Chapter 7

Elliptic Equations and Steady State Problems

Typical example Poisson's equation:

$$\Delta\phi = \rho$$

In discretized form this generates an algebraic system such as

$$\underline{\underline{\mathbf{A}}}(\mathbf{V}) \cdot \mathbf{V} = \mathbf{B} \tag{7.1}$$

with

$$\underline{\underline{\mathbf{A}}} = \begin{pmatrix} a_{11} & a_{12} & . & . \\ a_{21} & . & . & \\ & & & a_{NN} \end{pmatrix}, \quad \mathbf{V} = \begin{pmatrix} V_1 \\ . \\ V_N \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_1 \\ . \\ B_N \end{pmatrix}$$

- For $\underline{\underline{\mathbf{A}}}$ independent of $\mathbf{V}$ then solve $\underline{\underline{\mathbf{A}}} \cdot \mathbf{V} = \mathbf{B}$ directly.
- For $\underline{\underline{\mathbf{A}}}$ dependent of $\mathbf{V}$ then solve $\underline{\underline{\mathbf{A}}}(\mathbf{V}) \cdot \mathbf{V} = \mathbf{B}$ through iterative methods (nonlinear problem!)
- $\underline{\underline{\mathbf{A}}}$ can be full or sparse and effort to solve (7.1) depends on the particular structure

7.1 Newtons Method

Considering the set of equations defined by (7.1) the residual is

$$\mathbf{R} = \underline{\underline{\mathbf{A}}}(\mathbf{V}) \cdot \mathbf{V} - \mathbf{B} \tag{7.2}$$

Goal: To find a solution of $\mathbf{R} = 0$.

One dimensional example to illustrate Newton's method to solve (7.1): Consider

$$r = a(x)x - b$$

to find the solution of $r = 0$ or $a(x)x = b$.

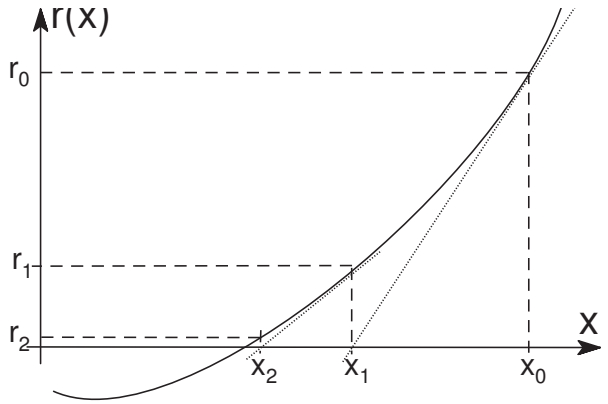


Figure 7.1: Illustration of the graphical solution with Newton's method.

From Figure 7.1 we find

$$-\frac{r(x_k)}{x_{k+1} - x_k} = \frac{dr}{dx}(x_k) = \left(\frac{dr}{dx}\right)_k$$

which yields the following iterative method to find the solution of $a(x)x = b$.

$$x_{k+1} = x_k - \left(\frac{dr}{dx}\right)_k^{-1} r_k$$

with $r_k = r(x_k)$.

Generalizing this iterative method to (7.1) with the residual (7.2) yields

$$\begin{aligned} \mathbf{V}^{(k+1)} &= \mathbf{V}^{(k)} - \left(\underline{\underline{\mathbf{J}}}^{(k)}\right)^{-1} \mathbf{R}^{(k)} \\ \text{with the Jacobian} \qquad & J_{ij} = \frac{\partial R_i}{\partial V_j} \end{aligned}$$

or equivalent one needs to solve

$$\underline{\underline{\mathbf{J}}}^{(k)} \Delta \mathbf{V}^{(k+1)} = -\mathbf{R}^{(k)}$$

For the maximum norms $\left\|\underline{\underline{\mathbf{J}}}\right\| = \max_i \sum_j \left|J_{ij}\right|$ and $\|\mathbf{V}\| = \max_i |V_i|$, Newton's method yields quadratic convergence

$$\left\|\mathbf{V}^{(k+1)} - \mathbf{V}_c\right\| = \left\|\mathbf{V}^{(k)} - \mathbf{V}_c\right\|^2$$

if the start value $\mathbf{V}^{(0)}$ is sufficiently close to to the actual solution $\mathbf{V}_c$. Convergence can be shown for the following conditions

$$\begin{aligned} \left\|\left(\underline{\underline{\mathbf{J}}}^{(0)}\right)^{-1}\right\| &\leq a \\ \left\|\mathbf{V}^{(1)} - \mathbf{V}^{(0)}\right\| = \left\|\left(\underline{\underline{\mathbf{J}}}^{(0)}\right)^{-1}\mathbf{R}^{(0)}\right\| &\leq b \\ \sum_i^N \left|\frac{\partial^2 \mathbf{R}^{(0)}}{\partial V_i \partial V_j}\right| &\leq \frac{c}{N} \end{aligned}$$

and $abc < 0.5$. Note that the radius of convergence for Newton's method decreases as N increases.

7.1.1 Example for Newton's method: Burgers equation

Consider the stationary two-dimensional momentum equations of the form

$$\begin{aligned} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{S}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) &= 0 \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{S}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) &= 0 \end{aligned}$$

with the velocity vector $\mathbf{v} = (u, v)$. These equations are also called the stationary Burgers equations. With the first and second derivative operators

$$\begin{aligned} L_x u_{j,k} &= \frac{u_{j+1,k} - u_{j-1,k}}{2\Delta x} \\ L_{xx} u_{j,k} &= \frac{1}{\Delta x^2}\left(u_{j+1,k} - 2u_{j,k} + u_{j-1,k}\right) \end{aligned}$$

the discretized equations are

$$\begin{aligned} Ru_{j,k} = u_{j,k}L_x u_{j,k} + v_{j,k}L_y u_{j,k} - \frac{1}{S}\left(L_{xx}u_{j,k} + L_{yy}u_{j,k}\right) &= 0 \\ Rv_{j,k} = u_{j,k}L_x v_{j,k} + v_{j,k}L_y v_{j,k} - \frac{1}{S}\left(L_{xx}v_{j,k} + L_{yy}v_{j,k}\right) &= 0 \end{aligned}$$

An exact solution can be obtained from the generating function

$$\phi = a_1 + a_2 x + a_3 y + a_4 xy + a_5 \left(e^{\lambda(x-x_0)} + e^{-\lambda(x-x_0)} \right) \cos(\lambda y)$$

with

$$\begin{aligned} u &= -\frac{2}{s\phi}\frac{\partial \phi}{\partial x} \\ v &= -\frac{2}{s\phi}\frac{\partial \phi}{\partial y} \end{aligned}$$

where a_1, a_2, a_3, a_4, and a_5 are constants which can be used to alter the exact solution. Boundary conditions for the numerical solution are taken directly from the exact solution. The dimension of the numerical domain is N_x in the x and N_y in the y direction. However boundary conditions are applied as Dirichlet conditions such that the actual x and y dimensions are $N_x - 2$ and $N_y - 2$. This leaves $(N_x - 2)(N_y - 2)$ values for u and the same number for v for a total of $2(N_x - 2)(N_y - 2)$ variables to be determined for the solution.

