BIT 20 (1980), 289-301

# ON THE STRUCTURE OF AN ADAPTIVE MULTI-LEVEL ALGORITHM

P. W. HEMKER

**Abstract.**

The structure of Multi-Level Adaptive Algorithms is explained. Their recursive character is exposed by means of the formal language ALGOL 68. The basic structure is given in two forms: coarse grid corrections to fine grids and fine grid corrections to coarse grids. The latter description can be used for the treatment of fully automatic adaptive grids. A new description of the FAS FMG algorithm of Brandt is given and the last section concludes with an ALGOL 68 procedure for the solution of boundary value problems with adaptive mesh-refinement.

*Key words and phrases*: Multi-Grid Algorithms, Multi-Level Technique, Adaptive Mesh-Selection, Defect Correction Principle.

## 1. Introduction.

In recent papers, Brandt [1, 2, 3] considers Multi-Level Adaptive Techniques (MLAT) for the computation of solutions to partial differential equations. The essential feature of these techniques is that the solution of one continuous problem on a region $\Omega$ is obtained by different discretizations of the problem on a hierarchy of finer and finer grids. The accuracy of the numerical solution finally obtained corresponds to the accuracy that can be attained with the discretization on the finest grid in the hierarchy and the iterative process for the solution of the discrete system on the finest grid largely benefits from the interaction with the discretization on the coarser ones.

There is overwhelming evidence (cf. e.g. [1, 4, 7], that the iterative method thus obtained is among the most efficient ones for a very large class of linear and non-linear problems. It is an additional advantage of the Multi-Grid approach that it allows for adaptive local grid-refinement, i.e. during the computation a mesh-refinement can be generated in those parts of $\Omega$ where this refinement is required for the accurate representation of the solution.

These are arguments in favour of the Multi-Grid (MG-)method, but a disadvantage is that the fundamental MG-algorithm has a much more complex structure than most discretization and iterative solution methods. In this paper it is our intention to clarify this basic structure of multi-level algorithms, to make it

---

AMS(MOS) subject classification scheme (1980): 65V05; 65B05, 65N20, 65Q05.
Received July 13, 1979. Revised August 20, 1980.

easier in the future both to analyze them mathematically and to implement them in a well structured way.

In Section 2 we give the basic principles of the MG-algorithm together with an ALGOL 68 representation of it. In Section 3 we give an ALGOL 68 representation of the FAS FMG algorithm of Brandt [3].

In Section 4 we give the dual representation of the MG-algorithm and in Section 5 a representation of the algorithm in the case of locally refined meshes and adaptive mesh-refinement. Further we indicate how adaptive order selection can be implemented.

## 2. Basic principles.

It is neither our intention to give here the heuristic background nor to give a theoretical account for the MLAT-technique. For this we refer to [1, 3]. However, we present enough of the theory to give some insight into the structure of the algorithm.

We consider the approximate solution of the (*nonlinear*) operator equation

$$Lu = f \tag{2.1}$$

on a domain $\Omega$ with boundary $\partial\Omega$. The domain $\Omega$ is covered by a sequence of finer and finer grids $G_0, G_1, \ldots, G_k, \ldots$ with mesh width $h_0, h_1, \ldots, h_k$. Usually $h_{j+1} = h_j/2$. There exist operators $I_h^H$ (the prolongation from $G^H$ to $G^h$) and $I_H^h$ (the restriction from $G^h$ to $G^H$) which transfer grid functions defined on a coarse grid $G^H$ to grid-functions defined on the next finer grid $G^h$ (vice versa), so that $I_H^h\, I_h^H$ is the identity operator on $G^H$.

