Semi-Coarsening in Three Directions for Euler-Flow Computations in Three Dimensions

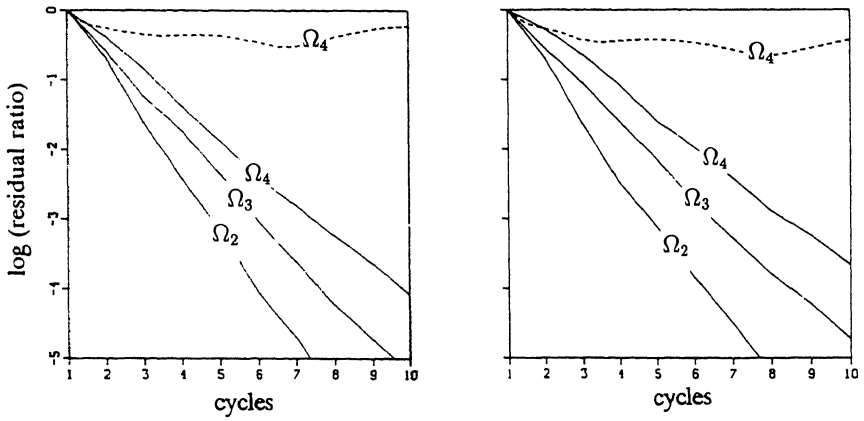
B. Koren, P.W. Hemker, P.M. de Zeeuw
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

A multiple semi-coarsened multigrid method for solving discretized, steady 3-D Euler equations of gas dynamics, is described and applied. Convergence results are presented for the case of the ONERA-M6 wing at transonic conditions. Comparisons are made with an optimal standard multigrid method, as well as with a single-grid method. Semi-coarsened multigrid appears to yield the best 3-D convergence behavior.

1 Introduction

A significant difficulty of standard multigrid methods for 3-D problems, when compared to application to 2-D problems, is that the requirements to be imposed on the smoother are much more severe. When cells are used as grid elements, in 3-D, standard coarsening implies restriction from each set of $2 \times 2 \times 2$ cells to a single cell only. Because the set of eight cells can support more high-frequency errors than the two-dimensional 2×2–set, 3-D standard multigrid imposes stronger requirements on the smoother than 2-D standard multigrid. Standard multigrid may not perform satisfactory for 3-D generalizations of 2-D problems, for which it does perform well. To illustrate this, we present standard-multigrid convergence results as obtained in solving first-order discretized, steady perfect-gas Euler equations for some 2-D and 3-D transonic test cases. The 2-D results, shown in Figure 1, have been taken from [8]. They show a reasonably grid-independent convergence behavior. In Figure 2 we show results, obtained with the same solution method, for a highly similar problem in 3-D. For this 3-D transonic test case, the convergence is far more grid-dependent than for both 2-D transonic cases. A fix might be found in deriving a more powerful smoother, keeping the other components of the numerical method the same. A more natural fix is not to apply standard, i.e. full coarsening, but to use multiple semi-coarsening instead. Figures 3a and 3b show standard coarsening and multiple semi-coarsening, respectively. (Semi-coarsening is the inverse of semi-refinement, as shown in Figure 1 of [6].) Though multigrid with multiple semi-coarsening is expected to be most fruitful for 3-D problems, as far as we know, applications of multiple semi-coarsening only exist in 2-D. The pioneering work has been done by Mulder [9], who has introduced multiple semi-coarsening as a fix for the poor convergence results observed in computing nearly grid-aligned flows governed by the steady, 2-D Euler equations. In [12], Radespiel and Swanson embroider on Mulder's approach for the steady, 2-D Euler equations. They pay particular attention to the prolongation operators. Semi-coarsened multigrid work for second-order elliptic



a. $M_\infty = 0.8$, $\alpha = 1.25°$. b. $M_\infty = 0.85$, $\alpha = 1°$.

Figure 1: Convergence behavior of standard multigrid method for NACA0012-airfoil at transonic conditions (solid lines: multigrid, dashed line: single grid).

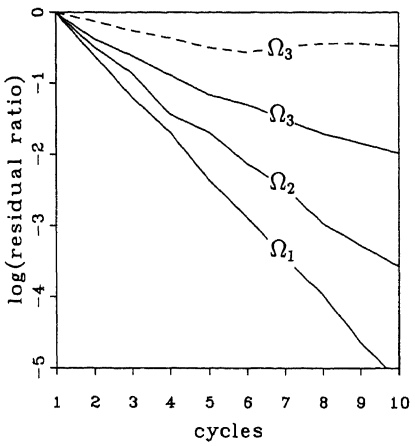
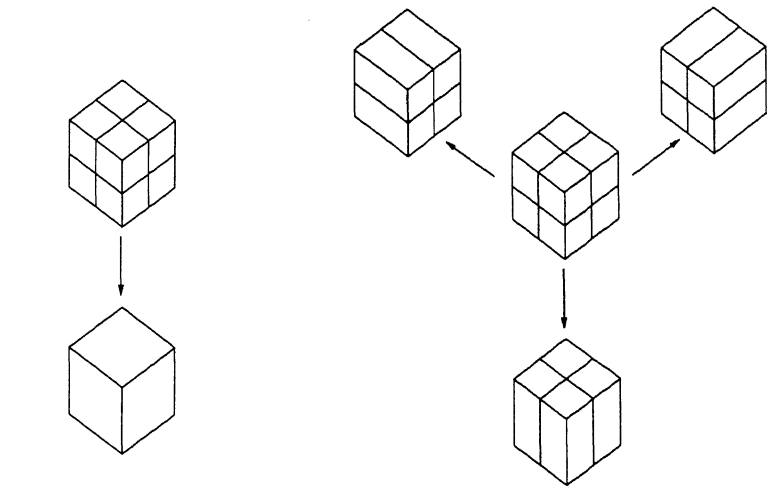


Figure 2: Convergence behavior of standard multigrid method for ONERA-M6 wing at transonic conditions, $M_\infty = 0.84$, $\alpha = 3.06°$ (solid lines: multigrid, dashed line: single grid).

(Poisson-type) equations has been studied by Naik and Van Rosendale [10] and – recently – by De Zeeuw [13]. Just as in [12], in [10, 13], much attention is paid to the choice of proper prolongation operators in the multigrid algorithm. In the present paper we consider semi-coarsened multigrid for the steady, 3-D Euler equations, and we also pay particular attention to the prolongation operators.



a. Full coarsening. b. Multiple semi-coarsening.

Figure 3: Two types of 3-D coarsenings.