To cast the steady Burger's equation into the form (7.2) one needs to arrange the variables $u_{j,k}$ and $v_{j,k}$ into a vector $\mathbf{V}$. Here we use the following association:

$u_{2,2}$	$v_{2,2}$	$u_{2,3}$	$v_{2,3}$	..	$u_{3,2}$	$v_{3,2}$	..	u_{N_x-1,N_y-1}	v_{N_x-1,N_y-1}
V_1	V_2	V_3	V_4	..	$V_{2(N_y-2)+1}$	$V_{2(N_y-2)+2}$	..	$V_{2(N_x-2)(N_y-2)-1}$	$V_{2(N_x-2)(N_y-2)}$

and we can treat the elements $Ru_{j,k}$ and $Rv_{j,k}$ in the same manner to arrange them into the form $\mathbf{R}$. Here the index l of V_l is computed as $l = 2(j-2)(N_y-2) + 2(k-2) + 1$ for the variable u and $l = 2(j-2)(N_y-2) + 2(k-2) + 2$ for the component v.

It remains to compute the Jacobian which for the derivatives of $Ru_{k,l}$ is

$$\begin{aligned} \frac{\partial Ru_{j,k}}{\partial u_{j-1,k}} &= -\frac{1}{2\Delta x}u_{j,k} - \frac{1}{S\Delta x^2} \\ \frac{\partial Ru_{j,k}}{\partial u_{j,k}} &= \frac{1}{2\Delta x}\left(u_{j+1,k} - u_{j-1,k}\right) + \frac{2}{S}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) \\ \frac{\partial Ru_{j,k}}{\partial u_{j+1,k}} &= \frac{1}{2\Delta x}u_{j,k} - \frac{1}{S\Delta x^2} \\ \frac{\partial Ru_{j,k}}{\partial u_{j,k-1}} &= -\frac{1}{2\Delta y}v_{j,k} - \frac{1}{S\Delta y^2} \\ \frac{\partial Ru_{j,k}}{\partial u_{j,+1k}} &= \frac{1}{2\Delta y}v_{j,k} - \frac{1}{S\Delta y^2} \\ \frac{\partial Ru_{j,k}}{\partial v_{j,k}} &= \frac{1}{2\Delta y}\left(u_{j,k+1} - u_{j,k-1}\right) \end{aligned}$$

To apply Newtons method to Burger's equation requires to invert the Jacobian for each iteration step. Since the length of the vector V is of order $N = 2N_xN_y$ the Jacobian has the dimension $N \times N$. Thus depending on the structure of the Jacobian this inversion can be very computationally expensive and is feasible only for relatively small numbers of grid points.

Illustration and explanation of the program Newburger:

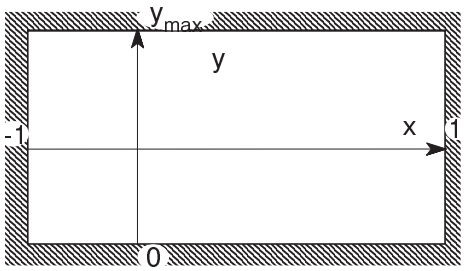


Figure 7.2: Illustration of the 2D domain for which Burgers equation is solved.

The program computes the solution to Burger's equation in a rectangular area with $N_x \times N_y$ grid points. Boundary conditions are Dirichlet conditions such that the boundary points do not need to be computed. The program needs 3 files to run:

Newburger.f - source code

newbin - include file with all variable declarations

newb.dat - input parameter file

The input parameters are set as:

```
400         itmx,max no of iterations
1           ird: 2-v,u from newb.sto; 1-v,u from initcon
10          ipn: iteration increment for print rms
.100e-04    eps: tolerance for rms error
.200e+00    S: reynoldsnumber
.300e-00    om: underrelaxation factor
.100e-01    dt: time step for pseudo transient version
110.13      coef a1 for exact solution
110.13      coef a2 for exact solution
0.0 coef    a3 for exact solution
0.0 coef    a4 for exact solution
1.0 coef    a5 for exact solution
5.0 coef    al for exact solution
```

The basic layout of the program is similar to other programs. The program is initiated by reading the program parameters and generating the grid and the start configuration. During the iterations the programs carries out the following main steps

- computation of the residual
- computation of the Jacobian
- factorization of the Jacobian
- solution (inversion) of the Jacobian and determination of the new values for u and v.

The iteration is terminated if the number of iterations steps reaches a maximum number of iteration steps or if the RMS error for the numerical solution becomes sufficiently small.

7.2 Direct Methods to Solve Linear Systems of Equations

7.2.1 Notes on the routines "fact" ad "sol":

Goal: Solution of

$$\underline{\underline{\mathbf{A}}} \cdot \mathbf{V} = \mathbf{B}$$

which requires to invert the matrix $\underline{\underline{\mathbf{A}}}$. Depending on the structure of $\underline{\underline{\mathbf{A}}}$ this can be a large or a smaller effort. One can distinguish three basic cases

- If $\underline{\underline{\mathbf{A}}}$ contains few or no elements which are 0 => $\underline{\underline{\mathbf{A}}}$ is called dense.
- If most elements of $\underline{\underline{\mathbf{A}}}$ are 0 => $\underline{\underline{\mathbf{A}}}$ is called sparse.
- If the nonzero elements of $\underline{\underline{\mathbf{A}}}$ are clustered around the diagonal => $\underline{\underline{\mathbf{A}}}$ is called banded.

In the present case $\underline{\underline{\mathbf{A}}}$ has many 0 elements but is not banded which requires a solution of the full system. This is achieved in two steps in which $\underline{\underline{\mathbf{A}}}$is factorized in a lower $\underline{\underline{\mathbf{L}}}$ and an upper diagonal matrix $\underline{\underline{\mathbf{U}}}$:

$$\underline{\underline{\mathbf{L}}} \cdot \underline{\underline{\mathbf{U}}} \cdot \mathbf{V} = \mathbf{B}$$

The first step in this process $\underline{\underline{\mathbf{L}}}$ is determined and the system of equations is brought into the form

$$\underline{\underline{\mathbf{U}}} \cdot \mathbf{V} = \underline{\underline{\mathbf{L}}}^{-1} \cdot \mathbf{B}$$

Example:

$$\underline{\underline{\mathbf{A}}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

The very first step in this procedure is to re-order rows in $\underline{\underline{\mathbf{A}}}$ such that the largest absolute values of the first element in the rows becomes the uppermost row.

