Technical Report
RAL-TR-2005-004

CCLRC

MI31: a conjugate gradient algorithm implementation with energy-norm stopping criteria

Mario Arioli and Gianmarco Manzini

March 24, 2005

Council for the Central Laboratory of the Research Councils

© Council for the Central Laboratory of the Research Councils

Enquires about copyright, reproduction and requests for additional copies of this report should be addressed to:

Library and Information Services
CCLRC Rutherford Appleton Laboratory
Chilton Didcot
Oxfordshire OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

CCLRC reports are available online at:
http://www.clrc.ac.uk/Activity/ACTIVITY=Publications;SECTION=225;

ISSN 1358-6254

Neither the Council nor the Laboratory accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

RAL-TR-2005-004

MI31: a conjugate gradient algorithm implementation with energy-norm stopping criteria

Mario Arioli[1] and Gianmarco Manzini[2]

ABSTRACT

We present an implementation of the preconditioned conjugate gradient method that is based on reverse communication to allow the user to incorporate his or her own matrix-vector multiplication routine and preconditioning algorithm. Our implementation also includes new stopping criteria that take into account some recent results on the approximation of the energy norm of the error during the conjugate gradient iterative process. We present the results of several numerical tests that experimentally validate the effectiveness of the stopping criteria on finite-element approximations of selected 2-D and 3D problems.

Keywords: Conjugate Gradient method, stopping criteria.

AMS(MOS) subject classifications: 65F10, 65F35, 65F50.

Current reports available by anonymous ftp to ftp.numerical.rl.ac.uk in directory pub/reports.
[1] m.arioli@rl.ac.uk, Rutherford Appleton Laboratory,
[2] Gianmarco.Manzini@ian.pv.cnr.it, IMATI - CNR, via Ferrata 1, 27100 Pavia, Italy
The work of first author was supported by EPSRC grant GR/S42170/01.
The work of second author was supported by EPSRC grant GR/R46427/01.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

May 9, 2005

Contents

1 Introduction

The conjugate gradient method is a very effective iterative algorithm for solving a linear system of equations

$$Au = b \tag{1}$$

where $A \in \mathbb{R}^{N\times N}$ is large, sparse, symmetric and positive definite and $b \in \mathbb{R}^N$. In particular, the conjugate gradient method (CG) has for long been successfully used to solve symmetric positive definite systems obtained by the finite-element approximation of elliptic partial differential equations. The convergence of the CG method is enhanced by introducing a matrix M to precondition the system (1) (see Greenbaum, 1997 for an introduction to this topic). The Preconditioned CG algorithm (PCG) requires the product of the matrix A by a vector and the product of the matrix M^{-1} by a vector. A rather common situation is one where the matrices A and M^{-1} are not explicitly available. This may occur either because these matrices cannot be directly built or because their size is too large to be stored in the CPU memory of the machine. In such cases, the PCG method can still be used by performing the matrix-vector products mentioned above as a sequence of simpler arithmetic operations. Consequently, a general purpose implementation of the preconditioned conjugate gradient has to allow the user to incorporate his/her own routines for matrix-vector products. *Reverse communication* makes it possible to achieve this task. For these reasons, the implementation of the PCG algorithm provided by the HSL Fortran95 package `MI31` is based on reverse communication. This implementation also includes the recent results by Arioli (2004), Golub and Meurant (1997), and Meurant (1999*b*) on the construction of stopping criteria that are reliable and computationally inexpensive.

The outline of the paper is as follows. In Section 2, we briefly review the mathematical aspects of the CG algorithm and the theoretical results on the error estimates that are pertinent to the current implementation. In Section 3 we discuss the technical aspects concerning the implementation of `MI31`; in particular, Sections 3.2 and 3.6 give a detailed description of the implementation of reverse communication. In Section 4 we illustrate the performance of the current implementation and discuss the use of error estimators for stopping the conjugate gradient iterations on a set of 2-D elliptic problems and on a set of 3-D models of structural engineering problems. Finally, in Section 5 we draw the final remarks and conclusions.

2 The CG algorithm

Let $r^{(k)} = b - Au^{(k)}$ be the residual of (1). We assume that the linear system (1) is symmetrically preconditioned by the nonsingular matrix U, in order to speed up the convergence of the method. We obtain the equivalent system

$$U^{-T}AU^{-1}y = U^{-T}b, \tag{2}$$

where $y = Uu$.

If we directly apply the CG method to (2), the iterates satisfy the following relations (Greenbaum 1997, Meurant 1999*a*)

$$\begin{aligned}
y^{(k)} &= y^{(k-1)} + \alpha_{k-1}\hat{p}^{(k-1)}, \quad \alpha_{k-1} = \frac{\hat{r}^{(k-1)T}\hat{r}^{(k-1)}}{\hat{p}^{(k-1)T}U^{-T}AU^{-1}\hat{p}^{(k-1)}}, \\
\hat{r}^{(k)} &= \hat{r}^{(k-1)} - \alpha_{k-1}U^{-T}AU^{-1}\hat{p}^{(k-1)}, \\
\hat{p}^{(k)} &= \hat{r}^{(k)} + \beta_{k-1}\hat{p}^{(k-1)}, \quad \beta_{k-1} = \frac{\hat{r}^{(k)T}\hat{r}^{(k)}}{\hat{r}^{(k-1)T}\hat{r}^{(k-1)}},
\end{aligned}$$

where $y^{(0)} = 0$ and $\hat{r}^{(0)} = \hat{p}^{(0)} = U^{-T}b$. In exact arithmetic, we have that $\hat{r}^{(k)} = U^{-T}b - U^{-T}AU^{-1}y^{(k)}$, and, therefore, by defining $u^{(k)} = U^{-1}y^{(k)}$, we have that

$$\hat{r}^{(k)} = U^{-T}(b - Au^{(k)}) = U^{-T}r^{(k)}.$$

Then, we have that

$$\|\hat{r}^{(k)}\|^2_{(U^{-T}AU^{-1})^{-1}} = \hat{r}^{(k)T}UA^{-1}U^T\hat{r}^{(k)} = \|r\|^2_{A^{-1}}\,.$$

Finally, if we define $p^{(k)} = U^{-1}\hat{p}^{(k)}$, and $M = U^TU$, we obtain the following variant of the PCG

Preconditioned Conjugate Gradient Algorithm (PCG)
Given an initial guess $u^{(0)}$, compute $r^{(0)} = b - Au^{(0)}$, and solve $Mz^{(0)} = r^{(0)}$. Set $p^{(0)} = z^{(0)}$, $\beta_0 = 0$, and $\alpha_{-1} = 1$.

for $k = 1, 2, \ldots,$ **until convergence**

$\alpha_{k-1} = \dfrac{r^{(k-1)T}z^{(k-1)}}{p^{(k-1)T}Ap^{(k-1)}}$;

$u^{(k)} = u^{(k-1)} + \alpha_{k-1}p^{(k-1)}$;

$r^{(k)} = r^{(k-1)} - \alpha_{k-1}Ap^{(k-1)}$;

Solve $Mz^{(k)} = r^{(k)}$;

$\beta_k = \dfrac{r^{(k)T}z^{(k)}}{r^{(k-1)T}z^{(k-1)}}$;

$p^k = z^k + \beta_k p^{(k-1)}$;

Figure 1: Preconditioned Conjugate Gradient Algorithm (PCG)

2.1 Error estimators and stopping criteria

The common stopping criterion uses the ratio between the residual and the right-hand side, i.e.

$$\omega = \frac{||r(u)||_2}{||b||_2}.$$

This implies that, if $\omega \neq \infty$, there is a vector δb, with $||\delta b||_2 \leq \omega||b||_2$, such that

$$Au = b + \delta b.$$

Nevertheless, the experiments described by Arioli (2004) give very good evidence that the stopping criteria based on the Euclidean norm of the residual $b - Au$ may be totally unsatisfactory and frequently misleading. For this reason, our implementation of the PCG method incorporates new stopping criteria that are based on a posteriori component-wise and norm-wise backward error theory (Arioli, Duff and Ruiz 1992).

2.1.1 Dual norm of the residual

If we use the conjugate gradient method, it is quite natural to have a stopping criterion that takes advantage of the minimization property of the iterative method. At each step k, the conjugate gradient approximation $u^{(k)}$ of the solution u of (1) minimizes the energy norm of the error $\delta u^{(k)} = u - u^{(k)}$ on the Krylov space $\mathcal{K}_k = span\{p^{(0)}, \ldots, p^{(k-1)}\}$ (Greenbaum 1997). Formally,

$$\delta u^{(k)} = \arg \min_{\delta u^{(k)} \in u - \mathcal{K}_k} \delta u^{(k)T} A \delta u^{(k)}.$$

Let us consider the space $\mathbb{R}^N$ endowed by the norm

$$\|y\|_A = (y^T A y)^{1/2},$$

and its dual space endowed by the dual norm

$$\|f\|_{A^{-1}} = (f^T A^{-1} f)^{1/2}.$$

The stopping criterion

$$\text{IF} \quad \|Au^{(k)} - b\|_{A^{-1}} \leq \eta \|b\|_{A^{-1}} \quad \text{THEN STOP}, \tag{3}$$

where $\eta < 1$ is a suitable a priori threshold chosen by the user, guarantees that the iterative approximation $u^{(k)}$ is the solution of the perturbed linear system

$$\begin{aligned} Au^{(k)} &= b - r^{(k)}, \\ \|r^{(k)}\|_{A^{-1}} &\leq \eta \|b\|_{A^{-1}}; \end{aligned}$$

see (Arioli, Noulard and Russo 2001) for details. The value of η may depend on the properties of the problem that we want to solve, and, in practical cases, may be much larger than ε, the roundoff unit of the computer finite-precision arithmetic. When the linear system (1) arises from the finite-element discretization of a partial differential equation, reasonable choices for η are provided by $\eta = h$ and $\eta = h^2$, where h is a parameter that takes into account the mesh density. In accordance with the definition proposed by Ciarlet (1978), the parameter h is usually the maximum diameter of the control volumes of the mesh covering the computational domain where the partial differential equation is formulated. The major implications of the two choices of η, which are mentioned above, on the effectiveness of the stopping criterion (3) are discussed by Arioli (2004).