2 Equations

2.1 Continuous equations

The steady, 3-D Euler equations are written as

$$\frac{\partial f(q)}{\partial x} + \frac{\partial g(q)}{\partial y} + \frac{\partial h(q)}{\partial z} = 0, \tag{1a}$$

with q the state vector

$$q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e \end{pmatrix}, \tag{1b}$$

$f(q)$, $g(q)$ and $h(q)$ the flux vectors

$$f(q) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho u(e + \frac{p}{\rho}) \end{pmatrix}, \quad g(q) = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho v(e + \frac{p}{\rho}) \end{pmatrix}, \quad h(q) = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho w(e + \frac{p}{\rho}) \end{pmatrix}, \tag{1c}$$

and with e the sum of internal and kinetic energy, satisfying the perfect-gas relation

$$e = \frac{1}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}\left(u^2 + v^2 + w^2\right). \tag{1d}$$

2.2 Discretized equations

The equations are discretized in the integral form

$$\oint_{\partial\Omega^*} (f(q)n_x + g(q)n_y + h(q)n_z)\,\mathrm{d}s = 0, \tag{2}$$

where $\partial\Omega^*$ is the boundary of an arbitrary subdomain Ω^* of the computational domain Ω, and where n_x, n_y and n_z are the x-, y- and z-components, respectively, of the outward unit normal on $\partial\Omega^*$. A straightforward and simple discretization is obtained by subdividing the entire computational domain Ω, in a structured manner, into disjunct, non-overlapping subdomains $\Omega_{i,j,k}$, $i = 0, 1, \ldots, i_{\max}$, $j = 0, 1, \ldots, j_{\max}$, $k = 0, 1, \ldots, k_{\max}$ (finite volumes) and by requiring that

$$\oint_{\partial\Omega_{i,j,k}} (f(q)n_x + g(q)n_y + h(q)n_z)\,\mathrm{d}s = 0, \quad \forall i, j, k. \tag{3}$$

Using the rotational invariance of the Euler equations

$$f(q)n_x + g(q)n_y + h(q)n_z = T^{-1}(\theta, \phi) f(T(\theta, \phi)q), \tag{4}$$

where $T(\theta, \phi)$ is the rotation matrix

$$T(\theta,\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta\cos\phi & \sin\theta\sin\phi & 0 \\ 0 & -\sin\theta & \cos\theta\cos\phi & \cos\theta\sin\phi & 0 \\ 0 & 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{5a}$$

$$\theta \equiv \frac{n_x}{\sqrt{n_x^2 + n_y^2 + n_z^2}}, \quad \phi \equiv \frac{n_y}{\sqrt{n_y^2 + n_z^2}}, \tag{5b}$$

(3) can be rewritten as

$$\oint_{\partial\Omega_{i,j,k}} T^{-1}(\theta, \phi) f(T(\theta, \phi)q)\mathrm{d}s = 0, \quad \forall i, j, k. \tag{6}$$

As finite volumes, arbitrarily shaped hexahedra are considered, the structured subdivision being such that – if existent – $\Omega_{i\pm1,j,k}$, $\Omega_{i,j\pm1,k}$ and $\Omega_{i,j,k\pm1}$ are the neighboring volumes of $\Omega_{i,j,k}$. The type of finite-volume method applied is the cell-centered one. Following the Godunov approach [1], along each cell face $\partial\Omega_{i,j,k}$, as in 2-D, the flux vector is assumed to be constant and to be determined by a uniformly constant left and right state, q^l and q^r, only. Doing so, the flux evaluation is identical to the numerical solution of the 1-D Riemann problem for a non-isenthalpic perfect-gas flow. For this, we apply the 3-D extension of the 2-D P-variant [5] of Osher's approximate Riemann solver [11]. This 3-D extension was first made by Houtman, see e.g. [7]. For the left and right cell-face states, we take the first-order accurate approximations

$$\begin{pmatrix} q^l_{i+\frac{1}{2},j,k} \\ q^r_{i+\frac{1}{2},j,k} \end{pmatrix} = \begin{pmatrix} q_{i,j,k} \\ q_{i+1,j,k} \end{pmatrix}, \quad \begin{pmatrix} q^l_{i,j+\frac{1}{2},k} \\ q^r_{i,j+\frac{1}{2},k} \end{pmatrix} = \begin{pmatrix} q_{i,j,k} \\ q_{i,j+1,k} \end{pmatrix}, \quad \begin{pmatrix} q^l_{i,j,k+\frac{1}{2}} \\ q^r_{i,j,k+\frac{1}{2}} \end{pmatrix} = \begin{pmatrix} q_{i,j,k} \\ q_{i,j,k+1} \end{pmatrix}. \tag{7}$$

At a later stage, these approximations can be replaced by higher-order accurate ones, in which case also limiters can be introduced.

3 Multigrid methods

In this section we first describe the standard 3-D multigrid algorithm. We use the 3-D generalization of the optimal 2-D multigrid approach, that was originally described in [5].

3.1 Standard multigrid

As the smoothing technique for the first-order discretized Euler equations, we prefer to apply collective symmetric point Gauss-Seidel relaxation. *Point* refers to the property that during the update of the local state vector $q_{i,j,k}$, all other state vectors are kept fixed. *Collective* refers to the property that the update of $q_{i,j,k}$ is done for all of its five components simultaneously. Further, *symmetric* means that after a relaxation sweep (i.e. an update of all state vectors $q_{i,j,k}$) in one direction, a new sweep in the reverse direction is made. The four different symmetric relaxation sweeps that are possible on a regular 3-D grid, are performed alternatingly. At each volume visited during a relaxation sweep, the system of five nonlinear equations is approximately solved by (exact) Newton iteration, the differential operator applied being $(\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial}{\partial w}, \frac{\partial}{\partial c}, \frac{\partial}{\partial z})^T$, where $c \equiv \sqrt{\gamma \frac{p}{\rho}}$, $z \equiv \ln\left(\frac{p}{\rho^\gamma}\right)$. This relaxation method is simple and robust.

As the standard multigrid method we apply the nonlinear version (FAS), preceded by nested iteration (FMG). For this we construct a nested set of grids such that each finite volume on a coarse grid is the union of $2 \times 2 \times 2$ volumes on the next finer grid (full coarsening, Figure 3a). Let $\Omega_0, \Omega_1, \ldots, \Omega_{\lambda_{\max}}$ be the sequence of such nested grids, with Ω_0 the coarsest and $\Omega_{\lambda_{\max}}$ the finest grid. Then, nested iteration is applied to obtain a good initial solution on $\Omega_{\lambda_{\max}}$, whereas nonlinear multigrid is applied to converge to the solution on the finest grid, $q_{\lambda_{\max}}$. The first iterate for the nonlinear multigrid cycling is the solution obtained by nested iteration. We proceed to discuss both stages in more detail.

