_{00} \) with nearby nodes and the actual data constraint. Consider the situation in Figure 27-1. Here, the central node (square) has a nearby constraining data point \( E \) that we would like to use. We will do so by first approximately the surface value \( z(x,y) \) at \( E \) by a 2nd-order Taylor expansion from the node. Using \( u \) and \( v \) as the relative distances from the node we get

\[
z_k = z_{00} + u_k \frac{\partial z}{\partial x} + v_k \frac{\partial z}{\partial y} + \frac{1}{2} u_k^2 \frac{\partial^2 z}{\partial x^2} + u_k v_k \frac{\partial^2 z}{\partial x \partial y} + \frac{1}{2} v_k^2 \frac{\partial^2 z}{\partial y^2} \quad (27.5)
\]

where the subscript \( k \) indicates the values at the off-node point. We saw earlier that the biharmonic equation could be written in terms of the curvatures at the node and the 4 nearest nodes. Note that the Taylor expansion has terms similar to that of the curvature operator \( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \). This means we may be able to use this expansion to express the curvature \( C_{00} \) in terms of \( z_k \). To see how this can be done, consider the four nodes A-D in addition to the data constraint at \( E \). We can also use the Taylor expansion to express the value at these nodes in terms of derivatives of \( z \). Let \( k \), the point index go from 1 to 5 (A-E). Multiply each Taylor expansion by a constant \( b_k \), and sum the five expressions:

\[
\sum b_k z_k = \sum b_k + \frac{\partial z}{\partial x} \sum b_k u_k + \frac{\partial z}{\partial y} \sum b_k v_k + \frac{1}{2} \sum \frac{\partial^2 z}{\partial x^2} \sum b_k u_k^2 + \frac{\partial^2 z}{\partial x \partial y} \sum b_k u_k v_k + \frac{1}{2} \sum \frac{\partial^2 z}{\partial y^2} \sum b_k v_k^2
\]

Now, if we cleverly choose the \( b_k \) such that

\[
\sum b_k u_k = \sum b_k v_k = \sum b_k u_k v_k = 0 \quad \sum b_k u_k^2 = \sum b_k v_k^2 = 2
\]

then the sum of the equations reduce to

\[
\nabla^2 z = \frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = C_{00} = \sum b_k z_k - z_{00} \sum b_k
\]

which relates the curvature at \((0,0)\) to the 5 points and the central node estimate. To find the \( b_k \) we must solve a matrix equation resulting from the imposed requirements. Assume that \( u_k, v_k \) have been normalized by the grid spacing \( \Delta \), e.g.,
\[ u_k = \frac{x_k - x_{00}}{\Delta}, \quad v_k = \frac{y_k - y_{00}}{\Delta} \]

the summation constraints gives these five equations

\[
\begin{bmatrix}
-1 & -1 & 0 & 1 & u_E \\
1 & 0 & -1 & -1 & v_E \\
1 & 1 & 0 & 1 & u_E \\
-1 & 0 & 0 & -1 & u_E v_E \\
1 & 0 & 1 & 1 & v_E \\
\end{bmatrix} \cdot \begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
\end{bmatrix} = \begin{bmatrix} 0 \\
0 \\
2 \\
0 \\
2 \\
\end{bmatrix}
\]

with solution

\[
b_5 = \frac{4}{(u_E + v_E)(1 + u_E + v_E)} \quad b_4 = 1 - \frac{1}{2} b_5 (u_E + u_E^2) \\
b_3 = v_E (1 + u_E) b_5 - 2 b_4 \\
b_2 = v_E (1 + u_E) b_5 - b_3 \\
b_1 = u_E b_5 + b_4 - b_2
\]

(27.7)

These expressions apply when the datum is in the first quadrant. Similar expressions arise when A–D are selected differently. We can now substitute our expression for \( \nabla^2 z = C_{00} \) into \( \nabla^4 z = 0 \) and solve for \( z_{00} \). This difference equation is then used at all nodes with data constraints while the simpler expressions developed earlier are used at “empty” nodes. These equations allow us to iteratively solve for all the \( z_{00} \) until the change at any node is minimal. The expression for the constrained node therefore gives (27.8):

\[
z_{00} = \frac{4(z_{10} + z_{-10} + z_{01} + z_{0-1}) - 2(z_{11} + z_{-11} + z_{1-1} + z_{-1-1}) - z_{20} - z_{-20} - z_{02} - z_{0-2} - 4 \sum_{k=1}^{5} b_k z_k}{4 \left( \sum_{k=1}^{5} b_k + 1 \right)}
\]

When the current node is on the edge of the grid or the next gridline inside, (27.3) refers to \( z \)-values at nodes that are non-existent. For these cases we must use the boundary equations. In 1-D we simply set the curvature to zero at the end-points; here we set

\[
\frac{\partial^2 z}{\partial n^2} = 0 \quad \frac{\partial}{\partial n} (\nabla^2 z) = 0 \quad \frac{\partial^2 z}{\partial x \partial y} = 0
\]