In order to build a suitable stopping criterion, we need to add some tool to the conjugate gradient algorithm for estimating the error $e_A^{(k)} = r^{(k)T} A^{-1} r^{(k)}$ at the step k. The estimation of the error can be performed by using either the rule proposed in the original paper by Hestenes and Stiefel (1952) or the Gauss and Gauss-Radau quadrature rules proposed by Golub and Meurant (1997). We should point out that the Gauss quadrature rule is mathematically equivalent to the Hestenes and Stiefel rule. Furthermore, Strakoš and Tichý (2002) proved that the Hestenes and Stiefel rule is numerically stable. These bounds are computed using the information of the last d steps of the conjugate gradient algorithm. As suggested by Golub and Meurant (1997) and by Strakoš and Tichý (2002), these rules provide both lower bounds and upper bounds for the error $e_A^{(k-d)}$.

In our implementation of stopping criteria for the PCG method, we consider the two following variants:

- the one based on the Gauss quadrature rule, as proposed by Hestenes and Stiefel, that gives the lower bound τ_k for the error $e_A^{(k-d)}$;

- the one based on the Gauss-Radau quadrature rule that provides the upper bound Ξ_k and the lower bound ξ_k for the error $e_A^{(k-d)}$;

The calculation of the upper bound Ξ_k requires an estimate of the smallest eigenvalue λ_{min} of the matrix A (or of the preconditioned matrix). Likewise, the calculation of the lower bound ξ_k requires an estimate of the largest eigenvalue of the matrix A (or of the preconditioned matrix). As

$$u = \sum_{j=0}^{N} \alpha_j p^{(j)}, \qquad u^{(k)} = \sum_{j=0}^{k} \alpha_j p^{(j)}, \qquad \text{and} \qquad p^{(i)} A p^{(j)} = 0, \qquad i \neq j,$$

we have that

$$u - u^{(k)} = \sum_{j=k}^{N} \alpha_j p^{(j)},$$

and the Hestenes and Stiefel rule can be reformulated by

$$\|u - u^{(k)}\|_A^2 = \sum_{j=k}^{N} \sum_{i=k}^{N} \alpha_i \alpha_j p^{(j)T} A p^{(i)} \approx \sum_{j=k}^{k+d} \alpha_j r^{(j)T} z^{(j)}.$$

Arioli (2004) discussed the choice of the delay parameter d. It turns out that $d = 5$ or $d = 10$ are successful compromises when the convergence is reasonably fast, i.e. the preconditioner is effective. The numerical experiments support this conclusion (see Arioli, 2004, Meurant, 1999*b*, and Golub and Meurant, 1997).

2.1.2 Dual norm of b

Finally, we must estimate $b^T A^{-1} b$. Taking into account that (Arioli 2004)

$$\|b\|_{A^{-1}}^2 = \|u\|_A^2 \geq b^T u^{(0)} + r^{(0)T} u^{(k)}, \tag{4}$$

and that the lower bound converges to $\|u\|_A^2$, we could replace $\|b\|_{A^{-1}}$ with its lower bound at step k of the conjugate gradient algorithm. Alternatively, we can use the values ψ_k computed during the conjugate gradient method to estimate the following lower bound for $\|u - u^{(0)}\|_A^2$:

$$\|u - u^{(0)}\|_A^2 \geq \|u^{(k)} - u^{(0)}\|_A^2 = \sum_{j=1}^{k} \psi_j. \tag{5}$$

Since

$$\|u - u^{(0)}\|_A^2 = \|u\|_A^2 + \|u^{(0)}\|_A^2 - 2 b^T u^{(0)},$$

we have the following lower bound for $\|b\|_{A^{-1}}^2$:

$$\|b\|_{A^{-1}}^2 = \|u\|_A^2 \geq r^{(0)T} u^{(0)} + b^T u^{(0)} + \sum_{j=1}^{k} \psi_j. \tag{6}$$

Strakoš and Tichý (2004) point out that this lower bound is much less sensitive to the roundoff than the lower bound (4).

The stopping criterion given by (3) can be replaced either by:

$$\text{IF } \varsigma_k \leq \eta^2 \left(b^T u^{(0)} + r^{(0)T} u^{(k)} \right) \text{ THEN STOP }, \tag{7}$$

or by

$$\text{IF } \varsigma_k \leq \eta^2 (r^{(0)T} u^{(0)} + b^T u^{(0)} + \sum_{j=1}^{k} \psi_j) \text{ THEN STOP }, \tag{8}$$

where ς_k is one of the estimates of the dual norm of the residual.

However, both (7) and (8) have stability problems when $u^{(0)} \neq 0$ has large entries and u has very small entries ($\approx \sqrt{\epsilon}$, ϵ machine precision) such as in the case of structural engineering problems with small displacements. The lower bound of a small energy norm of u is computed by a strong cancellation of large terms. In these situations, we can approximate $\|u\|_A^2$ by the more crude value $b^T u^{(k)}$, even if we cannot guarantee that it will be a lower bound. Finally, we poit out that the previous case is quite artificial. Nevertheless, we use the control (8), which requires only one floating-point operation per step, as the default stopping criterion of the code.

In Figure 2, we formulate the PCG algorithm that incorporates the proposed stopping criteria. Note that the introduction of a preconditioner to speed up the convergence rate of the conjugate gradient algorithm requires adapting the previous technique for the evaluation of $e_A^{(k)}$. Further details can be found in (Arioli 2004).

3 The implementation of MI31

3.1 Subroutine list

Our implementation in FORTRAN 90 is composed of ten subroutines, one integer function and one parameter module. For the sake of this presentation, the subroutines are logically grouped as listed below.

- Routines implementing the *user front-end*:

```
subroutine MI31AD ( n,rhs,w,ldw,icntl,cntl,Ido,info,rinfo,iptr );
subroutine MI31ID ( icntl,cntl ).
```

- Routines implementing the *initialization* phase and the check-up of the input arguments provided by the user:

```
subroutine sub_CG_INITIALISATION ( n,w,ldw,cg,icntl,cntl,Ido,info,rinfo );
subroutine sub_CHECK_PARAMETERS  ( icntl,cntl ).
```

- Routines implementing the *preconditioned conjugate gradient* method by reverse communication:

```
subroutine sub_CG_SETPARS ( flag,cg,Ido,info,rinfo );
subroutine sub_CG_PART_1  ( n,w,ldw,cg );
subroutine sub_CG_PART_2  ( n,w,ldw,cg ).
```

- Routines implementing the *stopping criteria*:

Let $u^{(0)}$ be the initial guess; compute $r^{(0)} = b - Au^{(0)}$, solve $Mz^{(0)} = r^{(0)}$, and, then, set $p^{(0)} = z^{(0)}$, $\beta_0 = 0$, $\alpha_{-1} = 1$, $\chi_1 = 1$, $\varsigma_0 = 1$, and $k = 0$:

while $\varsigma_k > \eta^2 norm\_u\_est$ **do**

$k = k + 1$;

$$\alpha_{k-1} = \frac{r^{(k-1)T} z^{(k-1)}}{p^{(k-1)T} A p^{(k-1)}}; \qquad \psi_k = \frac{(r^{(k-1)T} z^{(k-1)})^2}{p^{(k-1)T} A p^{(k-1)}}; \qquad \omega_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}};$$

if $k = 1$ **then**

$$\rho_1 = \omega_1; \qquad \phi_1 = \omega_1 - \lambda_{min}; \qquad \Phi_1 = \omega_1 - \lambda_{max};$$

else

$$\chi_k = \frac{\chi_{k-1}\pi_{k-1}}{\rho_{k-1}}; \qquad \rho_k = \omega_k - \frac{\pi_{k-1}^2}{\rho_{k-1}};$$
$$\phi_k = \omega_k - \omega_{k-1}^u; \qquad \Phi_k = \omega_k - \omega_{k-1}^l;$$

endif

$$u^{(k)} = u^{(k-1)} + \alpha_{k-1} p^{(k-1)};$$
$$r^{(k)} = r^{(k-1)} - \alpha_{k-1} A p^{(k-1)};$$