The nested iteration starts with a user-defined initial estimate for q_0, the solution on the coarsest grid. To obtain an initial solution on a finer grid $\Omega_{\lambda+1}$, first the solution on the coarser grid Ω_λ is improved by a single nonlinear multigrid cycle. Hereafter, this solution is prolongated to the finer grid $\Omega_{\lambda+1}$. These steps are repeated until the highest level (finest grid) has been reached.

Let $N_\lambda(q_\lambda) = 0$ denote the nonlinear system of first-order discretized equations on Ω_λ, then a single nonlinear multigrid cycle is recurrently defined by the following steps:

1. Improve on Ω_λ the latest obtained solution q_λ by application of n_{pre} relaxation sweeps.

2. Compute on the next coarser grid $\Omega_{\lambda-1}$ the right-hand side $r_{\lambda-1} = N_{\lambda-1}(q_{\lambda-1}) - I_\lambda^{\lambda-1} N_\lambda(q_\lambda)$, where $I_\lambda^{\lambda-1}$ is a restriction operator for right-hand sides.

3. Approximate the solution of $N_{\lambda-1}(q_{\lambda-1}) = r_{\lambda-1}$ by the application of n_{FAS} nonlinear multigrid cycles. Denote the approximation obtained as $\tilde{q}_{\lambda-1}$.

4. Correct the current solution by: $q_\lambda = q_\lambda + \tilde{I}_{\lambda-1}^{\lambda}(\tilde{q}_{\lambda-1} - q_{\lambda-1})$, where $\tilde{I}_{\lambda-1}^{\lambda}$ is a prolongation operator for solutions.

5. Improve again q_λ by application of n_{post} relaxations.

Steps (2),(3) and (4) form the coarse-grid correction (all three are skipped on the coarsest grid). The efficiency of a coarse-grid correction depends in general on the coarseness of the coarsest grid. The restriction operator $I_\lambda^{\lambda-1}$ and the prolongation operator $\tilde{I}_{\lambda-1}^{\lambda}$ are defined by

$$\begin{aligned}(r_{\lambda-1})_{i,j,k} = (I_\lambda^{\lambda-1} r_\lambda)_{i,j,k} \equiv\ & (r_\lambda)_{2i,2j,2k} + \\ & (r_\lambda)_{2i-1,2j,2k} + (r_\lambda)_{2i,2j-1,2k} + (r_\lambda)_{2i,2j,2k-1} + \\ & (r_\lambda)_{2i-1,2j-1,2k} + (r_\lambda)_{2i-1,2j,2k-1} + (r_\lambda)_{2i,2j-1,2k-1} + \\ & (r_\lambda)_{2i-1,2j-1,2k-1}, \end{aligned} \tag{8a}$$

$$\begin{aligned}(\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i,2j,2k} = \\ (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i-1,2j,2k} = (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i,2j-1,2k} = (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i,2j,2k-1} = \\ (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i-1,2j-1,2k} = (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i-1,2j,2k-1} = (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i,2j-1,2k-1} = \\ (\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1})_{2i-1,2j-1,2k-1} = \\ \equiv (q_{\lambda-1})_{i,j,k}. \end{aligned} \tag{8b}$$

As values for n_{pre}, n_{post} and n_{FAS}, we use here at all levels λ: $n_{\text{pre}} = 0$, $n_{\text{post}} = 1$ and $n_{\text{FAS}} = 1$; i.e. as nonlinear multigrid cycles we use sawtooth-cycles with a single post-relaxation per level only. Notice that these choices of the operators $I_\lambda^{\lambda-1}$ and $\tilde{I}_{\lambda-1}^{\lambda}$ guarantee the following nonlinear Galerkin relations between the discretizations on the different grids: for all discrete functions $q_{\lambda-1}$ on $\Omega_{\lambda-1}$, $\lambda = 1, \ldots, \lambda_{\max}$, we have

$$I_\lambda^{\lambda-1} N_\lambda \left(\tilde{I}_{\lambda-1}^{\lambda} q_{\lambda-1}\right) = N_{\lambda-1}\left(q_{\lambda-1}\right). \tag{9}$$

3.2 Multiple semi-coarsened multigrid method

Also in the case of the semi-coarsened multigrid method we use FAS as the basic multigrid algorithm, and on each grid we apply collective symmetric point Gauss-Seidel relaxation as the smoothing technique. In the semi-coarsened multigrid method, however, we replace the sequentially ordered set of grids Ω_λ, $\lambda = 0, \ldots, \lambda_{\max}$, by a partially ordered set of grids $\Omega_{l,m,n}$, $l = 0, 1, \ldots, l_{\max}$, $m = 0, 1, \ldots, m_{\max}$, $n = 0, 1, \ldots, n_{\max}$, with $\Omega_{0,0,0}$ the coarsest and $\Omega_{l_{\max},m_{\max},n_{\max}}$ the finest grid. Now $l+m+n$ is called the level of grid $\Omega_{l,m,n}$. The nesting and the semi-coarsening relation between these grids is described in [6, 4].

Also here nested iteration (FMG) is applied to obtain a good initial solution on the finest grid. We proceed to discuss the present nested iteration and nonlinear multigrid iteration in more detail. The nested iteration starts with a user-defined initial estimate on the coarsest grid, $\Omega_{0,0,0}$, which is improved by relaxation. The approximate solution $q_{0,0,0}$ is prolongated (level-by-level) to all grids up to and including level 3, with the 3-D prolongation according to formula (29) in [3] (see Appendix A for the implementation in the present 3-D Euler context). Next, the solution $q_{1,1,1}$ is improved by a single nonlinear multigrid cycle and prolongated to all grids up to and including level 6. For simplicity, we assume that $l_{\max} = m_{\max} = n_{\max}$. Then, the above process can be repeated in a straightforward manner up to and including level $3l_{\max}$.

A single nonlinear multigrid cycle on level $l + m + n$ is recurrently defined by the following steps:

1. Compute on all grids at the next coarser level, $(l + m + n) - 1$ the same right-hand sides as in standard multigrid, but use another restriction operator, viz. the

one described in Appendix B. (The restriction of defects is still natural, i.e. by summation over all sub-cells.)

2. Approximate the solutions on the coarser level $(l+m+n)-1$ by the application of a single nonlinear multigrid cycle on level $(l+m+n)-1$.

3. Correct the current solutions on level $l+m+n$ by one of two alternative correction prolongations. One prolongation can be seen as an extension to 3-D and to systems of equations, of the prolongation due to Naik and Van Rosendale [10]. (It uses prolongation weights that are proportional to the absolute values of the restricted defect components.) The other correction prolongation is the one proposed in [3]. (It is the correction-prolongation version of the solution prolongation described in Appendix A, it uses fixed prolongation weights.) In Appendix C, both correction prolongations are described explicitly.

