REPORT 92-40

An efficient algorithm for the computation of Galerkin coarse grid approximation for the incompressible Navier-Stokes equations

S. Zeng and P. Wesseling

ISSN 0922-5641

Reports of the Faculty of Technical Mathematics and Informatics no. 92-4

Delft 1992
Abstract

Efficient computation of coarse grid matrices is important to the overall efficiency of multigrid methods using Galerkin coarse grid approximation. A way to compute coarse grid matrices efficiently (algorithm CALRAP) with the non-zero pattern of coarse grid matrices determined by the algorithm STRURAP is discussed for operator-independent prolongations and restrictions with boundary modifications, assuming that the discretization matrix on the finest grid is derived from a scalar partial differential equation. By means of partition of grids, the computation of coarse grid matrices near boundaries is well treated in the same way as for interior grid points, with neither introducing if-then statements nor distinguishing between interior and boundary cases in the innermost loop of the algorithm CALRAP, which is expected to give an efficient computation of coarse grid matrices. Quasi-Algol descriptions of the two algorithms are developed, which can be used as predesigns for practical FORTRAN codes. A generalization of the algorithms is presented for the case that the discretization matrix is derived from a set of partial differential equations, particularly the incompressible Navier-Stokes equations, discretized on a staggered grid. A quasi-Algol description of the generalization is also given.
1 Introduction

By using Galerkin coarse grid approximation in the context of multigrid methods, the matrices and right-hand sides on coarse grids are computed, with the matrix and right-hand side on the finest grid provided. The efficiency of the computation of matrices and right-hand sides on coarse grids plays an important role in the overall efficiency of a multigrid method. In this paper, we will develop an efficient algorithm for this computation, restricting us to the case that prolongation and restriction operators are not operator-dependent. Here it suffices to describe the idea of Galerkin coarse grid approximation on two grids. Let a fine grid be $\mathcal{G}$ and a coarse grid be $\tilde{\mathcal{G}}$; coarse grid quantities will be indicated by an overbar. Let $A$ be a discretization matrix of a discretization of a boundary value problem on $\mathcal{G}$, and let $b$ be the right-hand side. The coarse grid approximation of $A$ is given by

$$\tilde{A} = \text{RAP},$$

with $R$ the restriction operator and $P$ the prolongation operator. This paper concentrates on the computation of $\tilde{A}$.

The discretization system to be considered here on the finest grid results from a discretization on a staggered grid of a set of partial differential equations, more precisely, the incompressible Navier-Stokes equations (or the Navier-Stokes equations accompanied by some transport equations, for example, the Boussinesq equations). The principles of this discretization are described in [3], [2], [5] and [6]. Staggered grids are used, so that different variables may be defined on different grid points, which makes performing RAP more difficult than on non-staggered grids. So we treat RAP without considering the staggering of grids first. The algorithm STRURAP which determines the non-zero pattern of $\tilde{A}$, assuming $A$ to be a matrix obtained from a discretization of a scalar partial differential equation, is discussed in section 2. Starting from an existing algorithm called CALRAP, section 3 analyzes the computation of $\tilde{A}$, which results in a revised version of the algorithm CALRAP, which is developed in section 4. Sections 5 and 6 present detailed descriptions of these algorithms, on which computer codes may be based. In section 7, these algorithms are generalized to staggered grids for a set of partial differential equations, and detailed descriptions of the generalized algorithms are given.

2 The Algorithm STRURAP

In stencil notation, as described in [4], prolongation, restriction and discrete equations can be written as:

$$ (Pu)_i = \sum_{k \in 2^n} P^*(k,j)\tilde{u}_k, \quad j = i - 2k, \ i \in \mathcal{G}, \quad (2.1) $$

$$ (Ru)_i = \sum_{j \in 2^n} R(i,j)u_k, \quad k = 2i + j, \ i \in \tilde{\mathcal{G}}, k \in \mathcal{G}, \quad (2.2) $$

$$ (Au)_i = \sum_{j \in 2^n} A(i,j)u_k, \quad k = i + j, \ i, k \in \mathcal{G}, \quad (2.3) $$
with \( u \) standing for some variable and \( nd \) being the number of space dimensions. \( P^* \) is the transpose of the prolongation \( P \). If \( j \) does not belong to the structure of the corresponding operator, or \( i \) does not belong to the grid, then the corresponding operator element is defined to be zero. The structures of \( P^* \), \( R \), \( A \) and \( \tilde{A} \) are defined respectively by the following sets:

\[
S_{P^*} = \{ j \in \mathbb{Z}^{nd} : \exists i \in \mathcal{G}, \text{ with } P^*(i, j) \neq 0 \},
\]

\[
S_R = \{ j \in \mathbb{Z}^{nd} : \exists i \in \mathcal{G}, \text{ with } R(i, j) \neq 0 \},
\]

\[
S_A = \{ j \in \mathbb{Z}^{nd} : \exists i \in \mathcal{G}, \text{ with } A(i, j) \neq 0 \},
\]

\[
S_{\tilde{A}} = \{ j \in \mathbb{Z}^{nd} : \exists i \in \mathcal{G}, \text{ with } \tilde{A}(i, j) \neq 0 \}.
\]

In terms of stencil notation, the computation of \( RAP \) can be realized by using the following formula (cf. [4]):

\[
\tilde{A}(i, n) = \sum_{m \in S_R} \sum_{k \in S_A} R(i, m) A(2i + m, k) P^*(i + n, m + k - 2n).
\]

Based on this formula, an algorithm called STRURAP is designed to determine the non-zero pattern of \( \tilde{A} \), and was given in [1] and outlined in [4], as follows:

```algorithm
Algorithm STRURAP

comment Calculation of \( S_{\tilde{A}} \)
begin
\( S_{\tilde{A}} = \emptyset \)
for \( q \in S_{P^*} \) do
  for \( m \in S_R \) do
    for \( k \in S_A \) do
      \( n = (m + k - q)/2 \)
      if \( n \in \mathbb{Z}^{nd} \) then
        \( S_{\tilde{A}} = S_{\tilde{A}} \cup n \)
      end if
    end for
  end for
end for
end STRURAP
```

3 The Algorithm CALRAP

Using formula (2.9), an algorithm called CALRAP is developed for the computation of \( \tilde{A} \) and is also given in [1] and [4]. The general structure of this algorithm is as follows:

```algorithm
Algorithm 1 CALRAP

comment Calculation of \( \tilde{A} = RAP \)
begin
\( \tilde{A} = 0 \)
for \( n \in S_{\tilde{A}} \) do
  for \( m \in S_R \) do
    \( \tilde{A}(i, n) = \sum_{k \in S_A} R(i, m) A(2i + m, k) P^*(i + n, m + k - 2n) \)
  end for
end for
end CALRAP
```

2
for $k \in S_A$ do
  $q = m + k - 2n$
  if $q \in S_{P^*}$ then
    $\tilde{G}_1 = \{i \in \tilde{G} : 2i + m \in \tilde{G}\} \cap \{i \in \tilde{G} : i + n \in \tilde{G}\}$
    for $i \in \tilde{G}_1$ do
      $\tilde{A}(i, n) = \tilde{A}(i, n) + R(i, m)A(2i + m, k)P^*(i + n, q)$
    od
  end if
od od od end CALRAP

In order to obtain high efficiency on vector computers, the innermost loop is designed to be the longest.

The innermost loop is usually very big. To achieve high efficiency, it is good to move as much work out of this loop as possible. In the case that restriction and prolongation operators do not depend on location, this can be done as follows:

Algorithm 2 CALRAP

begin $\tilde{A} = 0$
  for $n \in S_{\tilde{A}}$ do
    for $m \in S_R$ do
      for $k \in S_A$ do
        $q = m + k - 2n$
        if $q \in S_{P^*}$ then
          $\tilde{G}_1 = \{i \in \tilde{G} : 2i + m \in \tilde{G}\} \cap \{i \in \tilde{G} : i + n \in \tilde{G}\}$
          $\mu = R(m)P^*(q)$
          for $i \in \tilde{G}_1$ do
            $\tilde{A}(i, n) = \tilde{A}(i, n) + \mu \cdot A(2i + m, k)$
          od
        end if
      od od od end CALRAP

Here, $R(i, m)$ and $P^*(i + n, q)$ are simply written as $R(m)$ and $P^*(q)$, because they are independent of location. However, in practice, boundary modifications are needed for $R$ and $P^*$, which means that the above algorithm cannot be used uniformly over the whole grid $\tilde{G}_1$. Consequently, one has to introduce some if-then statements in order to determine whether $i$ or $i + n$ is on a boundary, or one has to separate boundary cases from the interior case and treat them differently. Use of if-then statements in the innermost loop degrades efficiency (parallelization and vectorization); but separating boundary cases brings about complexities in programming. For example, in three dimensions, there could be 27 boundary cases, for which one has to build 27 program segments. So, it is desirable to develop an algorithm that is not much more complicated than algorithm 2 and is also able to handle different
boundary cases, without introducing if-then statements in the innermost loop, and without constructing different codes for different boundary cases. We will show how this can be done.