Multiplying the first row with a_{21}/a_{11} and subtracting it from the second and similarly multiplying the first row with a_{31}/a_{11} and subtracting it from the third yields

$$\underline{\underline{\mathbf{A}}}' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} - \frac{a_{21}}{a_{11}} a_{12} & a_{23} - \frac{a_{21}}{a_{11}} a_{13} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

The manipulation of the second row can also be expressed in terms of a matrix multiplication:

$$\underline{\underline{\mathbf{A}}}' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} - \frac{a_{21}}{a_{11}} a_{12} & a_{23} - \frac{a_{21}}{a_{11}} a_{13} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \underline{\underline{\mathbf{T}}}_m \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$with \qquad \underline{\underline{\mathbf{T}}}_m = \begin{pmatrix} 1 & 0 & 0 \\ -m & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; m = \frac{a_{21}}{a_{11}}$$

After the second row the third row is manipulated in the same way and so forth. After the first step elements below the first row become

$$a'_{ij} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$$

In the k th step elements below the k th row become

$$a'_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj}$$

"solve":

The transformations which are applied to $\underline{\underline{\mathbf{A}}}$ are also applied to the vector $\mathbf{B}$, i.e., change of order of the elements in $\mathbf{B}$ if the order is changed among rows in $\underline{\underline{\mathbf{A}}}$ and multiplication with the matrix $\underline{\underline{\mathbf{T}}}_m$ when $\underline{\underline{\mathbf{A}}}$ is multiplied, e.g.,

$$\mathbf{B}' = \underline{\underline{\mathbf{T}}}_m \cdot \mathbf{B} = \begin{pmatrix} b_1 \\ b_2 - m b_1 \\ b_3 \end{pmatrix}$$

$$with \qquad \underline{\underline{\mathbf{T}}}_m = \begin{pmatrix} 1 & 0 & 0 \\ -m & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; m = \frac{a_{21}}{a_{11}}$$

After this first step elements below the first become

$$b_i' = b_i - \frac{a_{i1}}{a_{11}} b_1$$

and at the k th step the elements below the k th element become

$$b_i' = b_i - \frac{a_{ik}}{a_{kk}} b_1$$

Thus we have now transformed the equation into the structure

$$\begin{pmatrix} a_{11}' & a_{12}' & a_{13}' \\ 0 & a_{22}' & a_{23}' \\ 0 & 0 & a_{33}' \end{pmatrix} \cdot \mathbf{V} = \mathbf{B}'$$

The solution is now found through so-called backsubstitution starting from the last equation

- First: $a_{NN}' V_N = b_N' \;\Rightarrow\; V_N = b_N'/a_{NN}'$
- Second: $a_{N-1,N-1}' V_{N-1} + a_{N-1,N}' V_N = b_{N-1}' \;\Rightarrow\; V_{N-1} = \left(b_{N-1}' - a_{N-1,N}' V_N\right)/a_{N-1,N-1}'$
- and so forth.

Number of operations:

- Factorization $\sim N^3$
- Backsubstitution $\sim N^2$

7.2.2 Tridiagonal systems: Thomas algorithm and programs Banfac and Bansol

Factorization: Particularly suited for analytic solutions are banded matrices (here tridiagonal or pentadiagonal). Here a Gauss elimination has to be applied only to a small number of elements close to the diagonal. In the case of tridiagonal matrices such as the following

$$\begin{bmatrix} b_1 & c_1 & 0 & . & . & & & & \\ a_2 & b_2 & c_2 & 0 & . & & & & \\ 0 & a_3 & b_3 & c_3 & . & & & & \\ & & & . & . & . & . & & \\ & & & . & . & . & . & & \\ & & & & a_i & b_i & c_i & & \\ & & & & & . & . & . & \\ & & & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & & & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ v_i \\ . \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ . \\ . \\ d_i \\ . \\ d_{n-1} \\ d_n \end{bmatrix}$$

the matrix transformed into an upper diagonal matrix

$$\begin{bmatrix} b'_1 & c_1 & 0 & . & . & & & & \\ 0 & b'_2 & c_2 & 0 & . & & & & \\ 0 & 0 & b'_3 & c_3 & . & & & & \\ & & & . & . & . & . & & \\ & & & . & . & . & . & & \\ & & & & 0 & b'_i & c_i & & \\ & & & & & . & . & . & \\ & & & & & & 0 & b'_{n-1} & c_{n-1} \\ & & & & & & 0 & 0 & b'_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ v_i \\ . \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d'_1 \\ d'_2 \\ d'_3 \\ . \\ . \\ d'_i \\ . \\ d'_{n-1} \\ d'_n \end{bmatrix}$$

for the first equation no transformation is needed. The second equation uses the transformations

$$b'_2 = b_2 - c_1 \frac{a_2}{b_1} \qquad d'_2 = d_2 - d_1 \frac{a_2}{b_1}$$

This procedure can be applied recursively to all equations with the formula

$$b'_i = b_i - c_{i-1} \frac{a_i}{b'_{i-1}} \qquad d'_i = d_i - d'_{i-1} \frac{a_i}{b'_{i-1}}$$

The resulting matrix has only one row of elements in addition to the diagonal just above the diagonal elements.

Backsubstitution: The solution of the resulting set is straightforward. For equation n the solution is

$$v_n = d'_n / b'_n$$

The solution for all equations is found recursively going from index n to index 1 with the formula

$$v_i = \frac{d'_i - c_i v_{i+1}}{b'_i}.$$

This lead to the routines Banfac and Bansol for the solution of tridiagonal and pentadiagonal systems.

No of operations: $5N - 4$

Thomas algorithm realized in the routines Banfac and Bansol:

The routine Banfac uses the variables: b(5,n) which stores the matrix elements.

For input these are:

b(2,i)=a_i, b(3,i)=b_i, and b(4,i)=c_i

on output the variables are:

b(2,i)=a_i/b'_{i-1}, b(3,i)=b'_i, and b(4,i)=c_i

7.2.3 Pentadiagonal systems:

Higher order derivatives or quadratic finite elements generate a wider band for instance a pentadiagonal system.

$$\begin{bmatrix} b_1 & c_1 & f_1 & 0 & . & & & & \\ a_2 & b_2 & c_2 & f_2 & 0 & . & & & \\ e3 & a_3 & b_3 & c_3 & f_3 & . & & & \\ 0 & e_4 & a_4 & . & . & . & . & & \\ 0 & 0 & . & . & . & . & . & & \\ & & & e_i & a_i & b_i & c_i & f_i & \\ & & & & . & . & . & . & \\ & & & & & e_{n-1} & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & & 0 & e_n & a_n & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ v_i \\ . \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ . \\ . \\ d_i \\ . \\ d_{n-1} \\ d_n \end{bmatrix}$$

Here a first sweep eliminates the elements e_i in the system:

$$\begin{aligned} a'_i &= a_i - e_i b'_{i-1}/a'_{i-1} \\ b'_i &= b_i - e_i c'_{i-1}/a'_{i-1} \\ c'_i &= c_i - e_i f'_{i-1}/a'_{i-1} \\ f'_i &= f_i \\ d'_i &= d_i - e_i d'_{i-1}/a'_{i-1} \end{aligned}$$

and transforms the matrix transformed into

$$\begin{bmatrix} b'_1 & c'_1 & f'_1 & 0 & . & & & & \\ a'_2 & b'_2 & c'_2 & f'_2 & . & & & & \\ 0 & a'_3 & b'_3 & c'_3 & . & & & & \\ & & & . & . & . & . & & \\ & & & . & . & . & . & & \\ & & & 0 & a'_i & b'_i & c'_i & f'_i & \\ & & & & & . & . & . & \\ & & & & & 0 & a'_{n-1} & b'_{n-1} & c'_{n-1} \\ & & & & & & 0 & a'_n & b'_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ v_i \\ . \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d'_2 \\ d'_3 \\ . \\ . \\ d'_i \\ . \\ d'_{n-1} \\ d'_n \end{bmatrix}$$

The next sweep eliminates all a'_i terms where we choose to eliminate the b elements as well by first dividing the first row by b_1