Solve $Mz^{(k)} = r^{(k)}$;

$$\beta_k = \frac{r^{(k)T} z^{(k)}}{r^{(k-1)T} z^{(k-1)}}; \qquad \pi_k = \frac{\sqrt{\beta_k}}{\alpha_{k-1}};$$
$$p^k = z^k + \beta_k p^{(k-1)};$$
$$\omega_k^u = \lambda_{min} + \frac{\pi_k^2}{\Phi_k}; \qquad \omega_k^l = \lambda_{max} + \frac{\pi_k^2}{\phi_k};$$
$$\psi_k^l = \frac{\chi_k^2 \pi_k^2}{\rho_k(\omega_k^l \rho_k - \pi_k^2)}; \qquad \psi_k^u = \frac{\chi_k^2 \pi_k^2}{\rho_k(\omega_k^u \rho_k - \pi_k^2)};$$

if $k > d$ **then**

$$\tau_k = \sum_{j=k-d+1}^{k} \psi_j; \qquad \xi_k = \|r^{(0)}\|_{M^{-1}}^2 \psi_k^l + \tau_k; \qquad \Xi_k = \|r^{(0)}\|_{M^{-1}}^2 \psi_k^u + \tau_k;$$
$$\varsigma_k = \tau_k \qquad \text{or}\{= \xi_k\} \qquad \text{or}\{= \Xi_k\};$$
$$norm\_u\_est = (r^{(0)T} u^{(0)} + b^T u^{(0)} + \textstyle\sum_{j=1}^{k} \psi_j) \qquad \text{or}\{= b^T u^{(0)} + r^{(0)T} u^{(k)}\}$$

endif

end while.

Figure 2: Preconditioned conjugate gradient algorithm (PCG)

```
subroutine sub_CHECK_RESIDUAL     ( n,w,ldw,cg,icntl,cntl );
subroutine sub_ERROR_ESTIMATE_1_HS( n,w,ldw,cg,icntl,cntl,Ido,info,rinfo );
subroutine sub_ERROR_ESTIMATE_2_HS( n,w,ldw,cg icntl,cntl,Ido,info,rinfo ).
```

Furthermore, we use the integer function `next_action` defined as

`function next_action( last_action,icntl,cntl ),`

and a parameter module called

```
module CG_PARAMETERS.
```

3.2 Reverse Communication

Reverse communication is based on a double loop system. The user implements the external loop, which performs the matrix-vector product and, if requested, the preconditioning. The external loop has the logic structure depicted in Figure 3. The internal loop is provided by the package and performs the rest of the CG algorithm. Figure 3 illustrates the flow chart of the external loop of the user, while the following piece of source code exhibits a possible implementation of this loop.

```
do while ( Ido>0 )
   call mi31ad( n, w, ldw, icntl, cntl, Ido, info, rinfo, iptr )
   select case ( Ido )
   case ( 1 )
      call matvec( w(1,z), w(1,iptr), ip, ind, value, m, n, ne )
   case ( 2 )
      call precon( w(1,z), w(1,iptr), diag, n )
   end select
end do
```

The variable `Ido` contains the logical flag which drives the loop. This variable can take the values

0, stop the CG iterations;

1, perform the matrix-vector product;

2, perform the preconditioning (if requested).

If preconditioning is not required the variable `Ido` will never take the value 2.

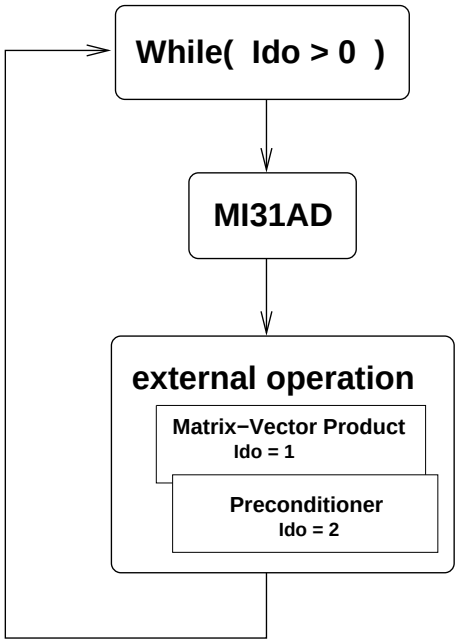

Figure 3: Flow chart of the user external loop

Any iteration of the CG algorithm is basically split as shown in Figure 4, where we also indicate the routines performing the various internal and external operations.

Matrix-vector product:

(external loop) $z = Ap^{(k)}$

First Part:

(`sub_CG_PART1`)

$\mathsf{curv}_k = z^T p^{(k)})$

$\alpha_{k-1} = \dfrac{\rho_k}{\mathsf{curv}_k}$

$u^{(k)} = u^{(k-1)} + \alpha_{k-1} p^{(k-1)}$

$r^{(k)} = r^{(k-1)} - \alpha_{k-1} z$

$\psi_k = \alpha_{k-1} * \rho_k$

$\mathsf{norm\_u\_est} = \mathsf{norm\_u\_est} + \psi_k$

or set $\mathsf{norm\_u\_est} = b^T u^{(k)}$

Preconditioning step:

(external loop) solve $Mz = r^{(k-1)}$

(`MI31AD`) or set $z = r^{(k-1)}$

Second Part:

(`sub_CG_PART2`)

$\rho_k = r^{(k)T} z$

$\beta_{k-1} = \dfrac{\rho_k}{\rho_{k-1}}$

$p^{(k)} = z + \beta_k p^{(k-1)}$

Stopping Criteria:

(`sub_CHECK_RESIDUAL`) if $||r^{(k)}||_2 \leq \max(||r^{(0)}||_2 * \texttt{CNTL(1)}, \texttt{CNTL(2)})$ then stop

(`sub_ERROR_ESTIMATE_1`) if Gauss error est. $\leq \mathsf{norm\_u\_est} * \texttt{CNTL(1)}^2$ then stop

(`sub_ERROR_ESTIMATE_2`) if Gauss-Radau error est. $\leq \mathsf{norm\_u\_est} * \texttt{CNTL(1)}^2$ then stop

Figure 4: The way a CG iteration is split (case $u^{(0)} = 0$)

3.3 The Subroutine `MI31AD`

The subroutine `MI31AD` implements the internal loop in a `do while` statement, and its flow chart is given in Figure 5.

When the execution stream enters into or exits from `MI31AD`, the subroutine `sub_CG_SETPARS` is called.

This subroutine is called at the beginning of `MI31AD` with the first header entry set to `INITIAL_SETUP`. In this case part of the entries of the arrays `INFO` and `RINFO` are copied into the data structure `cg` to restore the current status of the CG algorithm. Conversely, when the subroutine is called with the first option flag set to `FINAL_SETUP` at the end of `MI31AD`, the status of the CG algorithm in the data structure `cg` is saved into the arrays `INFO` and `RINFO`. The final statement `iptr = icntl(i_VECTOR_PTR)` set the integer pointer `iptr`. This pointer indicates the

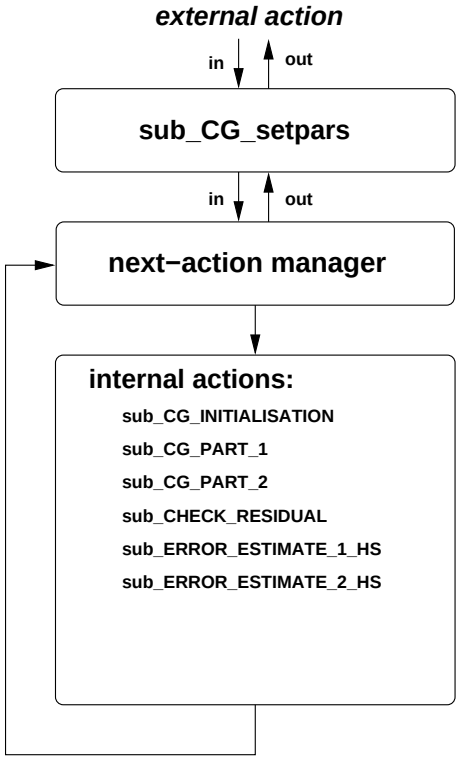

Figure 5: Flow chart of MI31AD

correct column of the array W that must be operated on by the user.

```
call sub_CG_SETPARS( INITIAL_SETUP, cg, Ido, info, rinfo )
do while ( .true. )
  cg % the_action = next_action( cg % the_action, icntl )
  select case ( cg % the_action )
  case ( ... )
  !! ... internal loop implementation
  end select
end do while
call sub_CG_SETPARS( FINAL_SETUP, cg, Ido, info, rinfo )
iptr = icntl(i_VECTOR_PTR)
```

The internal `do while` loop is expanded as follows.

```
do while ( .true. )
   cg % the_action = next_action( cg % the_action, icntl )
   select case ( cg % the_action )
   case ( INIT_PHASE )
      call sub_CG_INITIALISATION ( n, rhs, w, ldw, cg, icntl, cntl,
      Ido, info, rinfo )
   case ( CG_PART_1 )                        !! z = A p
      cg % iter = cg % iter+1                !! begin a new iteration
      call sub_CG_PART_1 ( n, w, ldw, cg )
   case ( CG_PART_2 )                        !! z = M r
```

```
      call sub_CG_PART_2 ( n, w, ldw, cg )
   case ( NO_PRECONDITIONER )
      w(1:n,z) = w(1:n,r)
   case ( CHECK_RESIDUAL )
      call sub_CHECK_RESIDUAL   ( n, w, ldw, cg, icntl, cntl )
   case ( ENERGY_ESTIMATE_1 )
      call sub_ERROR_ESTIMATE_1_HS ( n, w, ldw, cg, icntl, cntl,
      info, rinfo )
   case ( ENERGY_ESTIMATE_2 )
      call sub_ERROR_ESTIMATE_2_HS ( n, w, ldw, cg, icntl, cntl,
      Ido, info, rinfo )
   case default
      exit
   end select
end do
```

Any internal cycle performs an *action* decided by the *next-action manager* on the basis of the last action performed and the internal status of the routine. This latter one is recorded in the variable `rinfo(i_AUX_ACTION)`, and, during the initialization phase, `rinfo(i_EXT_ACTION)`. The next action to be performed by the code is returned by the integer function `next_action`, which implements the *next action manager* and is held in the variable `cg % the_action`.