4 A Revised Version of the Algorithm CALRAP

4.1 Partition of Grids

Because $P$ and $R$ are not operator-dependent, it is possible to partition the grid $\bar{G}$ into subgrids such that $P$ and $R$ are constant in each subgrid. Let a subgrid be identified by an integer $b = (b_1, b_2, \ldots, b_{nd}) \in Z^{nd}$, and let the collection of all $b$'s be denoted by the set $B$. A subgrid is denoted by $\bar{G}(b)$; $b = 0$ refers to the interior part of $\bar{G}$. We have

$$\bar{G} = \bigcup_{b \in B} \bar{G}(b),$$

(4.1)

$$\bar{G}(b) \cap \bar{G}(b') = \emptyset, \text{ for } b \neq b'.$$

(4.2)

So, $R(i, m)$ can be written as $R_4(m)$ if $i \in \bar{G}(b)$, and $P^*(i + n, q)$ can be written as $P^*_b(q)$ if $i + n \in \bar{G}(b')$.

4.2 The Starting Formula

For a given $i \in \bar{G}$ and $n \in SA$, we know in which subgrid $i$ and $i + n$ are and therefore which boundary case we have for $R(i, m)$ and $P^*(i + n, q)$. Suppose that we have $i \in \bar{G}(b)$ with $b \in B$ such that $i + n$ is located in the subgrid $\bar{G}(b')$ with $b' \in B$. In formula (2.9), for given $m \in S_R$, $q = m + k - 2n$ can be the same for different combinations of $k$ and $n$, which causes the multiplication of $R_4(m)$ and $P^*_b(q)$ to be carried out more than once. To avoid this repetition, we rewrite formula (2.9) as follows:

$$\bar{A}(i, n) = \sum_{m \in S_R} \sum_{q \in S_{P^*}} R(i, m)A(2i + m, q + 2n - m)P^*(i + n, q).$$

(4.3)

4.3 The Revised Version of CALRAP

Using formula (4.3), we reconstruct algorithm 1 as follows:

\emph{Algorithm 3 CALRAP}

\begin{verbatim}
comment Calculation of $\bar{A} = RAP$
begin $\bar{A} = 0$
    for $m \in S_R$ do
        $\bar{G}_1 = \{i \in \bar{G} : 2i + m \in \bar{G}\}$
        for $n \in S_A$ do
            $\bar{G}_2 = \{i \in \bar{G}_1 : i + n \in \bar{G}\}$
            for $q \in S_{P^*}$ do
                $k = q + 2n - m$
                if $k \in S_A$ then
                    for $i \in \bar{G}_2$ do

\end{verbatim}

4
\[ \tilde{A}(i, n) = \tilde{A}(i, n) + R(i, m)A(2i + m, k)P^*(i + n, q) \]

\[ \text{od} \]

\[ \text{end if} \]

\[ \text{od od od} \]

\[ \text{end CALRAP} \]

In order to take advantage of the fact that operators \( R \) and \( P^* \) are constant in each subgrid so that the multiplication between \( R \) and \( P^* \) can be moved outside of the innermost loop, as in algorithm 2, \( \mathcal{G}_2 \) is also divided into subgrids in a similar way as \( \tilde{\mathcal{G}} \): \( \tilde{\mathcal{G}}_3(0) \) is the interior of \( \tilde{\mathcal{G}}_2 \), \( \tilde{\mathcal{G}}_3(b), b \neq 0 \) is on the boundary of \( \tilde{\mathcal{G}}_2 \); the subdivision of \( \tilde{\mathcal{G}}_2 \) is called \( \tilde{\mathcal{G}}_3(b) \).

We know that \( \tilde{\mathcal{G}}_3(0) \) is in the interior of \( \tilde{\mathcal{G}} \). Also, because for \( i \in \tilde{\mathcal{G}}_2 \) we have \( i + n \in \tilde{\mathcal{G}} \), we know that \( i + n \) is in the interior of \( \tilde{\mathcal{G}} \) for \( i \in \tilde{\mathcal{G}}_3(0) \). If \( i \in \tilde{\mathcal{G}}_3(b) \) with \( b \neq 0 \), it has to be checked if and which boundary case we have for \( R(i, m) \) and \( P^*(i + n, q) \). Note that both for \( R(i, m) \) and \( P^*(i + n, q) \) we have only one boundary case in each \( \tilde{\mathcal{G}}_3(b) \). Which case we have is determined by \( r(b) = (r_1(b_1), r_2(b_2), \ldots, r_{nd}(b_{nd})) \) and \( p(b) = (p_1(b_1), p_2(b_2), \ldots, p_{nd}(b_{nd})) \).

For example, with \( i \in \tilde{\mathcal{G}}_3(b) \), we check for which \( b' \in B \) \( i \in \tilde{\mathcal{G}}(b') \). If \( i \in \tilde{\mathcal{G}}(b') \), \( r(b) = b' \).

Similarly, if \( i + n \in \tilde{\mathcal{G}}(b'') \), \( p(b) = b'' \).

Combining the above considerations, we arrive at algorithm 4, which satisfies our two requirements: no if-then statements inside the innermost loop; no special code segments for boundary cases.

\begin{algorithm}
  \caption{CALRAP}
  \begin{algorithmic}
    \State \textbf{comment} Calculation of \( \tilde{A} = \text{RAP} \)
    \State \textbf{begin} \( \tilde{A} = 0 \)
    \For {\( m \in S_R \)}
      \State \( \tilde{\mathcal{G}}_1 = \{ i \in \tilde{\mathcal{G}} : 2i + m \in \mathcal{G} \} \)
      \For {\( n \in S_{\tilde{A}} \)}
        \State \( \tilde{\mathcal{G}}_2 = \{ i \in \tilde{\mathcal{G}}_1 : i + n \in \tilde{\mathcal{G}} \} \)
        \For {\( b \in B \)}
          \State Define \( \tilde{\mathcal{G}}_3(b) \)
          \Comment{Construction of \( r(b) \) and \( p(b) \)}
          \For {\( b' \in B \)}
            \If {\( \tilde{\mathcal{G}}_3(b) \subseteq \tilde{\mathcal{G}}(b') \)}
              \State \( r(b) = b' \)
            \EndIf
            \If {\( \{ i + n : i \in \tilde{\mathcal{G}}_3(b) \} \subseteq \tilde{\mathcal{G}}(b') \)}
              \State \( p(b) = b' \)
            \EndIf
          \EndFor
        \EndFor
      \EndFor
    \EndFor
    \For {\( q \in S_{P^*} \)}
      \State \( k = q + 2n - m \)
      \If {\( k \in S_A \)}
        \For {\( b \in B \)}
          \State \( \mu = R_{r(b)}(m) \cdot P^*_{p(b)}(q) \)
        \EndFor
      \EndIf
  \EndAlgorithmic
\end{algorithm}
for $i \in \mathcal{G}_3(b)$ do \\
\hspace{1cm} \tilde{A}(i, n) = \tilde{A}(i, n) + \mu \cdot A(2i + m, k) \\
end if \\
end CALRAP

FORTRAN 77 has no language elements that represent loops over elements of sets. Therefore in the following sections we will translate STRURAP and CALRAP in quasi-Algol from which FORTRAN code is easily derived.

5 Quasi-Algol Formulation of the Algorithm STRURAP

5.1 The Scope of a Structure

Let $\mathcal{S}_{op}$ be the structure of an operator. We call $j \in \mathcal{S}_{op}$ the displacement of an element of the structure, and call the element $j = (0, 0)$ in two dimensions or $j = (0, 0, 0)$ in three dimensions the reference element. For $j \in \mathcal{S}_{op}$ with $\mathcal{S}_{op}$ given by $(2.5)-(2.8)$ and $op$ standing for $P^*$, $R$, $\mathbf{A}$ and $\tilde{A}$, respectively, we have

\[
\begin{align*}
\hat{s}_{1d}^{op} &= \min(j_d : j = (j_1, j_2, \ldots, j_{nd}) \in \mathcal{S}_{op}), \\
\hat{s}_{2d}^{op} &= \max(j_d : j = (j_1, j_2, \ldots, j_{nd}) \in \mathcal{S}_{op})
\end{align*}
\]  

with $d = 1, 2, \ldots, nd$. The following set

\[
\hat{\mathcal{S}}_{op} = \prod_{d=1}^{nd} \{ s_{1d}^{op}, s_{1d}^{op} + 1, \ldots, s_{2d}^{op} \}
\]  

is called the scope of $\mathcal{S}_{op}$, and $s_{1d}^{op}$ and $s_{2d}^{op}$ are called the lower and upper bounds of $\mathcal{S}_{op}$ in direction $d$. Obviously, $\mathcal{S}_{op} \subset \hat{\mathcal{S}}_{op}$ and $\hat{\mathcal{S}}_{op}$ is the smallest 'rectangle' set which contains $\mathcal{S}_{op}$. For example, if the operator $\mathbf{A}$ defined on the grid $\mathcal{G}$ has a standard five-point stencil, then for a given point $i \in \mathcal{G}$,

\[
\begin{align*}
\mathcal{S}_A &= \{(0,0), (0, \pm 1), (\pm 1, 0)\}; \\
\hat{\mathcal{S}}_A &= \{(0,0), (0, \pm 1), (\pm 1, 0), (\pm 1, \pm 1)\}; \\
s_{11}^A &= -1, \ s_{21}^A = 1, \ s_{12}^A = -1, \ s_{22}^A = 1.
\end{align*}
\]  

