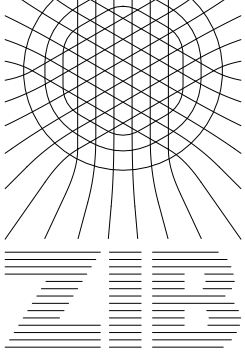
Konrad-Zuse-Zentrum für Informationstechnik Berlin



U. Nowak L. Weimann

GIANT – A Software Package for the Numerical Solution of Very Large Systems of Highly Nonlinear Equations

Technical Report TR 90–11 (December 1990)

GIANT – A Software Package for the Numerical Solution of Very Large Systems of Highly Nonlinear Equations

U. Nowak L. Weimann

Abstract

This report presents the final realization and implementation of a global inexact Newton method proposed by Deuflhard. In order to create a complete piece of software, a recently developed iterative solver (program GBIT) due to Deuflhard, Freund, Walter is adapted and serves as the standard iterative linear solver. Alternative linear iterative solvers may be adapted as well, e.g. the widely distributed code GMRES. The new software package GIANT (Global Inexact Affine Invariant Newton Techniques) allows an efficient and robust numerical solution of very large scale highly nonlinear systems. Due to the user friendly interface and its modular design, the software package is open for an easy adaptation to specific problems. Numerical experiments for some selected problems illustrate performance and usage of the package.

Contents

List of Figures

List of Tables

0. Introduction

For the numerical solution of systems of highly nonlinear equations

$$\begin{array}{ll} a) & F(x) = 0, \qquad x \in \mathbb{R}^n \\ b) & x^0 \text{ given initial guess} \end{array} \tag{0.0.1}$$

global affine invariant Newton techniques turn out to be quite efficient and robust (compare Deuflhard [2] and the more recent monograph [3]). Within the course of the Newton iteration a sequence of linear problems must be solved. As long as the dimension n of the system (0.0.1) is of moderate size, this can be done very efficiently by applying direct methods, e.g. Gaussian elimination techniques. Using special implementations of direct methods, like band mode elimination or sparse matrix techniques, even large systems with special structures can be attacked. Based on global affine invariant *exact* Newton techniques, software for this type of problem is presented in Nowak/Weimann [10]. But for very large scale problems, e.g. discretized partial differential equations in 2 or 3 dimensions, an *iterative* solution of the linear systems may be the only method of choice. In this case, one has two nested iteration processes: the Newton iteration, appearing as the *outer* iteration, and the iterative solution of the arising linear systems appearing as the *inner* iteration. In view of an efficient realization the question of how to choose the (required) accuracy for the inner iteration is of essential importance. Too weak accuracy requirements may destroy convergence (or slow down convergence speed) of the outer (Newton) iteration. Too stringent accuracy requirements will increase computing time drastically without increasing convergence speed of the outer iteration. Besides this, note that an only approximate solution of the linear systems means that only a so called *inexact* Newton scheme is available. The derivation of a cheaply implementable extension of affine invariant Newton methods to the case of inexact Newton methods, including an accuracy matching strategy, has been studied by Deuflhard [4].

It is the purpose of the present paper to describe the final realization and implementation of these techniques within the software package GIANT (mnemotechnically for Global Inexact Affine invariant Newton Techniques). In order to get a running code, two iterative linear solvers are presently included in the GIANT package. First, there is the well-known and widely used "Generalized Minimum Residual" (GMRES) code due to Brown/Hindmarsh/ Seager from the SLAP package of Seager/Greenbaum [11, 7]. As the underlying minimization principle of GMRES does not match the theoretical background of global affine invariant Newton methods, a recently developed iterative linear solver due to Deuflhard/Freund/Walter [5] has been implemented and adapted for the use in GIANT. This fast secant method GBIT1,

which uses so-called "Good Broyden" updates and a special, adapted line search principle, fits perfectly into the theoretical frame of GIANT. Both iterative methods for the solution of the linear systems may be combined with preconditioning techniques to improve their convergence properties.

This paper is organized as follows: In Section 1.1 the GIANT algorithm is presented in detail. The accuracy matching strategy for the inner and outer iteration and the consequence for an "optimal" iterative linear solver are discussed. Section 1.2 presents the iterative linear solver GBIT1 and it's adaptation for use in GIANT. Section 1.3 deals with details and variants of the algorithmic realization, e.g. choice of norms, internal scaling and restricted damping strategy. Chapter 2 deals with the implementation of the software package. First, a general overview is given in Section 2.1. Section 2.2 contains the description of the user interfaces and Section 2.3 describes some special features of the package. Typical numerical experiments are reported in Chapter 3. First, in Section 3.1, a set of test problems is established and fully described. The numerical results of solving these test examples with GIANT+GBIT1 and GIANT+GMRES respectively, are presented and discussed in Section 3.2. Finally, some concluding remarks are made.

1. Algorithms

1.1 Global Inexact Newton Scheme

The global *inexact* Newton methods proposed in [4] are extensions of the global *exact* Newton techniques where the arising linear systems are solved by direct ("exact") methods (cf. [2, 3]). The term "global" indicates that the usual Newton scheme is combined with an *affine invariant damping strategy* in order to extend the convergence domain of the method. Omitting details which are of minor interest in this context, the exact damped affine invariant Newton algorithm to solve (0.0.1) reads as follows:

Global Exact Affine Invariant Newton Scheme (Algorithm E)

Input:

x_0 initial guess for the solution
λ_0 initial damping factor
tol required accuracy for the solution
user routine to evaluate the nonlinear system function $F(x)$
user routine to evaluate the Jacobian of the system $J(x) := \frac{\partial F}{\partial x}$
(may be dummy as internal numerical differentiation procedures
may be used)
standard routines for direct solution of linear equations

Start:

$$k := 0$$

evaluate system

$$F_k := F(x_k)$$

Newton step:

evaluate Jacobian

$$J_k := J(x_k)$$

compute ordinary Newton correction

$$\Delta x_k := -J_k^{-1} F_k \tag{1.1.1}$$

compute a priori damping factor

$$\begin{array}{ll} \text{if } (k>0) & \lambda_k^{(0)} := \min\left\{1, \dfrac{1}{[h_k^{(0)}]}\right\} \quad ([h_k^{(0)}] \text{ see below (1.1.6) }) \\ \text{else} & \lambda_k^{(0)} := \lambda_0 \end{array} \tag{1.1.2}$$

$$j := 0$$
$$\lambda := \lambda_k^{(0)}$$

a posteriori loop

compute trial iterate

$$x_{k+1}^{(j)} := x_k + \lambda \Delta x_k$$

evaluate system

$$F_{k+1}^{(j)} := F(x_{k+1}^{(j)})$$

compute simplified Newton correction

$$\overline{\Delta x}_{k+1}^{(j)} := -J_k^{-1} F_{k+1}^{(j)} \tag{1.1.3}$$

termination check

$$\text{exit, if } \|\Delta x_k\| \le tol$$

compute a posteriori damping factor

$$\lambda_k^{(j+1)} := \min\left\{1, \frac{1}{[h_k^{(j+1)}]}\right\} \quad ([h_k^{(j+1)}] \text{ see below (1.1.7) }) \tag{1.1.4}$$

monotonicity check

$$konv := \|\overline{\Delta x}_{k+1}^{(j)}\| \le \|\Delta x_k\| \tag{1.1.5}$$

$$\begin{array}{lll} \text{if } konv & : & \overline{\Delta x}_{k+1} := \overline{\Delta x}_{k+1}^{(j)} \\ & & x_{k+1} := x_{k+1}^{(j)} \\ & & F_{k+1} := F_{k+1}^{(j)} \\ & & \lambda_k := \lambda \\ & & k := k+1 \\ & & \text{proceed at } \textit{Newton step} \\ \text{else:} & & j := j+1 \\ & & \lambda := \min\{\lambda_k^{(j)}, \dfrac{\lambda}{2}\} \\ & & \text{proceed at } \textit{a posteriori loop} \end{array}$$

To evaluate the damping factor $\lambda_k^{(j)}$ in (1.1.2) and (1.1.4) the following local estimates $[h_k^{(j)}]$ are used:
A priori estimate:

$$[h_k^{(0)}] := \frac{\|\overline{\Delta x}_k - \Delta x_k\|}{\lambda_{k-1}\|\Delta x_{k-1}\|} \cdot \frac{\|\Delta x_k\|}{\|\overline{\Delta x}_k\|} \tag{1.1.6}$$

A posteriori estimate:

$$[h_k^{(j)}] := \frac{2}{\lambda^2} \cdot \frac{\|\overline{\Delta x}_{k+1}^{(j)} - (1-\lambda)\Delta x_k\|}{\|\Delta x_k\|} \tag{1.1.7}$$

Note that this scheme requires, at least, two linear system solutions (with the same matrix) per Newton step — compared to just one in a "classic" Newton step. But within the implementation for systems of moderate size or with special structure in the matrix J_k (Codes NLEQ1, NLEQ1S, NLEQ2 — see [10]) this additional amount of work is comparatively small. As the matrix J_k is decomposed first, the solution of (1.1.1) and (1.1.3) reduces to backward-forward substitutions only. Thus, the additional costs are usually small compared to the total amount of work for one Newton step. Now, if the *direct* solution of (1.1.1) and (1.1.3) is straightforward replaced by an *iterative* solution, this additional amount of work for the computation of the simplified Newton correction may be significant. Therefore, one objective of the accuracy matching strategy of the global inexact affine invariant Newton scheme presented in [4] is to overcome this difficulty. It is not the purpose of this paper to repeat the detailed considerations made in [4] to derive this scheme and an associated accuracy matching strategy. Nevertheless, the basic ideas and consequences for the algorithm (E) are shortly summarized to facilitate the understanding of the final algorithm and to support an efficient application of the software package GIANT.

1.1.1 Adaptation of outer iteration

Using an iterative method for the computation of the ordinary and simplified Newton correction means to replace the "exact" solutions Δx_k and $\overline{\Delta x}_{k+1}$ of (1.1.1) and (1.1.3) respectively by only approximate "inexact" solutions of these systems. To be more precise, let s_k denote the inexact ordinary Newton correction and $\overline{s}_{k+1}$ the inexact simplified Newton correction. Assume that reasonable error estimates ε_k , $\overline{\varepsilon}_{k+1}$ are available for the relative errors of s_k and $\overline{s}_{k+1}$:

$$\begin{aligned} a) \quad & \varepsilon_k \approx \varepsilon_k^{true} := \frac{\|s_k - \Delta x_k\|}{\|s_k\|} \\ b) \quad & \overline{\varepsilon}_{k+1} \approx \overline{\varepsilon}_{k+1}^{true} := \frac{\|\overline{s}_{k+1} - \overline{\Delta x}_{k+1}\|}{\|\overline{s}_{k+1}\|} \end{aligned} \tag{1.1.8}$$

Then, as a first consequence of the replacement

$$\Delta x_k, \overline{\Delta x}_{k+1} \longrightarrow s_k, \overline{s}_{k+1} \tag{1.1.9}$$

in (E), the formulas for the computation of the optimal damping factors $\lambda_k^{(j)}$ ((1.1.2), (1.1.4)) must be modified to

$$\lambda_k^{(0)} = \min\left\{1, \frac{1-\hat{\varepsilon}_k}{[h_k^{(0)}]}\right\} \tag{1.1.10}$$

$$\lambda_k^{(j)} = \min\left\{1, \frac{1-\hat{\overline{\varepsilon}}_{k+1}}{[h_k^{(j)}]}\right\} \quad j = 1, 2, \ldots \tag{1.1.11}$$

where

$$\hat{\varepsilon}_k := \frac{\varepsilon_k}{1-\varepsilon_k}\,, \qquad \hat{\overline{\varepsilon}}_{k+1} := \frac{\overline{\varepsilon}_{k+1}}{1-\overline{\varepsilon}_{k+1}}\,. \tag{1.1.12}$$

Besides this partial consequence, the replacement (1.1.9) affects the algorithm also as a whole. Errors of s_k , $\overline{s}_{k+1}$ may disturb the damping strategy due to (1.1.6) and (1.1.7), and due to (1.1.5) the convergence behavior, and finally, the superlinear convergence of the inexact Newton iterates s_k to the solution x^* of (0.0.1) may be destroyed.

1.1.2 Accuracy matching strategy

In order to retain the behavior of the exact Newton scheme also for the inexact scheme, a special accuracy matching strategy is used (see [4] for details). Roughly speaking, this strategy sets only weak accuracy requirements for the errors ε_k , $\overline{\varepsilon}_{k+1}$ as long as the Newton iteration is "far away" from the solution ($\lambda_k < 1$, $h_k > 1$) and sets successively more stringent accuracy requirements for the error ε_k if the Newton iteration approaches to the solution x^* ($\lambda_k = 1$, $h_k < 1$) , thus ensuring superlinear convergence. Note that only ε_k is reduced, while $\overline{\varepsilon}_{k+1}$ not. Due to this fact, the additional costs for the computation of the inexact simplified Newton correction are again comparatively small - as in the standard case.

Within this accuracy matching strategy the *required* accuracy $(\varepsilon_k^{req}, \overline{\varepsilon}_{k+1}^{req})$ for the solutions s_k , $\overline{s}_{k+1}$ is set by

$$\begin{aligned}
&a) \qquad \hat{\varepsilon}_k^{req} := \rho \cdot \min\left(\frac{1}{1+\rho}, [h_k^{(0)}]\right) \\
&b) \qquad \varepsilon_k^{req} := \frac{\hat{\varepsilon}_k^{req}}{1+\hat{\varepsilon}_k^{req}} \\
&c) \qquad \overline{\varepsilon}_{k+1}^{req} := \frac{\rho}{1+2\rho}
\end{aligned} \tag{1.1.13}$$

with

$$d) \qquad \rho := \text{ safety factor, to be chosen } \leq \frac{1}{6}$$

Notice that the considerations made in [3] allow a minimum value of $\rho = 1/2$ ($\Rightarrow \varepsilon_k^{req} = 1/4$), which means that just one binary digit in $[h_k^{(j)}]$ and two binary digits in s_k , $\overline{s}_{k+1}$ must be correct. In spite of this theoretical upper bound a maximum value $\rho_{max} = 1/6$ should be used in order to synchronize the inexact scheme with the exact one.

Concerning the evaluation of (1.1.13), note that s_k enters via $[h_k^{(0)}]$ (c.f. (1.1.6)) into the right hand side of (1.1.13.a), which means, that the required accuracy for the solution s_k of the iterative solver depends on this solution itself. To overcome this difficulty, this iterative solution process is split up into an a priori and an a posteriori part — see Section 1.1.3. The robustness of the total global inexact Newton scheme strongly depends on the assumption, that the prescribed accuracy is really achieved. Due to this fact, the iterative linear solver should have a reasonable and robust error estimate for it's solution. Finally, one should mention, that under some circumstances the needed accuracy to have reasonable $[h_k^{(0)}]$-estimates at hand can be achieved without fulfilling (1.1.13). To utilize this fact, a special stopping criterion for the iterative solver may be used — see formula (1.1.18.b) of Section 1.1.3.

1.1.3 Adaptation of the inner iteration

In general, the efficiency of an iterative linear solver strongly depends on the quality of the initial guess. Within the iterative Newton scheme quite natural initial guesses for successive iterative linear system solutions are available. For the iteration towards s_k, $k > 0$, one may take $\overline{s}_k$ as initial guess and for the iteration towards $\overline{s}_{k+1}$ one may use $(1-\lambda_k)s_k$. Due to the special accuracy requirements of the inexact Newton iteration, a special formulation of the inner iteration is suggested (c.f. [4]). Recall that within the computation of the estimates $[h_k^{(j)}]$ $j = 0, 1, \ldots$ (c.f. (1.1.6),(1.1.7)), the differences

$$\begin{aligned} &a) \quad \delta s_k := s_k - \overline{s}_k \\ &b) \quad \delta\overline{s}_{k+1} := \overline{s}_{k+1} - (1-\lambda_k)s_k \end{aligned} \tag{1.1.14}$$

are required. With this in mind, an iteration with direct computation of (1.1.14.a,b) would be preferable to avoid cancellation of leading digits. To be precise, instead of an iteration in the form

$$\begin{aligned} &a) \quad s_k^0 := \overline{s}_k \\ &b) \quad s_k^{i+1} := s_k^i + \Delta_k^i \qquad\qquad i = 0, 1, 2, \ldots \end{aligned} \tag{1.1.15}$$

the inner iteration should be realized in the following way:

$$\begin{aligned} &a) \quad \delta s_k^0 := 0 \ , \quad s_k^0 := \overline{s}_k \\ &b) \quad \delta s_k^{i+1} := \delta s_k^i + \Delta_k^i \qquad\qquad i = 0, 1, 2, \ldots \\ &c) \quad s_k^{i+1} := s_k^0 + \delta s_k^{i+1} \end{aligned} \tag{1.1.16}$$

Accordingly, the inner iteration for $\overline{s}_{k+1}$ should read:

$$\begin{aligned} &a) \quad \delta \overline{s}_{k+1}^0 := 0 \ , \quad \overline{s}_{k+1}^0 := (1 - \lambda_k) s_k \\ &b) \quad \delta \overline{s}_{k+1}^{i+1} := \delta \overline{s}_{k+1}^i + \overline{\Delta}_k^i \qquad\qquad i = 0, 1, 2, \ldots \\ &c) \quad \overline{s}_{k+1}^{i+1} := \overline{s}_{k+1}^0 + \delta \overline{s}_{k+1}^{i+1} \end{aligned} \tag{1.1.17}$$

Furthermore, having an inner iteration of type (1.1.16) and (1.1.17) respectively, the realization of a termination criterion which fits perfectly into the requirements of the accuracy matching strategy is facilitated.

Let $\alpha(\overline{\Delta}^i)$ denote an estimate of the absolute error of both $\overline{s}_{k+1}^i$ and $\delta \overline{s}_{k+1}^i$. With $\overline{\varepsilon}_{k+1}^{req}$ defined by (1.1.13.c,d) the inner iteration can be stopped if either

$$\begin{aligned} &a) \quad \alpha(\overline{\Delta}_{k+1}^i) \leq \overline{\varepsilon}_{k+1}^{req} \|\overline{s}_{k+1}^{i+1}\| \\ &\text{or} \\ &b) \quad \alpha(\overline{\Delta}_{k+1}^i) \leq \tfrac{1}{4} \|\delta \overline{s}_{k+1}^{i+1}\| \end{aligned} \tag{1.1.18}$$

holds. In both cases, $\overline{s}_{k+1}$ is accurate enough to yield reasonable a posteriori estimates $[h_k^{(j)}], \quad j = 1, 2, \ldots$ via (1.1.7) and a satisfactory performance of the inexact monotonicity test (1.1.22) — see Section 1.1.5.

For the iteration towards s_k an analogue termination criterion is used first. In order to overcome the problem of the implicit definition of ε_k^{req} via (1.1.13.a,b,d), the inner iteration is split up into an a priori part which may be followed by an a posteriori part. Let $\varepsilon_k^{req,0}$ be defined by (1.1.13.c,d) - which is just (1.1.13.a,b,d) ignoring the $[h_k^{(0)}]$ term. Then the inner iteration is stopped if either

$$\begin{aligned} &a) \quad \alpha(\Delta_k^i) \leq \varepsilon_k^{req,0} \|s_k^{i+1}\| \\ &\text{or} \\ &b) \quad \alpha(\Delta_k^i) \leq \tfrac{1}{4} \|\delta s_k^{i+1}\| \end{aligned} \tag{1.1.19}$$

holds. Again, there is enough accuracy (now in s_k) to get a reasonable a priori estimate $[h_k^{(0)}]$ via (1.1.6) and a satisfactory performance of the inexact monotonicity test (1.1.22).

But only in the case $\lambda_k^{(0)} < 1$ and stopping criterion (1.1.19.a) has been activated, s_k is accurate enough to allow a satisfactory performance of the inexact Newton scheme. Thus, the iteration must be continued in other

cases. But for this a posteriori inner iteration, an $\varepsilon_k^{req,1}$ can now be defined via (1.1.13.a,b,d), as a reasonable estimate $[h_k^{(0)}]$ is at hand. This a posteriori inner iteration is continued until the criterion (1.1.19.a) is met — with $\varepsilon_k^{req,0}$ replaced by $\varepsilon_k^{req,1}$. In order to get a hint for a crude violation of the theoretical assumptions which led to the above stopping conditions one may realize a simple check. With s_k from the a posteriori iteration at hand, one can recompute $[h_k^{(0)}]$ and compare this value with that one computed with s_k from the a priori iteration.