$c''_1 = c'_1/b'_1$ and $f''_1 = f'_1/b'_1$ and so forth to yield

$$\begin{bmatrix} 1 & c_1'' & f_1' & 0 & . & & & & \\ 0 & 1 & c_2'' & f_2' & . & & & & \\ 0 & 0 & 1 & c_3'' & . & & & & \\ & & & . & . & . & . & & \\ & & & . & . & . & . & & \\ & & & 0 & 0 & 1 & c_i'' & f_i' & \\ & & & & & . & . & . & \\ & & & & & 0 & 0 & 1 & c_{n-1}'' \\ & & & & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ v_i \\ . \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2'' \\ d_3'' \\ . \\ . \\ d_i'' \\ . \\ d_{n-1}'' \\ d_n'' \end{bmatrix}$$

where

$$\begin{aligned} c_i'' &= \frac{c_i' - a_i' f_{i-1}''}{b_i' - a_i' c_{i-1}''} \\ f_i'' &= \frac{f_i'}{b_i' - a_i' c_{i-1}''} \\ d_i'' &= \frac{d_i' - a_i' d_{i-1}''}{b_i' - a_i' c_{i-1}''} \end{aligned}$$

Here the division of the double primed elements arises from eliminating the b'' terms by dividing the through them. The final solution is obtained again through back-substitution

$$v_i = d_i'' - c_i'' v_{i+1} - f_i'' v_{i+2}$$

Compared to the Thomas algorithm for tridiagonal matrices the pentadiagonal system requires two sweeps and one back-substitution. The method can be generalized to reduce a matrix $\underline{\underline{\mathbf{A}}}$ to its upper triangular form. However, a wider the band of a banded matrix requires more effort than narrowly banded matrices.

7.2.4 Block Tridiagonal systems

The Thomas algorithm can also be applied to a block tridiagonal system where the individual elements in a matrix are blocks of matrices.

$$\begin{bmatrix} \underline{\underline{\mathbf{b}}}_1 & \underline{\underline{\mathbf{c}}}_1 & 0 & . & . & & & & \\ \underline{\underline{\mathbf{a}}}_2 & \underline{\underline{\mathbf{b}}}_2 & \underline{\underline{\mathbf{c}}}_2 & 0 & . & & & & \\ 0 & \underline{\underline{\mathbf{a}}}_3 & \underline{\underline{\mathbf{b}}}_3 & \underline{\underline{\mathbf{c}}}_3 & . & & & & \\ & & & . & . & . & . & & \\ & & & . & . & . & . & & \\ & & & & \underline{\underline{\mathbf{a}}}_i & \underline{\underline{\mathbf{b}}}_i & \underline{\underline{\mathbf{c}}}_i & & \\ & & & & & . & . & . & \\ & & & & & & \underline{\underline{\mathbf{a}}}_{n-1} & \underline{\underline{\mathbf{b}}}_{n-1} & \underline{\underline{\mathbf{c}}}_{n-1} \\ & & & & & & 0 & \underline{\underline{\mathbf{a}}}_n & \underline{\underline{\mathbf{b}}}_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ . \\ . \\ \mathbf{v}_i \\ . \\ \mathbf{v}_{n-1} \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \\ . \\ . \\ \mathbf{d}_i \\ . \\ \mathbf{d}_{n-1} \\ \mathbf{d}_n \end{bmatrix}$$

The procedure to solve this system is equivalent to the normal Thomas algorithm using the appropriate matrix operations. No transformation is needed for the first row of matrix elements (first equation). The second row uses the transformations

$$\underline{\underline{\mathbf{b}}}'_2 = \underline{\underline{\mathbf{b}}}_2 - \underline{\underline{\mathbf{a}}}_2 \cdot \left(\underline{\underline{\mathbf{b}}}_1\right)^{-1} \cdot \underline{\underline{\mathbf{c}}}_1 \qquad \mathbf{d}'_2 = \mathbf{d}_2 - \underline{\underline{\mathbf{a}}}_2 \cdot \left(\underline{\underline{\mathbf{b}}}_1\right)^{-1} \cdot \mathbf{d}_1$$

This procedure is applied recursively to all rows with the formula

$$\underline{\underline{\mathbf{b}}}'_i = \underline{\underline{\mathbf{b}}}_i - \underline{\underline{\mathbf{a}}}_i \cdot \left(\underline{\underline{\mathbf{b}}}_{i-1}\right)^{-1} \cdot \underline{\underline{\mathbf{c}}}_{i-1} \qquad \mathbf{d}'_i = \mathbf{d}_i - \underline{\underline{\mathbf{a}}}_i \cdot \left(\underline{\underline{\mathbf{b}}}_{i-1}\right)^{-1} \cdot \mathbf{d}'_{i-1}$$

The resulting matrix has only one row of elements in addition to the diagonal just above the diagonal elements.

Backsubstitution:

The solution of the resulting set is straightforward. For equation n the solution is

$$\mathbf{v}_n = \left(\underline{\underline{\mathbf{b}}}'_n\right)^{-1} \cdot \mathbf{d}'_n$$

The solution for all equations is found recursively going from index n to index 1 with the formula

$$\mathbf{v}_i = \left(\underline{\underline{\mathbf{b}}}'_i\right)^{-1} \cdot \left(\mathbf{d}'_n - \underline{\underline{\mathbf{c}}}_i \cdot \mathbf{v}_{i+1}\right)$$

7.3 Iterative methods

7.3.1 Basic approach and simple iteration methods

Iterative methods use a complementary approach to solve the equation

$$\underline{\underline{\mathbf{A}}} \cdot \mathbf{V} = \mathbf{B}$$

While direct methods attempt to solve the system of equations exact through Gauss elimination iterative methods attempt to find a solution through a promising iteration scheme. A basic consideration for iterative methods is the following. Matrix $\underline{\underline{\mathbf{A}}}$ is split into matrices $\underline{\underline{\mathbf{N}}}$ and $\underline{\underline{\mathbf{P}}}$ where $\underline{\underline{\mathbf{N}}}$ is in some fashion close to $\underline{\underline{\mathbf{A}}}$ (for instance containing the same diagonal elements) but easier to factorize.

$$\left(\underline{\underline{\mathbf{N}}} - \underline{\underline{\mathbf{P}}}\right) \cdot \mathbf{V} = \mathbf{B}$$

i.e., $\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{N}}} - \underline{\underline{\mathbf{P}}}$. This suggests an iteration scheme

$$\mathbf{V}^{(n+1)} = \underline{\underline{\mathbf{N}}}^{-1} \left(\underline{\underline{\mathbf{P}}} \cdot \mathbf{V}^{(n)} + \mathbf{B}\right)$$

One can expressalso using the residual value. Defining the residual error at iteration level n as

$$\mathbf{R}^{(n)} = \left(\underline{\underline{\mathbf{N}}} - \underline{\underline{\mathbf{P}}}\right) \cdot \mathbf{V}^{(n)} - \mathbf{B} = \underline{\underline{\mathbf{N}}} \cdot \mathbf{V}^{(n)} - \left(\underline{\underline{\mathbf{P}}} \cdot \mathbf{V}^{(n)} + \mathbf{B}\right)$$

or

$$\underline{\underline{\mathbf{P}}} \cdot \mathbf{V}^{(n)} + \mathbf{B} = \underline{\underline{\mathbf{N}}} \cdot \mathbf{V}^{(n)} - \mathbf{R}^{(n)}$$

the iteration can also be written as

$$\mathbf{V}^{(n+1)} = \mathbf{V}^{(n)} - \underline{\underline{\mathbf{N}}}^{-1} \cdot \mathbf{R}^{(n)}$$

Convergence is achieved if the residual error $\left\|\mathbf{R}^{(\mathbf{n})}\right\|$ approaches 0 for large n. Convergence can be expected if the spectral radius (magnitude of max. Eigenvalue of $\underline{\underline{\mathbf{N}}}^{-1} \cdot \underline{\underline{\mathbf{P}}}$) is less than unity. This is typically the case if matrix $\underline{\underline{\mathbf{A}}}$ is diagonally dominated.