$\mathcal{S}_A$ and $\hat{\mathcal{S}}_A$ can be represented by the following stencils:

\[
[\mathcal{S}_A] = \begin{bmatrix} * & \star & * \\ * & \star & * \end{bmatrix}, \quad [\mathcal{S}(\mathcal{S}_A)] = \begin{bmatrix} * & * & * \\ * & \star & * \end{bmatrix},
\]

6
with ' * ' here denoting \( j \in S_A \) or \( j \in \hat{S}_A \), and with the square indicating the reference element. To make the above notations and definitions clearer, we give another example:

\[
\begin{align*}
S_{P^*} &= \{(0,0),(0,\pm1),(0,-2),(-1,0),(-1,\pm1),(-1,-2),(-2,0),(-2,\pm1),(-2,-2)\}; \\
\hat{S}_{P^*} &= S_{P^*}; \\
\hat{s}_{11}^{P^*} &= -2, \; s_{21}^{P^*} = 0, \; s_{12}^{P^*} = -2, \; s_{22}^{P^*} = 1; \\
[S_{P^*}] &= [\hat{S}_{P^*}] = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}.
\end{align*}
\]

(5.5)

5.2 The Characteristic Function of a Structure

For a given \( j \in \hat{S}_{op} \), in order to determine whether or not \( j \in S_{op} \), a natural way is to compare \( j \) with the elements of \( S_{op} \) one by one. Another way may be easier and is as follows. We define a characteristic function \( C_{op}(j) : \hat{S}_{op} \rightarrow \mathbb{N} \cup \{0\} \) as follows:

\[
C_{op} = \begin{cases} 
1 & \text{if } j \in S_{op} \\
0 & \text{otherwise},
\end{cases}
\]

(5.6)

where \( op \) may be \( A \), \( P^* \) or \( R \). The characteristic function of \( \hat{A} \) is defined differently:

\[
C_{\hat{A}}(j) = \begin{cases} 
c > 0 & \text{if } j \in S_{\hat{A}} \setminus S_A \\
C_A(j) & \text{if } j \in S_{\hat{A}} \cap S_A \\
0 & \text{if } j \notin S_{\hat{A}}.
\end{cases}
\]

(5.7)

The value of \( c \) will be specified later. In order to determine if \( j \in S_{op} \), we check whether \( C_{op}(j) = 0 \) or not. \( C_{\hat{A}}(j) \) gets a value \( c \) when enlargement of stencil of \( \hat{A} \) occurs. \( c \) is chosen differently from the non-zero value of \( C_A \) in order to distinguish the new members in the stencil of \( \hat{A} \) with the old members in the stencil of \( A \), so that we know how the enlargement occurs.

Using the concepts of scopes and characteristic functions, to input \( S_{op} \), for instance \( S_A \) of the standard five-point stencil given by (5.4), we can input \( C_A(j) \) by giving the following data:

\[
\begin{array}{ccc}
-1 & 1 & -1 & 1 \\
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0
\end{array}
\]

by which we mean

\[
\begin{align*}
& s_{11}^{A} = -1, \; s_{21}^{A} = 1, \; s_{12}^{A} = -1, \; s_{22}^{A} = 1, \\
& C_A((\pm1,\pm1)) = 0, \\
& C_A((\pm1,0)) = C_A((0,\pm1)) = C_A((0,0)) = 1.
\end{align*}
\]
This method of defining \( S_A \) by means of the scope and \( C_A(j) \) gives a clear image of how \( S_A \) is arranged.

### 5.3 Storage of Structures and Characteristic Functions

Let the structures \( S_R, S_A \) and \( S_{P*} \) be stored in the integer arrays \( S_R(1 : nd, 1 : n_R), S_A(1 : nd, 1 : n_A) \) and \( S_{P*}(1 : nd, 1 : n_{P*}) \), respectively; \( n_R, n_A \) and \( n_{P*} \) are the number of elements in \( S_R, S_A \) and \( S_{P*} \). For example, if the elements of \([A]\) are numbered as follows

\[
\begin{bmatrix}
5 \\
2 \\
3 \\
4 \\
1
\end{bmatrix}
\]

then \( S_A(1,4) = 1, S_A(2,4) = 0 \), indicating that \( j = (1,0) \) for element 4. The values of the characteristic function \( C_A \) are given by the integer array \( C_A(s_{A1}^A : s_{A2}^A, s_{A1}^A : s_{A3}^A : s_{A4}^A) \) (assuming \( nd = 3 \)). The integer array \( S_A(1 : nd, 1 : n_A) \) is used to store the structure \( S_A \), with \( n_A \) the number of elements in \( S_A \), and the integer array \( C_A(s_{A1}^A : s_{A2}^A, s_{A1}^A : s_{A3}^A : s_{A4}^A) \) to store the characteristic function \( C_A \). Because \( n_A \), \( s_{A1}^A \) and \( s_{A4}^A \) are not known prior to the computation of \( S_A \), they have to be assumed, because some computer languages such as FORTRAN do not allow dynamic declaration of array length. In fact, the array \( C_A \) and \( C_A \) are both declared as \( C_A(s_1 : s_2, s_1 : s_2, s_1 : s_2) \) and \( C_A(s_1 : s_2, s_1 : s_2, s_1 : s_2) \), with \(-s_1 = s_2 = 2\); the array \( S_A \) is declared as \( S_A(1 : nd, 1 : n_A') \), with \( n_A' \) given by \((s_2 - s_1 + 1)^{nd} \). It is easy to increase \(|s_1|\) and \(|s_2|\) if larger stencils have to be handled. The program STRURAP checks whether \(|s_1|\) and \(|s_2|\) are large enough. For our choices of \( R \) and \( P^* \), this choice of \( s_1 \) and \( s_2 \) seems to be sufficient.

### 5.4 Loop over a Structure

Let us take the for \( m \in S_R \) do loop as an example. In \( nd \) dimensions, a displacement in \( S_R \) is expressed by \( nd \) integers, each for a separate direction, according to our convention given in [7] that associates grid points and grid cell index in groups. So the loop

\[
\text{for } m \in S_R \text{ do}
\]

\[
\text{...}
\]

\[
\text{od}
\]

is equivalent to

\[
\text{for } i_R = 1 \text{ until } n_R \text{ do}
\]

\[
\text{for } d = 1 \text{ until } nd \text{ do}
\]

\[
m(d) = S_R(d, i_R)
\]

\[
\text{...}
\]

\[
\text{od}
\]

\text{...}

\[
\text{od}
\]
5.5 Even Number Check

For a given \( n \in \mathbb{Z}^{nd} \), to see if \( n/2 \in \mathbb{Z}^{nd} \) we use the following code:

\[
\text{even} = \text{true} \\
\text{for } d = 1 \text{ until } nd \text{ do} \\
\quad \text{even} = \text{even and } \text{mod}(n(d), 2) \text{ eq 0} \\
\text{od}
\]

If \( \text{even} \) equals \( \text{true} \) after the loop we have \( n/2 \in \mathbb{Z}^{nd} \) else \( n/2 \notin \mathbb{Z}^{nd} \).