1.1.4 Simplified adaptation

The above optimal adaptation of inner and outer iteration may exclude most of the available software from a direct use within GIANT. Therefore one may realize a simplified interaction between inner and outer iteration. Assume, an iterative solver is at hand which requires an initial guess s^0 and a prescribed tolerance ε^{req} and yields an approximate solution s^{iter} with an associated error estimate ε^{est}. Then, the interaction between GIANT and such a solver may be realized by the following simple procedures:

For simplified Newton correction:

$$\begin{aligned}
&\overline{\varepsilon}_{k+1}^{req} := \frac{\rho}{1+2\rho} \\
&\overline{s}_{k+1}^{0} := (1-\lambda_k) s_k \\
&\text{iterative linear solution} \\
&\qquad \hookrightarrow \overline{s}_{k+1}^{iter},\ \overline{\varepsilon}_{k+1}^{est} \\
&\overline{s}_{k+1} := \overline{s}_{k+1}^{iter} \\
&\text{proceed with inexact Newton scheme}
\end{aligned} \tag{1.1.20}$$

For ordinary Newton correction:

$$
\begin{aligned}
&\varepsilon_k^{req,0} := \frac{\rho}{1+2\rho} \\
&\text{if } (k>0) \quad s_k^0 := \overline{s}_k \\
&\text{else} \qquad\quad s_k^0 := 0 \\
&\text{iterative linear solution} \\
&\qquad \hookrightarrow s_k^{iter,0},\ \varepsilon_k^{est,0} \\
&[h_k^{(0)}]^0 := \frac{\|\overline{s}_k - s_k^{iter,0}\|}{\|x_k - x_{k-1}\|} \cdot \frac{\|s_k^{iter,0}\|}{\|\overline{s}_k\|} \\
&\lambda_k^{(0)} := \min\left\{1, \frac{1}{[h_k^{(0)}]^0}\right\} \\
&\text{if } (\lambda_k^{(0)} < 1) \\
&\qquad s_k := s_k^{iter,0} \\
&\qquad \text{proceed with inexact Newton scheme} \\
&\text{else} \\
&\qquad \varepsilon_k^{req,1} := \rho \cdot [h_k^{(0)}]^0 \\
&\qquad s_k^0 := s_k^{iter,0} \\
&\qquad \text{iterative linear solution (continued)} \\
&\qquad\qquad \hookrightarrow s_k^{iter,1},\ \varepsilon_k^{est,1} \\
&\qquad [h_k^{(0)}]^1 := \frac{\|\overline{s}_k - s_k^{iter,1}\|}{\|x_k - x_{k-1}\|} \cdot \frac{\|s_k^{iter,1}\|}{\|\overline{s}_k\|} \\
&\qquad \text{if } ([h_k^{(0)}]^1 - [h_k^{(0)}]^0) \text{ "not small"} \ \rightarrow\ \text{warning message} \\
&\qquad s_k := s_k^{iter,1} \\
&\qquad \text{proceed with inexact Newton scheme}
\end{aligned}
\tag{1.1.21}
$$

As mentioned above, these schemes are simplifications of the optimal strategy. But the the loss of efficiency and robustness is acceptable. Nevertheless, for the adaptation of the standard iterative solver "Good Broyden" (see Section 1.2) the optimal strategy is realized.

1.1.5 Inexact monotonicity test

Instead of just replacing $\Delta x_k, \overline{\Delta x}_{k+1}$ by $s_k, \overline{s}_{k+1}$ in the exact monotonicity test (1.1.5) an inexact monotonicity test in a less restrictive form turns out

to be favorable:

$$\|\overline{s}_{k+1}\| \cdot (1 - \overline{\varepsilon}_{k+1}) \leq \|s_k\| \cdot (1 + \varepsilon_k^{req}) \tag{1.1.22}$$

where

$$\varepsilon_k^{req} := \begin{cases} \varepsilon_k^{req,1} , & \text{if a posteriori inner iteration is done} \\ \varepsilon_k^{req,0} , & \text{else} \end{cases}$$

$$\overline{\varepsilon}_{k+1} := \max\{\overline{\varepsilon}_{k+1}^{req}, \overline{\varepsilon}_{k+1}^{est}\}$$

$$\overline{\varepsilon}_{k+1}^{est} \quad : \quad \text{estimated accuracy of solution } \overline{s}_{k+1}$$

As long as the inner iteration yields solutions which really have the required accuracy, (1.1.22) works quite satisfactory. In cases, when the true accuracy is too poor but the associated error estimate deceives enough accuracy, the monotonicity test may make the "wrong" decision. If (1.1.22) fails, the damping factor will be reduced and the check is subsequently activated with a recomputed simplified correction. But even with an exact solution for $\overline{s}_{k+1}$ at hand, this (and all further) test may fail, as the error of the ordinary Newton correction s_k may be too large. This difficulty may lead to a stop of the Newton iteration due to successive failures of the monotonicity test (1.1.22).

1.1.6 Inexact scheme

Based on the considerations above, one is now ready to present the whole inexact Newton scheme (with the simplified adaptation procedures). In order to have a direct connection to the associated software package GIANT the following algorithmic representation includes also the formal evaluation of the Jacobian. Note that within the Newton scheme this Jacobian is not directly used; only the routine for solving the linear problem may use this information. Therefore, GIANT is open for different ways of generating and storing the Jacobian — in connection with an associated realization of the iterative linear solver. A more detailed discussion of that topic can be found in Chapter 2, especially in Section 2.3.

Global Inexact Affine Invariant Newton Scheme (Algorithm I)

Input:
x_0 initial guess for the solution
λ_0 initial guess of damping factor
tol required accuracy for the solution
user routine to evaluate the nonlinear system function $F(x)$
special routines for iterative solution of linear equations
[user routine to evaluate the Jacobian of the system $J(x) := \frac{\partial F}{\partial x}$]

Start:

$$k := 0$$

evaluate system

$$F_k := F(x_k)$$

Newton step:

evaluate Jacobian

$$J_k := J(x_k)$$

compute ordinary Newton correction

$$\varepsilon_k^{req,0} := \frac{\rho}{1+\rho}\,,\ \rho \leq \frac{1}{6}$$

$$\text{if } (k>0) \quad : \ s_k^0 := \overline{s}_k$$

$$\text{else} \qquad : \ s_k^0 := 0$$

iterative linear solution procedure (see (1.1.21))

$$\text{input} \quad : \ s_k^0\,,\ \varepsilon_k^{req,0}$$

$$\text{output} \quad : \ s_k^{iter}\,,\ \varepsilon_k^{est,0}\,,\ [\varepsilon_k^{req,1}\,,\ \varepsilon_k^{est,1}]$$

$$s_k := s_k^{iter}$$

compute a priori damping factor

$$\text{if } (k>0) \quad \hat{\varepsilon}_k^{est} := \frac{\varepsilon_k^{est}}{1-\varepsilon_k^{est}}$$

$$[h_k^{(0)}] := \frac{\|\overline{s}_k - s_k\|}{\lambda_{k-1}\|s_{k-1}\|} \cdot \frac{\|s_k\|}{\|\overline{s}_k\|}$$

$$\lambda_k^{(0)} := \min\left\{1, \frac{1-\hat{\varepsilon}_k^{est}}{[h_k^{(0)}]}\right\}$$

$$\text{else} \qquad \lambda_k^{(0)} := \lambda_0$$

$$j := 0$$

$$\lambda := \lambda_k^{(0)}$$

a posteriori loop

compute trial iterate

$$x_{k+1}^{(j)} := x_k + \lambda s_k$$

evaluate system

$$F_{k+1}^{(j)} := F(x_{k+1}^{(j)})$$

compute simplified Newton correction

$$\overline{\varepsilon}_{k+1}^{req} := \frac{\rho}{1+\rho}\,,\ \rho \leq \frac{1}{6}$$

$$\overline{s}_{k+1}^{0} := (1-\lambda)s_k$$

iterative linear solution procedure (see (1.1.20))

input : $\overline{s}_{k+1}^{0}\,,\ \overline{\varepsilon}_{k+1}^{req}$

output : $\overline{s}_{k+1}^{iter}\,,\ \overline{\varepsilon}_{k+1}^{est}$

$$\overline{s}_{k+1}^{(j)} := \overline{s}_{k+1}^{iter}$$

$$\overline{\varepsilon}_{k+1} := \max\{\overline{\varepsilon}_{k+1}^{req}, \overline{\varepsilon}_{k+1}^{est}\}$$

termination check

$$\text{exit, if } \|s_k\| \leq tol$$

compute a posteriori damping factor

$$\hat{\overline{\varepsilon}}_{k+1} := \frac{\overline{\varepsilon}_{k+1}}{1-\overline{\varepsilon}_{k+1}}$$

$$[h_k^{(j+1)}] := \frac{2}{\lambda^2} \cdot \frac{\|\overline{s}_{k+1}^{(j)} - (1-\lambda)s_k\|}{\|s_k\|}$$

$$\lambda_k^{(j+1)} := \min\left\{1, \frac{1-\hat{\overline{\varepsilon}}_{k+1}}{[h_k^{(j+1)}]}\right\}$$

monotonicity check (see (1.1.22))

$$konv := \|\overline{s}_{k+1}^{(j)}\|(1-\overline{\varepsilon}_{k+1}) \leq \|s_k\|(1+\varepsilon_k^{req})$$

if *konv* :
- $\overline{s}_{k+1} := \overline{s}_{k+1}^{(j)}$
- $x_{k+1} := x_{k+1}^{(j)}$
- $F_{k+1} := F_{k+1}^{(j)}$
- $\lambda_k := \lambda$
- $k := k+1$
- proceed at *Newton step*

else:
- $j := j+1$
- $\lambda := \min\{\lambda_k^{(j)}, \frac{\lambda}{2}\}$
- proceed at *a posteriori loop*

1.2 Good Broyden Scheme

The standard method to perform the inner iterations within GIANT is the iterative linear solver "Good Broyden" due to Deuflhard/Freund/Walter [5]. To be more precise, it is the storage restricted variant of the algorithm "Good Broyden" with update (A) and linesearch (a) (notation from [5]). Numerical experiments with other variants of "Good Broyden" within the frame of GIANT confirm the observation in [5] that the above mentioned variant is preferable.

To fix notation, assume that the linear system

$$\begin{array}{ll} a) & As = b \\ b) & s_0 \text{ given initial guess} \end{array} \tag{1.2.1}$$

has to be solved. (One may identify A with J_k, b with F_k or $F_{k+1}^{(j)}$, and s with s_k or $\overline{s}_{k+1}$, to have the direct relation to Section 1.1). A preconditioning matrix H may be available, i.e. H is an approximation of A^{-1} and H may be easily computable. Furthermore, let $< \cdot, \cdot >$ denote an inner product and let $\| \cdot \|$ denote an induced norm of $< \cdot, \cdot >$ — see Section 1.3 for further discussion. Then, adapted for use in GIANT and omitting some technical details, the algorithm reads:

Good Broyden Iterative Scheme (Algorithm GBIT)

Input:

s_0 : initial guess for the solution
b : right hand side of the linear system
ε^{req} : required accuracy for the solution
k_{max} : storage restriction parameter
user routine to perform a matrix vector multiply: $y = Ax$
user routine to perform preconditioning: $y = Hx$

Preprocess:

$iter := 0$
$s_{iter} := s_0$
$\delta s_{iter} := 0$

Start/Restart:

$k := 0$

$r := b - As_{iter}$
$\Delta := Hr$
$\sigma :=< \Delta, \Delta >$

Iteration step:

$iter := iter + 1$
$q := A\Delta$
$z := Hq$

Update loop: $l = 1, 2, \cdots, k$ (for $k > 0$)

$f_1 :=< \Delta(l), z > /\sigma(l)$
$f_2 := 1 - \tau(l)$
if $(l < k)$: $z := z + f_1(\Delta(l+1) - f_2\Delta(l))$
else : $z := z + f_1(\Delta - f_2\Delta(l))$

end of update loop

$\gamma :=< \Delta, z >$
$\tau := \sigma/\gamma$
perform restart monitor (see below)
$\delta s_{iter} := \delta s_{iter-1} + \tau\Delta$
$s_{iter} := s_0 + \delta s_{iter}$
if $(k < k_{max})$
$\Delta(k+1) := \Delta$
$\sigma(k+1) := \sigma$
$\tau(k+1) := \tau$
endif
correction update: $\Delta := \Delta - \tau z$
$k := k + 1$
perform convergence monitor (see below)
if $(k \leq k_{max})$: proceed at *iteration step*
else : proceed at *start/restart*

The above mentioned procedures "restart monitor" and "convergence monitor" are of essential importance for the applicability and efficiency of the algorithm (GBIT). They have to be discussed in more detail.

Restart monitor

Theoretical considerations show, that the steplength parameter τ should be positive and not "too large". This is ensured by checking the so-called restart conditions:

$$\begin{array}{ll} a) & \tau < \tau_{min} \\ b) & \tau > \tau_{max}\,. \end{array} \tag{1.2.2}$$

In cases where either (1.2.2.a) or (1.2.2.b) is met, the iteration is proceeded at "restart", just as if the restriction $k > k_{max}$ has been activated. Numerical experiments suggest the choice:

$$\tau_{min} := 10^{-8}\,,\ \tau_{max} := 100\,. \tag{1.2.3}$$

Indeed, the behavior of (GBIT) is not too sensitive against a change of these thresholds. But another problem may occur. In critical examples the restart condition may be activated for $k = 0$, i.e. just at the start or at a restart of the iteration. To take this as an indicator for divergence and to stop the iteration would reduce the applicability of (GBIT) significantly. To proceed the iteration in such a case, a simple ad hoc device turns out to be quite helpful.

$$\begin{aligned}&\text{if}\ \ \tau < \tau_{min}\ \ \text{or}\ \ \tau > \tau_{max}\ \ \text{and}\ \ k = 0\ \ :\\ &\qquad \hat{\tau} := \tau\\ &\qquad \tau := 1\\ &\text{endif}\end{aligned}$$

After this resetting of τ the iteration is (re-)started as usual, with one exception. Instead of the usual correction update in (GBIT) a theoretically backed variant (c.f. [5])

$$\Delta := (1 - \tau + \hat{\tau})\Delta - \hat{\tau} z$$

is used (for $k = 0$).

Convergence monitor

The development of a robust convergence criterion turns out to be a rather crucial problem. As already mentioned in [5] the simple and quite natural estimation of the relative error

$$\tilde{\varepsilon}_{GB}^{est} := \frac{\|\Delta\|}{\|s\|} = \frac{\sqrt{\sigma}}{\|s\|} \tag{1.2.4}$$

may show an unpleasant behavior. Although the true error decreases nearly monotone, the associated error estimate $\tilde{\varepsilon}_{GB}^{est}$ shows an oscillatory decrease. As an example take Figure 1.1, which is just Figure 5.5 of [5]. Especially for the application within GIANT where only weak accuracy requirements are claimed for the most linear system solutions this fact may disturb the optimal matching of outer and inner iteration. To overcome this difficulty a modified error estimator has been developed which furthermore nicely fits into the requirements of GIANT, c.f. (1.1.19.a,b). First, to smooth the error estimates a so called hanging window is used. As soon as three accepted corrections Δ are available a smoothed value $\overline{\sigma}$ is computed via

$$\overline{\sigma} := \frac{1}{4}(\sigma_{new} + 2\sigma_{new-1} + \sigma_{new-2}) \tag{1.2.5}$$

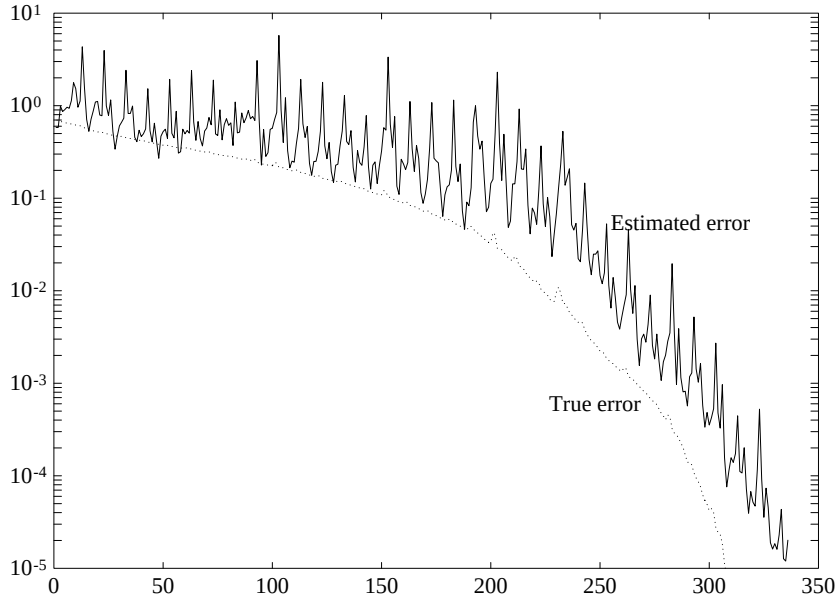


Figure 1.1 Estimated ($\tilde{\varepsilon}_{GB}^{est}$) and true error for GBIT

where

$$\sigma_l :=< \Delta_l, \Delta_l > \ .$$

Herein Δ_{new} denotes the newest available correction Δ and Δ_{new-1}, Δ_{new-2} are the latest previous (accepted) corrections. Second, the smoothed norm, i.e. $\sqrt{\overline{\sigma}}$ is enlarged by introducing a safety factor $\overline{\rho}$ ($\overline{\rho} = 4$ turns out to be a quite reasonable choice — see Section 3.2) :

$$\sqrt{\overline{\sigma}} \longrightarrow \overline{\rho}\sqrt{\overline{\sigma}} \tag{1.2.6}$$

Finally, the iteration is terminated only if the termination criterion (1.2.7) — see below — was met for two successive iterations. In order to realize the optimal adaptation to GIANT (c.f. (1.1.18), (1.1.19)) the final termination criteria for (GBIT) in GIANT read:

$$\begin{aligned} &a) \quad \overline{\rho}\sqrt{\overline{\sigma}} \leq \varepsilon^{req}\|s\| \\ &b) \quad \overline{\rho}\sqrt{\overline{\sigma}} \leq \frac{1}{4}\|\delta s\| \end{aligned} \tag{1.2.7}$$

According to the requirements in GIANT one has to distinguish two cases. In the first case, the Good Broyden iteration is stopped if (1.2.7.a) or (1.2.7.b) holds. In the second case the iteration is stopped only if (1.2.7.a) holds. In any case, the internally estimated error is computed by

$$\varepsilon_{GB}^{est} := \frac{\overline{\rho}\sqrt{\overline{\sigma}}}{\|s\|} \, . \tag{1.2.8}$$

Note that the scheme (GBIT) is realized in such a way that a continuation iteration to perform an a posteriori iteration (c.f. (1.1.21)) must not necessarily begin at "*Start/Restart*" due to missing flexibility of the implementation. Rather, the iteration is continued as if no exit has taken place.

In order to terminate a divergent iteration before an upper bound of *itmax* iterations is reached two heuristic divergence criteria have been implemented. One criterion checks for a certain cyclic behavior of the steplength parameter τ. The other one checks for too large corrections Δ. Finally, one should mention that the above described iterative linear solver is realized as a separate piece of numerical software named GBIT1. Thus, it may be used, independent of GIANT, as an iterative solver for large unsymmetric linear systems. But note that the development of some essential details has been made in view of the application within GIANT.

1.3 Details of Algorithmic Realization

In order to describe the underlying algorithms of GIANT some details of the algorithmic realization are worth mentioning. In practical applications the robustness and efficiency of the algorithms presented in Section 1.1 and 1.2 may e.g. strongly depend on the selected norm and scalar product, the reasonable choice of termination criteria and the chance to select special variants or modifications of the basic schemes.

1.3.1 Norm and scalar product

Generally speaking, to control the performance of the exact and inexact Newton algorithms (E) and (I) *smooth* norms (such as the Euclidean norm $\|\cdot\|_2$) are recommended. Non-smooth norms (such as the max-norm $\|\cdot\|_\infty$) may lead to some non-smooth performance, e.g. alternating between competing components of the iterates. Concerning the "Good Broyden" iteration the same general statement holds. Especially for the smoothing process in the error estimation of "Good Broyden" the choice of a smooth norm plays an important role. For the termination criterion of the Newton iteration, however, the max-norm may be used. Within the actual realization of GIANT the so called *root mean square* norm is used. This norm and the underlying scalar product is defined by

$$\begin{aligned} &a) \quad \|v\|_{rms} := \sqrt{< v, v >} \\ &b) \quad < u, v >:= \frac{1}{n} \sum_{i=1}^{n} u_i v_i \end{aligned} \qquad u, v \in \mathbb{R}^n \tag{1.3.1}$$

Note that for large scale systems this norm may significantly differ from the usual $\|\cdot\|_2$ or $\|\cdot\|_\infty$ norms, which are defined as follows:

$$\|v\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2} \qquad v \in \mathbb{R}^n \tag{1.3.2}$$

$$\|v\|_\infty := \max\{|v_i|\} \tag{1.3.3}$$

The choice of (1.3.1) is motivated by the following consideration. Assume that the problem (0.0.1) represents a discretized PDE. In order to check the quality of the discretization, one may solve the underlying continuous problem on grids with different levels of fineness, i.e. varying dimension n of the discretized problem. For adequate discretizations one may expect the same behavior of the inexact Newton scheme — (almost) independent of n. To achieve this aim, the algorithm must use quantities which are independent of the dimension of the problem — like $\|\cdot\|_{rms}$ or $\|\cdot\|_\infty$. Note that for special classes of applications, the use of (1.3.1) within GIANT is certainly not the best choice, but for an algorithm which is designed to solve general problems of the form (0.0.1) this choice turns out to be quite reasonable. Observe that the Newton- and Good Broyden schemes as presented here, are exclusively controlled by norms and scalar products to be evaluated in the space of the iterates (and not in the space of the residuals) — a necessary condition for an affine invariant method. Furthermore, in order to control the algorithmic performance ratios of norms are used, whereas the absolute value of a norm is just used for the termination criteria. These facts are of essential importance for the reliability of the algorithms.