If, on the grid $G^h$, we have an approximate solution $y^h$ to the $G^h$-discretization of (2.1)

$$L^h u^h = f^h\,, \tag{2.2}$$

then we can get better approximations, i.e. approximations with smaller residuals

$$z^h = f^h - L^h y^h\,, \tag{2.3}$$

either by relaxation methods on the grid $G^h$ (Jacobi, Gauss-Seidel, SOR etc.), which are efficient for reducing the high-frequency components of the residual, or by solving the residual equation on a coarser grid $G^H$; i.e. solving

$$L^H U^H = I_H^h f^h - I_H^h L^h y^h + L^H I_H^h y^h\,, \tag{2.4}$$

using $I_H^h y^h$ as a starting approximation. A better approximation to $u^h$ is obtained by

$$y^h + I_h^H (U^H - I_H^h y^h)\,. \tag{2.5}$$

These coarse grid corrections are efficient for reducing the low-frequency components of the residual. Equation (2.2) can be solved efficiently by using both methods successively.

We can also consider the relation between both grids from another point of view. If we solve

$$L^H u^H = f^H , \tag{2.6}$$

we obtain a $G^H$-approximation to the solution of (2.1). As usual, the truncation error, $\tau^H$, of (2.6) is defined by

$$L^H I_H u = f^H + \tau^H , \tag{2.7}$$

where $u$ denotes the solution of (2.1) and $I_H$ denotes the restriction operator which restricts a continuous function on $\Omega$ to a grid function on $G^H$. Analogously, the relative truncation error, $\tau_h^H$, between $G^H$ and $G^h$ can be defined by

$$L^H I_H^h u^h = f^H + \tau_h^H , \tag{2.8}$$

where $u^h$ is the solution of (2.2). Hence, once having computed

$$\tau_h^H \approx L^H I_H^h y^h - I_H^h L^h y^h , \tag{2.9a}$$

an approximation to

$$\tau_h^H = L^H I_H^h u^h - I_H^h L^h u^h , \tag{2.9b}$$

the solution of the coarse-grid problem (2.8) gives us (an approximation to) $I_H^h u^h$, i.e. the fine grid (accurate) solution restricted to the coarse grid.

Thus, the relation between the finer and coarser grids can be looked at in two different ways: either the coarse grid problems can be considered as means to accelerate the rate of convergence for the fine grid discretization, or the fine grid problems can be considered as means to estimate the truncation error of the coarse grid discretization.

We shall describe the MLAT algorithms using the formal language ALGOL 68 (van Wijngaarden et al., [6]). Because MLAT techniques are recursive ALGOL 68 is an elegant and (in a numerical research environment) efficient language for their implementation. The complex data-structures that are necessary in MLAT-codes (cf. [2]) are easily programmed in ALGOL 68 because of e.g. the handling of pointers by the ref-principle, the implicit garbage-collection and the possibility to define ones own modes and operators in this language.

In Text 1b we describe the basic algorithm but first – in Text 1a – we give an informal description of a set of ALGOL 68 modes and operators that correspond to the mathematical objects and operators that were considered in the first part of this section. Their formal description (i.e. their implementation in an ALGOL 68 program) depends on details such as the dimension of $\Omega$, the definition of the operator $L$, the discretization method used to obtain $L^h$, the relaxation method used etc.. The basic structure of the algorithm is independent of such details, and we omit them.

**mode function** = #some procedure mode which represents a continuous (vector-valued) function $\mathsf{R}^k \to \mathsf{R}^n$, where $k$ is the dimension of $\Omega$; e.g. with $k=2$, $n=1$, we have
**mode function = proc (real, real) real**#

**mode net** = #some structure to represent a net function on $G^h$, e.g. with $k=2$, $n=1$,
**mode net = ref [,] real**#

**op lh** = (**net** *yh*) **net:**
#some representation of the operator $L^h$ in eq. (2.2)#

**op restrict** = (**net** *yh*) **net:**
#some representation of the operator $I_H^h$#

**op prolongate** = (**net** *yh*) **net:**
#some representation of the operator $I_h^H$#

**op interpolate** = (**net** *yh*) **net:**
#some interpolation operator from $\mathsf{G}^H$ onto $\mathsf{G}^h$, possibly more accurate than $I_h^H$#