5.6 The Program STRURAP

Applying the concepts introduced above, the algorithm STRURAP may be formulated in quasi-Algol as follows:

\[
\text{Program STRURAP} \\
\text{comment Calculation of } S_A \text{ for } nd = 1, 2 \text{ or } 3 \\
\text{comment } S_R, S_A \text{ and } S_{P^*} \text{ are provided. Choose } -s_1 = s_2 = 2 \\
\text{begin } C_A = 0 \\
\quad i_A = 0 \\
\quad \text{for } k = nd + 1 \text{ until } 3 \text{ do} \\
\quad 
\quad \quad n(d) = 0 \\
\quad \text{od} \\
\text{enlarge = false} \\
\quad \text{for } i_{P^*} = 1 \text{ until } n_{P^*} \text{ do} \\
\quad \quad \text{for } d = 1 \text{ until } nd \text{ do} \\
\quad \quad 
\quad \quad \quad q(d) = S_{P^*}(d, i_{P^*}) \\
\quad \quad \text{od} \\
\quad \text{for } i_R = 1 \text{ until } n_R \text{ do} \\
\quad \quad \text{for } d = 1 \text{ until } nd \text{ do} \\
\quad \quad 
\quad \quad \quad m(d) = S_R(d, i_R) \\
\quad \quad \text{od} \\
\quad \text{for } i_A = 1 \text{ until } n_A \text{ do} \\
\quad \quad \text{even = true} \\
\quad \quad \text{for } d = 1 \text{ until } nd \text{ do} \\
\quad \quad 
\quad \quad \quad k(d) = S_A(d, i_A) \\
\quad \quad \quad n(d) = m(d) + k(d) - q(d) \\
\quad \quad \quad \text{even = even and } \text{mod}(n(d), 2) \text{ eq 0} \\
\quad \quad \text{if (even) then} \\
\quad \quad 
\quad \quad \quad n(d) = n(d)/2 \\
\quad \quad \quad \text{if } (s_1 \text{ gt } n(d) \text{ or } s_2 \text{ lt } n(d)) \text{ then} \\
\quad \quad \quad \quad \text{write error message} \\
\quad \quad \quad \quad \text{stop} \\
\quad \quad \quad \text{end if}
\]
end if
od
if (even) then
  if $C_A(n(1), n(2), n(3)) \neq 0$ then
    $i_A = i_A + 1$
  for $d = 1$ until $nd$ do
    $S_{\tilde{A}}(d, i_A) = n(d)$
  od
  if $C_A(n(1), n(2), n(3)) \neq 0$ then
    $C_{\tilde{A}}(n(1), n(2), n(3)) = C_A(n(1), n(2), n(3))$
  else
    $C_{\tilde{A}}(n(1), n(2), n(3)) = c$
    enlarge = true
  end if
end if
end if
end od od od
end STRURAP

Note that the divisions involved are integer division. If enlarge equals true then the stencil of $\tilde{A}$ is larger than that of $A$. The value of $c$ can be chosen to be 2, since there are only two grid levels. If there are $l_f$ grids, one may choose $c = l_f - l + 1$ on grid level $l$, with $l_f$ the finest grid level index.

The above code is valid for $nd = 1, 2$ or 3, because the number of subscripts of the arrays for the characteristic functions $C_A$ and $C_{\tilde{A}}$ is 3.

6 Quasi-Algol Formulation of the Algorithm CALRAP

6.1 Definitions of Grids

We restrict ourselves to rectangular or cuboid grids. A grid can be defined by specifying the lower and upper bounds in each direction of the grid. Here we define $\mathcal{G}$ and $\mathcal{G}$ in this way. The definition of other grids or subgrids follows the same rule.

$\mathcal{G}$ and $\mathcal{G}$ are specified by the arrays $G_c(1:2, 1:nd)$ and $G_f(1:2, 1:nd)$, respectively. For example, $G_c(1,d)$ and $G_c(2,d)$ are the lower and upper bounds in direction $d$ of the grid $\mathcal{G}$, with $d = 1, 2, \ldots, nd$.

6.2 Computation of $\tilde{G}_1$ and $\tilde{G}_2$

Let the array $G_1(1:2, 1:nd)$ store the lower and upper bounds of $\tilde{G}_1$ in each direction. $i \in \tilde{G}_1$ means

$$G_c(1,d) \leq i(d) \leq G_c(2,d), \quad d = 1, 2, \ldots, nd, \quad (6.1)$$
$$G_f(1,d) \leq 2 \cdot i(d) + m(d) \leq G_f(2,d) \quad d = 1, 2, \ldots, nd. \quad (6.2)$$
Figure 6.1: A partition of a grid in two dimensions

More precisely, a given grid $\mathcal{G}$ is partitioned in parts $\mathcal{G}(b)$, $b \in \mathcal{B}$, as follows. Each $\mathcal{G}(b)$ is a tensor product of one-dimensional grids $\mathcal{G}(b_d, d)$:

$$\mathcal{G}(b) = \prod_{d=1}^{nd} \mathcal{G}(b_d, d)$$  \hspace{1cm} (6.12)

where

$$\mathcal{G}(b_d, d) = \{i \in \mathcal{Z} : i1 \leq i \leq i2\}$$  \hspace{1cm} (6.13)

with

$$i1 = i2 = G(1, d) \text{ if } b_d = -1,$$
$$i1 = G(1, d) + 1, \quad i2 = G(2, d) - 1 \text{ if } b_d = 0,$$  \hspace{1cm} (6.14)
$$i1 = i2 = G(2, d) \text{ if } b_d = 1.$$

This partition satisfies

$$\mathcal{G}(b_d, d) \cap \mathcal{G}(b'_d, d) = \emptyset \text{ for } b_d \neq b'_d.$$  \hspace{1cm} (6.15)

The same applies to subdivisions of grids $\mathcal{G}$, $\mathcal{G}_1$, $\mathcal{G}_2$. But, the partition of $\mathcal{G}_2$ is called $\mathcal{G}_3(b)$. The lower and upper bounds of $\mathcal{G}_3(b)$ in direction $d$ are given by $G_3(1, b_d, d)$ and $G_3(2, b_d, d)$ respectively.

If $\mathcal{G}$ is not large enough, most likely on the coarsest grid, we could have $G_2(1, d) = G_2(2, d)$. When this happens, $\mathcal{G}_3(b_d, d)$ should be made empty for either $b_d = -1$ or $b_d = 1$, because of the partition given by (6.12)-(6.15). As for which should be made empty we check $G_2(1, d)$ and $G_2(2, d)$: if $G_2(1, d) = G_2(1, d)$, $\mathcal{G}_3(1, d)$ is made empty; if $G_2(2, d) = G_2(1, d)$, $\mathcal{G}_3(-1, d)$ is made empty; if $G_2(1, d)$ (or $G_2(2, d)$) is not equal to either $G_2(1, d)$ nor $G_2(2, d)$, either $\mathcal{G}_3(-1, d)$ or $\mathcal{G}_3(1, d)$ can be made empty (here we choose to set $\mathcal{G}_3(-1, d)$ empty). To make a empty grid we just let the lower bound be greater than the upper bound in any direction.

As already explained, $\mathcal{G}_3(0)$ is certainly in the interior of $\mathcal{G}$, since $\mathcal{G}_2 \subseteq \mathcal{G}$; $i + n$ is also certainly in the interior of $\mathcal{G}$ if $i \in \mathcal{G}_3(0)$, since $i + n \in \mathcal{G}$ holds for $i \in \mathcal{G}_2$. A useful consequence
resulting from the above partition of \( \mathcal{G}_2 \) is that for \( i \in \mathcal{G}_3(b) \) there is only one boundary case for both \( R(i, m) \) and \( P^*(i + n, q) \), i.e., we have \( i \in \mathcal{G}(b') \) and \( i + n \in \mathcal{G}(b'') \) and constant \( R \) and \( P^* \) given by \( R_{b'}(m) \) and \( P^*_{b''}(q) \).

6.4 Boundary Case Selection

Of course \( i_d \in \mathcal{G}_3(b_d, d) \), for all \( d = 1, 2, \ldots, nd \) when \( i = (i_1, i_2, \ldots, i_{nd}) \in \mathcal{G}_3(0) \). \( i \in \mathcal{G}_3(0) \) gives \( i_d \in \mathcal{G}_3(0, d) \subseteq \mathcal{G}(0, d) \). So in direction \( d \) only for \( b_d = \pm 1 \) we have to check which boundary case occurs for \( R(i, m) \) and \( P^*(i + n, q) \) with \( i_d \in \mathcal{G}_3(b_d, d) \). Two possibilities exist for \( \mathcal{G}_3(b_d, d) \): either \( \mathcal{G}_3(b_d, d) \subseteq \mathcal{G}(b_d, d) \) or \( \mathcal{G}_3(b_d, d) \subseteq \mathcal{G}(0, d) \). The case that \( \mathcal{G}_3(b_d, d) \subseteq \mathcal{G}(b_d, d) \) cannot happen for the following reason. If \( \bigcup_{d=-1}^{1} \mathcal{G}_3(b_d, d) \) consists of more than one point in direction \( d \), then \( \mathcal{G}_3(b_d, d) \subseteq \mathcal{G}(-b_d, d) \) would imply \( \mathcal{G}_2 \subseteq \mathcal{G} \), which is false. If \( \bigcup_{d=-1}^{1} \mathcal{G}_3(b_d, d) \) consists of only one point which is in \( \mathcal{G}(b_d, d) \), then \( \mathcal{G}_3(-b_d, d) = \emptyset \) according to the construction of \( \mathcal{G}_3 \) described in section 6.3. So, if \( \mathcal{G}_3(b_d, d) \subseteq \mathcal{G}(b_d, d) \), then \( r_d(b_d) = b_d \), otherwise \( r_d(b_d) = 0 \); if \( \{i + n : i \in \mathcal{G}_3(b_d, d)\} \subseteq \mathcal{G}(b_d, d) \), then \( p_d(b_d) = b_d \), otherwise \( p_d(b_d) = 0 \). The arrays \( r(-1 : 1, 1 : nd) \) and \( p(-1 : 1, 1 : nd) \) are used to store \( r(b) = (r_1(b_1), r_2(b_2), \ldots, r_{nd}(b_{nd})) \) and \( p(b) = (p_1(b_1), p_2(b_2), \ldots, p_{nd}(b_{nd})) \), with \( r(b_d, d) \) referring to \( r_d(b_d) \) and \( p(b_d, d) \) to \( p_d(b_d) \). The boundary case selection is carried out as follows:

```
if (G3(1, -1, d) eq Gc(1, d)) then
  r(-1, d) = -1
else
  r(-1, d) = 0
end if
if (G3(1, 1, d) eq Gc(2, d)) then
  r(1, d) = 1
else
  r(1, d) = 0
end if
r(0, d) = 0
if (G3(1, -1, d) + n(d) eq Gc(1, d)) then
  p(-1, d) = -1
else
  p(-1, d) = 0
end if
if (G3(1, 1, d) + n(d) eq Gc(2, d)) then
  p(1, d) = 1
else
  p(1, d) = 0
end if
p(0, d) = 0
```