1.3.2 Scaling and weighting

A proper *internal scaling* plays an important role for the efficiency and robustness of an algorithm. A desirable property of an algorithm is the so called *scaling invariance*. This means, e.g. regauging of some or all components of the vector of unknowns x (say, from Å to km) should not effect the algorithmic performance — although the problem formulation may change. In order to discuss this essential point consider a *scaling transformation* defined by:

$$\begin{array}{ll} a) & x \mapsto y := S^{-1}x \\ & \text{with a diagonal transformation matrix} \\ b) & S := \mathrm{diag}(s_1, \ldots, s_n) \end{array} \tag{1.3.4}$$

Insertion into the original problem (0.0.1) leads to a transformed problem

$$H(y) := F(Sy) = F(x) = 0 \tag{1.3.5}$$

where the associated solution y^* and Jacobian matrix H_y are given by

$$\begin{aligned} y^* &= S^{-1}x^* \\ H_y(y) &= F_x(x) \cdot S = J(x) \cdot S \end{aligned} \tag{1.3.6}$$

The problem (0.0.1) is said to be *covariant* under the scaling transformation (1.3.4) — a property which is shared by the ordinary Newton Method, as for $k = 0, 1, \ldots$

$$\begin{aligned} &a) \quad \Delta y_k = -H_y^{-1}(y_k)H(y_k) = -S^{-1}J^{-1}(x_k)F(x_k) = S^{-1}\Delta x_k \\ &b) \quad y_{k+1} = y_k + \Delta y_k = S^{-1}x_k + S^{-1}\Delta x_k = S^{-1}x_{k+1} \end{aligned} \tag{1.3.7}$$

holds. Note that the theoretical covariance property $y_k = S^{-1}x_k$ may be disturbed in real computations due to roundoff, except for special realizations like symbolic computations. As long as the Newton update is done via (1.3.7b) the simplified Newton correction is covariant also:

$$\overline{\Delta y}_{k+1} = -H_y^{-1}(y_k)H(y_{k+1}) = -S^{-1}J^{-1}(x_k)F(x_{k+1}) = S^{-1}\overline{\Delta x}_{k+1} \tag{1.3.7.c}$$

But, if norms (in the space of the iterates) enter into the algorithm, e.g. to perform a damping strategy or an error estimation, the covariance property of the algorithm is lost. As, in general

$$\|\Delta y_k\| = \|S^{-1}\Delta x_k\| \neq \|\Delta x_k\| \tag{1.3.8}$$

holds, the control and update procedures within the algorithms (E), (I) and (GBIT) will generate a different algorithmic performance if they are applied to problem (1.3.5) instead of (0.0.1). To overcome this difficulty one may internally replace the usual norm (e.g. (1.3.1)) by an associated *scaled* or *weighted norm*:

$$\|v\| \longrightarrow \|D^{-1}v\| \tag{1.3.9}$$

where D is a diagonal matrix to be chosen. Consider now the first Newton step of algorithm (E). Assume, a choice

$$D := \mathrm{diag}(x_1^0, \ldots, x_n^0) \quad , \qquad x^0 \text{ initial guess for } x^* \tag{1.3.10}$$

is possible ($x_i^0 \neq 0, i = 1, \ldots, n$). Inserting (1.3.10) into (1.3.9) yields for system (0.0.1):

$$\|\Delta x_0\| \longrightarrow \|D^{-1}\Delta x_0\| \tag{1.3.11}$$

Applied to the transformed system (1.3.6) one has

$$\overline{D} := \mathrm{diag}(y_1^0, \ldots, y_n^0) \quad , \qquad y^0 \text{ initial guess for } y^*$$

and due to

$$y^0 = S^{-1}x^0$$

one has

$$\overline{D}^{-1} = D^{-1}S$$

thus:

$$\|\Delta y_0\| \longrightarrow \|\overline{D}^{-1}\Delta y_0\| = \|D^{-1}SS^{-1}\Delta x_0\| = \|D^{-1}\Delta x_0\| \ . \tag{1.3.12}$$

In contrast to the case of unscaled norms (c.f. (1.3.8)) for the scaled norms (1.3.11) and (1.3.12) the norms of the first Newton corrections coincide. The same holds for the norms of the first simplified Newton correction. From this follows, that the first monotonicity test (1.1.5) will lead to the same algorithmic consequences independent of an eventual a priori transformation of type (1.3.4). Even all subsequent decisions of algorithm (E) will be invariant. In order to have invariance also for the algorithm (GBIT) one may interpret the replacement (1.3.9) as the introduction of a scaled scalar product:

$$< u, v > \longrightarrow < D^{-1}u, D^{-1}v > \ . \tag{1.3.13}$$

With that, the fixed choice of (1.3.9) for the internal scaling matrix D will yield invariance for all inexact Newton steps. But concerning the termination criteria of the Newton- and the Good Broyden scheme an *adaptive* choice is indispensable. Consider the natural stopping criterion for a Newton method in its unscaled form:

$$\begin{aligned} &err := \|\Delta x_k\| \\ &\text{stop, if} \quad err \leq tol \\ &tol \ : \text{prescribed (required) tolerance (accuracy)} \end{aligned} \tag{1.3.14}$$

In this unscaled form, *err* is a measure for the *absolute* error of the numerical solution x_k. Using a scaled norm

$$err := \|D_*^{-1}\Delta x_k\| \tag{1.3.15}$$

with

$$D^* := \mathrm{diag}(x_1^*, \ldots, x_n^*) \tag{1.3.16}$$

err is a measure of the *relative error* of x_k. Note that $\|D_*^{-1}(x^* - x_k)\|$ is the true relative error of x_k (still depending on the selected norm), whereas $\|D_*^{-1}\Delta x_k\|$ is just an estimate of it, but a quite reasonable one, as Newton's method converges quadratically near the solution x^*. Again, similar to (1.3.10), $x_i^* \neq 0, \ i = 1, \ldots, n$ is required, but in any case, x^* is usually not available. To avoid the difficulties coming from zero components and to connect the natural scaling matrices D_0, D_* ((1.3.10), (1.3.15)) within the course of the Newton iteration the following scaling strategy is applied. An internal *weighting vector* xw is used to define local scaling matrices D_k by

$$D_k := \mathrm{diag}(xw_1, \ldots, xw_n) \tag{1.3.17}$$

and xw may be locally defined by:

$$xw_i := \max\{|x_i^k|, \mathit{thresh}\} \tag{1.3.18}$$

where

$$\mathit{thresh} > 0 : \text{ threshold value for scaling.}$$

This scaling procedure yields reasonable values for the scaled norms used in the algorithms (E), (I) and (GBIT) and xw is also used in the Good Broyden scheme in order to realize scaled scalar products via (1.3.13). Note that the actual value of *thresh* determines a componentwise switch from a pure relative norm to a modified absolute norm. As long as $x_i^k > \mathit{thresh}$ holds, this component contributes with

$$\frac{\Delta x_i^k}{|x_i^k|}$$

to the norm, whereas for $x_i^k \leq \mathit{thresh}$ this component contributes with

$$\frac{\Delta x_i^k}{\mathit{thresh}}$$

to the total value of the norm. In order to allow a componentwise selection of *thresh* and to take into account that the damped Newton algorithm (I) uses information from two successive iterates the following extension of (1.3.17) and (1.3.18) is used in GIANT.

Input:

$xw^u :=$ user given weighting vector

Initial check:

a) if $(|xw_i^u| = 0)$

$$xw_i^u := \begin{cases} \mathit{rtol} & \text{if problem is highly nonlinear} \\ 1 & \text{if problem is mildly nonlinear} \\ & \text{(see Table 1.1 for problem type)} \end{cases} \tag{1.3.19}$$

Initial update:

b) $xw_i^0 := \max\{|xw_i^u|, |x_i^0|\}$

Iteration update:

c) $xw_i^k := \max\left\{|xw_i^u|, \frac{1}{2}(|x_i^{k-1}| + |x_i^k|)\right\}$

Thus, the scaling matrix, scalar product and norm (*weighted root mean*

square) used in GIANT are given by:

$$\begin{aligned} a) \quad & D_k := \mathrm{diag}(xw_1, \ldots, xw_n) \\ b) \quad & < u, v >:= \frac{1}{n} \sum_{i=1}^{n} \frac{u_i v_i}{xw_i^2} \\ c) \quad & \|v\| := \sqrt{< v, v >} \quad . \end{aligned} \tag{1.3.20}$$

Remarks:

(1) The final realization of scaling within (E) and (I) must be done carefully to achieve properly scaled norms for terms which include values from different iterates.

(2) In order to allow a scaling totally under user control, the updates (1.3.19.b,c) can be inhibited optionally.

(3) Within the realization of algorithm (E) (Code NLEQ1) the linear systems to be solved with a direct method are internally scaled systems:

$$J_k \Delta x_k = -F_k \longrightarrow (D_k^{-1} J_k D_k)(D_k^{-1} \Delta x_k) = -(D_k^{-1} F_k)$$

Thus, a user rescaling via (1.3.4) doesn't change the performance of the direct linear solver. To avoid an explicit usage of the Jacobian within GIANT this transformation is omitted. Note that the Good Broyden scheme is nevertheless invariant as a scaled scalar product is used within (GBIT). A realization of GIANT with other iterative methods may destroy the scaling invariance property of GIANT.

1.3.3 Termination criteria

First, recall that all norms in GIANT are evaluated in the space of the iterates x_k (and s_k^i respectively) and not in the space of the residuals. Concerning the termination criteria this means, that the associated error estimates yield a direct measure for the error of the associated solution vector. As scaled norms are used (*relative* error criterion) an interpretation in terms of correct leading decimal digits is possible. Assume, a termination criterion of the form

$$err_{rel} \approx \|\Delta x_k\| \leq tol \tag{1.3.21}$$

holds, where $\|\cdot\|$ is the weighted root mean square norm (1.3.20). Then, one has roughly

$$cld := -\log_{10}(tol)$$

correct decimal leading digits in the mantissa of each component x_i^k, independent of the actual exponent — except $x_i^k \ll xw_i^k$. In such a case the number of correct digits in the mantissa is approximately

$$cld := -(\log_{10}(tol) - (\log_{10}(|x_i^k|) - \log_{10}(xw_i^k))) \; .$$

In other words, the componentwise *absolute* error is, for both cases, approximately given by

$$err^i_{abs} \approx tol \cdot xw^k_i \ .$$

In contrast to this, an unscaled termination criterion in the space of the residuals

$$\|F(x_k)\| \leq tol$$

neither controls the error in the computed solution nor shows any invariance property. A simple reformulation of the original problem (0.0.1) of the form

$$F \longrightarrow \hat{S}F =: \hat{F} \ , \quad \hat{S} = \mathrm{diag}(tol^{-1}, \ldots, tol^{-1})$$

will lead to

$$\|\hat{F}(x_k)\| \approx 1$$

whereas a stopping criterion like (1.3.21) is not affected. In order to realize invariance against such a rescaling one may again use a scaled check, e.g.

$$\|\hat{D}^{-1}F\| \leq tol$$

where

$$\hat{D} := \mathrm{diag}(F_1(x_0), \ldots, F_n(x_0)) \ .$$

However, there is some arbitrariness in the choice of $\hat{D}$ and it is not clear how to develop an adaptive selection of further scaling matrices $\hat{D}_k$. In any case, the disadvantage of checking in the wrong space is still remaining.

Remark: Assume that the problem (0.0.1) is well scaled, i.e. unscaled norms yield meaningful numbers. If in such a situation

$$\|\Delta x_k\| \leq tol \quad \text{with} \quad \|F(x_k)\| \text{ "large"}$$

holds, the underlying problem is said to be ill-conditioned. That means, $\|F(x)\|$ "large" may occur even for $x := float(x^*)$ — just because x^* can't be represented exactly due to the finite length of the mantissa. For a badly scaled problem a check for the condition of the problem must use scaled norms, i.e.

$$\|D^{-1}\Delta x_k\| \leq tol \ \text{ with } \|\hat{D}^{-1}F(x_k)\| \text{ "large"} \tag{1.3.22}$$

indicates an ill-conditioned problem, provided that $D, \hat{D}$ are properly chosen. Note that within GIANT an optional printout of $\|F(x_k)\|$ is possible — but in unscaled form. Thus, situations like (1.3.22) may be pretended to users, but due to $\hat{D} = I$ the problem is well-conditioned but ill-scaled.

The actual implemented termination criterion for the Newton iteration in GIANT is not directly the one presented in the basic schemes (E) and (I). Analogous to the exact case, instead of using the inexact ordinary Newton

correction s_k, the criterion is realized in terms of the inexact simplified Newton correction $\overline{s}_{k+1}$:

$$\|\overline{s}_{k+1}\| \leq tol\,. \tag{1.3.23}$$

In order to overcome pathological situations, the above criterion (1.3.23) is only accepted for termination if additionally the condition

$$\|s_k\| \leq \sqrt{tol \cdot 10} \tag{1.3.24}$$

holds. Note that for the Newton iteration no heuristic divergence criterion is needed. Clearly, a maximum number of iterations may be prescribed, but usually an internal fail exit condition

$$\lambda_k^{(j)} < \lambda_{min} \tag{1.3.25}$$

will stop a divergent iteration before.

Concerning the termination criteria of the Good Broyden iteration the detailed description in Section 1.2 just needs some further comments. The general advantage of checking in the space of the iterates carries over. The problem of getting good error estimates in a linear convergent scheme is a general one and is attacked by introducing the safety factor $\overline{\rho}$. The distinction of this factor $\overline{\rho}$ and the safety factor ρ within the accuracy matching condition (c.f.(1.1.13.d)) is worth to discuss. An increase of $\overline{\rho}$ generates more accurate solutions s_k,$\overline{s}_{k+1}$ of the inner iteration. But the error estimates yielded from GBIT1 do not change. Furthermore, all linear solutions are affected by changing $\overline{\rho}$ - as long as $\overline{\rho}$ is modified only once at the beginning. In contrast to this, a decrease of ρ in (1.1.13) generates not only more stringent solution requirements for the linear system solution accompanied by (hopefully) more accurate solutions s_k,$\overline{s}_{k+1}$ but also smaller error estimates. Besides this, due to (1.1.13.a,b) and (1.1.13.c) respectively the influence of ρ depends on the estimate $[h_k^{(0)}]$, i.e. as long as $\lambda < 1$ holds both accuracy requirements ε_k^{req} and $\overline{\varepsilon}_{k+1}^{req}$ are influenced in the same way, whereas for $\lambda = 1$ the effect on the computation of ε_k^{req} changes. Roughly speaking, a decrease of ρ will force the performance of the inexact Newton scheme towards the performance of the exact one (trusting in error estimates of the linear solver) while an increase of $\overline{\rho}$ will close a gap between true and estimated error of the iterative solver.

1.3.4 Damping strategy

In order to start the computation an initial damping factor λ_0 is needed. For mildly nonlinear problems, where an undamped Newton scheme converges, the choice $\lambda_0 = 1$ is optimal. Even for highly nonlinear problems $\lambda_0 = 1$ may be used, as the a posteriori damping strategy will correct a too optimistic choice of λ_0. The additional costs are usually low, but note that the a priori

factor λ_0 enters via (1.1.7) into the a posteriori estimate. Furthermore, in critical applications the necessary evaluation of $F(x_0 + \lambda_0 \Delta x_0)$ may cause problems for $\lambda_0 = 1$ (e.g. overflow, invalid operand) as the first undamped trial iterate can be out of bounds. So, a "small" initial guess λ_0 is recommended. As pointed out above, a further parameter of the damping strategy, the minimal permitted damping factor λ_{min}, has to be selected. Besides this, for extremely sensitive problems a recently developed modification (see [3]) of the damping strategy may be selected. Within this *restricted* damping strategy the estimates $[h_k]$ are replaced by $[h_k]/2$. Note that this restricted strategy shows some nice theoretical properties. In order to simplify the choice of λ_0 , λ_{min} and the type of the strategy the following specification of problem classes and associated internal selection of parameters is made.

problem class	λ_0	λ_{min}	λ-strategy
linear	1	—	—
mildly nonlinear	1	10^{-4}	standard
highly nonlinear	10^{-2}	10^{-4}	standard
extremely nonlinear	10^{-4}	10^{-4}	restricted

Table 1.1 Definition of problem classes

If no user classification of the problem is available the problem is assumed to be "highly nonlinear". Observe that for the classification of a problem not only the nonlinear function F is relevant but also the quality of x_0. The case "linear" is realized in GIANT to allow the solution of a linearized problem with the same piece of software as for the original nonlinear problem. Because a specification "mildly nonlinear" at least forces the computation of the first simplified Newton correction, a problem specification "linear" includes the computation of the first ordinary Newton correction with the update $x_1 := x_0 + \lambda_0 \Delta x_0$ only. Thus, the computational overhead for the solution of a linear problem with GIANT is small. Note that in such a case the required tolerance *tol* is used as required accuracy for the iterative linear solver ($\varepsilon_0^{req} := tol$).

2. Implementation

2.1 Overview

The algorithms presented in Chapter 1 are realized as two pieces of numerical software, the codes GIANT and GBIT1 respectively. They belong to the numerical software library CodeLib of the Konrad Zuse Zentrum Berlin, thus they are available for interested users. Both codes are written in ANSI standard FORTRAN 77 (double precision) and are realized as subroutines which must be called from a user written driver program. All communication between the codes and the calling program is done via the arguments of the calling sequence of the subroutines, except for some (optional) data- and monitor output which is written to special FORTRAN units. Besides this interface to the calling program GIANT has three further interfaces. There is a call from GIANT to a subroutine named FCN which has to evaluate the nonlinear function F of (0.0.1) and a call to a subroutine named JAC which may evaluate the Jacobian F_x of the system. Finally, GIANT performs a call to a subroutine named ITSOL which has to solve a linear system — at least approximately. Note that no information from JAC is used within GIANT but only passed to subroutine ITSOL. In order to have the software as flexible as possible all three functions are input arguments of the calling sequence of GIANT and are, in principle, not part of the GIANT package. But the adaptation of an iterative linear solver for an efficient use by GIANT may be quite cumbersome, thus the code GBIT1I (which is just the internal core subroutine of the GBIT1 package) has been added to the basic GIANT package. This extended package represents the algorithms and their optimal telescoping as presented in Chapter 1. An internal interface from GIANT to GBIT1I is realized in addition to the above mentioned standard interface to an user supplied iterative solver. Thus, by setting an associated option one may easily switch from the standard iterative linear solver to an user supplied one — and vice versa. In order to have also sufficient flexibility for the iterative linear solver software the usual functional units of such methods, namely a subroutine named PRECON to perform the preconditioning and a subroutine named MULJAC to perform a matrix times vector multiplication are external subroutines for the code GBIT1 (i.e. GBIT1I). Consequently, these subroutines are additional input arguments for GIANT. The subroutines are not used within the basic GIANT package but just passed to both interfaces for the iterative linear solver. The drawback of this modularization is that an "easy use" of GIANT/GBIT1I is prevented. But this structure reflects the fact that the efficiency and robustness of GIANT strongly depends on the iterative linear solver while the efficiency and robustness of this solver may strongly depend on the preconditioner. Thus, the most efficient way to

adapt GIANT to a special problem class is to supply well suited ITSOL and PRECON/MULJAC routines respectively.

In order to facilitate the application, the codes GIANT and GBIT1 may be used with a supplementary package, named EASYPACK. This package contains some software from the Sparse Linear Algebra Package (SLAP) of Seager/Greenbaum [11, 7] mainly some preconditioners and a special matrix vector multiplication subroutine. Additionally, EASYPACK supplies appropriate interface subroutines to support the use of this software in combination with GIANT/GBIT1. In order to have no dependency on this supplementary package within GIANT the interface subroutines of EASYPACK are implemented as a frame around the extended GIANT package. Thus, the combination GIANT+GBIT1+EASYPACK reduces the flexibility but increases easy applicability of the basic code GIANT. One essential consequence is that now the storage mode of the user supplied Jacobian matrix is fixed to SLAP Triad or SLAP Column format. Furthermore, the Jacobian must be explicitly evaluated and stored within a subroutine with the prescribed name JAC1.

Alternatively, writing own subroutines JAC, PRECON and MULJAC enables the user to choose his own storage format as the workspace which may contain the Jacobian is passed unchanged from the interface of JAC to the interface of PRECON and MULJAC. Even a realization without an explicit storage of a matrix is possible. Instead of evaluating the Jacobian in JAC one may e.g. realize problem dependent subroutines PRECON and MULJAC.

