Additional Material from Chapter 11

- Solving a system of linear equations using iterative methods

Relationship of Matrices to Linear Equations

A system of linear equations can be written in matrix form:

$$\mathbf{Ax} = \mathbf{b}$$

Matrix **A** holds the *a* constants

x is a vector of the unknowns

b is a vector of the *b* constants.

Solving a System of Linear Equations

$$
\begin{array}{llll}
a_{n-1,0}x_0 + a_{n-1,1}x_1 + a_{n-1,2}x_2 & \ldots & + a_{n-1,n-1}x_{n-1} & = b_{n-1} \\
 & . & & \\
 & . & & \\
 & . & & \\
a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 & \ldots & + a_{2,n-1}x_{n-1} & = b_2 \\
a_{1,0}x_0 + a_{1,1}x_1 + a_{1,2}x_2 & \ldots & + a_{1,n-1}x_{n-1} & = b_1 \\
a_{0,0}x_0 + a_{0,1}x_1 + a_{0,2}x_2 & \ldots & + a_{0,n-1}x_{n-1} & = b_0
\end{array}
$$

which, in matrix form, is

$$\mathbf{Ax} = \mathbf{b}$$

Objective is to find values for the unknowns, $x0$, $x1$, …, xn-1, given values for $a0,0$, $a0,1$, …, an-1,n-1, and $b0$, …, bn .

Solving a System of Linear Equations

Dense matrices

Gaussian Elimination - parallel time complexity O(n2)

Sparse matrices

By iteration - depends upon iteration method and number of iterations but typically O(log n)

- Jacobi iteration
- Gauss-Seidel relaxation (not good for parallelization)
- Red-Black ordering
- Multigrid

Iterative Methods

Time complexity of direct method at O(N^2) with N processors, is significant.

Time complexity of iteration method depends upon:

- the type of iteration,
- number of iterations
- number of unknowns, and
- required accuracy

but can be less than the direct method especially for a few unknowns i.e a sparse system of linear equations.

Jacobi Iteration

Iteration formula - *i*th equation rearranged to have *i*th unknown on left side:

$$x_i^k = \frac{1}{a_{i,i}}\left[b_i - \sum_{j \neq i} a_{i,j} x_j^{k-1}\right]$$

Superscript indicates iteration:

x_i^k *is k*th iteration of x_i , x_j^{k-1} is (*k*–1)th iteration of x_j.