6.5 Loop over Subgrids

Because the partition of the grid \( \mathcal{G}_2 \) is known, the loop over all subgrids \( \mathcal{G}_3(b) \) for \( b \in B \)
for $b \in B$ do
  for $i \in \tilde{G}_3(b)$ do
    ...
  od od

is realized, in three dimensions, by

for $b_3 = -1$ until 1 do
  for $b_2 = -1$ until 1 do
    for $b_1 = -1$ until 1 do
      for $i_3 = G_3(1, b_3, 3)$ until $G_3(2, b_3, 3)$ do
        for $i_2 = G_3(1, b_2, 2)$ until $G_3(2, b_2, 2)$ do
          for $i_1 = G_3(1, b_1, 1)$ until $G_3(2, b_1, 1)$ do
            ...
          od od od od od od

The above can be also used for the number of space dimensions is less than 3, if the array $G_3$ is properly initialized in the unused direction(s).

6.6 The Program CALRAP

Combining the above implementations, algorithm 4 CALRAP can be formulated in quasi-Algol as follows:

\textit{Program CALRAP}

comment Computation of $\tilde{A} = \text{RAP}$
comment $G$, $\tilde{G}$, $S_R$, $\bar{S}_A$, $S_{p*}$ and $C_A$ are defined; $R$, $A$ and $P^*$ are known;
$-t_1 = \omega_2 = 2$
begin $A = 0$
  comment Initialize some arrays in unused directions
  for $d = nd + 1$ until 3 do
    $k(d) = 0$
    for $b_d = -1$ until 1 do
      $r(b_d, d) = 0$
      $p(b_d, d) = 0$
      $G_3(1, b_d, d) = 1$
      $G_3(2, b_d, d) = 1$
    od od
  comment Loop over $n$
  for $i_R = 1$ until $n_R$ do
    for $d = 1$ until $nd$ do
      $m(d) = S_R(d, i_R)$
      $G_1(1, d) = \max(G_x(1, d), (G_j(1, d) - m(d) + 1)/2)$
      $G_1(2, d) = \min(G_x(2, d), (G_j(2, d) - m(d))/2)$
    od
  comment Loop over $n$

14
for \( i_{\tilde{A}} = 1 \) until \( n_{\tilde{A}} \) do
  for \( d = 1 \) until \( nd \) do
    \( n(d) = S_{\tilde{A}}(d, i_{\tilde{A}}) \)
    \( G_2(1, d) = \max(G_1(1, d), G_c(1, d) - n(d)) \)
    \( G_2(2, d) = \min(G_1(2, d), G_c(2, d) - n(d)) \)

comment Partitioning of \( G_2 \) in direction \( d \)

\[
\begin{align*}
G_3(1,-1, d) &= G_2(1, d) \\
G_3(2,-1, d) &= G_2(1, d) \\
G_3(1,1, d) &= G_4(2, d) \\
G_3(2,1, d) &= G_4(2, d) \\
G_3(1,0, d) &= G_3(1, d) + 1 \\
G_3(2,0, d) &= G_3(2, d) - 1 \\
\end{align*}
\]

comment Make \( \tilde{G}_3(-1, d) \) or \( \tilde{G}_3(1, d) \) empty if necessary

if \( (G_2(1, d) \text{ eq } G_2(2, d)) \) then
  if \( (G_3(1, -1, d) \text{ eq } G_c(1, d)) \) then
    \( G_3(1,1, d) = 1 \)
    \( G_3(2,1, d) = -1 \)
  else
    \( G_3(1, -1, d) = 1 \)
    \( G_3(2, -1, d) = -1 \)
end if

comment Select boundary case in \( \tilde{G}_3(-1, d) \) and \( \tilde{G}_3(1, d) \) for \( R(i, m) \)

if \( (G_3(1, -1, d) \text{ eq } G_c(1, d)) \) then
  \( r(-1, d) = -1 \)
else
  \( r(-1, d) = 0 \)
end if

if \( (G_3(1,1, d) \text{ eq } G_c(2, d)) \) then
  \( r(1, d) = 1 \)
else
  \( r(1, d) = 0 \)
end if

\( r(0, d) = 0 \)

comment Select boundary case \( \tilde{G}_3(-1, d) \) and \( \tilde{G}_3(1, d) \) for \( P^*(i + n, q) \)

if \( (G_3(1, -1, d) + n(d) \text{ eq } G_c(1, d)) \) then
  \( p(-1, d) = -1 \)
else
  \( p(-1, d) = 0 \)
end if

if \( (G_3(1,1, d) + n(d) \text{ eq } G_c(2, d)) \) then
  \( p(1, d) = 1 \)
else
\[ p(1, d) = 0 \]
\[ p(0, d) = 0 \]
\[ \text{end if} \]
\[ q(d) = S_{p^*}(d, i_{p^*}) \]
\[ k(d) = q(d) + 2 \cdot n(d) - m(d) \]
\[ \text{outside} = \text{outside and (} k(d) \text{ lt } s_1 \text{ or } s_2 \text{ lt } k(d) \text{)} \]
\[ \text{od} \]
\[ \text{if (not outside) then} \]
\[ \text{comment Check whether } k \in S_A \]
\[ \text{if } (C_A(k(1), k(2), k(3)) \text{ ne } 0) \text{ then} \]
\[ \text{comment Specify } b \text{ for } \tilde{G}_3(b) \]
\[ \text{for } b_3 = -1 \text{ until } 1 \text{ do} \]
\[ \text{for } b_2 = -1 \text{ until } 1 \text{ do} \]
\[ \mu = R(r(b_1, 1), r(b_2, 2), r(b_3, 3), i_{R^*}) \]
\[ P^*(p(b_1, 1), p(b_2, 2), p(b_3, 3), i_{p^*}) \]
\[ \text{comment Loop over } \tilde{G}_3(b) \]
\[ \text{for } i_3 = G_3(b_3, 1, 3) \text{ until } G_3(b_3, 2, 3) \text{ do} \]
\[ \text{for } i_2 = G_3(b_2, 1, 2) \text{ until } G_3(b_2, 2, 2) \text{ do} \]
\[ \text{for } i_1 = G_3(b_1, 1, 1) \text{ until } G_3(b_1, 2, 1) \text{ do} \]
\[ \tilde{A}(i, n) = \tilde{A}(i, n) + \mu \cdot A(2i + m, k) \]
\[ \text{od od od od od od} \]
\[ \text{end if} \]
\[ \text{end if} \]
\[ \text{end CALRAR} \]

7 Generalization

In this section, we discuss the generalization of the algorithms STRURAP and CALRAR to a set of partial differential equations, the Navier-Stokes equations (maybe accompanied by transport equations), discretized on a staggered grid.