Summing up, one may distinguish three levels of GIANT software:

Level 0 : basic GIANT code	— no specification of an (iterative) linear solver
Level 1 : extended GIANT code	— software to perform the Good Broyden iteration is internally adapted
Level 2 : supplemented GIANT code	— extended GIANT within an easy to use frame

Besides this, in order to carry out the numerical experiments presented in Chapter 3 a level 3 has been implemented. The iterative solver GMRES is used as a user supplied iterative linear solver within GIANT level 2. To a certain extend, this combination realizes the simplified adaptation given by (1.1.20) and (1.1.21) — see Section 1.1.4. Figures 2.1 and 2.2 give a schematic representation of the different GIANT levels. Detailed diagrams, describing also the internal modularization, can be found in appendix A .
The following sections give a short introduction in how to use the GIANT package. A detailed documentation including all technical details is part of the code and is not reproduced here. Rather, the relation of the software

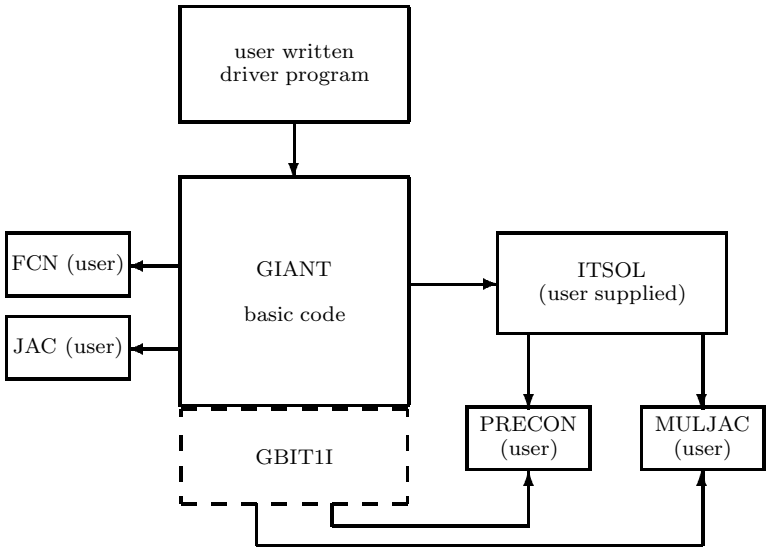


Figure 2.1 GIANT level 0 and level 1

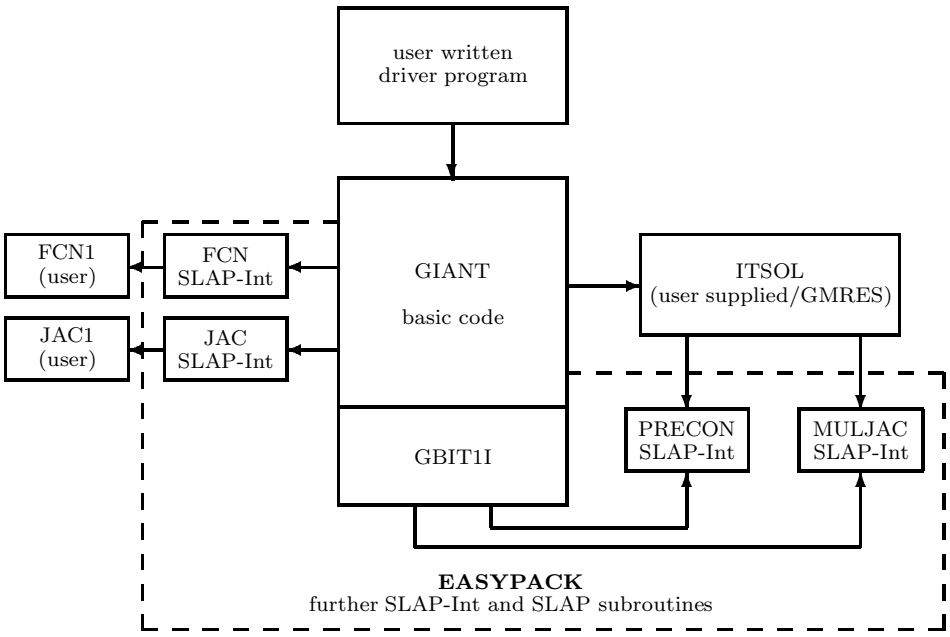


Figure 2.2 GIANT level 2 and level 3

to the underlying algorithms of Chapter 1 is pointed out and some general comments are made.

2.2 Interfaces

The actual implemented user interface subroutine (to the calling program) reads:

```
SUBROUTINE GIANT( N, FCN, JAC, X, XSCAL, RTOL, IOPT, IERR,
                  LIWK, IWK, LRWK, RWK, LIWKU, IWKU,
                  LRWKU, RWKU, MULJAC, PRECON, ITSOL )
```

Within this calling sequence one may distinguish three groups of arguments: one group for the problem definition and the associated solution requirements, another which refers to the algorithm and finally one which is used to define the linear system solution. The arguments of the first group are:

N integer, input :
dimension of the nonlinear system to be solved

FCN external subroutine, input :
evaluation of the nonlinear function $F(x)$

JAC external subroutine, input :
evaluation of the Jacobian matrix $J = F_x(x)$

X real array(N), in-out :
in : initial guess x^0
out : actual approximation x^k of the solution x^*

XSCAL real array(N), in-out :
in : initial user scaling vector xw^u
out : actual internal scaling vector xw^k

RTOL real, in-out :
required (in) / achieved (out) relative accuracy of x^k

In order to make a proper initial setting for these arguments the strong internal coupling of the arguments XSCAL, RTOL, X and even N should be observed. Recall that the internal scaling procedure (1.3.19) connects X-input, XSCAL and RTOL. Due to (1.3.20) the actual scaling vector xw^k influences the whole algorithmic performance, especially all termination criteria (c.f. (1.3.23), (1.3.24), (1.2.7)). Furthermore, an interpretation of the achieved accuracy can only be made in connection with xw^k and x^k and with regard of the internally used norm (c.f. (1.3.20)) — where N enters. A zero

initiation of XSCAL is possible but may lead to an unpleasant behavior of the algorithm — especially together with $x^0 = 0$.

The second group of arguments is:

IOPT integer array(50), in-out :
selection of options

IERR integer, output :
error flag (IERR > 0 indicates failure)

LIWK integer, input :
declared length of the integer work array IWK (LIWK ≥ 50)

IWK integer array(LIWK) - in-out :
first 50 elements: selection of special internal parameters (e.g. limits on iteration counts) (in), statistics of the algorithmic performance (out)
elements up from the 51 th : integer workspace for an user supplied iterative linear solver — if needed

LRWK integer, input :
declared length of the real array RWK
(LRWK $\geq$ 50+7N + 17+2KMAX+(4+KMAX)N), where KMAX is an integer parameter of GBIT1, namely the storage restriction parameter k_{max} of the algorithm (GBIT) c.f. Section 1.2

RWK real array(LRWK), in-out :
first 50 elements: selection of special internal parameters of GIANT (e.g. starting and minimum damping factor) and possibly of the linear solver (in), actual values of some internal parameters (out)
elements up from the 51 th : real workspace for GIANT and the linear solver.

Besides providing workspace these arguments can be used to control and monitor the performance of GIANT. This can be done by assigning special values to (a part of) the first fifty elements of IOPT, IWK and RWK. Note that a zero initiation forces an internal assignment with the default values.

Furthermore, some of these elements will hold helpful information after return — e.g. the minimum needed length of IWK and RWK to solve the given nonlinear problem (LIWK_{min} = IWK(18) , LRWK_{min} = IWK(19)). Observe that the internal default values are chosen according to the suggestions made in Chapter 1. The features of GIANT which can be influenced by this type of option selection are described in Section 2.3 below.

The last group of arguments concerns the iterative linear solution. As GBIT1I is part of the actual GIANT package the external subroutine ITSOL may be a dummy argument.

LIWKU integer, input :
declared length of the integer user workspace IWKU

IWKU integer array(LIWKU), in-out :
integer workspace array, reserved for user purposes

LRWKU integer, input :
declared length of the real user workspace RWKU

RWKU real array(LRWKU), in-out :
real workspace array, reserved for user purposes

MULJAC external subroutine, input :
subroutine which calculates $w = J \cdot v$ (J :=Jacobian matrix)

PRECON external subroutine, input :
subroutine for preconditioning $z = Hy$ (H :=preconditioning matrix)

ITSOL external subroutine, input :
iterative linear solver subroutine (solves $J\Delta x = F$)

The above mentioned user workspace arrays RWKU and IWKU may serve to store the Jacobian J and the preconditioning matrix H. GIANT and GBIT1 do not alter or use these arrays. They are just passed from and to the interfaces of FCN, JAC, PRECON and MULJAC.

From GIANT the following requirements are made:

FCN must provide the function value $F(x)$ for that vector x which is input to FCN. MULJAC must provide the result of the matrix vector multiplication $w = J \cdot v$ for a given vector v. J must be the Jacobian evaluated at that specific point x which was input to JAC at the preceding call of JAC. PRECON must solve the preconditioning system $H^{-1}z = y$ (or perform $z = Hy$ respectively) for a given vector y. Herein H should be a "good approximation" of J^{-1}, again evaluated at that point x which was input to JAC at it's preceding call. The interfaces of these routines are:

SUBROUTINE FCN(N,X,F,RWKU,IWKU,NFCN,IFAIL)

X	real array(N), input	actual (trial) iterate $x_k^{(i)}$
F	real array(N), output	function values $F(x_k^{(i)})$
NFCN	integer, input	counter
IFAIL	integer, in-out	error flag, =0 on input on output: if < 0, GIANT terminates
N, RWKU, IWKU	see above	

SUBROUTINE JAC(FCN,N,X,XSCAL,F,RWKU,IWKU,NJAC,IFAIL)

XSCAL	real array(N), input	actual scaling vector xw^k
F	real array(N), input	function values $F(x_k^{(i)})$
NJAC	integer, input	counter
FCN, N, X,RWKU, IWKU, IFAIL		see above

SUBROUTINE MULJAC(N,V,W,RWKU,IWKU)

V	real array(N), input	the vector to be multiplied with the Jacobian
W	real array(N), output	resulting vector $w = J \cdot v$
N, RWKU, IWKU		see above

SUBROUTINE PRECON(N,Y,Z,RWKU,IWKU)

Y	real array(N), input	the vector to be multiplied with the preconditioning matrix H
Z	real array(N), output	resulting vector $z = H \cdot y$
N, RWKU, IWKU		see above

In order to facilitate the correct and efficient adaptation of an user provided iterative solver the ITSOL interface needs some general comments. The ITSOL routine is used to solve the linear systems for the ordinary and simplified Newton corrections s_k and $\overline{s}_k$ respectively, i.e. at least two linear systems (of type $As = b$) have to be solved per Newton step — with the same matrix but different right hand sides. The right hand side b passed to ITSOL is just the output vector F from the last call to FCN. The matrix A of the linear system must be the Jacobian of $F(x)$ at that point x which was input to JAC at the last call to JAC. Recall that the sequence of the linear systems to be solved reads:

$$
\begin{aligned}
& J(x_0)s_0 = F(x_0) \\
& J(x_0)\overline{s}_1^i = F(x_1^i)\ ,\ i = 0 \\
& \text{ocassionally:}\quad J(x_0)\overline{s}_1^i = F(x_1^i)\ ,\ i = 1, 2, \ldots \\
& (\ x_1 := x_1^i\ ) \\
& J(x_1)s_1 = F(x_1) \\
& J(x_1)\overline{s}_2^i = F(x_2^i)\ ,\ i = 0 \\
& \text{ocassionally:}\quad J(x_1)\overline{s}_2^i = F(x_2^i)\ ,\ i = 1, 2, \ldots \\
& (\ x_2 := x_2^i\ ) \\
& J(x_2)s_2 = F(x_2) \\
& \qquad \vdots
\end{aligned}
$$

Observe that the vector X contains on input to ITSOL already a quite good initial guess (except for the very first call) for the required solutions s_k and $\overline{s}_{k+1}$ respectively. Due to the accuracy matching strategy, the solution of a specific linear system must be continued sometimes, in order to get a more accurate solution. The in-out argument XDEL may facilitate the use of iterative solvers, where the iteration is performed directly in the difference $\delta s := s - s^0$ (to avoid cancellation of leading digits). In any case, the values of XDEL are directly used in GIANT for the computations of $[h_k^{(i)}],\ i = 0, 1, \ldots$. Finally, note that the linear systems to be solved by ITSOL read $Js = +F$, i.e. the sign conversion necessary to get the associated Newton corrections is done within GIANT.

```
SUBROUTINE ITSOL( N, B, X, XDEL, XSCAL, MULJAC, PRECON,
                  TOL, ITMAX, ITER, ERR, IERR, IOPT,
                  LRWK, RWK, NRW, LIWK, IWK, NIW,
                  LRWKU, RWKU, LIWKU, IWKU )
```

B real array(N), input
right hand side of linear system

X real array(N), in-out
on input: initial estimate of the solution
(if continuation call: output from previous ITSOL call) on output: final approximation of the solution

XDEL real array(N), in-out
on input: (only, if continuation call) output from previous ITSOL call
on output: the difference X(out) − X(in)

TOL real, input
prescribed tolerance ε^{req}, c.f. (1.1.19)

ITMAX integer, input
maximum number of iterations allowed

ITER integer, output
number of iterations done

ERR real, output
error estimate ε^{est} for the approximate solution X(out), c.f. (1.1.8)

IERR integer, output
error indicator (=0 means no error)

IOPT integer array(50), in-out
options vector for ITSOL (not the same as for GIANT !)
on output, element IOPT(50) should contain information about the activated stopping criterion

LRWKL integer, input
length of real workspace supplied for ITSOL

RWK real array(LRWKL), in-out
real workspace for ITSOL

NRW integer, output
amount of real workspace actually used

LIWKL integer, input
length of integer workspace supplied for ITSOL

IWK integer array(LIWKL), in-out
integer workspace for ITSOL

NIW integer, output
amount of integer workspace actually used

N, XSCAL, MULJAC, PRECON, LRWKU, RWKU, LIWKU, IWKU
see above

2.3 Options

Though the underlying algorithm (I) of GIANT is self-adaptive in the sense that the damping factor λ is automatically adapted to the problem at hand, there are still some algorithmic parameters and variants open for an adjustment by the user. In general, the influence on the overall performance is not dramatic but in special applications a skillful matching may increase efficiency and robustness drastically. Concerning the internal options of GBIT1 the same statement holds. Besides these algorithmic options some other useful options, e.g. output generation, are available for the user. As pointed out in the preceding Section the adaptation can be easily performed by assigning special values to specific positions of the arrays IOPT, IWK and RWK. As far as possible, the input is checked for correctness.

Linear solver selection

The basic decision which iterative linear solver has to be used by GIANT can be made by setting IOPT(8). The user supplied iterative solver ITSOL is used if IOPT(8)=9 holds. An assignment of 0 or 1 to IOPT(8) causes GIANT to use GBIT1.

Internal scaling

The internal update of the user scaling (weighting) vector XSCAL according to (1.3.19) can be switched off by setting IOPT(9)=1. Note that the initial check (1.3.19a) is done in any case. The problem classification which may influence the performance of (1.3.19a) is done via IOPT(31) — see below.

Output generation

The amount of output produced by GIANT is internally controlled by the actual values of some output flags and directed to associated FORTRAN units. In order to monitor the algorithmic performance of the code, the internal flag MPRMON can be modified by the user by setting the associated element of the IOPT array (MPRMON corresponds to IOPT(13) and the associated output unit LUMON to IOPT(14)). An user assignment IOPT(13)=0 produces no monitor print output, whereas a setting IOPT(13)=3 will generate a detailed iteration monitor, i.e. the unscaled norm $\|F(x_k)\|$ (where F denotes the problem function introduced in (0.0.1) and x_k the Newton iterate), the scaled norms of the actual Newton corrections $\|s_k\|, \|\overline{s}_{k+1}\|$ and the actual damping factor λ_k are written to FORTRAN unit IOPT(14). Similar flag/unit pairs are available for error/warning printout, data output (actual iterate vector x_k) and output from the iterative solver GBIT1 — c.f. Table 2.1.

Option	Selection	Range	Default	Unit
error/warning messages	IOPT(11)	0-3	0	IOPT(12)
Newton iteration monitor	IOPT(13)	0-6	0	IOPT(14)
Solution output	IOPT(15)	0-2	0	IOPT(16)
Good Broyden monitor	IOPT(17)	≤ 1	0	IOPT(18)
Time monitor	IOPT(19)	0-1	0	IOPT(20)

Table 2.1 Options for output generation

Modification of Newton iteration

The damping strategy of the inexact Newton scheme can be partially modified by the user. First, a general problem classification can be made by the user by setting the parameter NONLIN (c.f. Table 1.1) to the desired value (1 – linear problem, 2 – mildly nonlinear, 3 – highly nonlinear, 4 – extremely nonlinear). But besides this, the values of some special internal parameters can be set separately. The initial and minimum allowed damping factors λ_0, λ_{min} may be set individually, whereas the general type of damping strategy (standard or restricted) still depends on NONLIN. Note that due to (1.3.25) λ_{min} is internally used as an emergency stopping criterion (in addition to the selectable maximum iteration limit ITMAX) and the choice of λ_0 even influences the iterative linear solution as the initial value for the simplified Newton correction computation $\overline{s}_1^0$ depends on λ_0. The essential safety factor ρ of the accuracy matching strategy (1.1.13) is under user control also. An assignment of a smaller value than the internal default value $\rho = 1/6$ may help in critical applications whereas the theoretical upper bound of $\rho_{max} = 1/6$ should be observed even for noncritical problems. An overview

on the options related to the inexact Newton scheme is given in Table 2.2.

Option	Selection	Range	Default
problem classification	IOPT(31)	0-4	3
Newton iteration limit	IWK(31)	≥ 1	50
initial damping factor	RWK(21)	≤ 1	see Table 1.1
minimum damping factor	RWK(22)	≤ 1	see Table 1.1
accuracy matching factor	RWK(32)	≤ 0.167	0.167

Table 2.2 Options for the inexact Newton iteration

Modification of Good Broyden iteration

The actual implementation of GIANT includes GBIT1I as the standard iterative linear solver. Thus, some of the options of the GIANT interface concern this inner iteration scheme. In view of the storage requirements of GIANT (level 1) the parameter KMAX of GBIT1I is certainly the most important option. Note that the storage restriction parameter k_{max} of the scheme (GBIT) of Section 1.2 is just KMAX. The restart monitor of (GBIT) can be influenced by choosing values for the minimum (τ_{min}) and maximum (τ_{max}) allowed steplength parameter τ — c.f. (1.2.2). Furthermore, the safety factor $\overline{\rho}$ of (1.2.6) can be modified. On this occasion one may specify different factors $\overline{\rho}_{ord}$ and $\overline{\rho}_{sim}$. They replace $\overline{\rho}$ in (1.2.7), (1.2.8) depending on whether (GBIT) iterates for an ordinary or simplified Newton correction computation.

Option	Selection	Range	Default
workspace restriction	IOPT(41)	≥ -2	9
iteration limit	IWK(41)	≥ 2	1000
minimum steplength	RWK(43)	$0 < \tau_{min} < 1$	10^{-8}
maximum steplength	RWK(44)	> 1	100.
safety factor $\overline{\rho}_{ord}$	RWK(41)	≥ 1	4
safety factor $\overline{\rho}_{sim}$	RWK(42)	≥ 1	4

Table 2.3 Options for the Good Broyden iteration

Workspace management

The essential part of workspace required by GIANT (level 0) consists of some arrays of dimension N. Within the actual implementation a real workspace RWK of roughly $7N$ elements is needed to perform the inexact Newton algorithm (I) including the adaptive internal scaling procedure (1.3.19). All further workspace is passed to the iterative linear solver interfaces. To be

precise, the elements 1 up to $\mathrm{LRWK}_N = 50 + 7N$ of RWK and the elements 1 up to $\mathrm{LIWK}_N = 50$ of IWK are checked and divided up within GIANT (level 0). The remaining parts (RWK($\mathrm{LRWK}_N + 1$,...,LRWK) and IWK($\mathrm{LIWK}_N + 1$,...,LIWK)) are supplied as RWK and IWK to the user supplied ITSOL subroutine.

If GIANT is used in connection with GBIT1I, these remaining parts are controlled and divided up within GIANT (level 1) also. Depending on the storage restriction parameter KMAX of algorithm (GBIT) the remaining part of RWK must be at least of length $\mathrm{LRWK}_G = 17 + 2\mathrm{KMAX} + (4 + \mathrm{KMAX})N$ whereas no integer workspace is needed for GBIT1I. Recall that KMAX can be set by the user via IOPT(41) (default value: KMAX=9). In order to increase the flexibility of GIANT a special workspace distribution device is available. A user setting of $\mathrm{KMAX} = -1$ forces an internal a priori computation of the maximum possible value of KMAX according to the declared size LRWK of RWK. If $\mathrm{KMAX} \geq 2$ holds, this value is used for the subsequent computations and returned as IOPT(41).

One step mode

Within special applications it may be useful to perform the Newton iteration step by step, i.e. the program control is passed back to the calling program after one Newton step. This mode can be selected by setting the mode flag IOPT(2)=1. In order to distinguish the first call — certain initializations and checks are made by GIANT — and successive calls an associated flag IOPT(1) is used. IOPT(1)=0 indicates that this is the first call whereas IOPT(1)=1 indicates a successive call. Note that GIANT internally sets IOPT(1)=1 on return if it was called in stepwise mode. Furthermore, the error flag IERR is set to -1 as long as the stopping criterion (1.3.23), (1.3.24) does not yet hold. As an example for an application of this option, just this internal stopping criterion can be substituted by a user defined one as the continuation of the iteration is under user control.

Matrix free method

In order to realize GIANT as a matrix free method one may replace the usual explicit multiplication of the Jacobian J with a given vector v (done within MULJAC) by directly approximating the result with a difference quotient of the form

$$J(x) \cdot v = \frac{\partial F(x)}{\partial x} \cdot v \approx \frac{F(x + \sigma v) - F(x)}{\sigma} \,. \tag{2.3.1}$$

The application of this widely used trick avoids the storage of the matrix J but inhibits the use of some standard preconditioners — like incomplete factorization techniques. Beyond that, the increment σ must be chosen carefully as the approximation error of (2.3.1) may disturb both iteration schemes — the nonlinear Newton iteration as well as the linear solvers iteration e.g. the Good Broyden iteration. As long as the approximation error is small com-

pared to the required accuracy for the iterative linear solution (c.f. 1.1.13) the inexact Newton scheme of Section 1.1 is still in its theoretical frame but beyond that scope, at least the superlinear convergence is destroyed. Concerning the Good Broyden iteration, an approximation error in the order of the required accuracy may prevent convergence. The implementation of the matrix free trick is left to the user — i.e. is not available as an option — but a numerical experiment illustrating this technique is presented in Section 3.2.4. It should be mentioned that the function values $F(x)$ in (2.3.1) have been already computed within the course of the Newton iteration and may be stored in RWKU for passing them to MULJAC. But when solving the linear system for the simplified Newton correction (1.1.3) the preceding call to FCN has an inappropriate argument. Therefore the function values $F(x)$ which are input to JAC must be used in order to approximate the correct Jacobian $J(x)$ for $x = x_k$.