The statement `select case ( cg % the_action )` switches to the call of the corresponding subroutine or exits from the loop. The different switches are self-explanatory (see Figure 5). Other than the values of this switch, `cg % the_action` can take the following values:

- `EX_PRECONDITIONER`, exit and perform the user preconditioner;
- `MAT_VEC_PRODUCT`, exit and perform the user matrix-vector product;
- `STOP_CG`, stop the CG iterations.

When one of these values is set, the routine enters the default switch and executes the `exit` statement. Notice that a new iteration starts when the condition

```
cg % the_action == MAT_VEC_PRODUCT .and. icntl(i_VECTOR_PTR) == p
```

is satisfied.

3.4 The part of a CG iteration performed internally.

As shown in Figure 4, the matrix-vector product and the preconditioner are performed externally, but the rest of the CG iteration is performed in the two subroutines `sub_CG_PART_1` and `sub_CG_PART_2` briefly described in this section.

The array `w` contains five columns, whose entries are the CG vectors $u^{(k)}$, $p^{(k)}$, $r^{(k)}$, the auxiliary vector z, and the initial residual $b - Au^{(0)}$.

These columns of `w` are addressed by the integer pointers

- `u` $= 1$,
- `p` $= 2$,
- `r` $= 3$,

- z = 4,
- b = 5.

The first three vectors are the usual vectors of the CG definition. The vector **z** is used to store temporarily the result of the external operations. The fifth vector is set to the initial residual and is used during the iterations to perform the error estimates.

3.4.1 The subroutine sub_CG_PART_1

The subroutine sub_CG_PART_1 executes the first part of the CG algorithm as indicated in Figure 4. Basically, the subroutine consists in the following piece of source code, which calls the BLAS routines ddot and daxpy:

```
cg % curv = ddot( n, w(1,z), 1, w(1,p), 1 )
cg % alpha = cg % rho / cg % curv
call daxpy( n,  cg % alpha, w(1,p), 1, w(1,u), 1 )
call daxpy( n, -cg % alpha, w(1,z), 1, w(1,r), 1 )
```

The updated value of the scalar parameters curv_k and α_{k-1} are stored in the data structure cg.

3.4.2 The subroutine sub_CG_PART_2

The subroutine sub_CG_PART_2 executes the second part of the CG algorithm as indicated in Figure 4. Basically, the subroutine consists in the following piece of source code, which calls the BLAS routines ddot, dscal and daxpy:

```
cg % rho  = ddot( n, w(1,r), 1, w(1,z), 1 )
cg % beta = cg % rho / cg % rho1
cg % rho1 = cg % rho
call dscal( n, cg % beta, w(1,p), 1 )
call daxpy( n,       one, w(1,z), 1, w(1,p), 1 )
```

The updated value of the scalar parameters ρ_{k-1}, ρ_k, and β_k are stored in the data structure cg.

3.5 The initialisation routines

The package contains three routines that are called in the initialisation phase. These ones are MI31ID, sub_CG_INITIALISATION and sub_CHECK_PARAMETERS.

3.5.1 The subroutine MI31ID

This routine sets the default values for the control arrays ICNTL and CNTL.

3.5.2 The subroutine sub_CG_INITIALISATION

This routine initialises the CG solver. The way this routine interacts with the next-action manager is the most complex part of the entire implementation. Indeed, during the initialisation phase, both the external operations of matrix-vector product and preconditioning may be required as well as several internal set-ups. The part of the routine to be performed each time it is entered is driven by the flags.

```
ext_action = icntl(i_EXT_ACTION)    !! last external action
set_r0 = icntl( i_SET_R0 ) == TRUE  !! r0 needs init
set_p0 = icntl( i_SET_P0 ) == TRUE  !! p0 needs init
set_b  = icntl( i_SET_B  ) == TRUE  !! b  needs init
```

The variable `icntl(i_EXT_ACTION)` is set to the `INIT_PHASE` the first time by `MI31ID`. At other times it contains the name of the last external operation performed by the code. This value allows the code to select between the two following updates:

```
select case ( ext_action )
case ( MAT_VEC_PRODUCT )
   call daxpy( n, -one, w(1,z), 1, w(1,b), 1 )  !! z = A u0, b <-- b-z = r0
   w(1:n,r) = rhs(1:n)                          !! set r0
case ( EX_PRECONDITIONER )
   w(1:n,p) = w(1:n,z)
end select
```

that respectively complete the set-up of the initial residual stored in `w(1:n,r)` and in `w(1,b)` when a guess solution is provided by the user, and the conjugate direction vector, stored in `w(1:n,p)`, when a user preconditioner is available. The piece of code

```
if ( set_b ) then
   !!  performs the copy the first time the routine is entered
   icntl( i_SET_B  ) = FALSE
   w(1:n,b) = rhs(1:n)
end if
```

sets the right-hand-side vector into the array column `w(1:n,b)`. The piece of code

```
if ( set_r0 ) then
   if ( icntl(i_USER_U0).eq.TRUE ) then
      icntl(i_SET_R0) = FALSE
      icntl(i_VECTOR_PTR) = u
      icntl(i_AUX_ACTION) = MAT_VEC_PRODUCT !! --> set_r0 = T
      icntl(i_EXT_ACTION) = MAT_VEC_PRODUCT !! --> uso interno
   else
      set_r0   = .false. !! --> set_r0 = F
   end if
end if
```

controls whether a user guess $u^{(0)}$ for the solution is provided. In such a case, the code calls for the external matrix-vector product $Au^{(0)}$ to complete the calculation of the initial residual, i.e. $r^{(0)} = b - Au^{(0)}$. We also set `icntl(i_SET_R0) = FALSE`, so that this fragment of the code is not rerun when the routine is reentered.

Note that, at this stage, the value of `set_r0` is kept equal to `TRUE`, so that the following `if/end if` pieces of source code are not to be executed. If no user guess $u^{(0)}$ is given, we set `set_r0=.false.`. This is also required to perform the next `if/end if` pieces of source code. The piece of code

```
if ( set_p0 .and. .not.set_r0 ) then
   if ( icntl(i_USER_PRECON).eq.TRUE ) then
      icntl(i_SET_P0)     = FALSE
```

```
      icntl(i_VECTOR_PTR) = r
      icntl(i_AUX_ACTION) = EX_PRECONDITIONER !! uso last-action manager
      icntl(i_EXT_ACTION) = EX_PRECONDITIONER !! uso interno
   else
      print *,' cg-init: p0 OK!!! '
      w(1:n,z) = w(1:n,r)
      w(1:n,p) = w(1:n,z)
      set_p0 = .false.
   end if
end if
```

controls whether a user preconditioner is in use to set the initial value of the conjugate direction $\mathbf{p}_0$. This block of statements is driven by the logical variables `set_p0` and `set_r0` and its logic is the same as the previous `if/end if` block. Notice that these statements are run only after that the set-up of the initial residual has been terminated. Eventually, the initialization routine executes the `if-end if` block:

```
if ( .not.set_r0 .and. .not.set_p0 ) then
   !! set the beginning of the real CG loop
   icntl(i_INIT_PHASE) = FALSE
   icntl(i_VECTOR_PTR) = p
   icntl(i_AUX_ACTION) = MAT_VEC_PRODUCT

   cg % r0_nrm2 = dnrm2( n, w(1,r), 1 )
   if ( cg % r0_nrm2 .eq. 0.d0 ) then
      write(ICNTL(i_STDERR_UNIT),'(a)')    &
           'ERROR MESSAGE: INITIAL RESIDUAL HAS 2-NORM EQUAL TO ZERO.'
      info(i_ERROR_FLAG) = i_ERRFLAG_ZERO
      icntl(i_AUX_ACTION) = STOP_CG
   else
      cg % resid = cg % r0_nrm2
   end if
   cg % rho     = ddot ( n, w(1,r), 1, w(1,z), 1 )
   cg % rho1    = cg % rho

   rinfo(i_B0X0) = ddot ( n, rhs, 1, w(1,u), 1 )
   rinfo(i_Z0R0) = cg % rho

   if ( CNTL(i_TOL_2) == 0.d0 ) then
      CNTL(i_TOL_2) = CNTL(i_TOL_1)
   end if

   if ( ICNTL(i_MAX_ITER) .eq. 0 ) then
      ICNTL(i_MAX_ITER) = n !! max n iterations
   end if

   !! final check of parameters
   call sub_check_parameters( info, icntl, cntl )

end if
```

The first three statements set the control array `icntl` for the next-action manager. Then, it initializes the variables used in the first CG iteration and stored in `cg`, and calculates $b^T u^{(0)}$, stored in `rinfo(i_B0X0)`, and ρ_0, stored in `rinfo(i_Z0R0)`. Finally, it calls the routine `sub_check_parameters` to perform some checks on the input values of the control arrays `icntl` and `cntl`.