Jacobi method

Here $\underline{\underline{\mathbf{N}}}$ is taken as the diagonal of $\underline{\underline{\mathbf{A}}}$ and $\underline{\underline{\mathbf{P}}}$ contains the remain elements of $\underline{\underline{\mathbf{A}}}$

$$\underline{\underline{\mathbf{N}}} = \underline{\underline{\mathbf{D}}}_{IA} \qquad \underline{\underline{\mathbf{P}}} = -\left(\underline{\underline{\mathbf{L}}}_{A} + \underline{\underline{\mathbf{U}}}_{A}\right)$$

such that the iteration yields

$$\underline{\underline{\mathbf{D}}}_{IA} \cdot \mathbf{V}^{(n+1)} = \mathbf{B} - \left(\underline{\underline{\mathbf{L}}}_{A} + \underline{\underline{\mathbf{U}}}_{A}\right) \cdot \mathbf{V}^{(n)}$$

In components

$$a_{ii} V_i^{(n+1)} = b_i - \sum_{j=1}^{i-1} a_{ij} V_j^{(n)} - \sum_{j=i+1}^{N} a_{ij} V_j^{(n)}$$

Explicitly this yields the simple iteration

$$V_i^{(n+1)} = \left(b_i - \sum_{j \neq i}^{N} a_{ij} V_j^{(n)} \right) / a_{ii}$$

While simple the Jacobi method yields rather slow convergence, it is simple and provides an iteration method that can easily be vectorized or parallelized.

Gauss-Seidel method

Here the idea is to use updated values of $V_i^{(n+1)}$ as soon as they are available. Formally one chooses

$$\underline{\underline{\mathbf{N}}} = \underline{\underline{\mathbf{D}}}_{IA} + \underline{\underline{\mathbf{L}}}_{A} \qquad \underline{\underline{\mathbf{P}}} = -\underline{\underline{\mathbf{U}}}_{A}$$

which yields for the iteration

$$\left(\underline{\underline{\mathbf{D}}}_{IA} + \underline{\underline{\mathbf{L}}}_{A}\right) \cdot \mathbf{V}^{(n+1)} = \mathbf{B} - \underline{\underline{\mathbf{U}}}_{A} \cdot \mathbf{V}^{(n)}$$

or

$$a_{ii} V_i^{(n+1)} + \sum_{j=1}^{i-1} a_{ij} V_j^{(n+1)} = b_i - \sum_{j=i+1}^{N} a_{ij} V_j^{(n)}$$

Since the $V_j^{(n+1)}$ with $j < i$ have already been computed one can move the sum over the $V_j^{(n+1)}$ to the right side of the equation yielding

$$V_i^{(n+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} V_j^{(n+1)} - \sum_{j=i+1}^{N} a_{ij} V_j^{(n)} \right) / a_{ii}$$

It is important to note that here every iteration step depends on the prior step and is therefore recursive. Because of this it is not possible to parallelize the GS method. However, GS iteration is about twice as fast as the Jacobi method.

Successive Overrelaxation (SOR)

This methods uses the Gauss-Seidel iteration but computes a weighted average for the new iteration level, i.e.,

$$V_i^{(n+1)} = \lambda \left[\left(b_i - \sum_{j=1}^{i-1} a_{ij} V_j^{(n+1)} - \sum_{j=i+1}^{N} a_{ij} V_j^{(n)} \right) / a_{ii} \right] + (1-\lambda) V_j^{(n)}$$

Another way to express this iteration is

$$\begin{aligned} V_i^{(*)} &= \left(b_i - \sum_{j=1}^{i-1} a_{ij} V_j^{(n+1)} - \sum_{j=i+1}^{N} a_{ij} V_j^{(n)} \right) / a_{ii} \\ V_i^{(n+1)} &= V_j^{(n)} + \lambda \left(V_i^{(*)} - V_j^{(n)} \right) \end{aligned}$$

Formally, SOR is expressed through

$$\underline{\underline{\mathbf{N}}} = \frac{1}{\lambda} \underline{\underline{\mathbf{D}}}_{IA} + \underline{\underline{\mathbf{L}}}_{A} \qquad \underline{\underline{\mathbf{P}}} = \frac{\lambda - 1}{\lambda} \underline{\underline{\mathbf{D}}}_{IA} - \underline{\underline{\mathbf{U}}}_{A}$$

Convergence of SOR depends on the choice of λ. Note that for $\lambda = 1$ SOR is identical to the GS method. faster iteration implies a choice of $\lambda > 1$. The optimum choice is given by

$$\lambda_{opt} = \frac{2}{1 + \sqrt{1 - \mu^2}} \qquad \mu \text{ largest EV of } \underline{\underline{\mathbf{1}}} - \underline{\underline{\mathbf{D}}}_{IA}^{-1} \cdot \underline{\underline{\mathbf{A}}}$$

although the computation of μ represents an increase in computational effort as well.

- Similar to GS, SOR is not suited to parallelization in this form.
- For reasonable choices of λ, SOR is faster than GS iteration.

Note that ordinary SOR cannot be used to apply acceleration techniques. However, with some modification one can generate a symmetric (SSOR) scheme where it is possible to apply acceleration techniques. SSOR iteration is less efficient than SOR.

7.4 Duct flow with iterative methods

Consider the equation we derived for viscous flow through a rectangular duct.

$$\left(\frac{b}{a}\right)^2 \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + 1 = 0$$

with $w = 0$ at $x = \pm 1$ and $y = \pm 1$. Considering a three-point finite difference discretization the algebraic system becomes

$$\left(\frac{b}{a}\right)^2 \frac{1}{\Delta x^2}\left(w_{j-1,k} - 2w_{j,k} + w_{j+1,k}\right) + \frac{1}{\Delta y^2}\left(w_{j,k-1} - 2w_{j,k} + w_{j,k+1}\right) + 1 = 0$$

For the different iteration methods we obtain the following discretization

a) Jacobi

$$\begin{aligned} w_{j,k}^{(n+1)} &= \frac{1}{2c_0}\left[1 + \left(\frac{b}{a}\right)^2 \frac{w_{j-1,k}^{(n)} + w_{j+1,k}^{(n)}}{\Delta x^2} + \frac{w_{j,k-1}^{(n)} + w_{j,k+1}^{(n)}}{\Delta y^2}\right] \\ \text{with} \qquad c_0 &= \left(\frac{b}{a}\right)^2 \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \end{aligned}$$

b) Gauss-Seidel

$$w_{j,k}^{(n+1)} = \frac{1}{2c_0}\left[1 + \left(\frac{b}{a}\right)^2 \frac{w_{j-1,k}^{(n+1)} + w_{j+1,k}^{(n)}}{\Delta x^2} + \frac{w_{j,k-1}^{(n+1)} + w_{j,k+1}^{(n)}}{\Delta y^2}\right]$$

c) SOR

Here the GS solution is called $w^{(*)}$ and use

$$w_{j,k}^{(n+1)} = w_{j,k}^{(n)} + \lambda\left(w^{(*)} - w_{j,k}^{(n)}\right) \tag{7.3}$$

The number of iterations as a function of the value of λ for finite differences and linear finite elements for the duct problem with 11×11 grid points is shown in table 7.1. Basically the two methods (fdm and fem) are well comparable with a slight advantage for the finite element approach.