7.1 The Equation Index Set and Variable Index Set

Here we introduce some concepts that will be used in the generalization. Without loss of generality, let us take the Boussinesq equations in two dimensions as an example. Instead of only one scalar partial differential equation and one variable, we now have four partial
differential equations. For simplicity we introduce an index to refer to them. Using the letter \( \alpha \) for the equation index, we stipulate that the momentum equation in the \( \xi^1 \)-direction and the \( \xi^2 \)-direction have respectively the index values \( \alpha = 1 \) and \( 2 \); the energy equation has the index value \( \alpha = 3 \); the continuity equation has the index value \( \alpha = 4 \). So by equation 1 we mean the momentum equation in the \( \xi^1 \)-direction. There are four variables, i.e., the velocities in the \( \xi^1 \)-direction and the \( \xi^2 \)-direction, the temperature \( T \) and the pressure \( p \), which are indexed by \( \beta \in \{1, 2, 3, 4\} \). The set of values of the index \( \alpha \) is called the equation index set \( \mathcal{E} \) and the set of values of the index \( \beta \) for \( \alpha \in \mathcal{E} \) is called the variable index set \( \mathcal{V}_\alpha \). Note that \( \mathcal{V}_\alpha \) depends on \( \alpha \). Not every equation has all variables present, for example, in equation 4, \( \mathcal{V}_4 = \{1, 2\} \). \( \mathcal{E} \equiv \{1, 2, \cdots, n_E\} \), where \( n_E \) is the number of partial differential equations. In our example, \( n_E = 4 \). The symbol \( V \) is used to represent the set \( \{V_1, V_2, \cdots, V_{n_E}\} \) and is also called the variable index set.

A discretization [3] of the partial differential equations on a staggered grid using contravariant variables results in the following discrete system:

\[
A x = b. \tag{7.1}
\]

More detailed,

\[
\begin{pmatrix}
A^{11} & A^{12} & A^{13} & A^{14} \\
A^{21} & A^{22} & A^{23} & A^{24} \\
A^{31} & A^{32} & A^{33} & A^{34} \\
A^{41} & A^{42} & A^{43} & A^{44}
\end{pmatrix}
\begin{pmatrix}
V^1 \\
V^2 \\
V^3 \\
V^4
\end{pmatrix}
= \begin{pmatrix}
b^1 \\
b^2 \\
b^3 \\
b^4
\end{pmatrix}. \tag{7.2}
\]

\( A^\alpha_\beta (\beta \in \mathcal{V}_\alpha, \alpha \in \mathcal{E}) \) are all sub-matrices. \( V^\beta \) are discrete variable vectors and \( b^\alpha \) are the right-hand sides. For the Boussinesq equations, \( A^{34} = A^{43} = A^{44} = 0 \). We have four discretized equation systems. A discretized equation set with index \( \alpha \) is a set of algebraic equations obtained by discretizing on a grid the corresponding partial differential equation with the same equation index \( \alpha \). Similarly, a discrete variable (or vector) with index \( \beta \) is the variable with index \( \beta \) defined on a grid. Therefore, by equation set \( \alpha \) we mean the equation set

\[
\sum_{\beta \in \mathcal{V}_\alpha} A^\alpha_\beta V^\beta = b^\alpha \tag{7.3}
\]

and by variable \( \beta \) we mean \( V^\beta \).

Galerkin coarse grid approximation takes place so that the coarse grid equation is given by

\[
\begin{pmatrix}
R^1 A^{11} P^{11} & R^1 A^{12} P^{12} & R^1 A^{13} P^{13} & R^1 A^{14} P^{14} \\
R^2 A^{21} P^{21} & R^2 A^{22} P^{22} & R^2 A^{23} P^{23} & R^2 A^{24} P^{24} \\
R^3 A^{31} P^{31} & R^3 A^{32} P^{32} & R^3 A^{33} P^{33} & R^3 A^{34} P^{34} \\
R^4 A^{41} P^{41} & R^4 A^{42} P^{42} & R^4 A^{43} P^{43} & R^4 A^{44} P^{44}
\end{pmatrix}
\begin{pmatrix}
V^1 \\
V^2 \\
V^3 \\
V^4
\end{pmatrix}
= \begin{pmatrix}
\tilde{b}_1 \\
\tilde{b}_2 \\
\tilde{b}_3 \\
\tilde{b}_4
\end{pmatrix}. \tag{7.4}
\]

where \( R^\alpha \) is the restriction operator for equation set \( \alpha \) and \( P^\alpha_\beta \) is the prolongation operator for variable \( \beta \) in equation set \( \alpha \). Note that \( P^\alpha_\beta \) may be chosen such that \( P^\alpha_\beta \neq P^{\alpha'}_\beta \) when \( \alpha' \neq \alpha \) in order to enlarge the stencils of the corresponding sub-matrices they are applied to [7]. We see that to perform RAP for a matrix derived from a set of partial differential
equations is to perform RAP for each sub-matrix in the matrix; for this we can use the
program CALRAP specified before.

7.2 Generalization of the Program STRURAP

Because the non-zero pattern of each sub-matrix on a coarse grid should be determined, the
program STRURAP as described in section 5.6 is used like a subroutine, with \(S_R, S_A\), and
\(S_P\) as its input and \(S_A\) as its output. So the program STRURAP is revised at the first line
as follows:

\[
\text{Subroutine STRURAP}(S_R, S_A, S_P, S_A)
\]

At each use of the subroutine, to the subroutine are sent the structure \(S_R\) of the restriction
operator corresponding to the equation set index of the sub-matrix, the structure \(S_A\) of
the sub-matrix and the structure \(S_P\) of the prolongation operator corresponding to the
variable index and equation set index of the sub-matrix. Let \(S_R^\beta, S_A^\beta, S_P^\beta\) and \(S_A^\beta\) refer
to the structures of the restriction operator \(R^\beta\), the sub-matrix \(A^\beta\), the transpose of the
prolongation operator \(P^\beta\) and the sub-matrix \(A^\beta\). The following simple algorithm computes
\(S_A^\beta\):

\[
\text{Algorithm COMPSTR}
\]

\[
\begin{align*}
&\text{comment Computation of } S_A^\beta, \forall \beta \in \mathcal{V}_\alpha, \forall \alpha \in \mathcal{E} \\
&\text{comment } \mathcal{V}, \mathcal{E}, S_R, S_A, S_P, S_A^\beta, \forall \beta \in \mathcal{V}_\alpha, \forall \alpha \in \mathcal{E} \text{ are defined} \\
&\text{begin} \\
&\text{for } \alpha \in \mathcal{E} \text{ do} \\
&\text{for } \beta \in \mathcal{V}_\alpha \text{ do} \\
&\text{call STRURAP}(S_R^\beta, S_A^\beta, S_P^\beta, S_A^\beta) \\
&\text{od od} \\
&\text{end COMPSTR}
\end{align*}
\]

7.3 The Program

A quasi-Algol program of the algorithm COMPSTR is implemented as follows. The equation
index set \(E\) can be simply defined by giving \(n_E\). To define the variable index set \(V\), the integer
array \(V_o(1 : n_E + 1, 1 : n_E)\) is used. For a given \(\alpha \in \mathcal{E}, V_o(1, \alpha)\) is the number of non-zero
sub-matrices in equation set \(\alpha\) and \(V_o(1 + \beta', \alpha)\) is \(\beta\), the value of the variable index of the
\(\beta'\)-th non-zero sub-matrix. The integer arrays \(S_R, S_A, S_P, S_A^\beta\) store the displacements of
the structures \(S_R^\beta, S_A^\beta, S_A^\beta\) and \(S_P^\beta\), and the integer arrays \(n_R, n_A, n_A^\beta\) and \(n_P^\beta\) store the
number of elements in these structures, for all \(\beta \in \mathcal{V}_\alpha\) and for all \(\alpha \in \mathcal{E} : S_R(\alpha), S_A(\beta, \alpha),
S_A^\beta(\beta, \alpha)\) and \(S_P^\beta(\beta, \alpha)\) give the displacements in the structures \(S_R^\beta, S_A^\beta, S_A^\beta\) and \(S_P^\beta\), and
\(n_R(\alpha), n_A(\beta, \alpha), n_A^\beta(\beta, \alpha)\) and \(n_P^\beta(\beta, \alpha)\) give the number of elements in these structures, for
a given \(\alpha\) and \(\beta\). Here, we just use, for example, the simple notation \(S_A(\beta, \alpha)\) to refer to all
displacements of \(S_A^\beta\), without giving more detailed information about how the storage of \(S_A^\beta\)
is arranged in the array \(S_A\) and how the array \(S_A\) is passed to the subroutine. The reason
is that if all these are to be explained, too many programming details have to be concerned,
which, of course, is specific to the computer language used to realize the algorithm and should be given somewhere else such as in a program guide.