Time monitor

In order to get more detailed information concerning the performance of the GIANT package together with the user supplied problem subroutines, a time monitor package which is designed for time measurements of multiple program parts is included within the GIANT (level 1) package and may easily be used when running GIANT together with GBIT1I. In this case, before using the monitor package for time measurements, because of the machine dependency of that stuff, the user has to adapt the subroutine SECOND in such a way that on output the only argument of this routine contains a "time stamp", measured in seconds. As distributed, SECOND is a dummy routine which always returns zero to this argument. If the time monitor is run together with a user supplied linear solver subroutine ITSOL the user additionally must include time monitor calls into the linear solvers code in order to the obtain correct measurements about the usage of ITSOL, PRECON and MULJAC. Refer to the GIANT code documentation (description of ITSOL) for technical details.

The monitor may be turned on by setting IOPT(19)=1. Its output will be written to the FORTRAN unit IOPT(20). The printout includes the total time used for solving the nonlinear problem and detailed statistics about the following listed program sections: linear solver (e.g. GBIT1I) excluding PRECON and MULJAC, subroutines PRECON, MULJAC, FCN, JAC (possibly including the preconditioner setup) and GIANT monitor- and data output. Statistics about the remaining code pieces are summarized as the item "GIANT". For each Section and for the remaining code the following statistics are printed out before the GIANT subroutine exits: number of calls of the code section, time used for all calls, average time used for one call, percent of time related to the total time used and related to the summarized time of the specific parts measured.

Of course, the time monitor package distributed within GIANT may be used as a separate piece of software within any program to print out statistics as described above for program sections which may be arbitrary defined by the user. These sections may even be nested — like GBIT1I with PRECON and MULJAC. The usage of the monitor package is described in detail within its code documentation.

Machine dependencies

One aspect of the modular structure of the GIANT package is the fact that the machine dependencies of the whole program are concentrated in two FORTRAN modules: the timing subroutine SECOND and the machine constants double precision function D1MACH. The routine SECOND is only called when the time monitor is switched on and therefore described within the context of this monitor. The function D1MACH returns, dependent on its input argument, typical machine dependent real constants, such as the machine precision, the smallest positive and the largest real number. It consists of comments which contain sets of FORTRAN data-statements with the appropriate real constants for several different machine types. Before using the GIANT code on a machine different from SUN, the user should comment out the data-statements related to these machines and uncomment the set of data-statements which is appropriate for the target machine.

Efficiency

Of course, the coding of GIANT/GBIT1 allows the automatic vectorization of all essential loops. Indeed, all these loops are still in-line code. The GIANT/GBIT1 version described in this paper, and applied for the numerical experiments of Chapter 3, doesn't use any routine from the BLAS-package. Mainly, because some of the innermost loops have a too complex structure for the standard BLAS interface. Consequently, these loops can't be replaced by just one call to a BLAS routine, and splitting up such a loop may slow down the performance speed. On the other hand, the gain by using machine adapted BLAS routines instead of in-line code seems to be extremely small — if a "good" compiler is available. As the overall computing time to solve a nonlinear problem is usually dominated by the time spent in FCN, JAC, MULJAC, PRECON and ITSOL this question is anyway of minor interest as for GIANT.

For GBIT1 another questions turns out to be more essential. Recall the discussion on scaling and weighting of Section 1.3, and the fact that the scalar product $< \cdot, \cdot >$ used in algorithm GBIT1 is given by (1.3.20). Therefore, the evaluation of a scalar product is quite costly. Within GBIT1 a lot of scalar products have to be evaluated — even within the innermost update loop. In order to reduce the additional costs resulting from the internal scaling the algorithm GBIT1 is realized in such a way that the iteration is done directly

in scaled variables, $\overline{\delta}_{siter}$, i.e.

$$\delta s_{iter} \longrightarrow D_k^{-1} \delta s_{iter} =: \overline{\delta s}_{iter} \ . \tag{2.3.2}$$

As a consequence, the vectors which are passed to the routines PRECON and MULJAC have to be descaled before the call and rescaled afterwards. But as this descaling/rescaling update occurs less frequent than the scalar product evaluation, the performance speed of GBIT1 is improved by about a factor 2.

3. Numerical Experiments

In this Chapter some computations illustrating the behavior of the algorithms described in Chapter 1 are presented. The discussion will focus on the performance of the inexact Newton scheme. But recall that the behavior of GIANT may strongly depend on the iterative linear solver. Therefore some attention is dedicated to this module of GIANT. For an efficient use of GIANT the realization of the user routines for the evaluation of the nonlinear function, the associated Jacobian and the matrix operations (c.f. Section 2.2) plays an important role. Concerning the solution of problems which are discretized partial differential equations even the used discretization may influence the performance of a Newton scheme — see [6]. Nevertheless, in order to concentrate on the performance of GIANT, the test frame used for the computations of this paper is as simple as possible, and thus certainly not well suited to solve challenging real life problems.

3.1 Test examples

The 2-D-discretizations of three quite different PDE-problems are used to generate test examples for the numerical experiments. For that purpose, the underlying PDE's are discretized with centered finite differences on tensor product grids. The finite difference approximations needed to discretize the test problems turn out to be:

$$\begin{aligned}
\Delta u = u_{xx} + u_{yy} \quad &\longrightarrow \quad \frac{1}{\Delta x^2}(u(x-\Delta x, y) - 2u(x,y) + u(x+\Delta x, y)) + \\
&\qquad \frac{1}{\Delta y^2}(u(x, y-\Delta y) - 2u(x,y) + u(x, y+\Delta y)) \\
u_x \quad &\longrightarrow \quad \frac{1}{2\Delta x}(u(x+\Delta x, y) - u(x-\Delta x, y)) \\
u_y \quad &\longrightarrow \quad \frac{1}{2\Delta y}(u(x, y+\Delta y) - u(x, y-\Delta y)) \ .
\end{aligned} \tag{3.1.1}$$

All computations presented are on equidistant grids but some experiments with slightly nonuniform rectangular grids show no essential change in the performance of the algorithms. In order to avoid another source of trouble for the interpretation of the numerical results, the analytical Jacobian of the discretized problems is used for the computations. A comparison to computations where the Jacobian is generated by numerical differentiation shows that the algorithmic behavior is scarcely influenced.

The following three boundary value problems are used to generate some test problems for the new software package GIANT.

Artificial Test Problem (atp)

This first example is an artificially generated scalar PDE for the function

$$\begin{array}{c} u(x,y) = \exp(-q) \ , \\ q = x^2 + y^2 \ . \end{array} \tag{3.1.2}$$

The equation used for numerical testing within this paper is

$$\Delta u - (0.9\exp(-q) + 0.1u)(4x^2 + 4y^2 - 4) + s \cdot g(x) = 0 \tag{3.1.3}$$

where

$$\begin{array}{ll} a) & g(x) = \exp(u) - \exp(\exp(-q)) \\ b) & s = \left\{ \begin{array}{lcl} +1 & : & \text{problem atp2} \\ -1 & : & \text{problem atp1} \end{array} \right. . \end{array} \tag{3.1.4}$$

The equation is solved in the domain $\Omega = [-3,3]^2$ with the simplified boundary conditions

$$u|_{\partial\Omega} = 0 \ . \tag{3.1.5}$$

The initial values are

$$u^0(x,y) = 0.2 \cdot \exp(-q) \quad \text{for} \quad (x,y) \in \Omega \ . \tag{3.1.6}$$

The PDE is discretized on a tensor product mesh of dimension $N = 31^2 = 961$ which is just the dimension of the nonlinear system to be solved. The distinction of the two cases ($s = \pm 1$) has been made in order to have a non-trivial linear (sub)problem. The case $s = +1$ (atp2) creates an indefinite (but still symmetric) Jacobian matrix — which may cause problems for the iterative linear solver. Concerning the Newton scheme both problems are not challenging.

SST Pollution (sst)

The following example is a straightforward extension of an instationary 1-D-model for the pollution of the stratosphere by supersonic transports (SSTs) — c.f. [12, 8]. The rather crude model describes the interaction of the chemical species O, O_3, NO and NO_2 along with a simple diffusion process. The equations for the stationary 2-D-model are:

$$\begin{array}{rcl} 0 & = & D\Delta u_1 + k_{1,1} - k_{1,2}u_1 + k_{1,3}u_2 + k_{1,4}u_4 - k_{1,5}u_1u_2 - k_{1,6}u_1u_4 \ , \\ 0 & = & D\Delta u_2 + k_{2,1}u_1 - k_{2,2}u_2 + k_{2,3}u_1u_2 - k_{2,4}u_2u_3 \ , \\ 0 & = & D\Delta u_3 - k_{3,1}u_3 + k_{3,2}u_4 + k_{3,3}u_1u_4 - k_{3,4}u_2u_3 + 800.0 + SST \ , \\ 0 & = & D\Delta u_4 - k_{4,1}u_4 + k_{4,2}u_2u_3 - k_{4,3}u_1u_4 + 800.0 \ , \end{array} \tag{3.1.7}$$

where

$$
\begin{aligned}
D &= 0.5 \cdot 10^{-9} \ , \\
k_{1,1}, \ldots, k_{1,6} &: 4 \cdot 10^5, 272.443800016, 10^{-4}, 0.007, 3.67 \cdot 10^{-16}, 4.13 \cdot 10^{-12} \ , \\
k_{2,1}, \ldots, k_{2,4} &: 272.4438, 1.00016 \cdot 10^{-4}, 3.67 \cdot 10^{-16}, 3.57 \cdot 10^{-15} \ , \\
k_{3,1}, \ldots, k_{3,4} &: 1.6 \cdot 10^{-8}, 0.007, 4.1283 \cdot 10^{-12}, 3.57 \cdot 10^{-15} \ , \\
k_{4,1}, \ldots, k_{4,3} &: 7.000016 \cdot 10^{-3}, 3.57 \cdot 10^{-15}, 4.1283 \cdot 10^{-12} \ , \\
SST &= \begin{cases} 3250 & \text{if } (x,y) \in [0.5, 0.6]^2 \\ 360 & \text{otherwise.} \end{cases}
\end{aligned}
$$

The boundary conditions are

$$
\left. \frac{\partial u_i}{\partial n} \right|_{\partial\Omega} = 0 \qquad \text{for } \ i = 1, \ldots, 4 \tag{3.1.8}
$$

and the domain Ω is just the unit square of $\mathbb{R}^2$. Two sets of initial values are used for the numerical experiments with this example. The first set consists of

$$
\begin{aligned}
u_1^0(x,y) &= 1.306028 \cdot 10^6 \ , \\
u_2^0(x,y) &= 1.076508 \cdot 10^{12} \ , \\
u_3^0(x,y) &= 6.457715 \cdot 10^{10} \ , \\
u_4^0(x,y) &= 3.542285 \cdot 10^{10}
\end{aligned}
\qquad (x,y) \in \Omega \ . \tag{3.1.9}
$$

The task of solving (3.1.7) (after discretization) with this initial values (problem sst1) can be considered as a mildly nonlinear problem which shows up in the fact that the undamped Newton method converges within a few iterations. In contrast to this, the second set of initial values

$$
\begin{aligned}
\overline{u}_1^0(x,y) &= 10^9 \ , \\
\overline{u}_2^0(x,y) &= 10^9 \ , \\
\overline{u}_3^0(x,y) &= 10^{13} \ , \\
\overline{u}_4^0(x,y) &= 10^7
\end{aligned}
\qquad (x,y) \in \Omega \tag{3.1.10}
$$

creates a highly nonlinear problem named sst2 (an undamped Newton scheme diverges). Both problems can be considered as "badly" scaled problems since the solution components are in the range from 10^6 upto 10^{12} and the residuals are in the range from 10^3 upto 10^{12} , $(r(\overline{u}^0) = 0.193 \cdot 10^{12}, r(\overline{u}^*_{GIANT}) = 0.181 \cdot 10^4$ where $\overline{u}^*_{GIANT}$ has approximately 5 correct decimal digits). Both problems are discretized on an uniform mesh with 26 mesh lines in both directions. Thus, the dimension of the discretized problems turns out to be $N = 4 \cdot 26^2 = 2704$. For some special experiments the test problem sst2 is

solved on finer grids also. Results for grids of size 51×51 ($N = 10404$) and 101×101 ($N = 40804$) are presented.

Driven Cavity Problem (dcp)

The last test problem is the classical driven cavity problem from incompressible fluid flow. The steady stream-function vorticity equations are

$$\begin{array}{ll} a) & \Delta\omega + \mathrm{Re}(\psi_x\omega_y - \psi_y\omega_x) = 0 \\ b) & \Delta\psi + \omega = 0 \end{array} \tag{3.1.11}$$

where ω is the vorticity, ψ is the stream function and Re represents the Reynolds number — see e.g. [9, 1]. The domain Ω is the unit square $[0,1]^2$. The usual boundary conditions are

$$\begin{array}{ll} a) & \psi|_{\partial\Omega} = 0 \\ b) & \left.\dfrac{\partial\psi}{\partial n}\right|_{\partial\Omega} = \begin{cases} 0 & \text{if } \ 0 \le y < 1 \\ 1 & \text{if } \ y = 1 \end{cases} . \end{array} \tag{3.1.12}$$

In order to avoid a discontinuity of the vorticity at the corners and to impose boundary conditions for the vorticity the formulation of [9] for (3.1.12.b) is used:

$$\frac{\partial\psi}{\partial y}(x,1) = -16x^2(1-x)^2 \tag{3.1.13}$$

$$\begin{aligned} \omega(x,0) &= -\frac{2}{\Delta y^2}\psi(x,\Delta y) \\ \omega(x,1) &= -\frac{2}{\Delta y^2}[\psi(x,1-\Delta y) + \Delta y\frac{\partial\psi}{\partial y}(x,1)] \\ \omega(0,y) &= -\frac{2}{\Delta x^2}\psi(\Delta x,y) \\ \omega(1,y) &= -\frac{2}{\Delta x^2}\psi(1-\Delta x,y) \end{aligned} \tag{3.1.14}$$

The equations (3.1.11) are discretized with usual finite differences on a uniform 31×31 mesh (c.f. [9]) which yields a nonlinear system with $N = 2\cdot31^2 = 1922$ unknowns. For the basic set of test problems the cases Re=100, Re=400 and Re=1000 are used (problems dcp100, dcp400 and dcp1000). For the experiments with larger Reynold numbers (Re=1000, 2000, 5000) a uniform mesh with 63^2 grid points (c.f. [1]) is used ($N = 7938$). For all cases, the initial estimates are given by

$$\psi(x,y) = \omega(x,y) = 0 \qquad \text{for} \quad (x,y) \in \Omega\ . \tag{3.1.15}$$

3.2 Numerical results

All experiments have been carried out with the extended GIANT package level 3 - c.f. Section 2.1. With that, the two linear solvers GBIT1 (c.f. Section 1.1) and GMRES (from SLAP) and the standard routines of SLAP for matrix-vector multiplication and preconditioning are available. If not otherwise stated, the default options and parameters for GIANT and GBIT1 are active — i.e. all problems are specified as being highly nonlinear (c.f. Table 1.1) and a value of $\rho = 1/6$ (c.f. (1.1.13)) is selected. For the required relative tolerance of the solution of the nonlinear problem a value of RTOL=10^{-5} is prescribed. Recall that this value has to be interpreted in connection with the associated scaling vector xscale. The internal scaling update is done via (1.3.1.a) and an initial value of XSCALE≡1 is used for all problems. Concerning the linear solvers two specifications are still open. For GBIT1, the storage restriction parameter is set to $k_{max} = 9$ which allows (at most) 10 iterations of the Good Broyden scheme before a restart is necessary. As the standard preconditioner the incomplete LU (ILU) factorization is selected. In order to apply the iterative linear solver GMRES within GIANT the simplified accuracy matching strategy (c.f.Section 1.1.4) is used. Some numerical testing suggested the following choice of the internal options of GMRES. For all tests, the linear solver is called with left preconditioning and without using the scaling vectors for the iterate and the right hand side. Denoting with H the preconditioning matrix as introduced in Section 1.2 (i.e. H "approximates" A^{-1}), and with

$$\tilde{\varepsilon}^{req} := \frac{\varepsilon^{req}}{\overline{\rho}}$$

the stopping criterion of GMRES reads as follows:

$$\|H(b - As_{iter})\| \leq \tilde{\varepsilon}^{req}\|Hb\|$$

and the error estimate returned by GMRES is defined by

$$\tilde{\varepsilon}^{est}_{GMRES} := \frac{\|H(b - As_{iter})\|}{\|Hb\|} \ .$$

Thus, disregarding the fact that the error is not really measured in the space of the iterates, the final error estimate returned to GIANT is

$$\varepsilon^{est} := \overline{\rho}\tilde{\varepsilon}^{est}_{GMRES} \ .$$

This simplification is made to avoid modifications of the GMRES code in order to compute an error estimate directly for the iterates, which would be quite cumbersome. As for GBIT1, at most 10 iterations are allowed in the standard case before a restart occurs, thus setting KMAX=10.

The above mentioned test examples atp1, atp2, sst1, sst2, dcp100, dcp400, dcp1000 are used as a basic testset for GIANT. Although the dimensions of these examples are comparatively small they are well suited to check some features of the code. The observations and conclusions which can be derived from the numerical experiments with this testset are confirmed by testing the codes on larger and more critical problems. These examples are derived from the basic examples by increasing the dimension of the grids and setting more challenging problem parameter values. All numerical experiments have been carried out either on SUN-SPARC1 workstations (16MB storage) in FORTRAN double precision or on the CRAY X-MP of the Konrad-Zuse-Zentrum Berlin using FORTRAN single precision. If not otherwise stated, the results from the workstation computations are presented.

3.2.1 Accuracy matching strategy

The most important question to be discussed is certainly whether the accuracy matching strategy works satisfactorily. As the examples are small enough to allow a direct solution of the linear systems, e.g. with a bandmode LU factorization, the performance of the *inexact* Newton scheme (algorithm I of Section 1.1) can be compared with it's *exact* counterpart (algorithm E). But the performance of GIANT depends on both, the *true* and the *estimated* error of the linear systems solution. To close the gap between the error estimate and the true error, the solver GBIT1 is used with different safety factors $\overline{\rho}$. The results of this first comparison are given in Table 3.1.

example	exact	$\overline{\rho}=1$	$\overline{\rho}=4$	$\overline{\rho}=40$	$\overline{\rho}=400$
atp1	5	5	5	5	5
atp2	7	failnew	faillin	faillin	faillin
sst1	5	6	6	5	5
sst2	23	39	23	23	23
dcp100	6	8	6	6	6
dcp400	8	10	9	9	9
dcp1000	10	11	11	10	10

Table 3.1 Number of function evaluations for the exact and the inexact Newton scheme (GBIT1 as linear solver)

The number of evaluations of the nonlinear function F ($\#F$) can be taken as a quite good indicator for the performance of the Newton scheme. As long as no corrector steps must be performed the number of Newton iterations and the number of Jacobian evaluations ($\#J$) are just $\#F-1$. Furthermore, the number of linear systems to be solved is usually (no corrector step) just

$2 \cdot (\#F - 1)$.

Analyzing the results presented in Table 3.1 one can state that already for $\overline{\rho} = 4$ there is a quite good agreement of the inexact and the exact Newton scheme. For $\overline{\rho} = 1$ a more distinct difference shows up, at least for the example sst2. A further increase of the safety factor $\overline{\rho}$ beyond $\overline{\rho} = 4$ obviously doesn't pay off. With $\overline{\rho} = 40$, the last discrepancy between the behavior of the inexact and the exact Newton scheme disappears and a choice of $\overline{\rho} = 400$ only increases the amount of work for linear algebra. The total number of iterations of the linear solver GBIT1 for the runs with $\overline{\rho} = 4$ and $\overline{\rho} = 400$ are summerized in Table 3.2. Within this Table, the number of linear iterations required to solve for the ordinary Newton correction Δx_k and the simplified correction $\overline{\Delta x_k}$ respectively are distinguished. Observe that within this Section the notation Δx_k and $\overline{\Delta x_k}$ is used for the inexact values s_k and $\overline{s}_k$ also.