**op restriction** = (**int** *k*, **function** *f*) **net:**
#some representation of the operator $I_{h_k}$ (see eq. (2.7))#

**function** *f* = #some representation of the right hand side function *f* in equation (2.1)#

**proc** *relax* = (**ref net** *yh*, **net** *rh*, **ref bool** *converged*, *slow convergence*) **void:**
#a procedure which performs one step of the relaxation iteration of $L^h y^h = r^h$; in *converged* and *slow convergence* the procedure delivers information about the size of the residual and the speed of convergence#

**proc** *solve directly* = (**ref net** *yh*, **net** *rh*) **void:**
#solves the discrete equation $L^h y^h = r^h$ on the coarsest grid $\mathsf{G}_0$. Since $L^h$ is non-linear this procedure in general needs an initial approximation to the solution (as input in *yh*)#

*Text* 1*a*. The modes, operators and procedures used in Text 1b.

The algorithmic structure is given in Text 1b in two procedures: *solve upto level* and *solve at level*; *solve upto level* is the driver procedure to which the number of levels $m$ and a start approximation on the zero-level grid $\mathsf{G}_0$ are given. It delivers the solution on a set of finer and finer grids $\mathsf{G}_0, \mathsf{G}_1, \ldots, \mathsf{G}_m$ that all cover the whole domain $\Omega$. Most of the work is done by the recursive procedure *solve on level*, in which relaxation and coarse grid corrections interchange with each other in order to reduce the residual.

**proc** *solve upto level* = (**int** *m*, **ref** [ ] **net** *yh*, **net** *start approximation*) **void:**
**begin net** *rh* = 0 **restriction** *f*;

```
  yh[0] := start approximation;
  solve directly (yh[0], rh);
  for k to m
  do net rh = k restriction f;
    yh[k] := interpolate yh[k-1];
    solve on level (k, yh[k], rh)
  od
end;
proc solve on level = (int k, ref net yh, net rh) void:
if k = 0
then solve directly (yh, rh)
else while relax (yh, rh, converged, slow convergence);
    not converged
    do  if slow convergence
        then net yhc := restrict yh;
             net rhc = restrict rh
               -restrict lh yh+lh restrict yh;                    #(2.4)#
             solve on level (k-1, yhc, rhc);
             yh := yh+
               prolongate (yhc-restrict yh)                       #(2.5)#
        fi
    od
fi;
```

*Text 1b.* The structure of the basic MG-algorithm.

The work done in the procedures is the following. First a starting approximation on the coarsest grid $G_0$ is used to solve the problem on $G_0$. Since $G_0$ only contains a small number of points, we assume that this problem can be solved efficiently by some "direct method". This can be either the direct solution of a linear system for linear problems or the use of e.g. some Newton process for nonlinear problems. The solution on $G_0$ is interpolated onto $G_1$ to obtain a starting approximation on $G_1$ and the problem is solved on $G_1$. Thus we get a solution on level 1, which is interpolated onto $G_2$, etc.. On each level $k$ ($k = 1, 2, \ldots, m$) the problem is essentially solved by the successive use of relaxation sweeps (reducing the high-frequency components in the residual) and the computation of coarse grid corrections on level $k-1$ (reducing the low-frequency components in the residual). Thus, in [0:$m$] **net** $yh$, one finally obtains the sequence $\{y^{h_j}\}_{j=0,\ldots,m}$ of solutions to the discrete problems $L^{h_j}y^{h_j} = f^{h_j}$ on $G_j$.

## 3. The FAS full multi-grid algorithm.

Of all variants of the MLAT algorithm (Cycle *A*, *B* and *C* in Brandt [1, 2] for linear problems and Cycle *C* FAS [1, 2] and FAS FMG [3] for nonlinear

problems) the FAS FMG (full approximation storage, full multi-grid) algorithm shows the complete structure best. With the basic structure of Section 2 in mind we are now able to translate the FAS FMG flowchart into two ALGOL 68 procedures. This translation is given in Text 2. The algorithm given here is essentially the same as the one given in Text 1, but now some strategy has been implemented (1) to decide on slow convergence, and (2) to solve the discrete systems of equations with a residual error which is of the same order of magnitude as the truncation error.