To pass a structure to the subroutine, both the number of elements of the structure and the displacements of the structure need to be passed to the subroutine. So the first line of the program STRURAP is modified as:

\[
\text{Subroutine STRURAP}(n_d, n_R, S_R, n_A, S_A, n_{P*}, S_{P*}, n_{\tilde{A}}, S_{\tilde{A}})
\]

The algorithm COMPSTR corresponds to the following program:

\[
\text{Program COMPSTR}\\
\text{comment} \text{ Computation of } S_{\tilde{A}}^{\hat{\beta}}, \forall \hat{\beta} \in V_{\tilde{A}}, \forall \alpha \in E\\
\text{comment} V, E, S_{\tilde{A}}^{\hat{\beta}}, S_{P*}^{\hat{\beta}}, \forall \hat{\beta} \in V_{\alpha}, \forall \alpha \in E \text{ are defined}\\
\text{begin}\\
\text{for } \alpha = 1 \text{ until } n_E \text{ do}\\
\quad \text{for } \beta = 1 \text{ until } V_\alpha(1, \alpha) \text{ do}\\
\quad \quad \beta = V_\alpha(1 + \beta, \alpha)\\
\quad \quad \text{call STRURAP}(n_d, n_R(\alpha), S_R(\alpha), n_A(\beta, \alpha), S_A(\beta, \alpha), n_{P*}(\beta, \alpha), S_{P*}(\beta, \alpha), n_{\tilde{A}}(\beta, \alpha), S_{\tilde{A}}(\beta, \alpha))\\
\quad \text{od od}\\
\text{end COMPSTR}
\]

7.4 Generalization of the Algorithm CALRAP

Modification of the algorithm CALRAP

The generalization of CALRAP is more complicated than that of STRURAP, because the definitions of grid points are involved. The algorithm itself needs to be modified besides only the first line, and therefore cannot be simply employed as a subroutine. The staggering of grids leads to this modification, which is not encountered for a scalar partial differential equation. The modification is discussed here.

Let us consider the definition of grid points. On a fine grid, instead of one set of grid points \( \mathcal{G} \) for a single scalar equation, one may have \( n_E \) sets of grid points, \( \mathcal{G}_\alpha \) for \( \alpha \in \mathcal{E} \), each for a different equation set. Of course, \( \mathcal{G}_{\alpha'} \equiv \mathcal{G}_{\alpha} \) for some \( \alpha' \neq \alpha \) is allowed. The same applies to the definition of grid points on a coarse grid. Then look at the computation of \( R^\alpha A^\alpha P^\alpha \). \( R^\alpha \) is applied to equation set \( \alpha \) defined on \( \mathcal{G}^\alpha \) and \( \tilde{\mathcal{G}}^\alpha \). \( P^\beta \) is applied to variable set \( \beta \) defined on \( \mathcal{G}^\beta \) and \( \tilde{\mathcal{G}}^\beta \). So the computation uses two types of grid point sets \( \mathcal{G}^\alpha \) (or \( \tilde{\mathcal{G}}^\alpha \)) and \( \mathcal{G}^\beta \) (or \( \tilde{\mathcal{G}}^\beta \)).

Problems arise first in the computation of \( \tilde{\mathcal{G}}_2 \), when \( \beta \neq \alpha \). In algorithm 4 of the algorithm CALRAP, the computation of \( \tilde{\mathcal{G}}_1 \) is given by

\[
\tilde{\mathcal{G}}_1 = \{ i \in \mathcal{G} : 2i + m \in \mathcal{G} \}. \tag{7.5}
\]

Since we are considering the restriction of equation set \( \alpha \), the set of the grid points involved should be \( \mathcal{G}^\alpha \) and \( \tilde{\mathcal{G}}^\alpha \). We can write (7.5) more precisely as

\[
\tilde{\mathcal{G}}^\alpha_1 = \{ i \in \mathcal{G}^\alpha : 2i + m \in G^\alpha \}. \tag{7.6}
\]
Next we consider the prolongation of variable set $\beta$ in equation set $\alpha \mathbf{P}^{\alpha \beta}$. The computation of $\bar{G}_2^\alpha$ is actually defined by

$$\bar{G}_2^\alpha = \{ i \in \bar{G}_1^\alpha : i + n \in \bar{G}^\beta \}. \quad (7.7)$$

Here we require $i + n \in \bar{G}^\beta$ because the prolongation is applied to a variable defined on $\bar{G}^\beta$. What does $i + n$ mean for $i \in \bar{G}_1^\alpha$ but $i + n \in \bar{G}^\beta$ with $n \in \mathbb{Z}^{nd}$? Recalling that grid points in a grid point set are one-to-one associated with primary grid cells (cf. [7]), for a point $i^1 \in \bar{G}_1^\alpha$, there exist corresponding points $i^2, i^3, \ldots, i^{\text{nd}}$, which are all associated with primary cell $i$, as illustrated in figure 7.1. This association gives $i^1 = i^2 = \ldots = i^{\text{nd}} = i$. Therefore, in (7.7)

![Diagram of a grid with points labeled $i^1, i^2, i^3, i^4$ and notes indicating grid points and their relations.]

---

Figure 7.1: Primary cell $i$ and its associated grid points

by $i + n \in \bar{G}^\beta$ we actually mean that $i^\beta + n \in \bar{G}_1^\alpha$ for $i^\alpha \in \bar{G}^\alpha$, where $i^\beta$ is a grid point like $i^\alpha$ associated with the same cell $i$. For simplicity, we simply write $i^1, i^2, \ldots, i^{\text{nd}}$ as $i$, since their value is equal. Keeping this association in mind, we get the following generalization of algorithm 4 CALRAP for the computation of $\bar{A}^{\alpha \beta}$:

**Algorithm 5 CALRAP**($\bar{G}_1^\alpha, \bar{G}^\beta, \mathcal{G}_A^\alpha, \mathcal{S}_R^\alpha, \mathcal{S}_P^\alpha, \mathcal{S}_F^\alpha, \mathcal{R}^\alpha, \mathcal{A}^{\alpha \beta}, \mathcal{P}^{\alpha \beta}, \mathcal{S}^{\alpha \beta}, \bar{A}^{\alpha \beta}$)

**Comment:** Calculation of $\bar{A}^{\alpha \beta} = \mathcal{R}^\alpha \mathcal{A}^{\alpha \beta} \mathbf{P}^{\alpha \beta}$

**Begin:**

1. For $m \in \mathcal{S}_R^\alpha$
   - $\bar{G}_1^\alpha = \{ i \in \bar{G}_1^\alpha : 2i + m \in \mathcal{G}_A^\alpha \}$
   - For $n \in \mathcal{S}_F^\alpha$
     - $\bar{G}_2^\alpha = \{ i \in \bar{G}_1^\alpha : i + n \in \bar{G}^\beta \}$
     - For $b \in \mathcal{B}$
       - Define $\bar{G}_3^\alpha(b)$
       - Comment: Construction of $r(b), p(b)$
     - For $b' \in \mathcal{B}$
       - If $\bar{G}_3^\alpha(b) \subseteq \bar{G}_3^\alpha(b')$ then
         - $r(b) = b'$
       - end if
       - If $\{ i + n : i \in \bar{G}_3^\alpha(b) \} \subseteq \bar{G}_3^\alpha(b')$ then
         - $p(b) = b'$
       - end if
od od
for q ∈ S_p^\beta do
    k = q + 2n - m
    if k ∈ S_A^\alpha then
        for b ∈ B do
            μ = R^\alpha_{(q)}(m) ⋅ P^\alpha\beta_{(p)}(q)
            for i ∈ G^\beta_3(b) do
                A^\alpha\beta(i, n) = A^\alpha\beta(i, n) + μ ⋅ A^\alpha\beta(2i + m, k)
            od od
        end if
    od od od
end CALRAP

The Subroutine CALRAP

Corresponding to the changes in algorithm 4 CALRAP, algorithm 5 CALRAP is changed as follows. Let the arrays G_{ce}(1 : 2, 1 : nd) and G_{cv}(1 : 2, 1 : nd) store the lower and upper bounds of \bar{G}^\alpha and \bar{G}^\beta, and let the array G_f(1 : 2, 1 : nd) store the lower and upper bounds of \bar{G}^\alpha. The subroutine CALRAP is given by


comment Computation of A^\alpha\beta = R A^\alpha\beta P^* A^\alpha\beta

comment - s_1 = s_2 = 2

begin A = 0
    comment Initialize some arrays in unused directions
    for d = nd + 1 until 3 do
        k(d) = 0
        for b_d = -1 until 1 do
            r(b_d, d) = 0
            p(b_d, d) = 0
            G_3(1, b_d, d) = 1
            G_3(2, b_d, d) = 1
        od od
    comment Loop over m
    for i_R = 1 until n_R do
        for d = 1 until nd do
            m(d) = S_R(d, i_R)
            G_1(1, d) = max(G_{ce}(1, d), (G_f(1, d) - m(d) + 1)/2)
            G_1(2, d) = min(G_{ce}(2, d), (G_f(2, d) - m(d))/2)
        od
    comment Loop over n
    for i_A = 1 until n_A do
        for d = 1 until nd do
            n(d) = S_A(d, i_A)
\[ G_2(1, d) = \max(G_1(1, d), G_{cv}(1, d) - n(d)) \]
\[ G_2(2, d) = \min(G_1(2, d), G_{cv}(2, d) - n(d)) \]