The table identifies the Jacobi method, Gauss-Seidel corresponds to $\lambda = 1$, and the other values of λ correspond to successive over relaxation ($\lambda > 1$) or under relaxation ($\lambda < 1$). It demonstrates that the Jacobi method is clearly the slowest iteration scheme. Gauss-Seidel iteration is faster but still relatively slow when compared to the best results for SOR. Another property for the finite difference methods is shown in Figure 7.3. The result confirms the numbers from the table demonstrating the by far fastest convergence for SOR.

Table 7.1: Number iterations for convergence ($b/a = 1.0$) for tolerance 10^{-6}

λ	FDM	FEM	ADI-FDM	ADI-FEM	λ_y
Jacobi	87	64	19	19	0.9
0.8	74	55	16	18	1.1
1.0	51	38	12	16	1.7
1.2	36	26	12	14	2.5
1.3	29	21	12	13	2.8
1.4	23	17	11	14	3.3
1.5	18	12	11	15	3.7
1.55	15	13	11	16	4.1
1.6	17	15	12	17	4.5

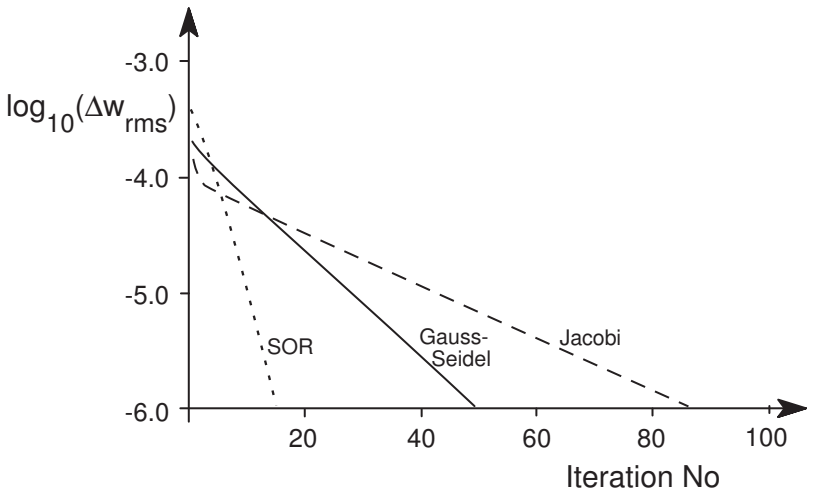


Figure 7.3: Convergence history for point finite difference methods for three basic iteration schemes.

d) Alternating Direction Implicit (ADI) method

All methods thus far have been explicit iteration methods. However, there are advantages to explore implicit formulations such as

$$\left(\frac{b}{a}\right)^2 \frac{1}{\Delta x^2}\left(-w^{(*)}_{j-1,k} + 2w^{(*)}_{j,k} - w^{(*)}_{j+1,k}\right) = 1 + \frac{1}{\Delta y^2}\left(w^{(n)}_{j,k-1} - 2w^{(n)}_{j,k} + w^{(n)}_{j,k+1}\right)$$

This is an implicit expression for $w^{(*)}_{j,k}$, however, since it involves only elements with indices $j-1$, j, $j+1$ one can consider a solution which for fixed k only involves the inversion of a tridiagonal matrix. Using the Thomas algorithm this system can be solved in a very efficient manner. This approach paired with (7.3) is called successive line over relation.

The approach above is also used in the alternating direction implicit (ADI) method. One such approach is expressed by the following sequence of iterations

$$
\begin{aligned}
(1) \qquad w_{j,k}^{(*)} - w_{j,k}^{(n)} &= \lambda \left(1 + \left(\frac{b}{a} \right)^2 L_{xx} w_{j,k}^{(*)} + L_{yy} w_{j,k}^{(n)} \right) \\
(2) \qquad w_{j,k}^{(n+1)} - w_{j,k}^{(*)} &= \lambda \left(1 + \left(\frac{b}{a} \right)^2 L_{xx} w_{j,k}^{(n+1)} + L_{yy} w_{j,k}^{(*)} \right)
\end{aligned}
$$

Note however that in this cases the iteration parameter λ is expected to depend strongly on the grid scale. To rmove this dependence we define

$$
\lambda = \lambda_y \Delta_y^2 = \lambda_x \left(\frac{b}{a} \right)^{-2} \Delta_x^2
$$

.

Re-arranging the explcit equations yields the following two step iteration procedure

1. step for fixed k

$$
-\lambda_x w_{j-1,k}^{(*)} + (1 + 2\lambda_x) w_{j,k}^{(*)} - \lambda_x w_{j+1,k}^{(*)} = \lambda_y \Delta y^2 + \lambda_y w_{j,k-1}^{(n)} + (1 - 2\lambda_y) w_{j,k}^{(n)} + \lambda_y w_{j,k+1}^{(n)} \tag{7.4}
$$

2. step for fixed j

$$
-\lambda_y w_{j,k-1}^{(n+1)} + (1 + 2\lambda_y) w_{j,k}^{(n+1)} - \lambda_y w_{j,k+1}^{(n+1)} = \lambda_y \Delta y^2 + \lambda_x w_{j-1,k}^{(*)} + (1 - 2\lambda_x) w_{j,k}^{(*)} - \lambda_x w_{j+1,k}^{(*)} \tag{7.5}
$$

It is noted that the ADI iteration is very similar to the ADI method for two-dimensional diffusion equation. The number of iterations for the ADI fem and fdm methods is also listed in table 7.1 which shows that the ADI methods converge faster than the corresponding regular fem and fdm methods.

Convergence is also a strong function of grid refinement:

Table 7.2: Grid refinement and convergence for the duct problem and SOR (regular fdm).

Grid	Optim. λ	No iterations for convergence
6×6	1.30	12
11×11	1.55	23
21×21	1.74	41
41×41	1.86	79

- There are various acceleration schemes, however, applied to Jacobi or Gauss-Seidel schemes no acceleration technique yields faster relaxation than SOR with the optimum choice of λ.
- For iteration techniques to be effective, the problem should be strongly elliptic (no presence of first order derivatives).
- If significant first order derivative are present iteration can fail or be rather slow (may require $\lambda < 1$).

Related methods and techniques:

Strongly Implicit Procedure (SIP): This method form of $\underline{\underline{\mathbf{N}}} = \underline{\underline{\mathbf{L}}} \cdot \underline{\underline{\mathbf{U}}}$ which generates a strongly implicit system of equation for the iteration. This solve by making use of the specific form of $\underline{\underline{\mathbf{N}}}$. Fast and robust.