3.5.3 The subroutine `sub_CHECK_PARAMETERS`

This routine verifies the consistency of the control parameters given in input to `MI31AD`. These parameters are set to their default values by `MI31ID`, but can be arbitrarily changed by the user before the CG algorithm starts the run..

3.6 The *Next-Action* Manager

Basically, any action of the CG iterative scheme, which has been performed internally or externally, is followed by a "next action". The "next action" to be performed is decided on the basis of the value of the variable `last_action`, which is taken as an input argument, and by controlling the internal flags `icntl(i_AUX_ACTION)` and `icntl(i_INIT_PHASE)`. These parameters are used during the initialisation phase and when an error-estimate routine is called. When an external operation is selected, the code must also specify the input field by setting suitably the variable `icntl(i_VECTOR_PTR)`. The structure of the routine, which implements the initialisation mechanism, is as follows.

```
function NEXT_ACTION( last_action, icntl )
  use CG_PARAMETERS
  implicit none

  integer :: next_action, last_action
  integer, dimension(ndim) :: icntl

  if ( icntl(i_INIT_PHASE) == TRUE ) then
     last_action = INIT_PHASE
  end if

  select case ( last_action )

  case ( INIT_PHASE )
     next_action = icntl(i_AUX_ACTION)
     icntl(i_AUX_ACTION) = INIT_PHASE

  case ( ... ) !! other case implementations

  end select

  if ( icntl(i_PRT_NEXT_ACTION) == TRUE ) then

    !! some printing stuff ...

  end if

  return
```

```
end function NEXT_ACTION
```

When `icntl(i_INIT_PHASE) == TRUE`, `last_action` is forced to `INIT_PHASE`, so that the next-action manager selects `case ( INIT_PHASE )`. The next action is then set to the value stored in `icntl(i_AUX_ACTION)`. This latter variable has been set the first time to `INIT_PHASE` by `MI31ID`. During the initialization phase, this variable is set by `sub_CG_INITIALISATION` to the external operations required by the initialization steps, and eventually to the first user matrix-vector product that starts the CG iterations. Notice that `icntl(i_AUX_ACTION) = INIT_PHASE`, that is the auxiliary action flag is re-set to the value `INIT_PHASE` to continue the initialization the next time the action manager is re-entered after an external operation. The external operation that has been performed during the initialization phase is indicated by the value of the flag stored in `icntl(i_EXT_ACTION)`, `sub_CG_INITIALISATION`.

The select statement is fully described as follows.

```
select case ( last_action )

case ( INIT_PHASE )
   next_action = icntl(i_AUX_ACTION)
   icntl(i_AUX_ACTION) = INIT_PHASE

case ( CG_PART_1 )
   if ( icntl(i_USER_PRECON) == TRUE ) then
      icntl(i_VECTOR_PTR) = r
      next_action         = EX_PRECONDITIONER
   else
      next_action = NO_PRECONDITIONER
   end if

case ( EX_PRECONDITIONER, NO_PRECONDITIONER )
   next_action = CG_PART_2

case ( CG_PART_2 )
   next_action = stopping_criteria( icntl( i_STOPPING ) )  !! select the stopping
                                                           !! criterion
case ( CHECK_RESIDUAL )
   if ( icntl(i_AUX_ACTION)==STOP_CG ) then
      next_action = STOP_CG
   else if ( icntl(i_AUX_ACTION)==LOOP_CG ) then
      icntl(i_VECTOR_PTR) = p
      next_action         = MAT_VEC_PRODUCT
   else
      stop '- WRONG ACTION.'
   end if

case ( ENERGY_ESTIMATE_1, ENERGY_ESTIMATE_2 )
   if      ( icntl(i_AUX_ACTION) == STOP_CG ) then
      next_action = STOP_CG
   else if ( icntl(i_AUX_ACTION) == LOOP_CG ) then
      icntl(i_VECTOR_PTR) = p
      next_action         = MAT_VEC_PRODUCT
```

```
      else
         stop '- WRONG ACTION.'
      end if

   case ( MAT_VEC_PRODUCT )
      next_action = CG_PART_1

   case ( STOP_CG )
      next_action = STOP_CG

   end select
```

3.7 The Module CG_PARAMETERS

All of the access entries of the control arrays ICNTL and CNTL, the info and working arrays INFO, RINFO, and W, and the values that any parameter flag can take are parameterised and referred to by a label name. The definitions are contained in the module CG_PARAMETERS, which is included at the beginning of the package. Such a parameterisation makes it possible to change the position of the control and info flags and the value that they attain consistently. The name of the entries and index values are self-explanatory, and for this reason are not included in this discussion.

4 Numerical experiments

In this section, we present the results of the PCG algorithms on two set of test problems. In all the tests, we used the incomplete Cholesky factorization, **icfs**, by Lin and Moré (1999) as the preconditioner.

4.1 Test problems

The first set of test problems is related to the solution by finite elements of a 2-dimensional Poisson problem defined on $\Omega = (0,1) \times (0,1)$ (the unit square) as

$$(\mathcal{P}_1) \qquad \begin{cases} \operatorname{div} c(\mathbf{x}) \operatorname{grad} u(\mathbf{x}) = 1 & in \quad \Omega \\ u(\mathbf{x}) = 0 & on \quad \partial\Omega \end{cases}$$

where (see Figure 6)

$$\begin{cases} c(\mathbf{x}) = 1 & \mathbf{x} \in \Omega_1 \\ c(\mathbf{x}) = 10^{-6} & \mathbf{x} \in \Omega \setminus \Omega_1 \end{cases}$$

The meshes and the discrete problem have been generated using FEMLAB (2004). In Figure 7, we show the mesh of the coarser problem where there are 8 points on each segment of $\partial\Omega$. The other meshes (see Table 1 for the details) have been built by doubling the number of nodes on each segment of the boundary of the previous mesh. On the quadrilateral meshes, we consider the finite-element approximation provided by the quadratic Lagrange elements.

The second set of test problems is related to the solution of a structural mechanics elasticity problem in three dimensions, and is formulated on the domain Ω, that represents an iron beam of length 10. Figure 8 shows the iron beam, and Figure 9 its cross section. The two extremes of the beam are clamped and the body force is gravity. The union of the two clamped faces ($x = 0$ and $x = 10$) is represented by the sub-domain $\partial\Omega_1$, while $\partial\Omega_2$ is the top face of the beam (see Figure 8).

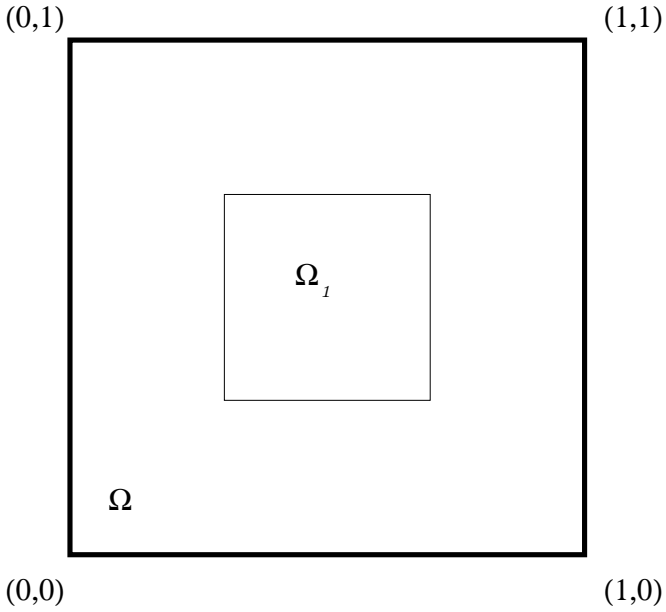

Figure 6: Domain Ω for problem $(\mathcal{P}_1)$

	Mesh 8	Mesh 16	Mesh 32	Mesh 64
n (# degree of freedom)	6241	25281	101761	408321
nz (# nonzeros)	51481	211721	858601	3457961
h	0.05	0.025	0.0125	0.00625

Table 1: Data relating to the meshes for problem $(\mathcal{P}_1)$.