4. Improve the solutions on level $l+m+n$ by the application of n_{post} relaxations sweeps.

Thus, here we also use sawtooth cycles as nonlinear multigrid cycles. Whereas in the standard multigrid method one may also apply pre-relaxations, in the present semi-coarsened multigrid technique this is not (yet) possible. Pre-relaxation leads to incoherent right-hand side representations. For an explanation of this we refer to [13]. (Note that by not yet applying pre-relaxations, the averages taken in the restriction operator as described in Appendix B, are not yet relevant in fact.)

4 Numerical results

4.1 Test case

As test case we consider the ONERA-M6 wing at the transonic conditions $M_\infty = 0.84$, $\alpha = 3.06°$. The grids used are of C-O-type (Figures 4a - 4d). The wing as well as the grids are symmetric with respect to the plane through the wing's leading and trailing edges. In the convergence results to be presented hereafter, the finest grid considered is the $64 \times 16 \times 16$-grid.

4.2 Convergence results

Convergence results obtained are given in Figure 5. (Note that the convergence results that have already been presented in Figure 2 are repeated in Figures 5a–b.) In all four graphs, the residual ratio is defined as $\|R^{i_{\text{FAS}}}\|_{L_1} / \|R^1\|_{L_1}$, where $R^{i_{\text{FAS}}}$ is the first component (i.e. the mass component) of $N_{l_{\max},m_{\max},n_{\max}}(q^{i_{\text{FAS}}}_{l_{\max},m_{\max},n_{\max}})$ and where i_{FAS} refers to the status before the i-th FAS-cycle. The improvement of both semi-coarsened multigrid methods with respect to the standard multigrid method is significant. Of both semi-coarsened methods, the one with the fixed prolongation weights (Figure 5d) performs best.

5 Outlook

The foregoing multiple semi-coarsening method is called a full-grid-of-grids semi-coarsening method. A disadvantage of full-grid-of-grids semi-coarsening is that many grid points

are needed in total. With N the total number of points on the finest grid, asymptotically standard multigrid uses $\frac{8}{7}N$ grid points versus $8N$ points for the full-grid-of-grids approaches. An efficiency improvement is obtained by thinning out the grid-of-grids. Most ambitious in this respect is the sparse-grid-of-grids approach (see [3] and the further references there in). For a sparse grid-of-grids, for all grids $\Omega_{l,m,n}$ holds the constraint: $l+m+n \leq l_{\max}$ ($l_{\max} = m_{\max} = n_{\max}$). With the full grid-of-grids depicted as a cube in Figure 6a, the sparse grid-of-grids is the subset given in Figure 6b. The gain in efficiency by the reduction of the numbers of grid points is enormous. Theoretically, the sparse-grid-of-grids approach has a much better ratio of discrete accuracy over number of grid points used [2]. In the ideal case the full grid-of-grids will be completely replaced by a sparse grid-of-grids. Work in this direction is in progress.

References

[1] GODUNOV, S.K.: Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics (Cornell Aeronautical Lab. Transl. from the Russian), *Math. Sbornik*, **47**, 271-306 (1959).

[2] GRIEBEL, M., ZENGER, C. AND ZIMMER, S.: Multilevel Gauss-Seidel-algorithms for full and sparse grid problems, *Computing*, **50**, 127-148 (1993).

[3] HEMKER, P.W.: Finite volume multigrid for 3D-problems (in this book).

[4] HEMKER, P.W. AND PFLAUM, C.: Approximation on partially ordered sets of regular grids, *Report NM-R9611*, CWI, Amsterdam (1996).

[5] HEMKER, P.W. AND SPEKREIJSE, S.P.: Multiple grid and Osher's scheme for the efficient solution of the steady Euler equations, *Appl. Numer. Math.*, **2**, 475-493 (1986).

[6] HEMKER, P.W. AND ZEEUW, P.M. DE: BASIS3, a data structure for 3-dimensional sparse grids (in this book).

[7] HOUTMAN, E.M. AND BANNINK, W.J.: Experimental and numerical investigation of the vortex flow over a delta wing at transonic speed, in: *Vortex Flow Aerodynamics*, 5.1-5.11, (AGARD, Neuilly-sur-Seine, 1991).

[8] KOREN, B.: *Multigrid and Defect Correction for the Steady Navier-Stokes Equations, Application to Aerodynamics, CWI Tracts*, **74** (Stichting Mathematisch Centrum, Amsterdam, 1991).

[9] MULDER, W.A.: A new multigrid approach to convection problems, *J. Comput. Phys.*, **83**, 303-323 (1989).

[10] NAIK, N.H. AND ROSENDALE, J. VAN: The improved robustness of multigrid elliptic solvers based on multiple semicoarsened grids, *SIAM J. Numer. Anal.*, **30**, 215-229 (1993).

[11] OSHER, S. AND SOLOMON, F.: Upwind difference schemes for hyperbolic systems of conservation laws, *Math. Comput.*, **38**, 339-374 (1982).

[12] RADESPIEL, R. AND SWANSON, R.C.: Progress with multigrid schemes for hypersonic flow problems, *J. Comput. Phys.*, **116**, 103-122 (1995).

[13] ZEEUW, P.M. DE: Multiple semi-coarsening techniques (in this book).

Appendix A. Prolongation of solution

Using the same notation as in [6] and defining p as the pointer to a complete patch at level l,m,n and s as the index of one of the five solution components (say s=1,2,3,4,5), the algorithm for the solution prolongation reads in quasi-ForTran:

3-D prolongation:

```
if (l>lmin.and.m>mmin.and.n>nmin) then
   setting pointers:
   pxyf=pntr(xf,pyf)
   pxzf=pntr(xf,pzf)
   pyzf=pntr(yf,pzf)
   pxyzf=pntr(xf,pyzf)
   prolongation:
   data(s,p)=+data(s,pntr(xf,p))
             +data(s,pntr(yf,p))
             +data(s,pntr(zf,p))
             -(data(s,pntr(xkl,pxyf))+data(s,pntr(xkr,pxyf))+
               data(s,pntr(ykl,pxyf))+data(s,pntr(ykr,pxyf)))/4
             -(data(s,pntr(xkl,pxzf))+data(s,pntr(xkr,pxzf))+
               data(s,pntr(zkl,pxzf))+data(s,pntr(zkr,pxzf)))/4
             -(data(s,pntr(ykl,pyzf))+data(s,pntr(ykr,pyzf))+
               data(s,pntr(zkl,pyzf))+data(s,pntr(zkr,pyzf)))/4
             +(data(s,pntr(xkl,pntr(ykl,pxyzf)))+
               data(s,pntr(xkr,pntr(ykl,pxyzf)))+
               data(s,pntr(xkl,pntr(ykr,pxyzf)))+
               data(s,pntr(xkr,pntr(ykr,pxyzf)))+
               data(s,pntr(xkl,pntr(zkl,pxyzf)))+
               data(s,pntr(xkr,pntr(zkl,pxyzf)))+
               data(s,pntr(xkl,pntr(zkr,pxyzf)))+
               data(s,pntr(xkr,pntr(zkr,pxyzf)))+
               data(s,pntr(ykl,pntr(zkl,pxyzf)))+
               data(s,pntr(ykr,pntr(zkl,pxyzf)))+
               data(s,pntr(ykl,pntr(zkr,pxyzf)))+
               data(s,pntr(ykr,pntr(zkr,pxyzf))))/12,
             s=1,2,3,4,5
endif
```