Recall that the example atp2 is constructed in such a way, that divergence of GBIT1 can be expected. Regarding the error message from GIANT, the cases $\overline{\rho} = 1$ and $\overline{\rho} \geq 4$ differ. In the first case, the monotonicity check (1.1.22) fails repeatedly and the Newton iteration terminates as the minimum damping factor is reached. Obviously, the Newton corrections $\|\Delta x_0\|$ and $\|\Delta x_1\|$ are not accurate enough. In the second case, the iterative linear solver GBIT1 signals divergence during the iteration for the first simplified Newton correction. From this example and some other testing the following hint is derived. If GIANT signals failure of the Newton iteration the actual problem should be solved again but with more stringent accuracy requirements for the linear solver (decrease ρ) or more pessimistic error estimation within the linear solver (increase of $\overline{\rho}$). But a general setting of a too small value for ρ (or too large $\overline{\rho}$) may increase the number of linear iterations drastically without any gain concerning the Newton iteration. Thus, the standard values $\rho = 1/6$ and $\overline{\rho} = 4$ are implemented in this code.

example	$\#J$		$\#F$		#it-lin-ord		#it-lin-sim		#it-lin	
atp1	4	4	5	5	33	48	22	37	55	85
atp2	fail									
sst1	5	4	6	5	92	165	30	97	122	262
sst2	22	22	23	23	235	498	127	388	362	886
dcp100	5	5	6	6	102	288	38	168	140	456
dcp400	8	8	9	9	274	799	78	375	352	1174
dcp1000	10	9	11	10	376	2349	144	681	520	3030

Table 3.2 $\#J, \#F$ and # linear iterations for $\overline{\rho} = 4$ and $\overline{\rho} = 400$ ($GBIT1$)

Using GMRES as an iterative linear within GIANT similar observations can

be made. Table 3.3 shows the performance of the exact and inexact Newton scheme, again for different values of $\overline{\rho}$. A detailed statistic for $\overline{\rho} = 4$ and $\overline{\rho} = 400$ is given in Table 3.4. For GMRES a safety factor of at least $\overline{\rho} = 40$ is required in order to have good accordance of the inexact and the exact Newton scheme.

example	exact	$\overline{\rho} = 1$	$\overline{\rho} = 4$	$\overline{\rho} = 40$	$\overline{\rho} = 400$
atp1	5	6	5	5	5
atp2	7	8	7	7	7
sst1	5	7	5	5	5
sst2	23	failnew	faillin	24	22
dcp100	6	8	7	7	6
dcp400	8	16	10	9	9
dcp1000	10	61	12	11	10

Table 3.3 Number of function evaluations for the exact and the inexact Newton scheme (GMRES as linear solver)

example	$\#J$		$\#F$		#it-lin-ord		#it-lin-sim		#it-lin	
atp1	4	4	5	5	27	42	16	29	43	71
atp2	6	6	7	7	85	120	38	73	123	193
sst1	4	4	5	5	117	205	16	77	133	282
sst2	–	21	–	22	–	420	–	285	–	705
dcp100	6	5	7	6	143	184	55	152	198	336
dcp400	9	8	10	9	218	442	99	281	317	723
dcp1000	11	9	12	10	223	437	100	293	323	730

Table 3.4 $\#J, \#F$ and # linear iterations for $\overline{\rho} = 4$ and $\overline{\rho} = 400$ (GMRES)

Note that the differences show up especially for the non-trivial (concerning the Newton iteration) examples sst2 and dcp1000. Thus, one should not simply compare the tables 3.2 and 3.4 in order to judge the efficiency of the new iterative solver GBIT1. The numerical experience from using both codes within the GIANT package suggest that a comparison of GBIT1 with $\overline{\rho} = 4$ and GMRES with $\overline{\rho} = 400$ is more adequate. But, recall that the adaptation of GMRES for the application within GIANT is certainly not optimal. A comparison of GBIT1 and GMRES (with corresponding values for *kmax*) can be done in terms of the iteration count, as the number of PRECON/MULJAC calls per iteration as well as the internal amount of work per iteration is quite similar — with a slight advantage for GMRES.

Relating to the example sst2, the results from GIANT+GMRES are quite interesting. First, similar to the case GIANT+GBIT1 in example atp2, an increase of $\overline{\rho}$ switches the error message of GIANT — but the reason for the fails is quite different. For $\overline{\rho} = 1$, the Newton iteration stops as the maximum number of Newton iterations (50) is reached, and, for $\overline{\rho} = 4$ the failure of GMRES occurs after 48 Newton iterations. Obviously, the achieved accuracy for the Newton corrections is not sufficient to allow convergence of the Newton iteration and, moreover, the Newton iteration may diverge in such a way that a point is reached where the iterative linear system solution breaks down. The difficulties in solving sst2 are removed, if $\overline{\rho}$ is increased further.

The results presented so far are open for two quite different interpretations. Believing in the error estimator of the linear solver, one may conclude from the above mentioned results that the accuracy requirements from GIANT for the linear systems solution are not sufficient to guarantee a performance of the inexact Newton scheme which is comparable with the exact one. Trusting in the theoretical consideration which led to the accuracy matching strategy of GIANT, one might conclude that the error estimation of the linear solver is often too optimistic. For the example, which needs the most (damped) Newton iterations (sst2), this question is analyzed in more detail. The performance of the exact and the inexact iteration (with GBIT1, $\overline{\rho} = 4$) is illustrated in Figures 3.1 and 3.2. Figure 3.1 presents the so called level functions $\|\Delta x_k\|$ of both iterations while Figure 3.2 shows the associated damping factors λ_k. Up to the 15 th iterate there is a quite good agreement. The first significant difference occurs for the level functions of the 16 th iterate which leads to an obvious disturbance of the damping factor λ_{16}. But after that, the inexact Newton iteration converges back to the exact iteration. Figure 3.3 shows the number of linear iterations which are required to solve the linear systems for $\|\Delta x_k\|$ and $\|\overline{\Delta x}_{k+1}\|$ respectively. Observe the drop in the iteration count just for $k = 16$. The required accuracy ε_k^{req} for the computation of the ordinary Newton correction Δx_k is given in Figure 3.4.

Now, in order to check the quality of the error estimation ε^{est} of the approximation s_k for Δx_k, the linear iteration is continued with a smaller value for the required accuracy, e.g. $\varepsilon_{new}^{req} := \varepsilon_{iter}^{est} \cdot 10^{-3}$. With the associated approximation $\tilde{s}_k$ at hand the true error of the iterate s_k^{iter} is estimated to be (c.f. (1.1.8))

$$\varepsilon_k^{true} := \frac{\|s_k^{iter} - \tilde{s}_k\|}{\|\tilde{s}_k\|} \, .$$

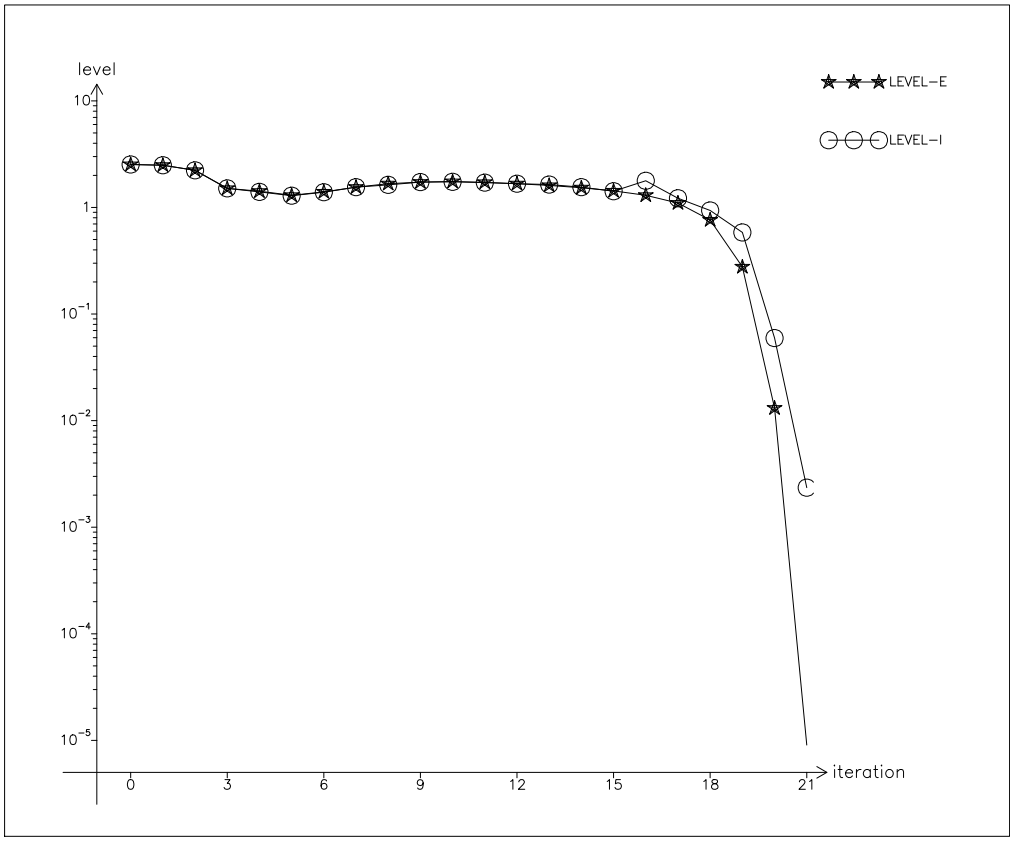


Figure 3.1 sst2: level functions $\|\Delta x_k\|$ for the exact and inexact scheme

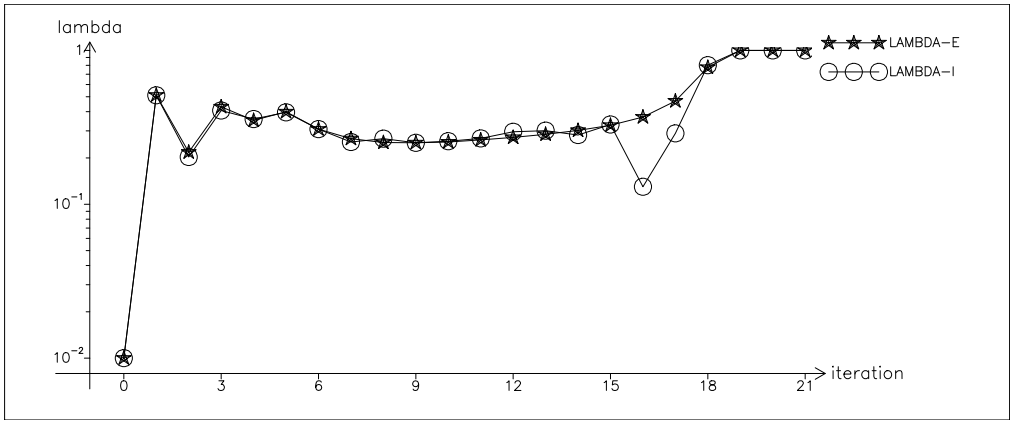


Figure 3.2 sst2: damping factors λ_k for the exact and inexact scheme

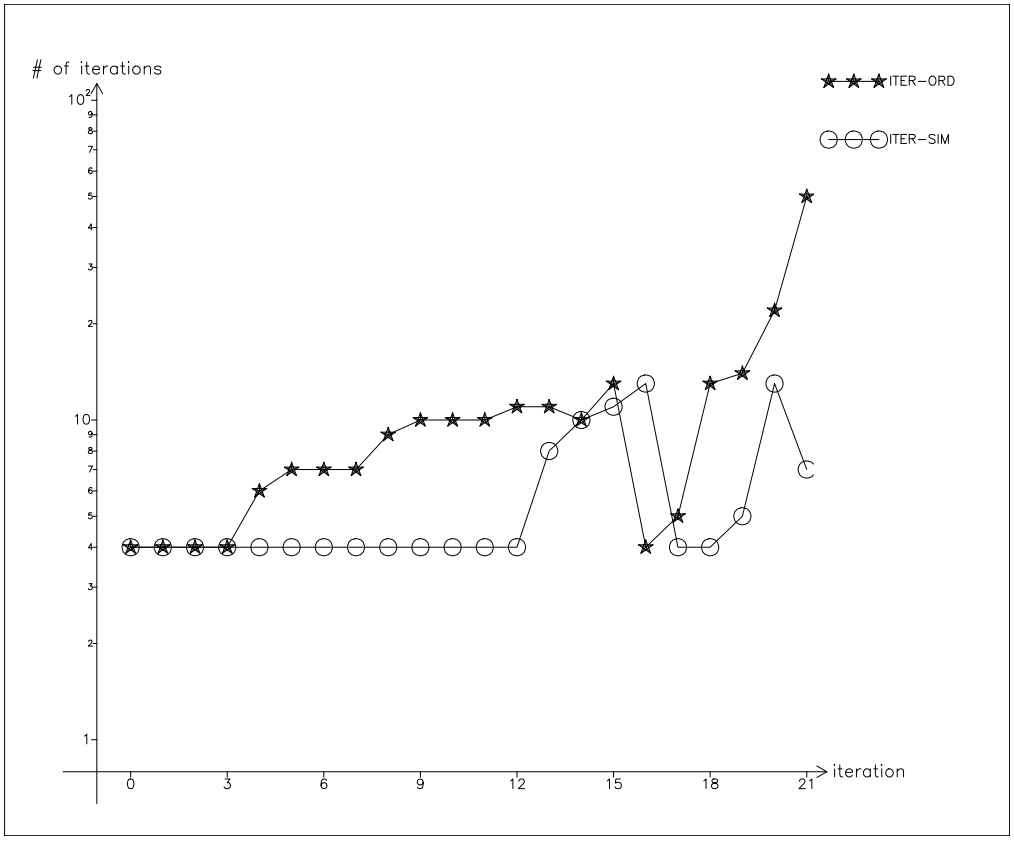


Figure 3.3 sst2: number of linear iterations

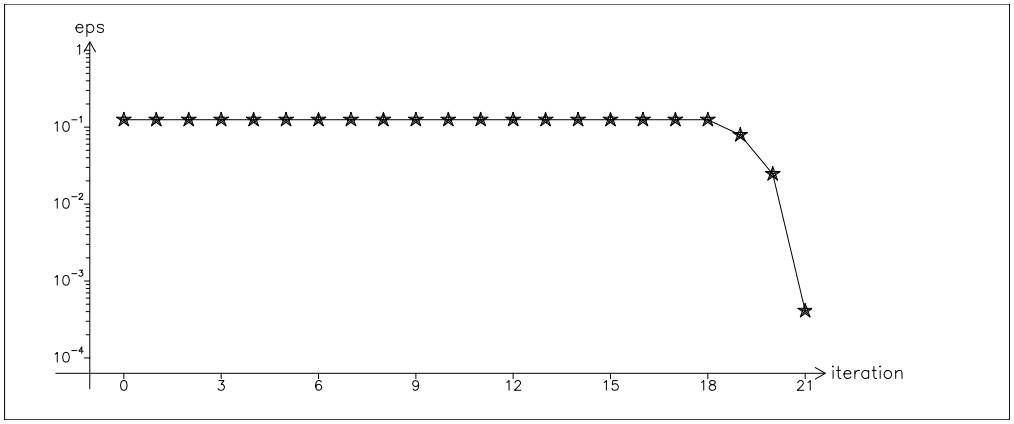


Figure 3.4 sst2: required accuracy ε_k^{req}

Figure 3.5 shows the required accuracy ε_k^{req}, the associated error estimate ε_k^{est} (from GBIT1 with $\overline{\rho} = 4$) and the approximation for the true error ε_k^{true}. Just for the 16 th iterate, the true error is larger than the required one — with the consequence of creating a distinct difference of the inexact and the exact Newton iteration. Figures 3.6 and 3.7 present the true error ε_k^{true} of s_k

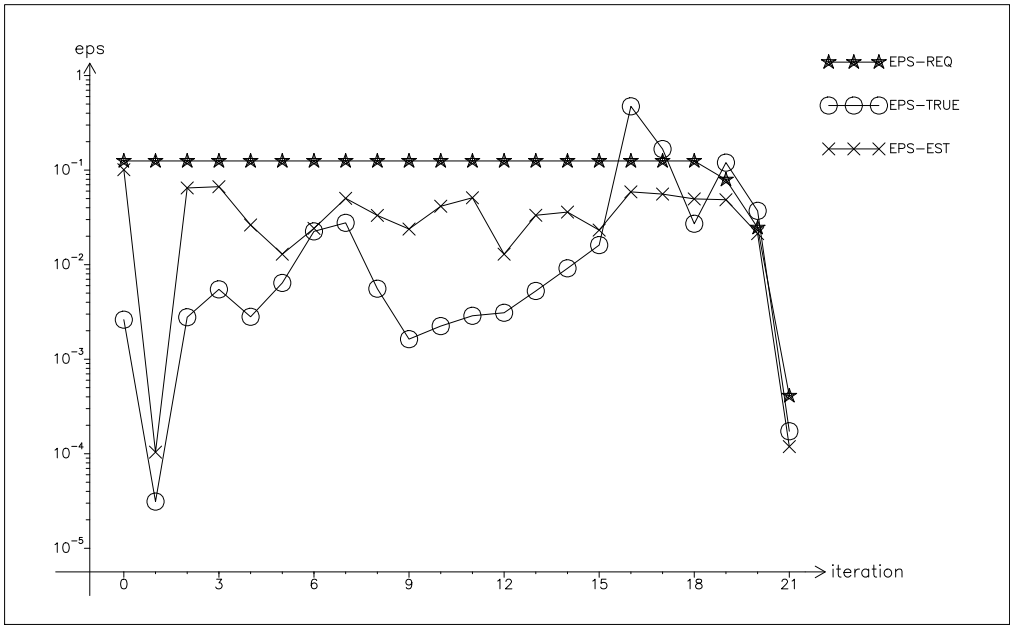


Figure 3.5 sst2: required, estimated and true accuracy of $\|\Delta x_k^{iter}\|$ for $\overline{\rho} = 4$

and the associated damping factors λ_k for the case $\overline{\rho} = 1$. Strong violations of the accuracy requirements of GIANT lead to drastically reduced damping factors λ_k. But the achieved accuracy of the linear solutions is still sufficient to guarantee convergence of GIANT — now within 33 instead of 22 iterations. Things change, if one solves the standard example sst2 on a refined grid. Using a mesh of dimension 51×51 for the finite difference discretization one ends up with a nonlinear system of dimension $N = 10404$. For the standard case $\overline{\rho} = 4$, the inexact Newton scheme stops after 23 iterations because the monotonicity check fails repeatedly. Restarting GIANT, but now with a safety factor $\overline{\rho} = 40$ for GBIT1, the problem is solved within 22 Newton iterations, just as before.

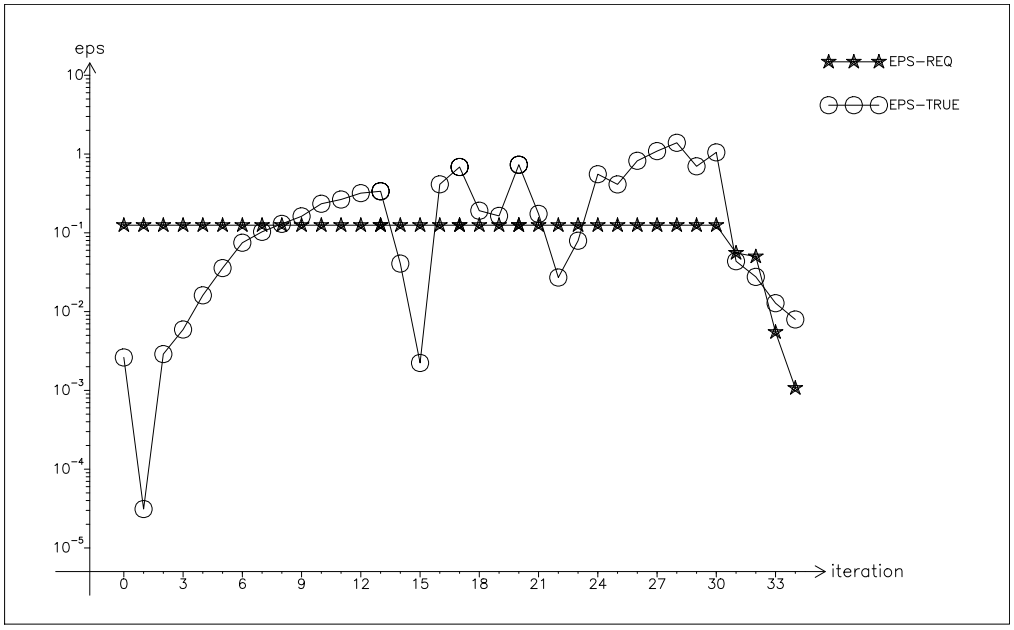


Figure 3.6 sst2: required and true accuracy of the inexact ordinary Newton corrections $\|\Delta x_k\|$ for $\overline{\rho} = 1$

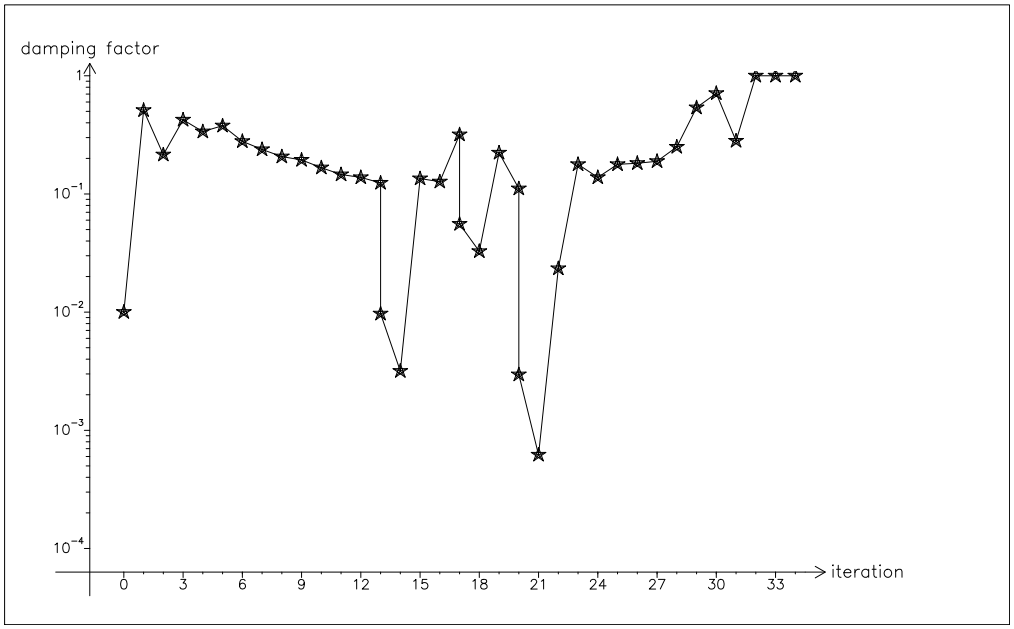


Figure 3.7 sst2: damping factors λ_k for $\overline{\rho} = 1$

Figure 3.8 illustrates for both cases the behavior of the required and the true accuracy of the inexact ordinary Newton correction. In contrast to the case of $\overline{\rho} = 1$ on a 26×26 mesh, for the case $\overline{\rho} = 4$ on a 51×51 mesh the repeated violation of the accuracy bound $\varepsilon^{req} = 1/8$ causes failure of the inexact Newton iteration. The behavior of the Newton scheme for slight violations of the accuracy bound nicely reflects the fact that the underlying theoretical considerations (c.f. [4]) expect convergence even for $\varepsilon^{true} = 1/4$ (i.e. $\rho = 1/2$). Increasing the condition of the matrices J_k a second time by