Hence, in this algorithm, we use three parameters $\eta$, $\alpha$ and $\delta$ in order to tune the strategy; $\eta$ is a parameter used in deciding when convergence is slow; its value depends on the difference scheme and the relaxation method used; $\alpha$ is a parameter which controls the factor by which a finer grid solution should be more accurate than a coarse grid solution; its value depends on the mesh ratio and the order of accuracy of the discretization method; $\delta$ is a parameter which insures that a coarse grid correction is not computed much more accurately than the relaxation corrections with which it shares the task of reducing the residual. More details about these parameters can be found in [3].

```
proc solve upto level = (int m, ref [ ] net yh) void:
begin real tolk := 0.0001 * alfa;
  solve directly (yh[0], 0 restriction f);
  for k to m
  do net rh := k restriction f;
    yh[k] := interpolate yh[k-1];
    tolk := tolk*alfa;
    solve on level (k, yh[k], rh, tolk, tolk)
  od
end;

proc solve on level = (int k, ref net yh, net rh, real tol, ref real nexttol) void:
begin real error, olderror := maxreal;
  while relax (yh, rh, error);
    error ≧ tol
  do if k = 0 or error/olderror ≦ eta
    then olderror := error
    else net yc := restrict yh;
      net residual = rh-lh yh;
      net rc = lh yc+restrict residual;
      real tau = norm(rc-(k-1)restriction f);
      real tolc := delta*error;

      solve on level (k-1, yc, rc, tolc, tolc);
```

```
        yh := yh + prolongate (yc - restrict yh);
        nexttol := alfa * tau;
        olderror := maxreal
      fi
    od
end;
```

*Text* 2. The FAS FMG-algorithm.
An ALGOL 68 translation of the algorithm published by Brandt [1979]. In this text the procedure *relax*(*yh*, *rh*, *error*) performs a relaxation iteration step in the solution of $L^h y^h = r^h$ and delivers in *error* an estimate of the norm of its residual: $error \approx \|r^h - L^h y^h\|$. The procedure *norm* delivers some suitable norm of a netfunction.

## 4. The dual representation of the MG-algorithm.

In the procedure *solve on level* in Section 2 the MG-algorithm can clearly be recognized from the point of view where coarse grid problems serve as fine grid correctors. Having obtained the fine grid solution on $G_m$, we trivially can obtain $G_m$-accurate solutions on lower levels by

```
for k from m by -1 to 1
do y[k-1] := restrict y[k] od;
```

However we can also reorganize the algorithm so that (i) these grid-functions are obtained immediately, and (ii) fine grid solutions can be recognized as generating corrections – in the form of corrected solutions and relative truncation errors – to the coarse grid solutions. To this end we have to turn the recursive part of the algorithm inside out. Thus we find the basic algorithm in its dual representation. *It computes the fine grid solution y[m] and its restrictions to coarser grids essentially the same way as in the representation in Text* 1 *in Section* 2.

The use of the dual representation is that it will enable us to introduce partial mesh-refinement (see Section 5). In that case finer grids do not extend over the whole range of $\Omega$ and we cannot apply the algorithmic structure in the form given in Section 2 because no "fine grid solution" has now been defined on $\Omega$.