2-D prolongations:

```
if (l=lmin.and.m>mmin.and.n>nmin) then
   setting pointer:
   pyzf=pntr(yf,pntr(zf,p))
   prolongation:
   data(s,p)=+data(s,pntr(yf,p))
             +data(s,pntr(zf,p))
             -(data(s,pntr(ykl,pyzf))+data(s,pntr(ykr,pyzf))+
               data(s,pntr(zkl,pyzf))+data(s,pntr(zkr,pyzf)))/4, s=1,2,3,4,5
endif
if (l>lmin.and.m=mmin.and.n>nmin) then
```

```
   setting pointer:
   pxzf=pntr(xf,pntr(zf,p))
   prolongation:
   data(s,p)=+data(s,pntr(xf,p))
             +data(s,pntr(zf,p))
             -(data(s,pntr(xkl,pxzf))+data(s,pntr(xkr,pxzf))+
               data(s,pntr(zkl,pxzf))+data(s,pntr(zkr,pxzf)))/4, s=1,2,3,4,5
endif
if (l>lmin.and.m>mmin.and.n=nmin) then
   setting pointer:
   pxyf=pntr(xf,pntr(yf,p))
   prolongation:
   data(s,p)=+data(s,pntr(xf,p))
             +data(s,pntr(yf,p))
             -(data(s,pntr(xkl,pxyf))+data(s,pntr(xkr,pxyf))+
               data(s,pntr(ykl,pxyf))+data(s,pntr(ykr,pxyf)))/4, s=1,2,3,4,5
endif

1-D prolongations:
if (l=lmin.and.m=mmin.and.n>nmin) then
   data(s,p)=data(s,pntr(zf,p)), s=1,2,3,4,5
endif
if (l=lmin.and.m>mmin.and.n=nmin) then
   data(s,p)=data(s,pntr(yf,p)), s=1,2,3,4,5
endif
if (l>lmin.and.m=mmin.and.n=nmin) then
   data(s,p)=data(s,pntr(xf,p)), s=1,2,3,4,5
endif
```

Appendix B. Restriction of defects

Using the same notation as in [6] and defining p as the pointer to a complete patch at level l,m,n and d as the index of one of the five corresponding defect components (say d=6,7,8,9,10), the algorithm for the restriction of the defects reads in quasi-ForTran (assuming that all finer-grid cells that are required exist):

3-D restriction:

```
if (l<lmax.and.m<mmax.and.n<nmax) then
   data(d,p)=1/3*(data(d,pntr(xkl,p))+data(d,pntr(xkr,p))+
                  data(d,pntr(ykl,p))+data(d,pntr(ykr,p))+
                  data(d,pntr(zkl,p))+data(d,pntr(zkr,p))), d=6,7,8,9,10
endif
```

2-D restrictions:

```
if (l=lmax.and.m<mmax.and.n<nmax) then
   data(d,p)=1/2*(data(d,pntr(ykl,p))+data(d,pntr(ykr,p))+
                  data(d,pntr(zkl,p))+data(d,pntr(zkr,p))), d=6,7,8,9,10
endif
if (l<lmax.and.m=mmax.and.n<nmax) then
   data(d,p)=1/2*(data(d,pntr(xkl,p))+data(d,pntr(xkr,p))+
                  data(d,pntr(zkl,p))+data(d,pntr(zkr,p))), d=6,7,8,9,10
endif
if (l<lmax.and.m<mmax.and.n=nmax) then
   data(d,p)=1/2*(data(d,pntr(xkl,p))+data(d,pntr(xkr,p))+
                  data(d,pntr(ykl,p))+data(d,pntr(ykr,p))), d=6,7,8,9,10
endif
```

1-D restrictions:

```
if (l=lmax.and.m=mmax.and.n<nmax) then
   data(d,p)=data(d,pntr(zkl,p))+data(d,pntr(zkr,p)), d=6,7,8,9,10
endif
if (l=lmax.and.m<mmax.and.n=nmax) then
   data(d,p)=data(d,pntr(ykl,p))+data(d,pntr(ykr,p)), d=6,7,8,9,10
endif
if (l<lmax.and.m=mmax.and.n=nmax) then
   data(d,p)=data(d,pntr(xkl,p))+data(d,pntr(xkr,p)), d=6,7,8,9,10
endif
```

Appendix C. Prolongation of solution corrections

With defect-dependent prolongation weights

Again using the same notation as in [6], defining p again as the pointer to a complete patch at level l,m,n, sold as the index for the solution components before update (say sold=6,7,8,9,10) and snew as the index for the updated solution components (snew=1,2,3,4,5), then the algorithm for the defect-dependent correction prolongation reads in quasi-ForTran:

3-D prolongation:

```
if (l>lmin.and.m>mmin.and.n>nmin) then
   calculation weights:
   dsum(d)=abs(data(d,pntr(xf,p)))+
           abs(data(d,pntr(yf,p)))+
           abs(data(d,pntr(zf,p))), d=6,7,8,9,10
   wxf(d)=abs(data(d,pntr(xf,p)))/dsum(d), d=6,7,8,9,10
   wyf(d)=abs(data(d,pntr(yf,p)))/dsum(d), d=6,7,8,9,10
   wzf(d)=abs(data(d,pntr(zf,p)))/dsum(d), d=6,7,8,9,10
   prolongation:
   data(snew,p)=data(sold,p)
                +wxf(d)*(data(snew,pntr(xf,p))-data(sold,pntr(xf,p)))
                +wyf(d)*(data(snew,pntr(yf,p))-data(sold,pntr(yf,p)))
                +wzf(d)*(data(snew,pntr(zf,p))-data(sold,pntr(zf,p))),
                (snew,d,sold)=(1,6,11),(2,7,12),(3,8,13),(4,9,14),(5,10,15)
endif
```