The physical model is described by the steady Navier's equation:

$$(\mathcal{P}_2) \qquad \begin{cases} -\mathrm{div}\,\mathbf{c}\,\mathrm{grad}\,\mathbf{u} = \mathbf{K} & in \quad \Omega \\ \mathbf{u} = 0 & on \quad \partial\Omega_1 \\ (\mathbf{n}\cdot\boldsymbol{\sigma})_z = -4000 & on \quad \partial\Omega_2 \\ (\mathbf{n}\cdot\boldsymbol{\sigma})_x = 0 & on \quad \partial\,\Omega \setminus \{\partial\,\Omega_1 \cup \partial\,\Omega_2\} \\ (\mathbf{n}\cdot\boldsymbol{\sigma})_y = 0 & on \quad \partial\,\Omega \setminus \{\partial\,\Omega_1 \cup \partial\,\Omega_2\} \\ (\mathbf{n}\cdot\boldsymbol{\sigma})_z = 0 & on \quad \partial\,\Omega \setminus \{\partial\,\Omega_1 \cup \partial\,\Omega_2\} \end{cases}$$

where the tensor $\mathbf{c}$ is a 9×9 constitutive matrix defined by the material properties, the tensor $\boldsymbol{\sigma}$ is the stress tensor, the vector $\mathbf{u} \in \mathbb{R}^3$ takes into account the displacements, and $\mathbf{K}$ is the body force. As for the first set, the meshes and the finite dimensional problems have been generated using FEMLAB (2004). In Figure 9, we exhibit the coarser mesh relative to a cross-section of the beam where, on each segment of the boundary, we have two points. The other cross-section meshes (see Table 2 for the details) have been built by doubling the number of nodes on each segment of the boundary of the previous mesh. The global meshes have been built by subdividing the beam along the x-axis with parallel cross-sections (see Table 2 for the details) and using bilinear Lagrange finite-element approximation on the resulting 3-rectangles (see Ciarlet, 1978, Chapter 2 Section 2.2).

In the next section, we use the first set of test problems to compare the different estimators for the energy norm of the error (see τ_k, ξ_k, and Ξ_k in Figure 2). The second test problem

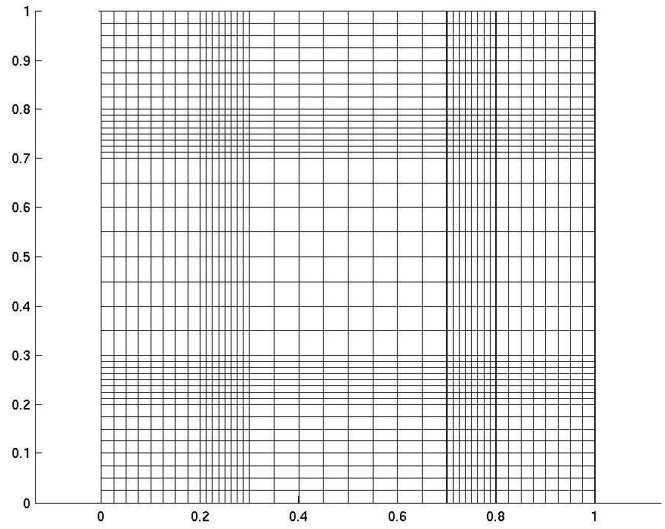

Figure 7: Mesh 8 for problem ($\mathcal{P}_1$)

	Mesh 2	Mesh 4	Mesh 8
n (# degree of freedom)	22113	78813	597789
nz (# nonzeros)	769248	2948472	23386248
# cross-sections	64	64	128
h	0.3	0.15625	0.078125

Table 2: Data relating to the meshes for problem ($\mathcal{P}_2$).

set is a realistic application. It will be used to illustrate the benefit of using the Hestenes and Stiefel criterion versus the residual based stopping criterion with the usual threshold of 10^{-8}. In particular, we will be able to indentify much sooner an iteration where the corresponding approximated solution is satisfactory from the mathematical and engineering point of views.

4.2 Numerical results

The lower and upper bounds based on the Gauss-Radau rule need an estimate of, respectively, the largest and smallest eigenvalues of the preconditioned matrix. We should point out that this is a serious handicap, insofar as the computation of these two eigenvalues can be as costly as the solution of the system by PCG. In particular, the Gauss-Radau lower bound that relies on the assumption that the largest eigenvalue of $M^{-1}A$ is less than $||A||_2 \approx 7.8$ does not improve the Hestenes and Stiefel bound. In Figure 10, we exhibit the ratio between the two estimates for problem $\mathcal{P}_1$ and for all the meshes. We have the same behaviour for problem $\mathcal{P}_2$. Owing to this poor gain, we omit in what follows the results of the Gauss-Radau lower bound.

We assume that the energy norm $\tilde{\mathfrak{E}}$ of the computed solution on the finest mesh is a good approximation of the true energy norm $\mathfrak{E}$ of the continuous solution. This assumption enables us to estimate the true energy norm error $\mathfrak{E}_k^{(p)}$, $p = 8, 16, 32$ at step k and on each of the coarser

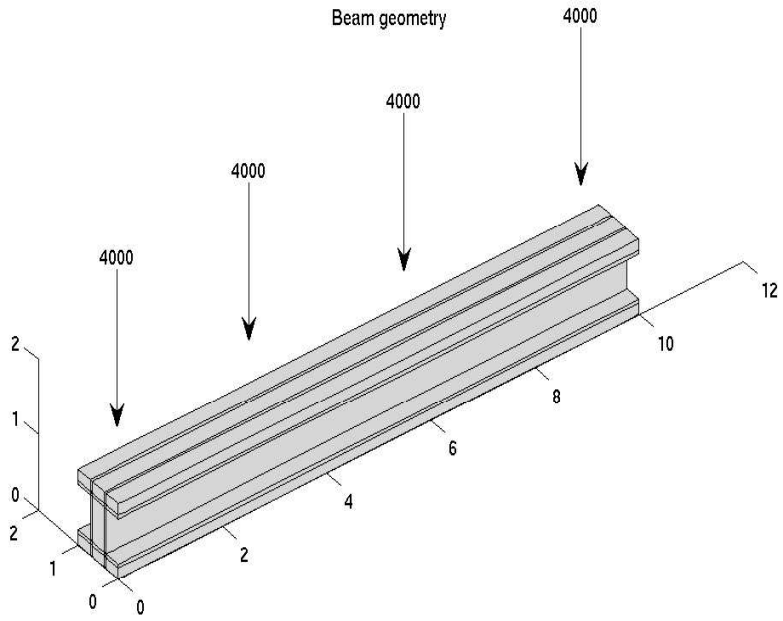

Figure 8: Geometry for problem $(\mathcal{P}_2)$

meshes using the simple formula

$$\mathfrak{E}_k^{(p)} = \sqrt{\frac{|\tilde{\mathfrak{E}}^2 - \texttt{rinfo(1)}|}{\tilde{\mathfrak{E}}^2}}, \tag{9}$$

where `rinfo(1)` stores the value of `norm_u_est` at step k (see Figure 4).

The error estimates are compared with the values of e_k given by the formula

$$e_k^{(p)} = \sqrt{\sum_{i=k}^{\tilde{k}} \psi_i^{(p)}}, \qquad p = \{8, 16, 32\}, \tag{10}$$

on each mesh, and where $\tilde{k}$ denotes the final iteration step index.

4.2.1 Discussion of results for the model problem $(\mathcal{P}_1)$

In Figures 11, 12, and 13, we compare the values $\mathfrak{E}_k^{(p)}$, $p = 8, 16, 32$ with the values of e_k given by the formula (10). The value of $\tilde{\mathfrak{E}}$ on mesh 64 is 3.391196410^4.

In all the plots, the two curves are very close until the approximation error in energy norm becomes dominant. The discontinuous coefficient in $\mathcal{P}_1$ makes the regularity of the solution very poor and we cannot expect a better global error (see Petzoldt, 2001) even if we use quadratic Lagrangian elements.

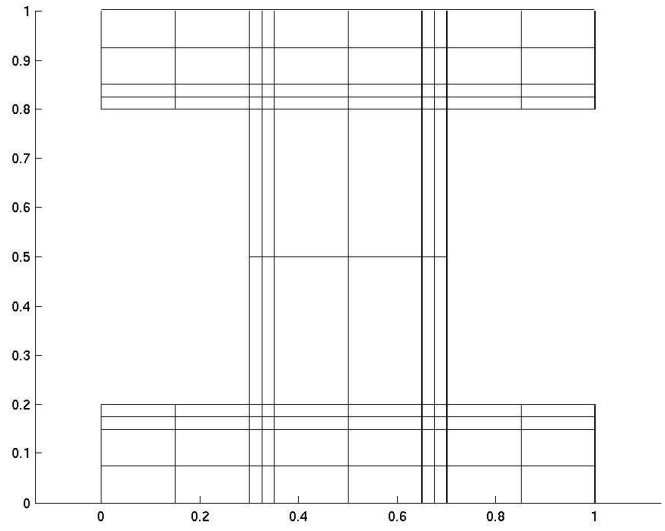

Figure 9: Cross section with a mesh example for problem $(\mathcal{P}_2)$

In Figures 14, 15, and 16, we compare the Hestenes and Stiefel lower bound, the Gauss-Radau upper bound, and the preconditioned residual with the error. In order to have the Gauss-Radau upper bound, we need to input a lower bound λ of the smallest eigenvalue of $M^{-1}A$. In our test, we choose

$$\lambda = h \min_{\mathbf{x}\in\Omega} c(\mathbf{x}).$$

This choice is the result of a "*trial-and-error*" procedure. In all the experiments, we choose the delay parameter $d = 10$, and, for this set of problems, the tolerance $\eta = h^2$. This will not guarantee a global error of order $\mathcal{O}(h^2)$ but will help in having $\mathcal{O}(h^2)$ error in the regular subsets of Ω_1 and of $\Omega \setminus \Omega_1$ far away from the discontinuity on $c(\mathbf{x})$.