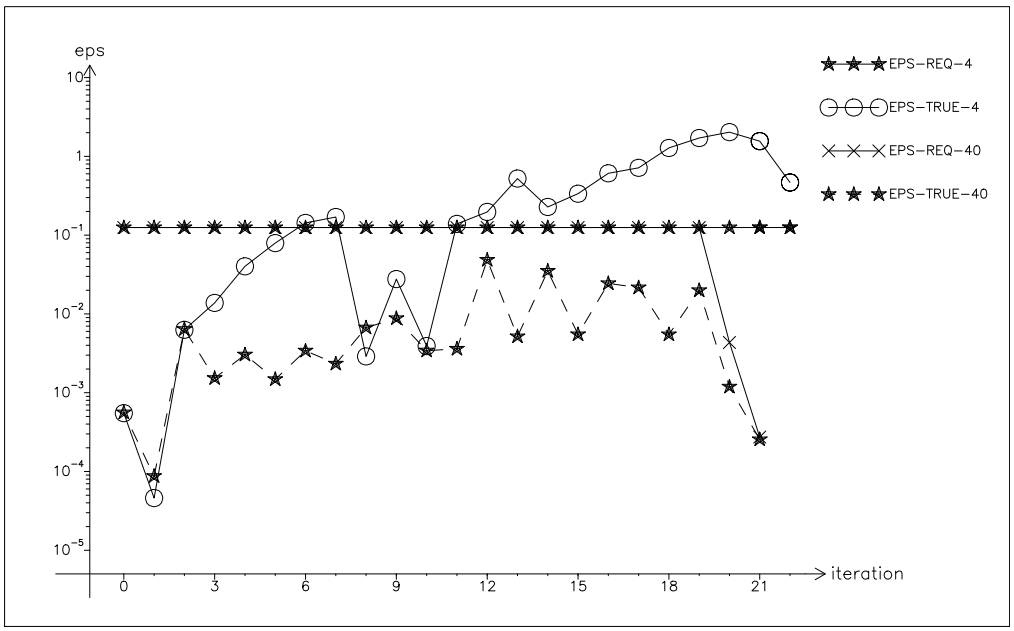


Figure 3.8 sst2 (51×51 grid): required and true accuracy of $\|\Delta x_k^{iter}\|$ for $\overline{\rho} = 4$ and $\overline{\rho} = 40$

solving the example sst2 on a 101×101 mesh ($N = 40804$), the problems for the iterative solver GBIT1 increase again. For the case $\overline{\rho} = 40$ the usual Newton performance (#Newton-iter = 22) is disturbed as now 25 Newton steps are required. By setting $\overline{\rho} = 400$ the solution is achieved within 22 Newton steps again. Another quite interesting performance results from changing the preconditioner. Using a block diagonal LU decomposition as preconditioner, the performance of the iterative solver is improved. The results for $\overline{\rho} = 4$ and $\overline{\rho} = 40$ are given in Table 3.5. For this specific example, the block diagonal preconditioning not only reduces the number of linear iterations but also increases the robustness of GBIT1.

The outcome of these results and some other testing is obvious. If the true achieved accuracy of the linear solutions fits into the requirements made by the accuracy matching strategy of GIANT, the performance of the inexact

example	status	#J	#F	#it-lin-ord	#it-lin-sim	#it-lin
$\overline{\rho} = 4$						
sst2-26-ilu	ok	22	23	235	127	362
sst2-26-bdiag	ok	22	23	163	99	262
sst2-51-ilu	failnew	23	26	146	162	308
sst2-51-bdiag	ok	22	23	219	107	326
sst2-101-ilu	faillin	22	22	205	95	300
sst2-101-bdiag	faillin	1	2	4	3	7
$\overline{\rho} = 40$						
sst2-26-ilu	ok	22	23	341	276	617
sst2-26-bdiag	ok	22	23	212	115	327
sst2-51-ilu	ok	22	23	440	212	652
sst2-51-bdiag	ok	22	23	402	140	542
sst2-101-ilu	ok	25	26	644	376	1020
sst2-101-bdiag	ok	22	23	409	171	580

Table 3.5 Performance of GIANT/GBIT1 for the modified example sst2 (ILU vs. blockdiagonal preconditioning, safety factors $\overline{\rho} = 4$ and $\overline{\rho} = 40$)

and exact Newton algorithm coincide extremely well. This experimentally confirms the theoretical considerations of Deuflhard [4].

3.2.2 Iterative linear solvers

The code GIANT can be considered as a special test frame for an iterative linear solver. As the efficiency and the robustness of GIANT strongly depends on the performance of the linear solver, some experiments with GBIT1 and GMRES are worth mentioning. Due to the different adaptation for the application within GIANT, a fair comparison of GBIT1 with GMRES is quite difficult. Rather than, one may change the preconditioner or modify *kmax* and compare the changes in the behavior of the codes. As the required storage of both codes is dominated by the actual value for the parameter *kmax*, one may study their performance for values different from $k_{max} = 9$ (GBIT1) and KMAX=10 (GMRES) — the standard values used so far. Table 3.6 shows the result of using GBIT1 with $k_{max} = 4$ and $k_{max} = 14$ respectively. In order to facilitate the interpretation of the results, the difference to the standard case $k_{max} = 9$ is directly presented, moreover, in symbolic form. An entry "$\circ$" means that the change is below 10%. "+" and "$-$" indicate a difference in the range $\pm 10\%$ up to $\pm 50\%$ and "++","$--$" represent a change beyond that range. In order to measure the different behavior, one must look to different indicators. As the performance of GIANT may be affected, the number of evaluations of the nonlinear function (#F) is given in Table 3.6 also. The changes in the total number of linear iterations (#iter)

are certainly the most important measure, but as the amount of work per iteration depends on *kmax*, the total CPU time for the linear iterations (CPU) is given additionally. As this time dominates the total CPU time required to solve the nonlinear problem, one can drop those numbers.

The results of Table 3.6 are somewhat surprising. An increase of *kmax* doesn't entails a better performance of GBIT1, even a change for the worth may occur. Consequently, the reduction of *kmax* can improve or deteriorate the behavior of GBIT1. Note that for dcp400 the slight reduction of the number of iterations is accompanied with a more significant gain in the CPU time.

	kmax=4			*kmax*=14		
example	#F	#iter	CPU	#F	#iter	CPU
atp1	∘	∘	∘	∘	∘	∘
atp2	faillin			faillin		
sst1	∘	+	+	∘	+	+
sst2	∘	+	+	∘	∘	∘
dcp100	+	+	+	+	+	+
dcp400	∘	∘	−	∘	−	−
dcp1000	∘	∘	∘	∘	−	−

Table 3.6 Comparison of GBIT1 ($\overline{\rho} = 4$) with different *kmax* (ref. to *kmax*=9)

Making the same experiment with GMRES the results are more uniform. An increase of *kmax* yields a better performance of GMRES, while a reduction of *kmax* leads to loss of efficiency and robustness (faillin for sst1, sst2). Again, note that the influence may be different concerning the number of iterations and the required CPU time (atp2, dcp1000). The effect of changing *kmax* is studied again in the special experiments with the driven cavity problem — see Section 3.2.3.

	kmax=5			*kmax*=15		
example	#F	#iter	CPU	#F	#iter	CPU
atp1	∘	+	+	∘	∘	∘
atp2	∘	++	++	∘	−	∘
sst1	faillin			∘	∘	∘
sst2	faillin			∘	−	−
dcp100	∘	∘	∘	∘	∘	∘
dcp400	∘	∘	∘	∘	∘	∘
dcp1000	∘	+	∘	∘	∘	∘

Table 3.7 Comparison of GMRES ($\overline{\rho} = 400$) with different *kmax* (ref. to *kmax*=10)

The performance of an iterative linear solver is strongly affected by the qual-

ity of the preconditioner. A "good" preconditioner may increase the convergence speed and the convergence domain of the basic iterative method. On the other hand, each linear iteration may be more expensive. The consequences of changing the preconditioner are summerized in Table 3.8 (for GBIT1) and Table 3.9 (for GMRES). Three alternatives for the standard ILU preconditioning are checked: the diagonal and the lower triangle preconditioner from the Sparse Linear Algebra Package (SLAP), as well as blockdiagonal preconditioning. These experiments show a clear result. Except for the sst examples, the above mentioned alternative preconditioning techniques are not competitive to ILU preconditioning. Even, if diagonal preconditioning would be "for free" (e.g. on vector machines), the ILU preconditioning still yields the fastest linear iteration. Beyond that, the use of more sophisticated preconditioners, e.g. an ILU factorization with fill-in, seems to be promising for the non-trivial examples of the basic testset. The good result of the blockdiagonal preconditioner for the example sst2 (c.f. Table 3.5) are due to the special structure of the underlying PDE — c.f. (3.1.7). A general comparison of Table 3.8 with 3.9 indicates a more robust behavior of GBIT1 compared with GMRES — disregarding the fact that GBIT1 is still not able to solve the linear subproblems of example atp2.

	Blockdiagonal			Diagonal			Lower triangle		
example	#F	#iter	CPU	#F	#iter	CPU	#F	#iter	CPU
atp1	∘	++	++	∘	++	+	∘	++	++
atp2	failnew			failnew			failnew		
sst1	∘	+	+	faillin			++	++	++
sst2	∘	−	−	failnew			faillin		
dcp100	+	++	++	+	++	++	+	++	++
dcp400	+	++	++	+	++	++	∘	++	++
dcp1000	+	++	++	∘	++	++	∘	++	++

Table 3.8 Comparison of different preconditioners used with GBIT1 ($\overline{\rho} = 4$)

example	Blockdiagonal			Diagonal			Lower triangle		
	#F	#iter	CPU	#F	#iter	CPU	#F	#iter	CPU
atp1	∘	++	++	∘	++	++	∘	++	++
atp2	∘	++	++	∘	++	++	∘	++	++
sst1	∘	−	−	faillin			faillin		
sst2	∘	−−	−−	faillin			faillin		
dcp100	+	++	++	+	++	++	∘	++	++
dcp400	faillin			∘	++	++	∘	++	++
dcp1000	faillin			faillin			faillin		

Table 3.9 Comparison of different preconditioners used with GMRES ($\overline{p} = 400$)

3.2.3 Special experiments for the driven cavity problem

The driven cavity problem is a widely used test example for iterative methods (linear and nonlinear), as well as for special discretizations schemes. In order to check the performance of GIANT, this example is now solved also for more critical Reynolds numbers. But this task requires at least a finer discretization, hence, a 63×63 mesh is used from now on. But no attempt has been made to use a more suitable discretization or to exploit the special structure of the equations. The problem is still solved as a coupled system of two PDE's. Thus, after discretization, a nonlinear system with 7938 unknowns arises. For comparison, the cases Re=1000, 2000 and 5000 are considered.

Table 3.10 gives the results of a whole sequence of runs. Herein, the storage restriction parameter varies between the extremely storage saving value *kmax*=2 and the quite large value *kmax*=24. The total CPU time (in seconds, on SUN-SPARC1) used by GIANT (f77 optimization level 3) is presented within the Table, as the average time per linear iteration may depend on the prescribed *kmax*. As a consequence, a run with more linear iterations but with a smaller *kmax* may be more efficient, because for a smaller *kmax* the average time for an iteration is usually reduced.

example	status	#J	#F	#it-ord	#it-sim	#it-lin	CPU(s)
dcp1000-2	ok	13	14	1087	290	1377	972
dcp1000-4	ok	11	12	1056	309	1365	927
dcp1000-9	ok	12	13	1013	214	1227	902
dcp1000-14	ok	12	13	918	211	1129	913
dcp1000-19	ok	13	14	824	250	1074	939
dcp1000-24	ok	11	12	1009	193	1202	1090
dcp2000-2	ok	14	15	1317	2258	3575	2396
dcp2000-4	ok	13	14	1356	261	1617	1090
dcp2000-9	ok	14	15	1238	279	1517	1121
dcp2000-14	ok	14	15	1623	249	1872	1496
dcp2000-19	ok	15	16	1263	292	1555	1318
dcp2000-24	ok	15	16	1027	242	1269	1126
dcp5000-2	ok	23	24	1805	668	2473	1750
dcp5000-4	ok	15	16	1447	282	1729	1185
dcp5000-9	ok	17	18	1316	314	1630	1213
dcp5000-14	ok	18	19	1199	381	1580	1283
dcp5000-19	ok	18	19	1464	273	1737	1483
dcp5000-24	ok	17	18	1379	293	1672	1507

Table 3.10 dcp1000, 2000, 5000 with GBIT1 ($\overline{\rho} = 4$)

In contrast to the observations made for the basic testset, the Newton iteration is now influenced by the variation of *kmax* — a clear hint that the achieved accuracy of GBIT1 is not sufficient. Rerunning the sequence of problems, but now with a safety factor $\overline{\rho} = 40$ for GBIT1, retrieves the independence of the GIANT performance with respect to variations within the iterative linear solver — see Table 3.11.

Furthermore, the increasing difficulty of the problem for larger Reynolds numbers requires now only one additional Newton iteration per increase of the Reynolds number — independent of *kmax*. In contrast to this, for the standard option $\overline{\rho} = 4$ the difference between Re=1000 and Re=5000 may be 2 up to 12 Newton iterations — depending on *kmax*. For both experiments the change of the GBIT1 performance due to the *kmax* variation shows again a quite irregular pattern. Concerning the overall efficiency, a comparison of the dcp5000 run with $\overline{\rho} = 4$ and $\overline{\rho} = 40$ respectively, shows a quite interesting result. Although for $\overline{\rho} = 4$ GIANT needs for all runs more Newton iterations, all runs are faster than the runs with $\overline{\rho} = 40$. This is due to the fact that the total CPU time of GIANT is mainly spent in the iterative linear solver (including preconditioning and matrix $\times$ vector routine). For $\overline{\rho} = 4$, this part uses about 92% of the total CPU time while for $\overline{\rho} = 40$ this part is even about 96%. The performance of GIANT/GBIT1 for the sequence of safety

example	status	#J	#F	#it-ord	#it-sim	#it-lin	CPU(s)
dcp1000-2	ok	10	11	1834	556	2390	1606
dcp1000-4	ok	10	11	1792	676	2468	1611
dcp1000-9	ok	10	11	1599	528	2127	1503
dcp1000-14	ok	10	11	1665	423	2088	1626
dcp1000-19	ok	10	11	1714	393	2107	1791
dcp1000-24	ok	10	11	1722	399	2121	1889
dcp2000-2	ok	11	12	1868	702	2570	1727
dcp2000-4	ok	11	12	1954	662	2616	1709
dcp2000-9	ok	11	12	1280	592	1872	1339
dcp2000-14	ok	11	12	1960	600	2560	2000
dcp2000-19	ok	11	12	1833	539	2372	1967
dcp2000-24	ok	11	12	1868	651	2519	2183
dcp5000-2	ok	12	13	3570	555	4125	2736
dcp5000-4	ok	12	13	2439	1087	3526	2293
dcp5000-9	ok	12	13	1509	667	2176	1567
dcp5000-14	ok	12	13	1652	524	2176	1714
dcp5000-19	ok	12	13	1684	531	2215	1858
dcp5000-24	ok	12	13	1444	624	2068	1806

Table 3.11 dcp1000, 2000, 5000 with GBIT1 ($\overline{\rho} = 40$)

factors $\overline{\rho} = 1, 4, 40, 400$ on the problem dcp5000 is given in Table 3.12.

example	status	#J	#F	#it-ord	#it-sim	#it-lin	CPU(s)
dcp5000-r001	FailLin	9	9	249	32	281	247
dcp5000-r004	ok	17	18	1316	314	1630	1213
dcp5000-r040	ok	12	13	1509	667	2176	1567
dcp5000-r400	ok	12	13	3276	1491	4767	3330

Table 3.12 dcp5000 with GBIT1 ($kmax = 9$)

Obviously, the case $\overline{\rho} = 4$ is just the limiting case for convergence of the inexact scheme. The iterative behavior for the standard case ($\overline{\rho} = 4$, $kmax$=9) is illustrated in Figures B.3 – B.6. Figure B.1 and Figure B.2 show contour plots for the associated numerical solution. The contour levels plotted are

$$\omega = -4, -2, -1, -0.5, 0, 1.0, 2, 4, 6, 10 \qquad \text{and}$$
$$\psi = -0.01, -0.002, -0.001, -0.0002, 0.00, 0.01, 0.02, 0.03, 0.04, 0.05 \ .$$

In order to solve the examples dcp1000, dcp2000 and dcp5000 with the combination GIANT+GMRES, a safety factor of $\overline{\rho} = 400$ is necessary. The results for the KMAX-sequence 5,10,15,20,25 are summerized in Table 3.13. For $\overline{\rho} = 4$ only the cases dcp1000-10 and dcp2000-25 converge. For $\overline{\rho} = 40$

there is only one successful run: dcp2000-20. All other runs (for $\overline{\rho} = 4$, $\overline{\rho} = 40$) show a failure of GMRES indicated by a value 2 of the error flag IERR, which means that GMRES fails to reduce any more the residual.

example	status	#J	#F	#it-ord	#it-sim	#it-lin	CPU(s)
dcp1000-5	ok	10	11	1508	1788	3296	1846
dcp1000-10	ok	10	11	1446	1527	2973	1819
dcp1000-15	ok	10	11	1299	1261	2560	1716
dcp1000-20	ok	10	11	1126	1002	2128	1551
dcp1000-25	ok	10	11	936	856	1792	1413
dcp2000-5	ok	12	13	1239	1245	2484	1425
dcp2000-10	ok	12	13	1054	1052	2106	1324
dcp2000-15	ok	12	13	991	938	1929	1320
dcp2000-20	ok	12	13	928	993	1921	1418
dcp2000-25	ok	12	13	939	852	1791	1420
dcp5000-5	ok	15	16	1685	747	2432	1419
dcp5000-10	ok	15	16	1150	669	1819	1175
dcp5000-15	ok	16	18	1346	1152	2498	1714
dcp5000-20	ok	15	17	997	745	1742	1304
dcp5000-25	ok	15	16	1057	3502	4559	3540

Table 3.13 dcp1000, 2000, 5000 with GMRES ($\overline{\rho} = 400$)

Analyzing the numbers of Table 3.13, the tendency of GMRES that an increase of *kmax* pays off, shows up only for dcp1000. Already for dcp2000 there is a slight disturbance and for dcp5000 the performance is even more irregular as for GBIT1. Although a direct comparison of the results from GIANT+GBIT1 and GIANT+GMRES is quite dubious, the choice GBIT1 ($\overline{\rho} = 4$) is obviously the most efficient one. The safe variant with GBIT1 ($\overline{\rho} = 40$) is still competitive to the GMRES ($\overline{\rho} = 400$) version of GIANT and, furthermore, shows a distinct increase of robustness. A comparison to the results presented in [1] is certainly even more dubious. But, since in [1] a quite different globalization technique for the Newton method is realized, a look to the results of NKSOL (Nonlinear Krylov SOLver for nonlinear systems of equations) is nevertheless interesting. The scheme with the dogleg globalization strategy and GMRES as linear solver requires for the solution of dcp5000: 88 Newton iterations, 1460 function evaluations, 1315 linear system solutions, 0 Jacobian evaluations (matrix free scheme). But, note that there is a lot of differences, e.g. different boundary conditions, different discrete problem formulations, different Jacobian generation and different preconditioning.

3.2.4 Performance analysis

Recall that the affine invariant damping strategy of GIANT requires at least one additional linear system solution per Newton step — compared to the undamped method (or to a "cheap" heuristic damping strategy). Hence, one may analyze the results of the numerical experiments in more detail in order to quantify this additional amount of work. Comparing the number of linear iterations for the computation of the ordinary corrections (#it-lin-ord) with that one needed for the simplified corrections (#it-lin-sim) one can see that the ratio

$$r_s^o := \#\text{it-lin-ord}/\#\text{it-lin-sim} \tag{3.2.1}$$

is within 2 — 6 for most of the experiments. A quite typical value turns out to be 3. This means that only 25% of the total amount of work for linear algebra computations is spent for the damping strategy. Furthermore, due to the special adaptation within GIANT, the inexact simplified Newton correction $\overline{s}_k$ is used as initial guess for the linear iteration towards the next ordinary Newton correction s_k. Thus, dropping the computation of $\overline{s}_k$ would increase the costs for the (iterative) computation of s_k.

The ratio r_s^o strongly depends on two things. First, the required tolerance RTOL for the solution x^* of the nonlinear problem enters as, for the last Newton iterations towards x^*, the required accuracy ε_k^{req} for the ordinary Newton corrections computations gets smaller and smaller — whereas $\overline{\varepsilon}_{k+1}^{req}$ (for simplified Newton corrections) not. Second, the nonlinearity of the problems plays an important role. For highly nonlinear problems, where $\lambda_k < 1$ is really required to ensure the convergence of GIANT, for the Newton steps with $\lambda_k < 1$ the required accuracies ε_k^{req} and $\overline{\varepsilon}_{k+1}^{req}$ may be equal — but of comparatively moderate size (c.f. Figures 3.3, 3.4, B.6).