2-D prolongations:

```
if (l=lmin.and.m>mmin.and.n>nmin) then
   calculation weights:
   dsum(d)=abs(data(d,pntr(yf,p)))+abs(data(d,pntr(zf,p))), d=6,7,8,9,10
   wyf(d)=abs(data(d,pntr(yf,p)))/dsum(d), d=6,7,8,9,10
   wzf(d)=abs(data(d,pntr(zf,p)))/dsum(d), d=6,7,8,9,10
   prolongation:
   data(snew,p)=data(sold,p)
                +wyf(d)*(data(snew,pntr(yf,p))-data(sold,pntr(yf,p)))
                +wzf(d)*(data(snew,pntr(zf,p))-data(sold,pntr(zf,p))),
                (snew,d,sold)=(1,6,11),(2,7,12),(3,8,13),(4,9,14),(5,10,15)
endif
if (l>lmin.and.m=mmin.and.n>nmin) then
   calculation weights:
   dsum(d)=abs(data(d,pntr(xf,p)))+abs(data(d,pntr(zf,p))), d=6,7,8,9,10
   wxf(d)=abs(data(d,pntr(xf,p)))/dsum(d), d=6,7,8,9,10
   wzf(d)=abs(data(d,pntr(zf,p)))/dsum(d), d=6,7,8,9,10
   prolongation:
   data(snew,p)=data(sold,p)
                +wxf(d)*(data(snew,pntr(xf,p))-data(sold,pntr(xf,p)))
                +wzf(d)*(data(snew,pntr(zf,p))-data(sold,pntr(zf,p))),
                (snew,d,sold)=(1,6,11),(2,7,12),(3,8,13),(4,9,14),(5,10,15)
endif
```

```
if (l>lmin.and.m>mmin.and.n=nmin) then
   calculation weights:
   dsum(d)=abs(data(d,pntr(xf,p)))+abs(data(d,pntr(yf,p))), d=6,7,8,9,10
   wxf(d)=abs(data(d,pntr(xf,p)))/dsum(d), d=6,7,8,9,10
   wyf(d)=abs(data(d,pntr(yf,p)))/dsum(d), d=6,7,8,9,10
   prolongation:
   data(snew,p)=data(sold,p)
                +wxf(d)*(data(snew,pntr(xf,p))-data(sold,pntr(xf,p)))
                +wyf(d)*(data(snew,pntr(yf,p))-data(sold,pntr(yf,p))),
                (snew,d,sold)=(1,6,11),(2,7,12),(3,8,13),(4,9,14),(5,10,15)
endif

1-D prolongations:
if (l=lmin.and.m=mmin.and.n>nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(zf,p)))-data(sold,pntr(zf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
if (l=lmin.and.m>mmin.and.n=nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(yf,p)))-data(sold,pntr(yf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
if (l>lmin.and.m=mmin.and.n=nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(xf,p)))-data(sold,pntr(xf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
```

With fixed prolongation weights

Again using the same notation as in [6], defining p as the pointer to a complete patch at level l,m,n and defining sold and snew as the indices for the solution components before and after update, respectively ((snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)), then the algorithm for the correction prolongation with fixed weights reads in quasi-ForTran:

```
3-D prolongation:
if (l>lmin.and.m>mmin.and.n>nmin) then
   setting pointers:
   pxf=pntr(xf,p)
   pyf=pntr(yf,p)
   pzf=pntr(zf,p)
   pxyf=pntr(xf,pyf)
   pxzf=pntr(xf,pzf)
   pyzf=pntr(yf,pzf)
   pxyzf=pntr(xf,pyzf)
   pxyll=pntr(xkl,pntr(ykl,pxyzf))
   pxyrl=pntr(xkr,pntr(ykl,pxyzf))
   pxylr=pntr(xkl,pntr(ykr,pxyzf))
```

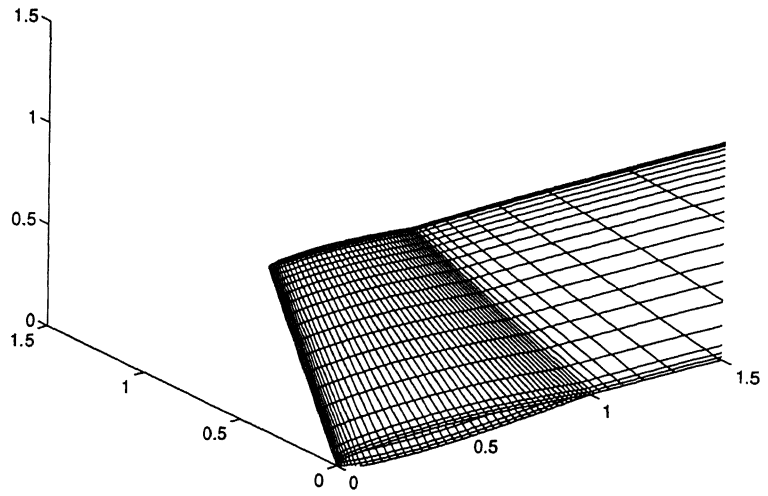
```
   pxyrr=pntr(xkr,pntr(ykr,pxyzf))
   pxzll=pntr(xkl,pntr(zkl,pxyzf))
   pxzrl=pntr(xkr,pntr(zkl,pxyzf))
   pxzlr=pntr(xkl,pntr(zkr,pxyzf))
   pxzrr=pntr(xkr,pntr(zkr,pxyzf))
   pyzll=pntr(ykl,pntr(zkl,pxyzf))
   pyzrl=pntr(ykr,pntr(zkl,pxyzf))
   pyzlr=pntr(ykl,pntr(zkr,pxyzf))
   pyzrr=pntr(ykr,pntr(zkr,pxyzf))
   prolongation:
   data(snew,p)=data(sold,p)
                +(data(snew,pxf)-data(sold,pxf))
                +(data(snew,pyf)-data(sold,pyf))
                +(data(snew,pzf)-data(sold,pzf))
                -((data(snew,pntr(xkl,pxyf))-data(sold,pntr(xkl,pxyf)))+
                  (data(snew,pntr(xkr,pxyf))-data(sold,pntr(xkr,pxyf)))+
                  (data(snew,pntr(ykl,pxyf))-data(sold,pntr(ykl,pxyf)))+
                  (data(snew,pntr(ykr,pxyf))-data(sold,pntr(ykr,pxyf))))/4
                -((data(snew,pntr(xkl,pxzf))-data(sold,pntr(xkl,pxzf)))+
                  (data(snew,pntr(xkr,pxzf))-data(sold,pntr(xkr,pxzf)))+
                  (data(snew,pntr(zkl,pxzf))-data(sold,pntr(zkl,pxzf)))+
                  (data(snew,pntr(zkr,pxzf))-data(sold,pntr(zkr,pxzf))))/4
                -((data(snew,pntr(ykl,pyzf))-data(sold,pntr(ykl,pyzf)))+
                  (data(snew,pntr(ykr,pyzf))-data(sold,pntr(ykr,pyzf)))+
                  (data(snew,pntr(zkl,pyzf))-data(sold,pntr(zkl,pyzf)))+
                  (data(snew,pntr(zkr,pyzf))-data(sold,pntr(zkr,pyzf))))/4
                +((data(snew,pxyll)-data(sold,pxyll))+
                  (data(snew,pxyrl)-data(sold,pxyrl))+
                  (data(snew,pxylr)-data(sold,pxylr))+
                  (data(snew,pxyrr)-data(sold,pxyrr))+
                  (data(snew,pxzll)-data(sold,pxzll))+
                  (data(snew,pxzrl)-data(sold,pxzrl))+
                  (data(snew,pxzlr)-data(sold,pxzlr))+
                  (data(snew,pxzrr)-data(sold,pxzrr))+
                  (data(snew,pyzll)-data(sold,pyzll))+
                  (data(snew,pyzrl)-data(sold,pyzrl))+
                  (data(snew,pyzlr)-data(sold,pyzlr))+
                  (data(snew,pyzrr)-data(sold,pyzrr)))/12,
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif

2-D prolongations:
if (l=lmin.and.m>mmin.and.n>nmin) then
   setting pointers:
   pyf=pntr(yf,p)
   pzf=pntr(zf,p)
   pyzf=pntr(yf,pntr(zf,p))
   prolongation:
   data(snew,p)=data(sold,p)
```