(27.9a, 27.9b, 27.9c)

where \( n \) is the direction normal to the edge, and (27.9c) is applied at the four corners. Expressing these equations as finite differences results in expressions that relate the outside nodes to the inside nodes. Substituting these expressions into (27.3) gives relations that only contains inside nodes. For example, the \( n \)-derivative along \( x = 0 \) gives

\[
z_{-10} - 2z_{00} + z_{10} = 0 \quad \Rightarrow \quad z_{-10} = 2z_{00} - z_{10}
\]

(27.10)
Equations (27.9b) will similarly result in

\[ z_{-20} = z_{20} + \left( z_{11} + z_{-1-1} - z_{-1-1} - z_{-10} \right) - 4 \left( z_{10} - z_{-10} \right) \]  (27.11)

Figure 27–2. Grid nodes (black circles) and auxiliary outside boundary points.

Similar expressions result for the outside nodes at the other boundaries. We will use these equations iteratively to solve the value \( z_{00} \) at each node in terms of neighboring nodes. Assume that all the nodes have some initial values (maybe assigned by a MA operation). Using (27.9) we first compute the values at all the outside points, then solve (27.4,8) at all inside nodes. Thus, a computer routine will visit all the nodes in the grid and update the value based on (27.4,8). This is repeated again and again until there is no change in the node value - we say that our procedure has converged. Unfortunately, convergence in numerical analysis is not the same as convergence in mathematics. While the set of equations \( \nabla^4 z = 0 \) and boundary conditions have a unique solution, we are likely not to spend enough time to find this solution but accept a slightly less accurate solution obtained in less than infinite time.

In practice, a few more tricks are applied to assure a rapid convergence. These are over-relaxation and multiple grid strategy.

1. Over-relaxation

This simply means that we anticipate that the node estimate \( z_{00} \) will fall somewhat short of the final estimate so we try to speed up the convergence by “overshooting” the estimate such that

\[ z_{00}^{new} \leftarrow (1 - w)z_{00}^{old} + wz_{00}^{new} \]  (27.12)

\( W \) must be in the \( 1 < w < 2 \) range, with \( w = 1.5 \) typically being a best choice. This technique reduces the total number of iterations. It is possible to determine the optimal \( w \) for many problems solved by iterative methods. However, it is simpler to find this empirically. We can do this by tracking the convergence for several runs with different \( w \). Typically, there will be a preferred value, which gives the fastest convergence. Smaller values give slower convergence and larger values make the solution diverge. (for some problems, in particular linearized nonlinear equations, we may have to use \( w < 1 \) which is called “under-relaxation”).
2. Multiple grids

Instead of integrating directly on the desired grid, we initially choose a coarse grid spacing (in multiples of final spacing) and solve the gridding equations on this intermediate grid. This is fast because the number of nodes is small. Once converged we interpolate onto a finer grid using a bilinear interpolant, then repeat the process until we arrive at the final grid spacing. This approach assures that the long-wavelength trends in the data are propagated quickly since they can be resolved on a coarser grid, while finer detail, which can be local in nature, is solved for only in the end. It is useful to make sure $nx$ and $ny$ have many factors in common so that many intermediate grid spacing can be useful to propagate the effects over the grid efficiently.

One problem with cubic splines was their behavior in the presence of sharp gradients in the data. To minimize the curvature the interpolant made large excursions to avoid sharp turns. By doing so the predicted values could easily exceed the data constraints on either side. We found that by pulling at the end of the curve we introduced tension, which effectively removed these artifacts. In the limit (infinite tension), the spline degenerates to a linear interpolation, with $\frac{\partial^2 z}{\partial x^2} = \delta(x_i)$. In 2-D we can add this feature by modifying our differential equation to take tension into account. We get

$$(1 - t)\nabla^4 z - t\nabla^2 z = \sum_{i=1}^{n} a_i \delta(r)$$

(27.13)

where $t$ is a constant that indicates the amount of tension ($t = 0$ is no tension, $t = 1$ is infinite tension). While this equation may seem more formidable to solve, the finite difference expression involves exactly the same nodes, but the weight assigned to them are slightly different. Thus an algorithm that can solve $\nabla^4 z = 0$ can easily be modified to solve the more general case in (27.13). Let us look closer at the weights implied

With no tension the weights actually goes negative. It is this characteristic that may cause oscillations since it implies a negative autocorrelation for a certain distance. We see the full tension solution ($t = 1$) only has positive weights and are more local. Hence, no ringing will take place and data constraints will only propagate their effect a very short distance. Intermediate values of $t$ will give intermediate positions. Thus, a solution derived from (27.13) will still have continuous curvature but it is no longer minimized. As in the case of splines in tension, the amount (value of $t$) is application dependent.

With tension, we keep the boundary conditions as before except $\frac{\partial^2 z}{\partial n^2} = 0$ is replaced by

$$(1 - t)\frac{\partial^2 z}{\partial n^2} + t\frac{\partial z}{\partial n} = 0$$

(27.14)

Because weights are more local the iterative solution is faster when $t > 0$. 