The first set of test problems shows that the Gauss-Radau upper bound is reliable but is also more expensive in terms of additional iterations. The Hestenes and Stiefel criterion may suffer from oscillations during the initial phase; nonetheless, after the super-linear convergence has started, it becomes very efficient. However, in all our tests the oscillation amplitude of the Hestenes and Stiefel lower bound is smaller than h and becomes even smaller by increasing the value of the parameter d to 20. Note that the value of the preconditioned residual is quite close to the Gauss-Radau upper bound. This fact suggests that the incomplete Cholesky preconditioner works fine and that the resulting matrix M preserves the energy properties of the matrix A.

4.2.2 Discussion of results for the model problem $(\mathcal{P}_2)$

In Figures 17, and 18, we compare the values $\mathfrak{E}_k^{(p)}$, $p = 8, 16, 32$ with the values of e_k given by the formula (10). The value of $\tilde{\mathfrak{E}}$ on mesh 8 is 1.147732910^{-1}. We found the same behaviour we had for the model problem $(\mathcal{P}_1)$.

In Figure 19 and Figure 20, we plot the errors and the estimates of Hestenes and Stiefel, Gauss-Radau (only the upper bound) and the 2-norm of the preconditioned residual. For the second set of test problems, the Gauss-Radau upper bound seems less efficient and more conservative than necessary. As before, we have chosen $d = 10$. Moreover, we have assumed that the smallest eigenvalue of $M^{-1}A$ is $\approx 10^{-4}$. The energy norm error is quite high owing to the poor global

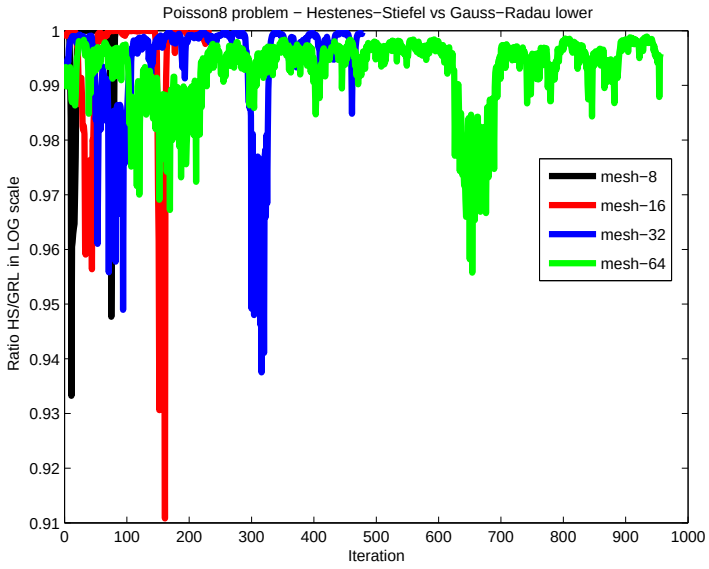

Figure 10: Ratio between the Hestenes-Stiefel and Gauss-Radau lower bounds for problem ($\mathcal{P}_1$)

regularity of the solution. This is a direct consequence of the geometry of the domain. The Hestenes and Stiefel bound is still reliable and suggests that a threshold $\eta = \mathcal{O}(h^2)$ will guarantee that the computed solution has an energy norm error smaller than the true error without requiring several additional iterations. We point out that for mesh 8, a stopping criteria requiring that the residual of the preconditioned problem be less than 10^{-8} would have required ≈ 1900 iterations to reach convergence. A stopping criterion based on the Hestenes and Stiefel bound requires 350 iterations to have the energy norm less than $\mathcal{O}(h^2)$.

5 Conclusions

The numerical experiments performed on the two sets of test problems support the use of stopping criteria based on the energy norm of the error. In particular, the Hestenes and Stiefel criterion is numerically stable and does not require any *a priori* information regarding the preconditioned problem. Moreover, the numerical experiments show that the conjugate gradient method, applied to matrices related to the solution of partial differential equations with not a very regular solution, needs one of our stopping criteria in order to avoid unnecessary iterations. The stopping criterion based on the Euclidean norm of the preconditioned residual is not related to the energy norm of the error and, thus, is not able to predict when we have a stable solution.

References

Arioli, M. (2004), 'A stopping criterion for the conjugate gradient algorithm in a finite element method framework', *Numer. Math.* **97**, 1–24. Electronic version: DOI: 10.1007/s00211-003-0500-y.

Arioli, M., Duff, I. S. and Ruiz, D. (1992), 'Stopping criteria for iterative solvers', *SIAM J. Matrix Anal. Appl.* **13**(1), 138–144.

Arioli, M., Noulard, E. and Russo, A. (2001), 'Stopping criteria for iterative methods: Applications to PDE's', *CALCOLO* **38**, 97–112.

Ciarlet, P. (1978), *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, The Netherlands.

FEMLAB (2004), 'FEMLAB ©COMSOL'. See http://www.femlab.com.

Golub, G. and Meurant, G. (1997), 'Matrices, moments and quadrature II; how to compute the norm of the error in iterative methods', *BIT* **37**, 687–705.

Greenbaum, A. (1997), *Iterative Methods for Solving Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Hestenes, M. and Stiefel, E. (1952), 'Methods of conjugate gradients for solving linear systems', *J. Res. Nat. Bur. Standards* **49**, 409–436.

Lin, C.-J. and Moré, J. J. (1999), 'Incomplete cholesky factorizations with limited memory', *SIAM Journal on Scientific Computing* **21**, 24–45.

Meurant, G. (1999*a*), *Computer Solution of Large Linear Systems*, Vol. 28 of *Studies in Mathematics and its Application*, Elsevier/North-Holland, Amsterdam, The Netherlands.

Meurant, G. (1999*b*), 'Numerical experiments in computing bounds for the norm of the error in the preconditioned conjugate gradient algorithm', *Numerical Algorithms* **22**, 353–365.

Petzoldt, M. (2001), Regularity and error estimators for elliptic problems with discontinuous coefficients, PhD thesis, Fachbereich Mathematik u. Informatik, Freie Universitt Berlin.

Strakoš, Z. and Tichý, P. (2002), 'On error estimation by conjugate gradient method and why it works in finite precision computations', *Electronic Transactions on Numerical Analysis* **13**, 56–80.

Strakoš, Z. and Tichý, P. (2004), 'Error estimation in preconditioned conjugate gradients', *To appear on BIT*.

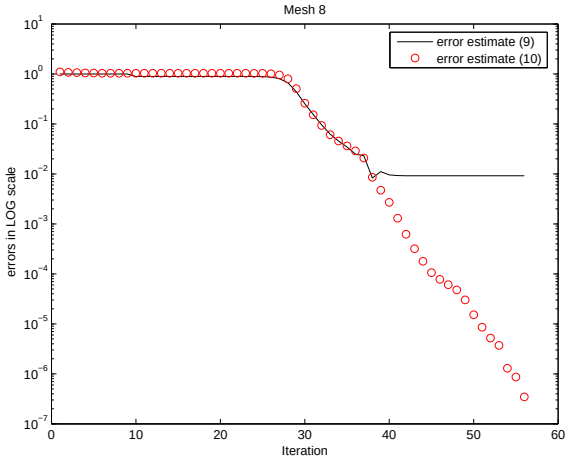

Figure 11: $\mathfrak{E}_k^{(8)}$ **(solid line)** vs $e_k^{(8)}$ **(circles)** for problem $(\mathcal{P}_1)$ and mesh 8

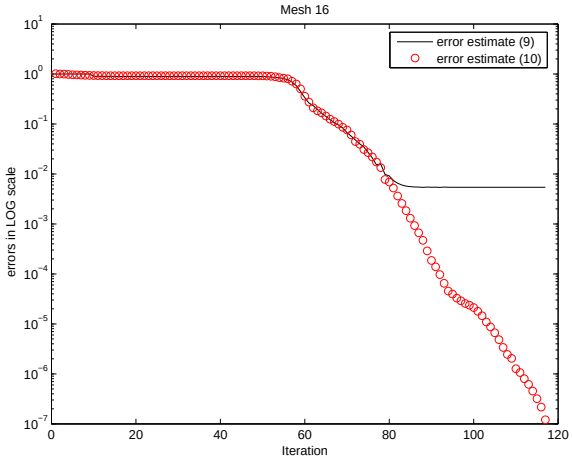

Figure 12: $\mathfrak{E}_k^{(16)}$ **(solid line)** vs $e_k^{(16)}$ **(circles)** for problem $(\mathcal{P}_1)$ and mesh 16