**comment** Partitioning \( \mathcal{G}_2 \) in direction \( d \)

\[ G_3(1, -1, d) = G_2(1, d) \]
\[ G_3(2, -1, d) = G_2(1, d) \]
\[ G_3(1, 1, d) = G_2(2, d) \]
\[ G_3(2, 1, d) = G_2(2, d) \]
\[ G_3(1, 0, d) = G_2(1, d) + 1 \]
\[ G_3(2, 0, d) = G_2(2, d) - 1 \]

**comment** Make \( \mathcal{G}_3(-1, d) \) or \( \mathcal{G}_3(1, d) \) empty if necessary

if \( (G_2(1, d) \text{ eq } G_2(2, d)) \) then
  if \( (G_3(1, -1, d) \text{ eq } G_{ce}(1, d)) \) then
    \[ G_3(1, 1, d) = 1 \]
    \[ G_3(2, 1, d) = -1 \]
  else
    \[ G_3(1, -1, d) = 1 \]
    \[ G_3(2, -1, d) = -1 \]
  end if
end if

**comment** Select boundary case in \( \mathcal{G}_3(-1, d) \) and \( \mathcal{G}_3(1, d) \) for \( R(i, m) \)

if \( (G_3(1, -1, d) \text{ eq } G_{ce}(1, d)) \) then
  \[ r(-1, d) = -1 \]
else
  \[ r(-1, d) = 0 \]
end if

if \( (G_3(1, 1, d) \text{ eq } G_{ce}(2, d)) \) then
  \[ r(1, d) = 1 \]
else
  \[ r(1, d) = 0 \]
end if

\( r(0, d) = 0 \)

**comment** Select boundary case \( \mathcal{G}_3(-1, d) \) and \( \mathcal{G}_3(1, d) \) for \( P^*(i + n, q) \)

if \( (G_3(1, -1, d) + n(d) \text{ eq } G_{cv}(1, d)) \) then
  \[ p(-1, d) = -1 \]
else
  \[ p(-1, d) = 0 \]
end if

if \( (G_3(1, 1, d) + n(d) \text{ eq } G_{cv}(2, d)) \) then
  \[ p(1, d) = 1 \]
else
  \[ p(1, d) = 0 \]
end if

\[ p(0, d) = 0 \]
od
comment Loop over q
for \(i_{p^*} = 1\) until \(n_{p^*}\) do
  outside = true
  for \(d = 1\) until \(ad\) do
    \(q(d) = S_{p^*}(d, i_{p^*})\)
    \(k(d) = q(d) + 2 \cdot n(d) - m(d)\)
    outside = outside and \((k(d) \lt s_1\) or \(s_2 \lt k(d))\)
  od
if (not outside) then
  comment Check whether \(k \in S_A\)
  if \((C_A(k(1), k(2), k(3))) \neq 0\) then
    comment Specify \(b\) for \(\hat{G}_3(b)\)
    for \(b_3 = -1\) until 1 do
      for \(b_2 = -1\) until 1 do
        for \(b_1 = -1\) until 1 do
          \(\mu = R(\tau(b_1, 1), \tau(b_2, 2), \tau(b_3, 3), i_R)\)
          \(P^*(p(b_1, 1), p(b_2, 2), p(b_3, 3), i_{p^*})\)
          comment Loop over \(\hat{G}_3(b)\)
          for \(i_2 = G_3(b_2, 1, 3)\) until \(G_3(b_3, 2, 3)\) do
            for \(i_2 = G_3(b_2, 1, 2)\) until \(G_3(b_2, 2, 2)\) do
              for \(i_1 = G_3(b_1, 1, 1)\) until \(G_3(b_1, 2, 1)\) do
                \(\hat{A}(i, n) = \hat{A}(i, n) + \mu \cdot A(2i + m, k)\)
              od
            od
          od
        od
      od
    od
  end if
end if
od od od od od od
end if
od od od
end CALRAP

The algorithm COMPRAP

With the previous changes, an algorithm called COMPRAP is easily designed, which uses algorithm 5 CALRAP as a subroutine:

\[\text{Algorithm COMPRAP}\]
\[\text{comment Computation of } \hat{A} = \text{RAPH}\]
\[\text{comment } V, \mathcal{E}, \hat{G}^\alpha, G^\alpha, S_R, S_A^\alpha, S_P^\alpha, \forall \beta \in V_\alpha, \forall \alpha \in \mathcal{E}, \text{ are defined}\]
\[\text{comment } S_A^\alpha, \forall \beta \in V_\alpha \text{ and } \forall \alpha \in \mathcal{E} \text{ are computed by using the algorithm COMPRAP-STR}\]
\[\begin{align*}
\text{begin} \\
\text{for } \alpha \in \mathcal{G} \text{ do} \\
\text{for } \beta \in V_\alpha \text{ do} \\
\text{call CALRAP}(\hat{G}^\alpha, \hat{G}^\beta, G^\alpha, G^\beta, S_R, S_A^\alpha, S_P^\alpha, R^\alpha, A^\alpha, P^\alpha, \alpha^\beta, S_A^\alpha, \hat{A}^\alpha) \\
\text{od}
\end{align*}\]
end COMPRAP

The program

We implement the algorithm COMPRAP as follows:

Program COMPRAP

comment $V, E, \tilde{b}^\alpha, g^\alpha, S_R^\alpha, S_A^\alpha, S_P^\alpha, \forall \beta \in V_\alpha, \forall \alpha \in E$, are defined by the array $V_\alpha$, the number $n_E$, the arrays $G_\alpha, G_f$, the number $n_R$ and the array $S_R$, the array $C_A$, the number $n_P$ and the array $S_P$, respectively. The operators $R^\alpha, A^\alpha \beta, P^\alpha \beta$ and $\hat{A}^\alpha \beta$ are referred to by $R(\alpha), A(\beta, \alpha), P^* (\beta, \alpha)$ and $\hat{A}(\beta, \alpha)$.

comment $S_A^\alpha, \forall \beta \in V_\alpha$ and $\forall \alpha \in E$ are computed by using the program COMP-STR and are given by the number $n_A$ and the array $S_A$.

begin

for $\alpha = 1$ until $n_E$ do

for $\beta' = 1$ until $V_\alpha(1, \alpha)$ do

$\beta = V_\alpha(1 + \beta', \alpha)$

call CALRAP($nd, G_\alpha(\alpha), G_c(\beta), G_f(\alpha), n_R(\alpha), S_R(\alpha), C_A(\beta, \alpha), n_P(\beta, \alpha), S_P(\beta, \alpha), R(\alpha), A(\beta, \alpha), P^* (\beta, \alpha), n_A(\beta, \alpha), S_A(\beta, \alpha), \hat{A}(\beta, \alpha)$)

od od

end COMPRAP
References


The following reports have appeared in this series:

<table>
<thead>
<tr>
<th>Report</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>92-30</td>
<td>R.E. Grundy, C.J. van Duijn, C.N. Dawson</td>
<td>Asymptotic profiles with finite mass in one-dimensional contaminant transport through porous media: the fast reaction case</td>
</tr>
<tr>
<td>92-31</td>
<td>J.E. Mebius</td>
<td>An intrinsic construction of the algebra of quaternions</td>
</tr>
<tr>
<td>92-32</td>
<td>Andrea J.S. Hin, Frits H. Post, Theo van Walsum</td>
<td>A testbed for flow visualization research</td>
</tr>
<tr>
<td>92-33</td>
<td>J.A. Bakker</td>
<td>Semantische modellen voor gedistribueerd database management I. De ontwikkeling van een metamodel voor fragmentatie</td>
</tr>
<tr>
<td>92-34</td>
<td>H. de Nivelle</td>
<td>Equivalentie van een aantal deductiesystemen</td>
</tr>
<tr>
<td>92-35</td>
<td>S. Zeng and P. Wesseling</td>
<td>Galerkin coarse grid approximation for the incompressible Navier-Stokes equations in general coordinates</td>
</tr>
<tr>
<td>92-36</td>
<td>J.P.M. de Vreught</td>
<td>Parallel contraction of binary dags</td>
</tr>
<tr>
<td>92-37</td>
<td>K. Szajowski and W.L. de Koning</td>
<td>Discrete-time Markovian jump linear systems</td>
</tr>
<tr>
<td>92-38</td>
<td>Robert E. Kooij and André Zegeling</td>
<td>Polynomial systems with centers and limit cycles</td>
</tr>
<tr>
<td>92-39</td>
<td>K.-H. Elster and R. Elster</td>
<td>Vector approximation problems in geometric vector optimization</td>
</tr>
<tr>
<td>92-40</td>
<td>S. Zeng and P. Wesseling</td>
<td>An efficient algorithm for the computation of Galerkin coarse grid approximation for the incompressible Navier-Stokes equations</td>
</tr>
</tbody>
</table>

Copies of these reports may be obtained from the bureau of the Faculty of Technical Mathematics and Informatics, Julianalaan 132, 2628 BL DELFT, phone 015-784568.