Acceleration techniques (conjugate gradients): Manipulation of the direction of the iteration. If the residual $\mathbf{R}$ is expanded in terms of eigenvectors of $\underline{\underline{\mathbf{A}}}$ the conjugate gradient method eliminates the contribution from one eigenvector at each iteration step.

7.5 Pseudotransient method

A conceptually rather simple approach to the solution of elliptic PDE's is the pseudo transient method. This method constructs an equivalent time-dependent problem and integrates in time until the stationary solution is reached

Instead of solving

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

we can solve

$$\frac{\partial v}{\partial t} = \alpha \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right)$$

Discretization:

$$\begin{aligned} v_{jk}^{n+1} &= (1-4s)\, v_{jk}^n + s \left(v_{j,k-1}^n + v_{j,k+1}^n + v_{j-1,k}^n + v_{j+1,k}^n \right) \\ with \qquad & s = \frac{\alpha \Delta t}{\Delta x^2} \end{aligned}$$

For $s = 1/4$ this is equivalent to the Jacobi method. Note that one can multiply the right side with any choice $c\,(x,y)$ which may speed up the relaxation for instance in certain regions of the domain.

Example: 2D, steady Burger's equation:

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{S}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) &= 0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{S}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) &= 0 \end{aligned}$$

Discretization:

$$\begin{aligned} \frac{u_{jk}^{n+1}-u_{jk}^{n}}{\Delta t}+Ru_{jk}^{n+1} &= 0 \\ \frac{v_{jk}^{n+1}-v_{jk}^{n}}{\Delta t}+Rv_{jk}^{n+1} &= 0 \end{aligned}$$

Expanding residuals Ru_{jk}^{n+1} andRv_{jk}^{n+1} at time level n yields

$$\begin{aligned} \frac{\Delta u_{jk}^{n+1}}{\Delta t}+Ru_{jk}^{n+1}+\left(\frac{\partial Ru_{jk}}{\partial q_{lm}}\right)^{n}\Delta q_{lm} &= 0 \\ \frac{\Delta v_{jk}^{n+1}}{\Delta t}+Rv_{jk}^{n+1}\left(\frac{\partial Rv_{jk}}{\partial q_{lm}}\right)^{n}\Delta q_{lm} &= 0 \end{aligned}$$

with $\Delta u_{jk}^{n+1}=u_{jk}^{n+1}-u_{jk}^{n}$. Here q_{lm} includes all terms (u and v) which yield nonzero derivative of Ru_{jk} and Rv_{jk}. Equations can be written as

$$\left(\frac{1}{\Delta t}\underline{\underline{\mathbf{1}}}+\underline{\underline{\mathbf{J}}}\right)\Delta\mathbf{q}=-\mathbf{R}$$

which can be solved with the usual Newton's method. Here Δt serves as parameter which can be used to make the Jacobian more diagonally dominated and thus increase the radius of convergence which is usually small for systems which contain convective derivatives.

7.6 Multigrid Method

The basic idea behind multigrid methods is to have a series of grid resolutions for the system with $\Delta_{m+1}/\Delta_m = 1/2$ such that high values of m correspond to a fine grid and low m to low resolution.

For each grid level one has to solve a system

$$\underline{\underline{\mathbf{A}}}^{m}\cdot\mathbf{V}^{m}=\mathbf{B}^{m} \tag{7.6}$$

Note that different from the notation above the upper index m corresponds to a particular grid resolution rather than to an iteration level. Here a solution for $\mathbf{V}^m$ can be used as an approximation for $\mathbf{V}^{m+1}$ or for that matter also as an approximation for $\mathbf{V}^{m-1}$.

Consider an approximate solution was obtained from level $m+1$ or $m-1$ and is denoted $\mathbf{V}^{m,a}$. The solution on grid level m can be written as

$$\mathbf{V}^{m}=\mathbf{V}^{m,a}+\mathbf{W}^{m}$$

where $\mathbf{W}^m$ is the required correction to $\mathbf{V}^{m,a}$ to obtain a solution we can define a residual on level m as

$$\underline{\underline{\mathbf{A}}}^m \cdot \mathbf{W}^m = \mathbf{B}^m - \underline{\underline{\mathbf{A}}}^m \cdot \mathbf{V}^{m,a} = \mathbf{R}^m \tag{7.7}$$

Approach: Solution is found through iteration at various grid levels going

- first from fine to coarse
- then from coarse to fine

Motivation: Typically relaxation methods yield a fast relaxation on the highest frequencies or spectral components. The mathematical reason aside physical intuition suggest this behavior. Some of the basic iteration schemes for elliptic equations (Poisson equation) are actually identical to the explicit solution of a corresponding (time-dependent) diffusion equation. Even in more complex situations it is often possible to identify the iteration number in some way with time in a diffusion equation. For the diffusion equation the evolution is fastest on the smallest spatial (grid) scale. For instance the discretization of the diffusion equation yields $\Delta t = \alpha s/\Delta x^2$ (with $s = const$). The required time resolution in explicit schemes needs to resolve the diffusion time on the grid scale. Therefore relaxation occurs first on the smallest scales corresponding to the largest k vector of the spectral components. This is the basis of the multigrid approach. For each grid level only a few relaxation steps are carried out. Going to a coarse grid implies that relaxation on larger physical scales is much better. This can then be a basis (through interpolation) for the finer grid resolution on which relaxation of the finest physical scales is achieved.

For any grid level only a few relaxation steps ν are sufficient to smooth the high frequency components (high for the given grid level).

$$\begin{aligned} \mathbf{W}^{m,\nu} &= Relax^{\nu}\left(\mathbf{W}^m, \underline{\underline{\mathbf{A}}}^m, \mathbf{R}^m\right) \\ \mathbf{R}^{m,\nu} &= \mathbf{R}^m - \underline{\underline{\mathbf{A}}}^m \cdot \mathbf{W}^{m,\nu} \end{aligned}$$

Here $\mathbf{W}^{m,\nu}$ is the correction achieved after ν iteration steps and $\mathbf{R}^{m,\nu}$ is the corresponding residual.

The cycle starts with an initial approximation $\mathbf{V}^{M,a}$ on the finest grid. Here one can choose $\mathbf{W}^m$ and $\mathbf{R}^m$ either according to (7.7) or as

$$\mathbf{W}^m = \mathbf{V}^{M,a}, \qquad \mathbf{R}^m = \mathbf{B}^M$$

with $m = M$ and solve $\underline{\underline{\mathbf{A}}}^m \cdot \mathbf{W}^m = \mathbf{R}^m$ which is equivalent to (7.6). The residual on the next coarser grid level $m-1$ is