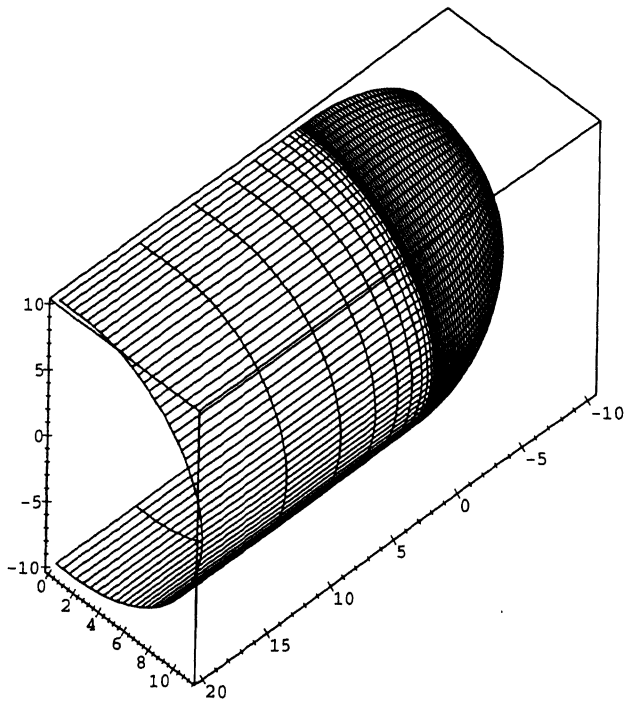
```
                +(data(snew,pyf)-data(sold,pyf))
                +(data(snew,pzf)-data(sold,pzf))
                -((data(snew,pntr(ykl,pyzf))-data(sold,pntr(ykl,pyzf)))+
                  (data(snew,pntr(ykr,pyzf))-data(sold,pntr(ykr,pyzf)))+
                  (data(snew,pntr(zkl,pyzf))-data(sold,pntr(zkl,pyzf)))+
                  (data(snew,pntr(zkr,pyzf))-data(sold,pntr(zkr,pyzf))))/4,
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
if (l>lmin.and.m=mmin.and.n>nmin) then
   setting pointers:
   pxf=pntr(xf,p)
   pzf=pntr(zf,p)
   pxzf=pntr(xf,pntr(zf,p))
   prolongation:
   data(snew,p)=data(sold,p)
                +(data(snew,pxf)-data(sold,pxf))
                +(data(snew,pzf)-data(sold,pzf))
                -((data(snew,pntr(xkl,pxzf))-data(sold,pntr(xkl,pxzf)))+
                  (data(snew,pntr(xkr,pxzf))-data(sold,pntr(xkr,pxzf)))+
                  (data(snew,pntr(zkl,pxzf))-data(sold,pntr(zkl,pxzf)))+
                  (data(snew,pntr(zkr,pxzf))-data(sold,pntr(zkr,pxzf))))/4,
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
if (l>lmin.and.m>mmin.and.n=nmin) then
   setting pointers:
   pxf=pntr(xf,p)
   pyf=pntr(yf,p)
   pxyf=pntr(xf,pntr(yf,p))
   prolongation:
   data(snew,p)=data(sold,p)
                +(data(snew,pxf)-data(sold,pxf))
                +(data(snew,pyf)-data(sold,pyf))
                -((data(snew,pntr(xkl,pxyf))-data(sold,pntr(xkl,pxyf)))+
                  (data(snew,pntr(xkr,pxyf))-data(sold,pntr(xkr,pxyf)))+
                  (data(snew,pntr(ykl,pxyf))-data(sold,pntr(ykl,pxyf)))+
                  (data(snew,pntr(ykr,pxyf))-data(sold,pntr(ykr,pxyf))))/4,
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif

1-D prolongations:
if (l=lmin.and.m=mmin.and.n>nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(zf,p))-data(sold,pntr(zf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
if (l=lmin.and.m>mmin.and.n=nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(yf,p))-data(sold,pntr(yf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
```

```
endif
if (l>lmin.and.m=mmin.and.n=nmin) then
   data(snew,p)=data(sold,p)
                +data(snew,pntr(xf,p))-data(sold,pntr(xf,p)),
                (snew,sold)=(1,11),(2,12),(3,13),(4,14),(5,15)
endif
```

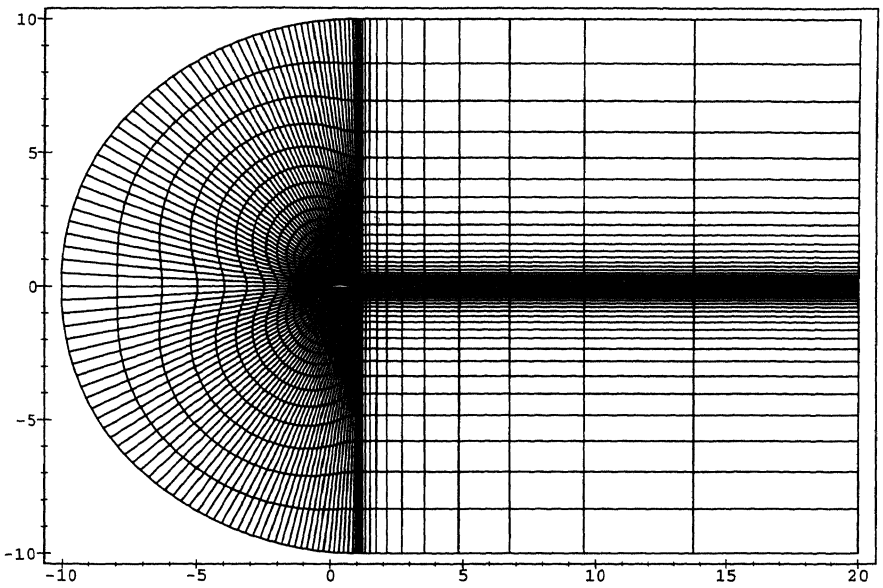


a. At upper side wing.