We can understand the recursive part of the procedure *correct for finer nets* in Text 3 by noting that for each call of the form

*correct for finer nets* $(k, y, r)$

we have

(1) at entrance: the coarse grid (level $k$) solution has been found, since:

(4.1) $$norm(rh[k] - \textbf{lh}\ yh[k]) \leqq tolerence;$$

(The procedure *norm* delivers some proper norm of a netfunction.)

```
proc solve upto level = (int m, ref [ ] net y, net startapproximation) void:
begin [0:m] net r;
  for i to m do y[i] := nil od;
  y[0] := startapproximation;
  r[0] := 0 restriction f;
  while solve directly (y[0], r[0]);
    not correct for finer nets (0, y, r)
  do skip od;
end;

proc correct for finer nets = (int k, ref [ ] net y, r) bool:
if k = upb r
then true
else bool converged; real olderror, error := maxreal;
  if net (y[k+1]) :=: net(nil)
  then #enter level k+1 for the first time#
       y[k+1] := interpolate y[k];
       r[k+1] := (k+1) restriction f
  else y[k+1] := y[k+1]+prolongate (y[k]-restrict y[k+1])
  fi;
  while not
    if error := olderror; relax (y[k+1], r[k+1], error);
      converged := error < tolerence
    then correct for finer nets (k+1, y, r)
    else (error/olderror) ≦ eta #i.e. slow convergence#
    fi
  do skip od;
  y[k] := restrict y[k+1];
  r[k] := restrict r[k+1]-restrict lh y[k+1]+lh restrict y[k+1];
  converged
fi;
```

*Text* 3. The dual representation of the basic MG-algorithm. In this text the meaning of the procedure *relax* is the same as in Text 2.

(2) on exit with the value **true**:

$y[m]$, $y[j]$, $r[j]$, $j=k, k+1, \ldots, m-1$ have been changed, and
$r[m]$, $y[j]$, $r[j]$, $j=0, 1, \ldots, k-1$ have not;
$y[m]$ contains the solution of the problem discretized on $G_m$, moreover,
$y[j]=$**restrict** $y[j+1]$, $j=k, k+1, \ldots, m$, (i.e. at all levels the $G_m$-accurate solution is delivered);

$r[j]$, $j=k,k+1,\ldots,m-1,m$, contains the relative truncation error of the $G_j$-discretization with respect to the $G_m$-discretization;

(3) on exit with the value **false**:

$r[k]$, $y[k]$ and $y[k+1]$ only have been changed and we shall not return at the level $k$ before $y[k]$ has been adapted – by coarse grid corrections – such that with the given $r[k]$ the inequality (4.1) has been satisfied.

REMARK. Notice that the ALGOL 68 phrase "**while** *a*; **not** *b* **do skip od**" can be understood as "**repeat** *a* **until** *b*".

## 5. Partial and adaptive grid refinement.

The discretization applied in connection with the MG-technique mostly uses uniform nets (either for finite difference or finite element methods). This implies that the set-up of the discrete equations is simple and requires very little administration of geometrical details. Mesh-refinement is obtained by adding finer and finer mesh-levels $G_j$. If a partial mesh-refinement is sufficient, this refinement can be restricted to (smaller and smaller) parts of the original domain $\Omega$. The only requirements are (i) that a few (e.g. two) coarse grids cover the original domain $\Omega$ and (ii) that the domain of each grid is covered by the domain of the next coarser one. A coarse-grid "father" can have different grid "sons", i.e. finer grids that extend over a subdomain. We restrict ourselves to the case where "brother"-grids (sons of the same father) extend over disjoint subregions of the father-region. Thus we obtain a tree-structure of net-functions that together form a grid-family. The data structure for such a grid family is in ALGOL 68 easily represented by

**mode gridf** = **struct** (**net** *net*, **ref** [ ] **gridf** *sons*) .

Each object of the mode **gridf** contains a **net** as in Section 2 and an array of references to its sons, which themselves are of the mode **gridf** again. If this array of sons is of length zero, there are no sons, i.e. there are no finer grids in the region covered by this **gridf**.