Example of a Sparse System of Linear Equations
Laplace's Equation

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

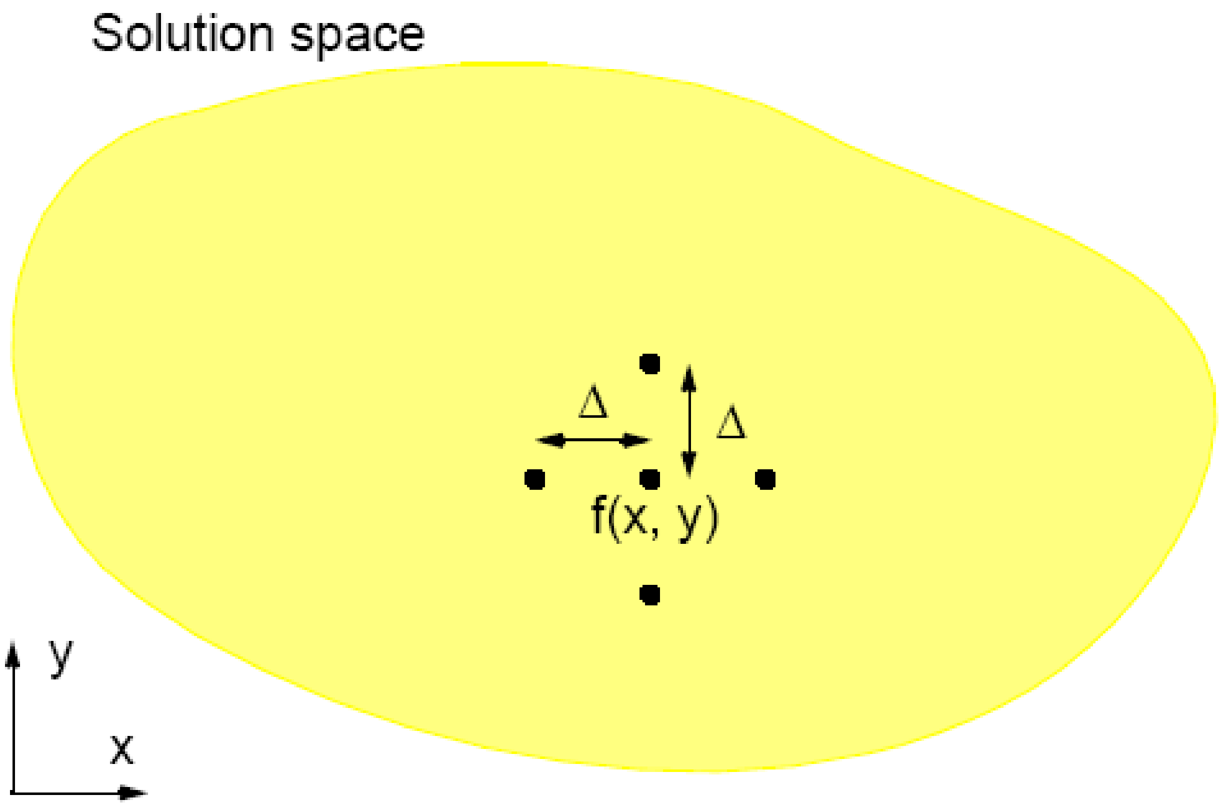
Also need to specify a domain
0 <= *x* <= *1*
0 <= *y* <= *1*
And also to specify
f(x,y) on the edges of the domain

Solve for *f* over the two-dimensional x-y space.

For a computer solution, *finite difference* methods are appropriate

Two-dimensional solution space is "discretized" into a large number of solution points.

Finite Difference Method



Method of Manufactured Solutions

A way to construct solutions for equations so that you cantest numerical accuracy

E.G consider $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$, u known on boundaries

We may not always know an exact solution u(x,y) so suppose We know an approximate solution v(x,y) we van then test the Accuracy of our code by solving the equation that v(x,y) does satisfy

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = f(x, y)$$

E.G if $v(x,y) = 6\,x^3 + 3\,y^2 \;\; then \;\; \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 36x + 6$

This is called the method of manufactured solutions

For analytical solutions see course web page

If distance between points, Δ, made small enough:

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{1}{\Delta^2}[f(x+\Delta, y) - 2f(x, y) + f(x-\Delta, y)]$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{1}{\Delta^2}[f(x, y+\Delta) - 2f(x, y) + f(x, y-\Delta)]$$

Substituting into Laplace's equation, we get

$$\frac{1}{\Delta^2}[f(x+\Delta, y) + f(x-\Delta, y) + f(x, y+\Delta) + f(x, y-\Delta) - 4f(x, y)] = 0$$