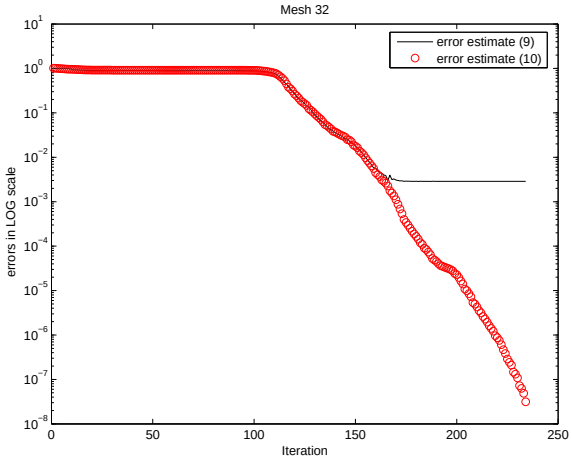

Figure 13: $\mathfrak{E}_k^{(32)}$ **(solid line)** vs $e_k^{(32)}$ **(circles)** for problem $(\mathcal{P}_1)$ and mesh 32

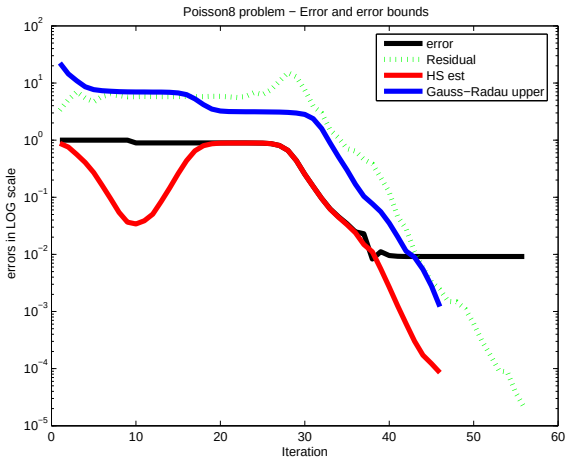

Figure 14: Error estimates for problem $(\mathcal{P}_1)$ and mesh 8

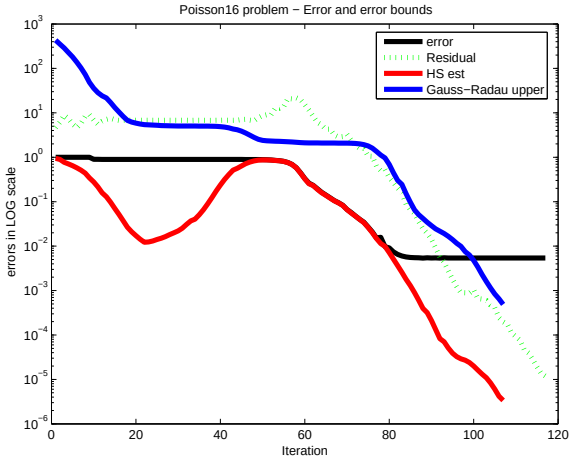

Figure 15: Error estimates for problem $(\mathcal{P}_1)$ and mesh 16

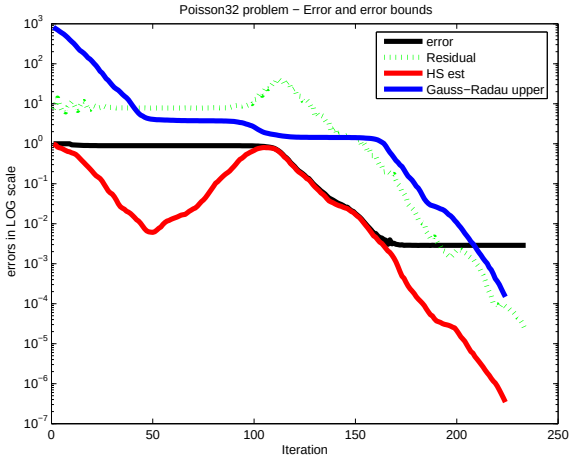

Figure 16: Error estimates for problem $(\mathcal{P}_1)$ and mesh 32

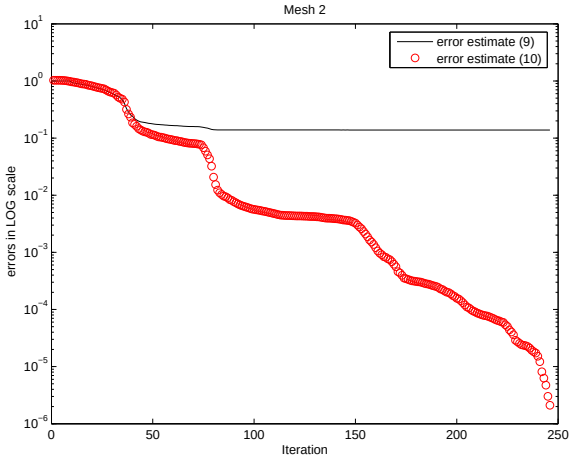

Figure 17: $\mathfrak{E}_k^{(8)}$ **(solid line)** vs $e_k^{(8)}$ **(circles)** for problem $(\mathcal{P}_1)$ and mesh 8

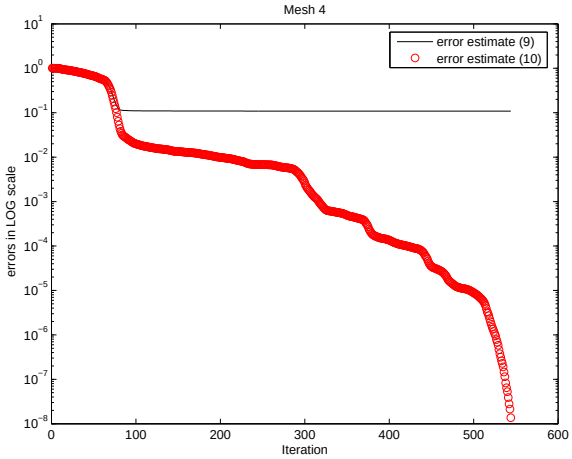

Figure 18: $\mathfrak{E}_k^{(16)}$ **(solid line)** vs $e_k^{(16)}$ **(circles)** for problem $(\mathcal{P}_1)$ and mesh 16

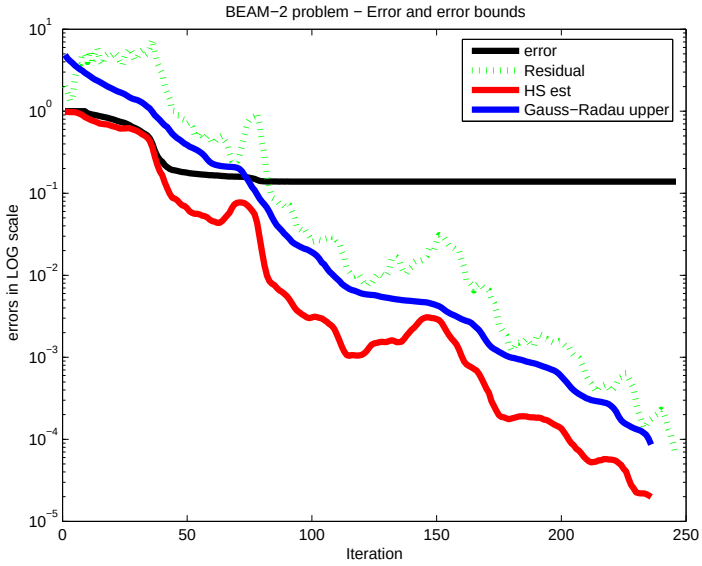

Figure 19: Error estimates for problem ($\mathcal{P}_2$) and mesh 8

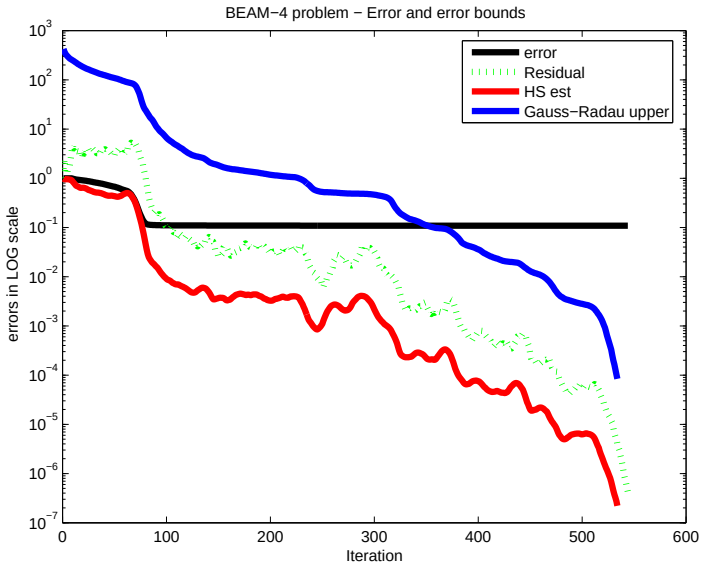

Figure 20: Error estimates for problem ($\mathcal{P}_2$) and mesh 16