The only tool we further need in order to formulate our MG-algorithm on a grid-family instead of on a set of coextensive grids $G_0, G_1, \ldots, G_m$, is an operator **subregion** so that the expression *gridson* **subregion** *gridfather* denotes the grid which is the restriction of the grid *gridfather* to the region that is covered by the grid *gridson*. Such an operator

**op subregion** = (**gridf** *gridson*, **ref gridf** *gridfather*)
**ref gridf**: . . . ,

the details of which depend on the explicit structure of a **net**, can be described in ALGOL 68 without difficulties. An auxiliary operator:

```
op <:= = (ref gridf a, gridf b)void: net of a := net of b;
```

is convenient for the assignment of net-values of grid $b$ to grid $a$.

With the aid of the mode **gridf** and the operators **subregion** and $<:=$, we describe the MG-algorithm for an arbitrary grid-family by the two procedures in Text 4.

(In order not to repeat the small complication of the first interpolation (initialization) on a new level for $y$, we have identified the two interpolation operators **interpolate** and **prolongate**; a distinction between both operators would lead to an algorithmic structure for initialization similar to the one in Section 4.)

```
proc solve = (ref gridf y, net startapproximation)void:
begin gridf r := y restriction f;
  initialize at zero all nets of the family (y);
  net of y := startapproximation;
  while solve directly (net of y, net of r);
    not correct for finer grids (y, r)
  do skip od
end;

proc correct for finer grids = (ref gridf yhf, rhf)bool:
begin bool ready := true;
  for i to upb (sons of yhf)
  do ref gridf yh = (sons of yhf)[i], rh = (sons of rhf)[i];
    yh := yh + prolongate (yh subregion yhf - restrict yh)
    while not
      if relax(net of yh, net of rh, ...); converged
      then correct for finer nets (yh, rh)
      else slow convergence
      fi
    do skip od;
    yh subregion yhf <:= restrict yh;
    rh subregion rhf <:= restrict rh - restrict lh yh + lh restrict yh;
    ready := ready and converged
  od; ready
end;
```

*Text* 4. The MG-algorithm for tree-structured partially refined grids. This algorithm solves the equation $Lu = f$ on the tree of grids called **gridf** $y$. In this procedure the operators **prolongate, restrict** and **lh,** working on a **gridf** *gridf*, have the same effect on *net* **of** *gridf* as their analogs working on a **net** in the previous sections. The expression *gridf* **restriction** $f$ creates a grid-family similar to (congruent with) *gridf* and initializes all its net-functions with the corresponding values of the **function** $f$.

We notice that for a PDE the boundary conditions for a subregion $\Omega_\alpha \subset \Omega$ are determined either by a part of the boundary conditions of $\Omega$ itself (if $\Omega_\alpha$ is partly along the boundary of $\Omega$) or by the most recently obtained approximation to the solution on its gridfather (which – at convergence – takes the same values as the gridson itself!).

In contrast with the representation given in Section 2, in the representation of the algorithms given in Section 4 and 5, at any time before the finest grid solution has finally been obtained, the gridfamily can be extended in any subregion and at any level, in order to adapt the grid to the solution. This means that both $y$ and $r$ should be extended in a similar way, that the offspring of $y$ should be initialized at approximate values of the solution and that the corresponding offspring of $r$ should be initialized at the corresponding function values of $f$. In the course of the computation the algorithm automatically corrects all values at all levels.

We illustrate this in Text 5, which is part of an actual ALGOL 68 program which solves a (two-point) boundary-value problem. In this program – as in Text 2 – some strategy has been implemented to decide on convergence and slow convergence. The program has been used to find and to resolve boundary layers in linear and nonlinear problems of the form

$$\varepsilon y'' + N(x, y, y') = f \quad \text{on } (a, b),$$
$$y(a) = ya,\ y(b) = yb\ .$$

Numerical results of the MG-method for this problem will be reported later. (We do not mean that in practice two-point boundary-value problems should be solved by iterative means (!) but Text 5 can easily be extended for 2- and 3-dimensional problems.)