Rearranging, we get

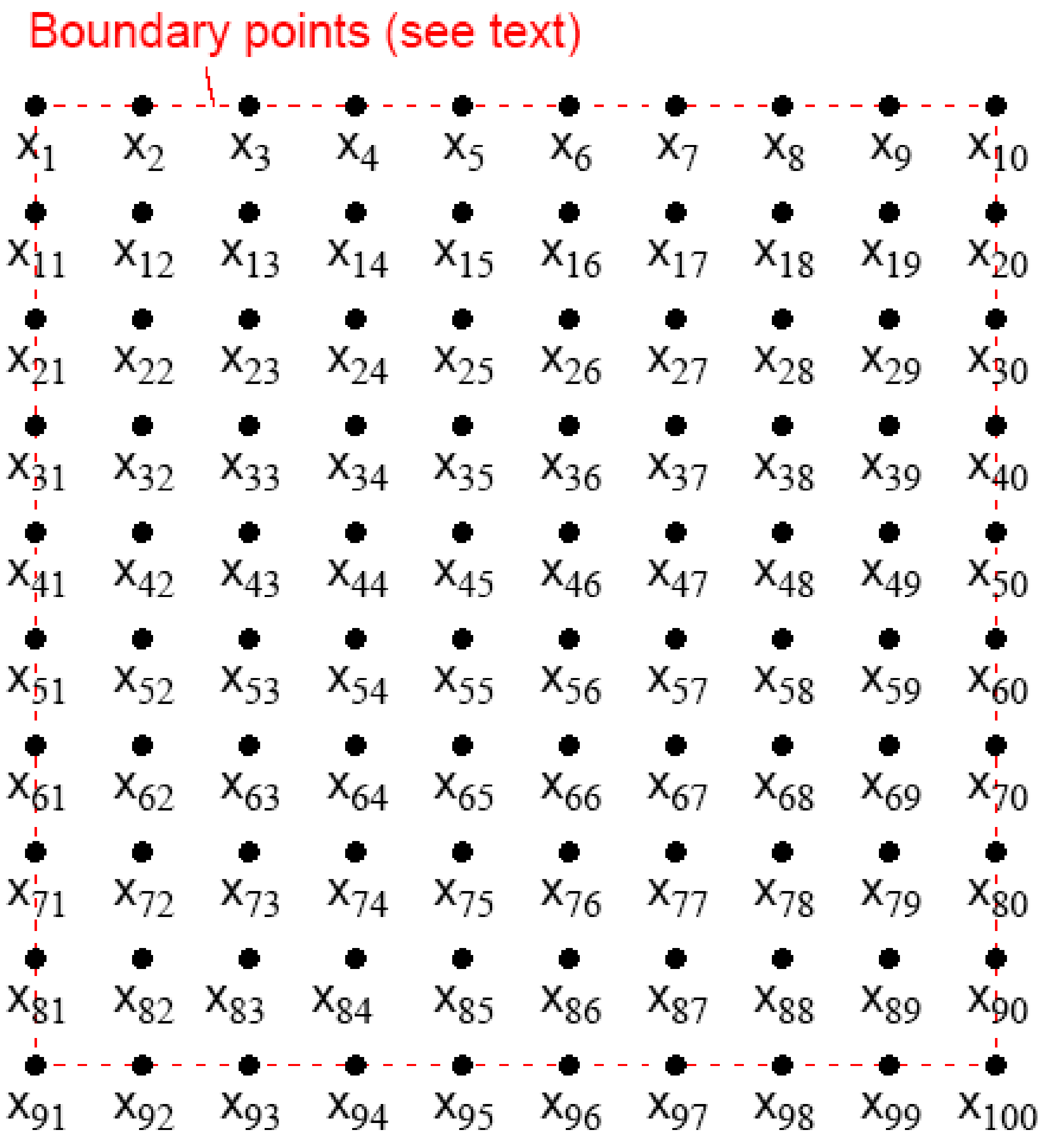
$$f(x, y) = \frac{[f(x-\Delta, y) + f(x, y-\Delta) + f(x+\Delta, y) + f(x, y+\Delta)]}{4}$$

Rewritten as an iterative formula:

$$f^k(x, y) = \frac{[f^{k-1}(x-\Delta, y) + f^{k-1}(x, y-\Delta) + f^{k-1}(x+\Delta, y) + f^{k-1}(x, y+\Delta)]}{4}$$

$f^k(x, y)$ - kth iteration, $f^{k-1}(x, y)$ - $(k-1)$th iteration.

Natural Order



Relationship with a General System of Linear Equations

Using natural ordering, *i*th point computed from *i*th equation:

$$x_i = \frac{x_{i-n} + x_{i-1} + x_{i+1} + x_{i+n}}{4}$$

or

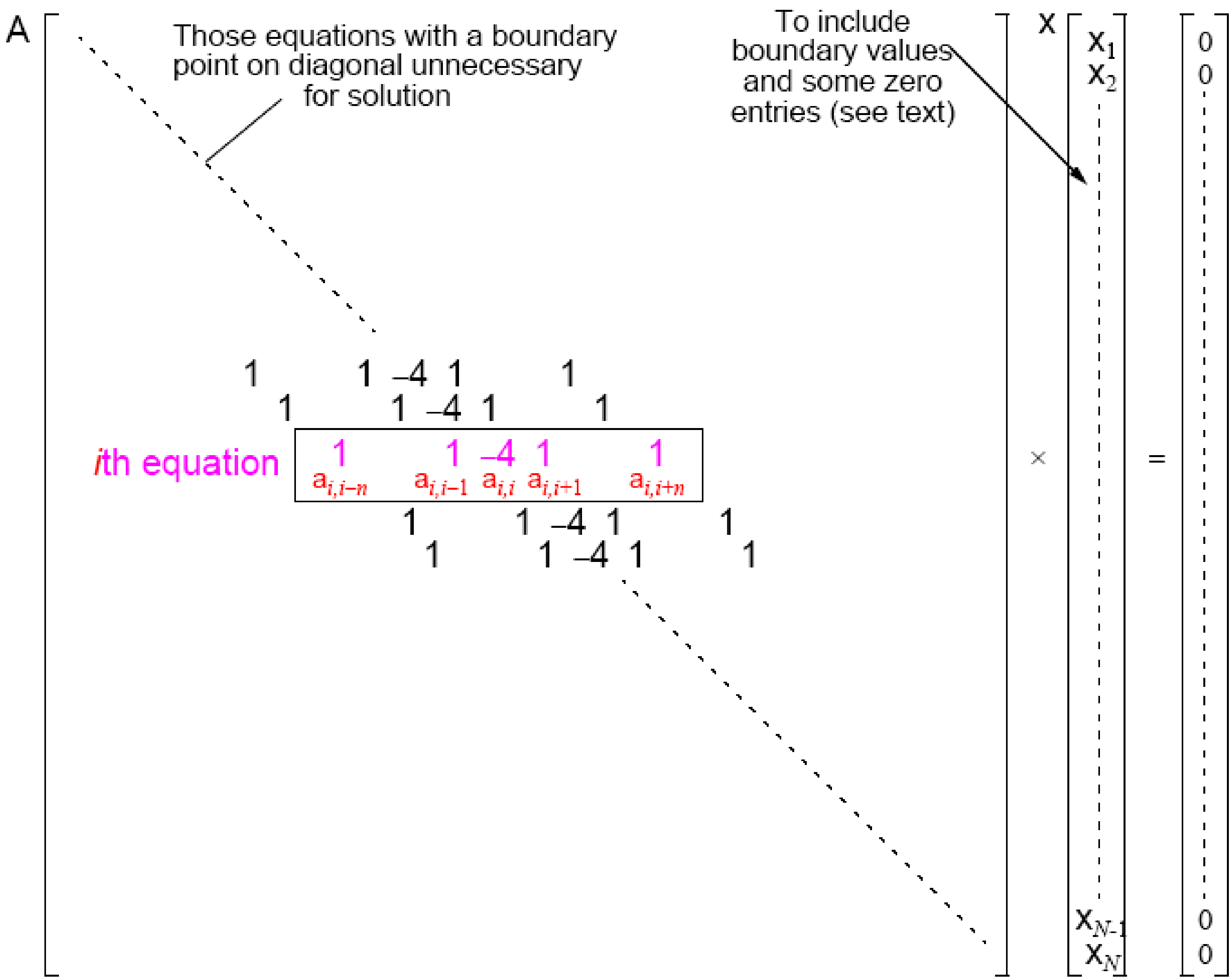
$$x_{i-n} + x_{i-1} - 4x_i + x_{i+1} + x_{i+n} = 0$$

which is a linear equation with five unknowns (except those with boundary points).

In general form, the *i*th equation becomes:

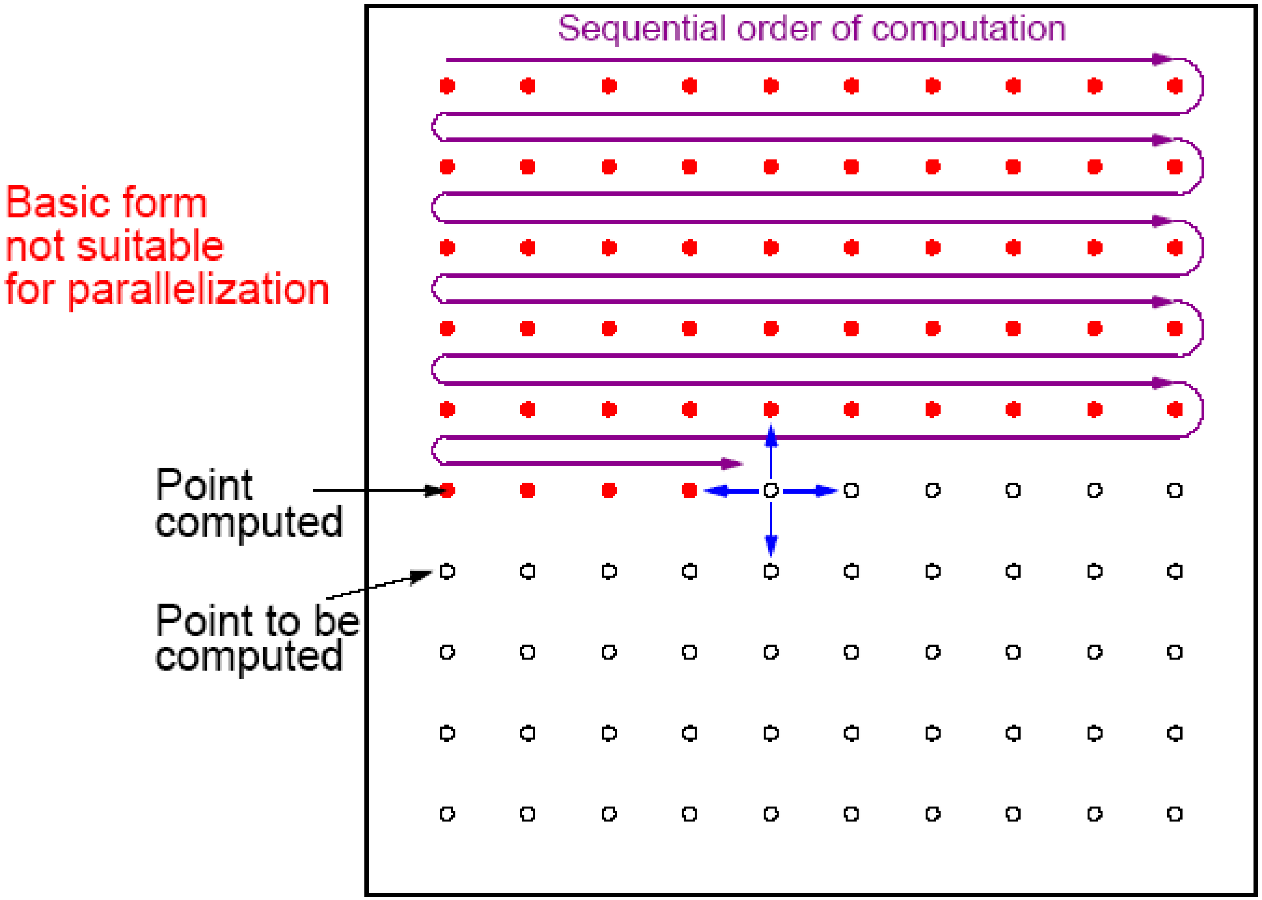
$$a_{i,i-n}x_{i-n} + a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} + a_{i,i+n}x_{i+n} = 0$$

where $a_{i,i} = -4$, and $a_{i,i-n} = a_{i,i-1} = a_{i,i+1} = a_{i,i+n} = 1$.


A
Those equations with a boundary point on diagonal unnecessary for solution
To include boundary values and some zero entries (see text)
x
x_1
x_2
x_{N-1}
x_N
0
0
0
0
1 1 –4 1 1
1 1 –4 1 1
ith equation
1 1 –4 1 1
$a_{i,i-n}$ $a_{i,i-1}$ $a_{i,i}$ $a_{i,i+1}$ $a_{i,i+n}$
1 1 –4 1 1
1 1 –4 1 1
×
=

Gauss-Seidel Relaxation

Uses some newly computed values to compute other values in that iteration.