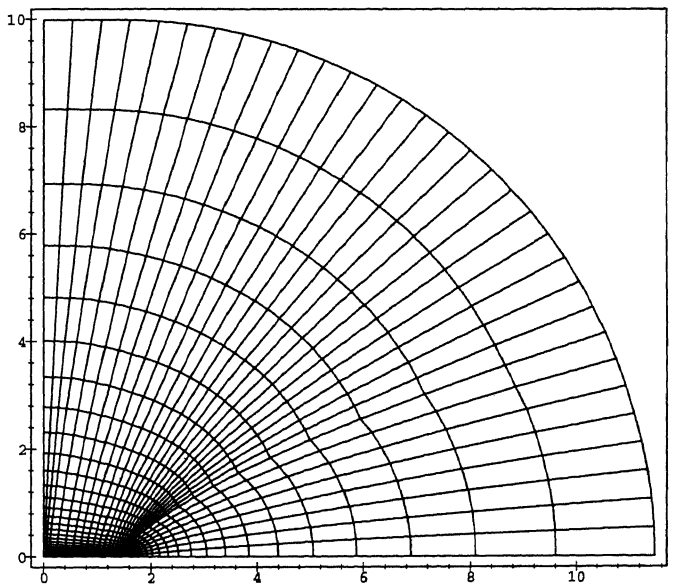


b. At far-field boundary.

Figure 4: Views at $128 \times 32 \times 32$ C-O-type grid ONERA-M6 wing.

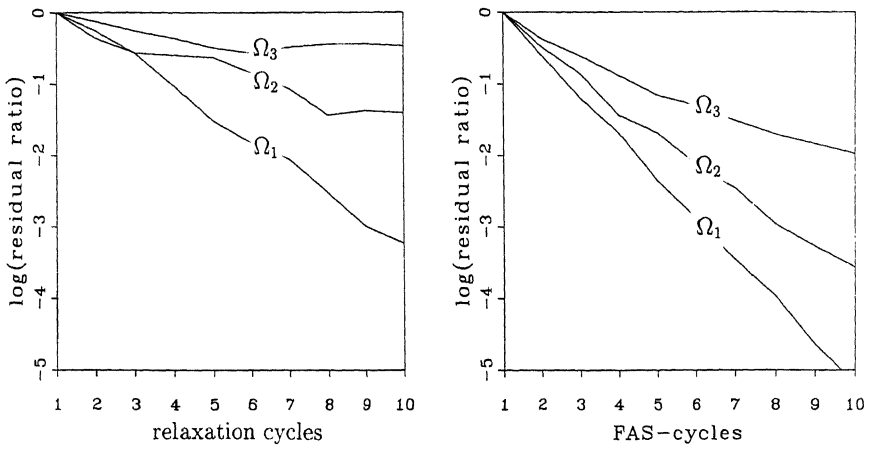


c. At symmetry boundary.



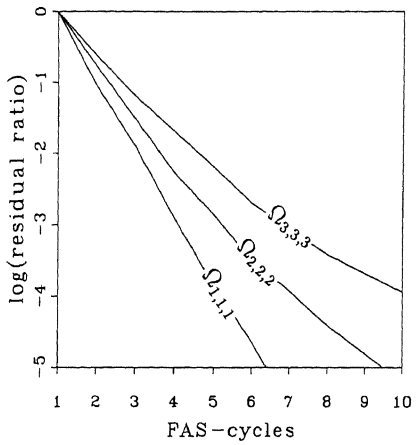
d. At upper part downstream boundary.

Figure 4: Views at $128 \times 32 \times 32$ C-O-type grid ONERA-M6 wing.

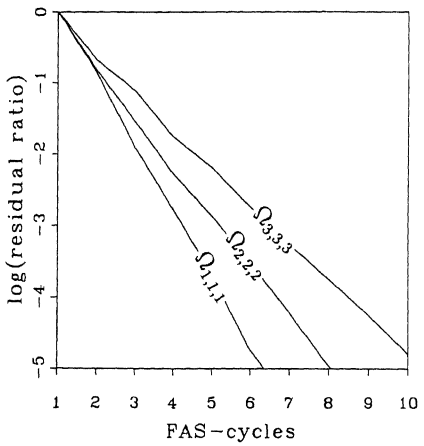


a. Single-grid.

b. Standard multigrid.

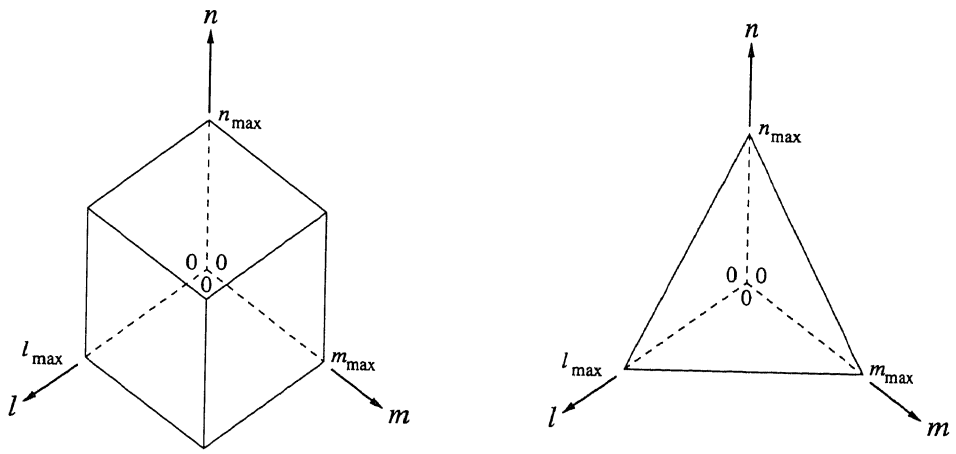


c. Semi-coarsened multigrid, with defect-dependent prolongation weights.



d. Semi-coarsened multigrid, with fixed prolongation weights.

Figure 5: Convergence behavior of different solution methods for ONERA-M6 wing at transonic conditions, $M_\infty = 0.84$, $\alpha = 3.06°$.



a. Full.

b. Sparse.

Figure 6: Grids of grids.