It is also easy to implement adaptive order of convergence. To this end one should have an operator **lh** in which the order of accuracy of the discretization depends on the mesh-size and/or on the subregion of $\Omega$ where it is applied. The high order of accuracy of the solution is obtained by evaluation of the discrete operator **lh** on the finest grids only.

The complex equations of the high order discretization are solved by the iterative use of this accurate operator in combination with the simpler approximate inverse operator. In this sense the method is similar to the method of deferred corrections (cf. [5]).

```
proc solve = (ref gridf y, real tol)void:
begin real tolc := 0.1, alfa := 0.25, eta := 0.75;
  proc solve directly = (ref gridf y, r)void:
  begin real error;
    while relaxation(y, r, error);
      not (error<tol)
    do skip od
  end;
```

```
proc correct for finer grids = (ref gridf yhf, rhf)bool:
begin bool ready := true;
  for i to upb (sons of yhf)
  do bool converged; gridf tau;
    real taunorm, error, olderror := maxreal;
    ref gridf yh = (sons of yhf)[i], rh = (sons of rhf)[i];
    yh := yh+prolongate(yh subregion yhf-restrict yh);
    while
      while not
        if relaxation(yh, rh, error); converged := error<tolc
        then correct for finer grids (yh, rh)
        else (error/olderror)>eta #i.e. slow convergence#
        fi
      do olderror := error od;
      tau := lh restrict yh-restrict lh yh;
      if upb (sons of yh) ≠ 0
      then false
      elif taunorm := norm tau;
        (tolc := taunorm*alfa; tolc<tol | tolc := tol); error > tolc
      then converged
      elif converged := true; taunorm*alfa**2 > tol
      then create sons(yh, tau, tol/alfa**2);
        rh := yh restriction f;
        (tolc *:= alfa; tolc<tol | tolc := tol);
        not correct for finer grids (yh, rh)
      else false
      fi
    do olderror := error od;
    yh subregion yhf <:= restrict yh;
    rh subregion rhf <:= restrict rh+tau;
    ready := ready and converged
  od; ready
end;

gridf r := y restriction f;
while solve directly (y, r);
  not correct for finer grids (y, r)
do skip od
end;
```

*Text* 5. An ALGOL 68 procedure for a Multi-Level Algorithm with adaptive mesh-refinement for the solution of $Lu=f$. A solution $y$ is obtained such that $\|Ly-f\|<tol$. The procedure call *create sons* (*yh*, *tau*, *t*) creates sons for the **gridf** *yh* on those parts of $\Omega$ where *net* **of** $tau>t$, i.e. where the values of the netfunction *tau* are greater than a given value *t*.

## REFERENCES

1. A. Brandt, *Multi-level adaptive solutions to boundary value problems*, Math. Comp. 31 (1977), 333–390.
2. A. Brandt, *Multi-Level Adaptive Solutions to Partial Differential Equations – Ideas and Software*, Proceedings of Symposium on Mathematical Software. (Mathematics Research Center, University of Wisconsin, March 1977), (John Rice, ed.) pp. 277–318, Academic Press, New York.
3. A. Brandt, *Multi-Level Adaptive Techniques (MLAT) for Singular Perturbation Problems.* (In: Numerical Analysis of Singular Perturbation Problems, P. W. Hemker and J. J. H. Miller eds., Academic Press, London, 1979.)
4. R. A. Nicolaides, *On the observed rate of convergence of an iterative method applied to a model elliptic difference equation*, Math. Comp. 32 (1978), 127–133.
5. H. J. Stetter, *The defect correction principle and discretization methods*, Num. Math. 29 (1978), 425–443.
6. Van Wijngaarden et al. Eds., *Revised Report on the Algorithmic Language ALGOL* 68, Springer-Verlag, New York, Heidelberg, Berlin (1976).
7. P. Wesseling, *A convergence proof for a multiple grid method*, Report NA-21 (1978), Delft Technical University, The Netherlands.

MATHEMATISCH CENTRUM
P.O. BOX 4079
1009 AB AMSTERDAM
THE NETHERLANDS