Gauss-Seidel Iteration Formula

$$x_i^k = \frac{1}{a_{i,i}}\left[b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^k - \sum_{j=i+1}^{N} a_{i,j}x_j^{k-1}\right]$$

where the superscript indicates the iteration.

With natural ordering of unknowns, formula reduces to

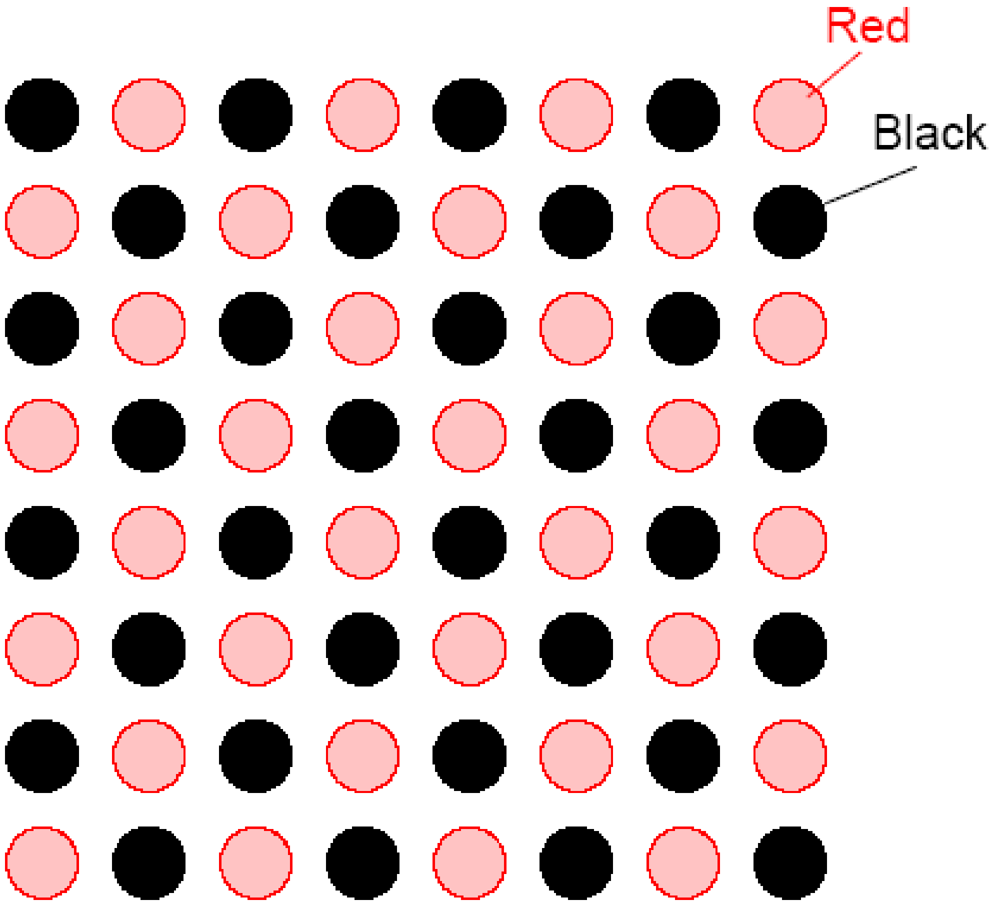
$$x_i^k = (-1/a_{i,i})[a_{i,i-n}x_{i-n}^k + a_{i,i-1}x_{i-1}^k + a_{i,i+1}x_{i+1}^{k-1} + a_{i,i+n}x_{i+n}^{k-1}]$$

At the *k*th iteration, two of the four values (before the *i*th element) taken from the *k*th iteration and two values (after the *i*th element) taken from the (*k*-1)th iteration. We have:

$$f^k(x,y) = \frac{[f^k(x-\Delta,y) + f^k(x,y-\Delta) + f^{k-1}(x+\Delta,y) + f^{k-1}(x,y+\Delta)]}{4}$$

Red-Black Ordering

First, black points computed. Next, red points computed. Black points computed simultaneously, and red points computed simultaneously.