$$\mathbf{R}^{m-1} = \mathbf{I}_m^{m-1} \cdot \mathbf{R}^{m,\nu}$$

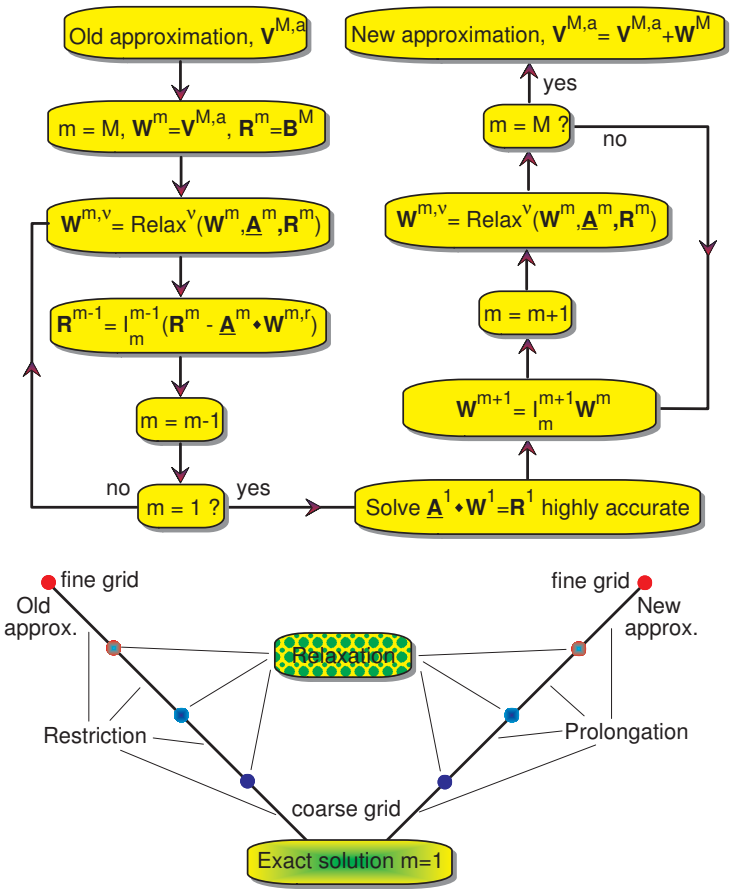


Figure 7.4: So-called V-cycle for the application of the multigrid method.

where $\mathbf{I}_m^{m-1}$ is a restriction operator:

$$\mathbf{I}_m^{m-1} = \frac{1}{16}\begin{pmatrix} 0 & 2 & 0 \\ 2 & 8 & 2 \\ 0 & 2 & 0 \end{pmatrix} \quad \text{or} \quad \mathbf{I}_m^{m-1} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

On the $m-1$ level the iteration is repeated for ν steps after which one switches again to the next coarser grid. The sequence is carried out up to the point where the coarsest grid is reached. For the coarsest grid the iteration is continued until a highly accurate solution is found on the coarsest grid.

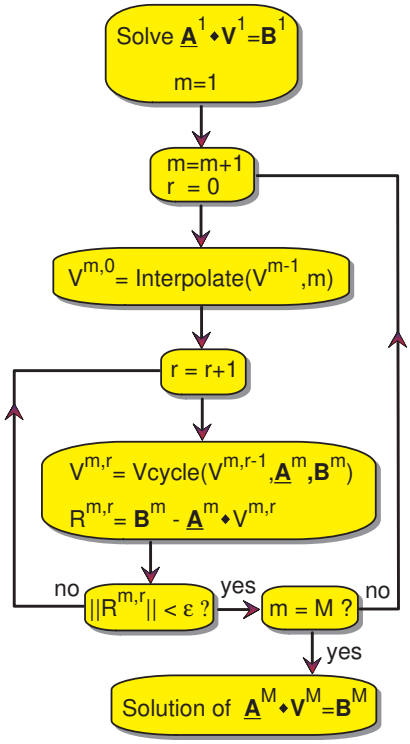
Subsequently this solution is now iterated through a series of increasingly finer grid levels. Here a solution is first approximated at the next finer grid level through a prolongation for instance through a simple bilinear interpolation:

$$
\begin{aligned}
w^m_{jk} &= w^{m-1}_{jk} \\
w^m_{j\pm1/2,k} &= \frac{1}{2}\left(w^{m-1}_{jk} + w^{m-1}_{j\pm1k}\right) \\
w^m_{j+1/2,k+1/2} &= \frac{1}{4}\left(w^{m-1}_{jk} + w^{m-1}_{j+1k} + w^{m-1}_{jk+1} + w^{m-1}_{j+1k+1}\right)
\end{aligned}
$$

For each grid level again relatively few relaxation steps are applied to smooth the high frequency components now with the large scale (low frequency components) determined by the coarsest grid solution.

$$\mathbf{W}^{m,\nu} = Relax^{\nu}\left(\mathbf{W}^m, \underline{\underline{\mathbf{A}}}^m, \mathbf{R}^m\right)$$

This V-cycle ends with the finest grid level M and since it requires only very few iteration steps for each level it represents a relatively fast iteration technique. Note that the relaxation method is not specified and the process can use any iteration method (GS, SOR, ADI, etc).
Regarding the basic properties of the cycle it also does not matter whether the equation that is solved is the original equation or whether it is linearized and the solution is for the correction only. Note that non-linear problem require to carry out the iteration for the original equation. Depending on the desired accuracy the V-cycle is carried out once or multiple times as indicated in the figure on the right.



Note that a big advantage of the multigrid method is that the low frequency portion of the spectrum (which is usually the most time consuming) is relaxed on the coarsest grid scale with a much smaller number of total grid points (orders of magnitudes for 3D problems) and is therefore a highly efficient relaxation method.

7.7 Summary of properties of elliptic solution methods

Newton's method:

Advantages:

- rapid convergence;

Disadvantages:

- small radius of convergence;
- $\underline{\underline{\mathbf{J}}} = \partial R_i / \partial V_j$ requires large memory;
- factorization is computationally expensive ($\sim N^3$ operations) for full matrices;
- no convergence if $\underline{\underline{\mathbf{J}}}$ is ill conditioned;

Jacobi method: Number of operations $\sim N^2$

Advantages:

- well suited for parallelization and acceleration techniques;
- simple;
- conserves symmetries;

Disadvantages:

- slow;
- more memory intensive than GS;

Gauss-Seidel method (GS): Number of operations $\sim N^2$

Advantages:

- well suited for parallelization and acceleration techniques;
- faster than Jacobi;
- simple;
- little memory required;

Disadvantages

- still slower than most other methods;

Successive over relaxation: Number of operations $\sim N^{1.5}$

Advantages:

- fast although some acceleration techniques or ADI may be faster;
- simple;

Disadvantages:

- dependence on empirical λ;
- difficult to parallelize and for acceleration;

Alternate direction implicit procedure (ADI): Number of operations $\sim N^{1.2}$

Advantages:

- fast
- relatively simple

Disadvantages:

- difficult for irregular domains
- less flexible

Acceleration techniques (conjugate gradient and other): Number of operations $\sim N^{1.2} - N^{1.5}$

Advantages:

- fast
- applicable to irregular domains

Disadvantages:

- may require preconditioning
- memory expensive
- programming effort

Multigrid method: Number of operations $\sim N^{1.2}$

Advantages:
fastest method (importance for large systems)
conceptually simple
Disadvantages:
difficult for irregular domains

Pseudotransient method (explicit): Number of operations per step $\sim N$. Properties for the linearized pseudotransient method combined with Newton's iteration are rather different than those summarized below.

Advantages:

- simple
- well suited for parallelization
- large radius of convergence
- detection of nonstationary solutions

Disadvantages:

- sometimes rather small time steps required (explicit)
- slow convergence