Red-Black Parallel Code

```
forall (i = 1; i < n; i++)
    forall (j = 1; j < n; j++)
      if ((i + j) % 2 == 0)                 /* compute red points */
          f[i][j] = 0.25*(f[i-1][j] + f[i][j-1] + f[i+1][j] + f[i][j+1]);
forall (i = 1; i < n; i++)
   forall (j = 1; j < n; j++)
     if ((i + j) % 2 != 0)                 /* compute black points */
         f[i][j] = 0.25*(f[i-1][j] + f[i][j-1] + f[i+1][j] + f[i][j+1]);
```

Higher-Order Difference Methods

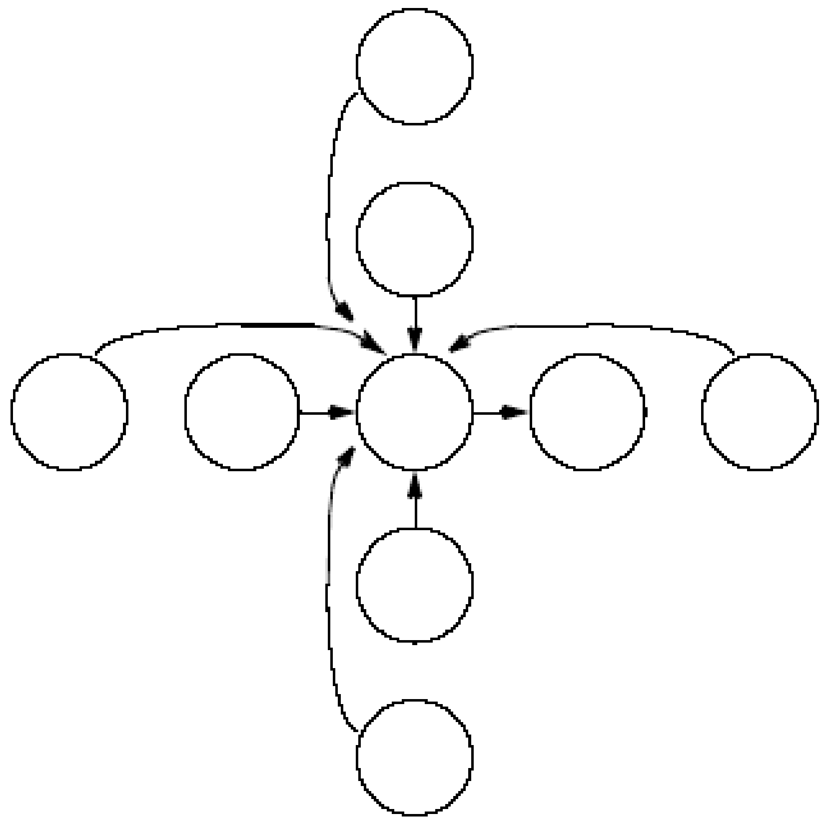
More distant points could be used in the computation. The following update formula:

$$f^{k}(x, y) =$$

$$\frac{1}{60}\Big[16 f^{k-1}(x-\Delta, y) + 16 f^{k-1}(x, y-\Delta) + 16 f^{k-1}(x+\Delta, y) + 16 f^{k-1}(x, y+\Delta).$$

$$- f^{k-1}(x-2\Delta, y) - f^{k-1}(x, y-2\Delta) - f^{k-1}(x+2\Delta, y) - f^{k-1}(x, y+2\Delta)\Big]$$

Nine-point stencil



Overrelaxation

Improved convergence obtained by adding factor $(1 - \omega)x_i$ to Jacobi or Gauss-Seidel formulae. Factor ω is the *overrelaxation parameter*.

Jacobi overrelaxation formula

$$x_i^k = \frac{\omega}{a_{ii}}\left[b_i - \sum_{j \neq i} a_{ij} x_i^{k-1}\right] + (1-\omega)x_i^{k-1}$$

where $0 < \omega < 1$.

Gauss-Seidel successive overrelaxation

$$x_i^k = \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_i^k - \sum_{j=i+1}^{N} a_{ij} x_i^{k-1}\right] + (1-\omega)x_i^{k-1}$$

where $0 < \omega \leq 2$. If ω = 1, we obtain the Gauss-Seidel method.