Just to have some numbers one may compare the standard runs of GIANT (+GBIT1, $\overline{\rho} = 4$) for the basic testset (c.f. Table 3.2) with runs where the required tolerance RTOL is reduced from 10^{-5} to 10^{-8}. There is no change for example sst1 as the standard run is already accurate enough. The most significant change occurs for dcp1000 (coarse grid). Only one additional Newton iteration is required. But the total number of linear iterations is more than doubled. The additional ordinary Newton correction computation costs 652 linear iterations, while the additional simplified correction costs just 19 iterations. These numbers must be compared to the costs for the 10 previous iterations (#it-lin-ord=376, #it-lin-sim=144). The ratio r_s^o changes from $r_s^o = 2.6$ (RTOL=10^{-5}) to $r_s^o = 6.3$.

In order to illustrate the general advantage of the accuracy matching strategy, one may compare the standard runs of GIANT with an experiment where $\varepsilon_k^{req} = \overline{\varepsilon}_{k+1}^{req} = RTOL = 10^{-5}$ is prescribed. Despite of the fact, that the examples of the basic testset are comparatively small, the number of lin-

ear iterations is already increased by factors between 2.2 (for the "simple" example atp1) and 10.6 (for the "challenging" example dcp1000).

In terms of CPU time, the efficiency of GIANT is mainly determined by the user given problem routines FCN, JAC and the routines for the linear system solution ITSOL, PRECON, MULJAC. The GIANT algorithm just tries to find the solution of the nonlinear within as few steps as possible, at which, the required accuracy for the linear system solution is as loose as possible. This is the reason for the absence of CPU times from most of the above tables. Nevertheless, a short look to the CPU time distribution may be interesting. Table 3.14 presents this distribution for some selected test examples and for a sequential (SUN-SPARC1) and a vector (CRAY X-MP) machine. The different capability of vectorization for the different parts of the codes

example	GIANT	J	F	GBIT1	PRECON	MULJAC
percentages for SUN-SPARC1						
atp1	5.9	23.5	12.6	31.0	15.0	12.0
sst2	4.0	23.7	2.7	27.6	22.3	19.7
dcp1000 (31^2)	1.6	10.9	1.1	34.4	27.8	24.2
dcp5000 (63^2)	0.8	6.5	0.6	39.9	27.9	24.3
percentages for CRAY X-MP						
atp1	0.5	34.6	8.8	2.0	36.0	18.1
sst2	0.2	38.3	1.0	1.3	37.2	22.0
dcp1000 (31^2)	0.1	19.9	0.6	1.9	46.7	30.8
dcp5000 (63^2)	0.1	12.3	0.3	1.9	51.5	33.9

Table 3.14 percentage CPU of total CPU for special parts

is nicely reflected. Only GIANT and GBIT1 vectorize quite good, whereas for MULJAC and FCN the innermost loops are too short and in regard of PRECON and JAC the vectorization may even increase the average CPU time per call. Observe that the CPU time summarized within J includes the setup of the preconditioner, i.e. the incomplete LU factorization. Furthermore, note that the implementation of FCN and JAC is extremely inefficient, as these routines are designed to evaluate discretized PDE-problems on general rectangular grids. In a similar way, the SLAP routines PRECON and MULJAC suffer from their generality. So, in order to combine advantages of GIANT with an efficient (fast) solution of a specific problem class, a special adaptation of all user routines, not only to the problem class but also to the target machine, is strongly recommended.

The final experiment reported in this paper deals with the use of GIANT / GBIT1 as a so called matrix-free method. The way how to realize such a technique, as well as the additional problems coming from the additional

approximation error due to (2.3.1), have been discussed in Section 2.3. The possible advantages and disadvantages (CPU time, storage) of such a technique will again strongly depend on the implementation of the routines FCN, JAC, PRECON, MULJAC (see above) and, furthermore, on the chance of finding a "Jacobian-free", but good, preconditioner. Therefore, the following experiment concentrates on checking the performance of GIANT+GBIT1 having introduced this new type of inexactness.

First, for the simple example atp1 and for four variants of the more critical example sst2 (26^2,51^2-grid, $\overline{\rho} = 4, 40$) a straightforward implementation of the matrix free trick (2.3.1) is used to replace the usual MULJAC routine. This means, the perturbation parameter σ of (2.3.1) is just set to a fixed, absolute value. Note that the perturbed vector is $x + \sigma v$, and σv can't be directly interpreted as a relative perturbation of x. This is an essential difference to the standard case of numerical differentiation, where the j-th column of the Jacobian is explicitly generated, e.g. by

$$\begin{aligned} J_j &:= \frac{F(x + \varepsilon x_j) - F(x)}{\varepsilon x_j} \\ x_j &:= j\text{-th component of } x \\ \varepsilon &:= \text{(relative) perturbation parameter} \\ (&\approx \sqrt{\text{epmach}} \cdot \text{safety\_factor.}) \end{aligned}$$

This usual choice of ε reflects the fact, that too small values for ε would increase the approximation error due to cancellation of leading digits, whereas too large values for ε would increase the discretization error.

In order to balance the magnitude of x and $x + \sigma v$ in a similar way, one may use a simple adaptive strategy. For each evaluation of (2.3.1) the parameter σ_{adapt} is chosen in such a way, that

$$\begin{aligned} &\sigma_{adapt}\|v\| = \varepsilon\|x\| \\ &\varepsilon := \text{(relative) perturbation parameter} \\ &(\varepsilon \approx \sqrt{\text{epmach}} \cdot \text{safety\_factor.}) \end{aligned}$$

holds. The results of the experiments with $\sigma_{fixed} = 10^{-1}, 10^{-3}, 10^{-5}$ and σ_{adapt} (for $\varepsilon = 10^{-3}, 10^{-5}, 10^{-7}$) are summarized in Tables 3.15 and 3.16. For all runs block diagonal preconditioning is used.

In both tables, the changes in relation to the associated standard run are given. Again, the performance of the Newton iteration is measured by counting the number of function evaluations (#F), while the total number of linear iterations (#it-lin) is used in order to check the performance of GBIT1. Obviously, the linear iteration is affected more distinct than the nonlinear Newton iteration — especially, regarding the fact, that all fail runs are due to the failure of GBIT1. All these failures occur for the "last" ordinary Newton correction computation near the solution x^*, i.e. with the most stringent accuracy

example	direct	σ_{fixed}			$\sigma_{adapted}/\varepsilon$		
		10^{-5}	10^{-3}	10^{-1}	10^{-7}	10^{-5}	10^{-3}
atp1	5	0	0	+1	0	0	0
sst-26-4	23	fail	0	0	0	0	0
sst-51-4	23	0	0	+11	0	0	+5
sst-26-40	23	fail	fail	-1	fail	0	0
sst-51-40	23	fail	0	0	0	0	+1

Table 3.15 changes of #F for varying $\sigma_{fixed}, \sigma_{adapted}$

example	direct	σ_{fixed}			$\sigma_{adapted}/\varepsilon$		
		10^{-5}	10^{-3}	10^{-1}	10^{-7}	10^{-5}	10^{-3}
atp1	99	-13	-2	+54	+2	+2	+4
sst-26-4	262	fail	+212	-6	+10	+6	+3
sst-51-4	326	-45	+129	+416	+58	-21	+23
sst-26-40	327	fail	fail	-70	fail	-1	+10
sst-51-40	542	fail	-3	+10	-128	-36	-58

Table 3.16 changes of #it-lin for varying $\sigma_{fixed}, \sigma_{adapted}$

requirement prescribed so far. As the approximation error for the matrix-vector product is now in the order of the required accuracy, and furthermore, varies from iteration to iteration, the convergence criterion of GBIT1 is never activated. Hence, the linear iteration stops after having reached the internal *itmax* restriction. A comparison of the results σ_{fixed} versus $\sigma_{adapted}$ clearly shows, that for the variation of $\sigma_{adapted}$ the performance changes are more uniform and less significant. Moreover, the range of non-critical values is larger. Finally, one may compare the average costs (CPU time) for the usual MULJAC routine in contrast to the matrix free realization. For atp1 there is a drastic increase by a factor of 15, as the evaluation of FCN is comparatively expensive, while for the sst examples this factor is about 3. One may consider carefully this loss in CPU time with the gain of reducing the storage requirements by approximately 25%. But, recall that these factors depend on the selected examples, the used preconditioner and the implementation of the associated routines.

4. Conclusion

A new software package for the numerical solution of very large systems of highly nonlinear equations has been presented. Based on the strategy proposed in [4], the new code GIANT realizes a very efficient telescoping of a globally convergent Newton iteration (outer iteration) with iterative methods for the linear system solution (inner iteration). Two of these methods have been applied for the numerical testing of GIANT — the widely used code GMRES and the rather new method GBIT [5]. In comparison to GMRES, the new code GBIT1 shows a quite good performance. This is certainly a result of the fact, that the underlying minimization principle of GBIT1 fits nicely into the theoretical frame of GIANT.

The numerical experiments with GIANT are very encouraging. Obtained on a set of rather diverse test problems, they clearly show, that the advantages of the underlying global exact Newton techniques carry over to the inexact case. Furthermore, the additional costs for the affine invariant globalization strategy are comparatively small. The modular design of the package allows an easy adaptation and modification, or even optimization, of the method. This can be done by selecting special numerical options as well as by replacing the standard routines for the iterative linear solution, preconditioning and matrix-vector multiplication.

Concerning the Newton iteration, GIANT shows a high degree of robustness and efficiency — as long as the required accuracy for the linear system solution is really achieved. Thus, the overall robustness and efficiency of GIANT mainly depend on the quality of the iterative linear solver. The convergence domain and the convergence speed of these methods can be improved drastically by using a good preconditioning technique. But a reasonable error estimator (for the solution) is still rarely available. In order to overcome this difficulty, one may try to develop special stopping criteria within GIANT. This, and other possible improvements of GIANT, will be part of future work.

References

[1] P. N. Brown, Y. Saad: *Hybrid Krylov Methods for Nonlinear Systems of Equations.* Lawrence Livermore National Laboratory, Preprint UCLR-97645, Rev.1 (1988)

[2] P. Deuflhard: *A Relaxation Strategy for the Modified Newton Method.* In Bulirsch/Oettli/Stoer (eds.): Optimization and Optimal Control. Springer Lecture Notes 477, p.59-73 (1975)

[3] P. Deuflhard: *Newton Techniques for Highly Nonlinear Problems - Theory and Algorithms.* Academic Press,Inc. (To be published)

[4] P. Deuflhard: *Global Inexact Newton Methods for Very Large Scale Nonlinear Problems.* Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, Preprint SC 90-2 (1990)

[5] P. Deuflhard, R. Freund, A. Walter: *Fast Secant Methods for the Iterative Solution of Large Nonsymmetric Linear Systems.* IMPACT Comput. Sci. Eng. 2, 244-276 (1990)

[6] P. Deuflhard, F. A. Potra: *Asymptotic Mesh Independence of Newton-Galerkin Methods via a Refined Mysovskii Theorem* Konrad-Zuse-Zentrum fuer Informationstechnik Berlin Preprint SC 90-9 (1990)

[7] A. Greenbaum: *Routines for Solving Large Sparse Linear Systems* Lawrence Livermore Nat. Laboratory, Livermore Computing Center, January 1986 Tentacle, pp 15-21

[8] M. C. MacCracken: *Fist annual report, DOT-CIAP program* Report UCRL-51336, Lawrence Livermore Lab., Livermore, California, February 1973.

[9] S. Murata, N. Satofuka, T. Kushiyama: *Numerical Analysis for Square Cavity Flow Using a New Poisson Solver* Computational Fluid Dynamics, Elsevier Science Publishers B.V. (North-Holland), 1988, p. 547-556.

[10] U. Nowak, L.Weimann: *A Family of Newton Codes for Systems of Highly Nonlinear Equations- Algorithm, Implementation, Application* Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, Technical Report TR 90-10 (1990)

[11] M. Seager: *A SLAP for the Masses* Lawrence Livermore Nat. Laboratory Technical Report, UCRL-100267, December 1988

[12] R. F. Sincovec, N. K. Madsen: *Software for Nonlinear Partial Differential Equations* ACM Transactions on Mathematical Software Vol. 1, No. 3, September 1975, p. 232-260.

A. Program structure diagrams

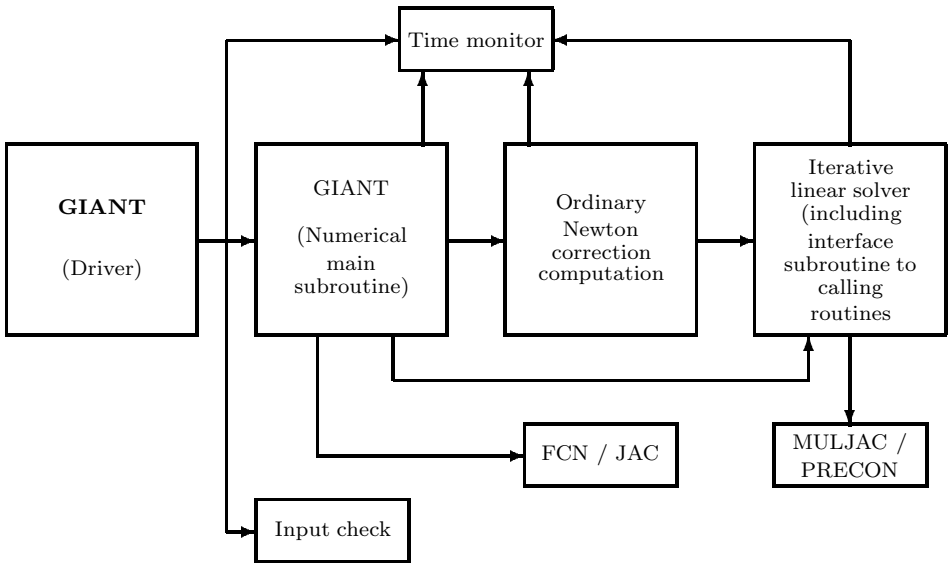


Figure A.1 GIANT: Program structure overview

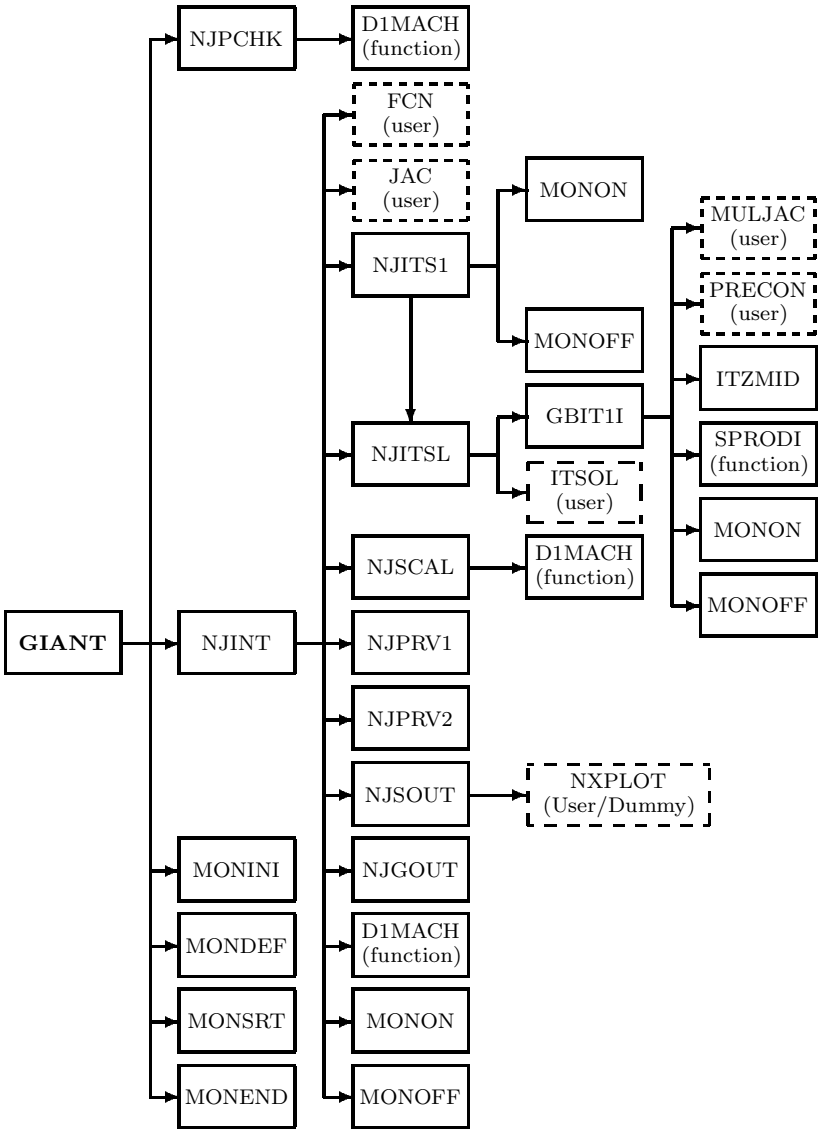


Figure A.2 GIANT with Good Broyden: Program structure (subroutines)

routine	purpose
Interface to the calling program	
GIANT	Generalized Inexact Affineinvariant Newton Techniques - User interface and workspace distribution subroutine
NJPCHK	Checks, if input parameters and options have reasonable values
Internal subroutines, realizing the algorithm (I)	
NJINT	Main core subroutine of GIANT - realizing the inexact Newton scheme
NJITS1	Computation of the inexact ordinary Newton correction including the a priori and the a posteriori iteration
NJSCAL	Calculates the scaling vector for the inexact Newton iteration
NJITSL	The internal interface subroutine to the iterative linear solvers
Iterative linear solver GBIT1 (Good Broyden)	
GBIT1I	The internal core subroutine of the the algorithm GBIT1
ITZMID	Computation of the "mean error" estimation for GBIT1
SPRODI	Evaluation of the scalar product used in GBIT1
Output subroutines	
NJPRV1	Does print monitor output
NJPRV2	Does print monitor output (another format, different data as in NJPRV1)
NJSOUT	Output of the sequence of Newton iterates (or the solution only)
NJGOUT	Special output of miscellaneous monitor data. Used to generate various diagrams
Time monitor	
MONON	Starts a specific time measurement part
MONOFF	Stops a specific time measurement part
MONINI	Initialization call of the time monitor package
MONDEF	Configuration of the time monitor - definition of one specific measurement part
MONSRT	Start of time monitor measurements
MONEND	Finishes time monitor measurements and prints table of time statistics
Machine dependent subroutines	
D1MACH	Returns machine dependent double precision constants
SECOND	Returns a time stamp measured in seconds - used for the time monitor
Routines to be supplied by the user of GIANT	
FCN	The nonlinear problem function
JAC	The Jacobian associated to the nonlinear problem function
MULJAC	Must evaluate the product Jacobian times vector
PRECON	Must perform the preconditioning
ITSOL	The interface subroutine to a user supplied alternative linear solver
NXPLOT	Optional output of the Newton iterates, e.g. in form of graphics.

Table A.1 purpose of GIANT subroutines

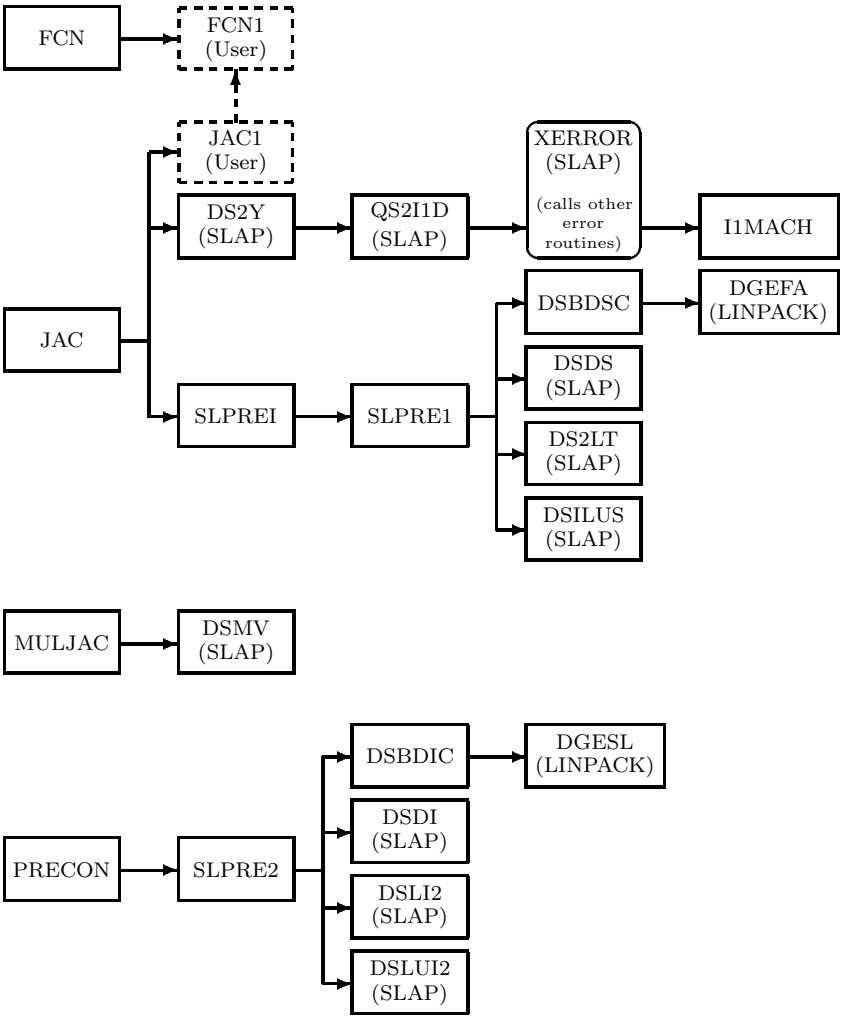


Figure A.3 SLAPInt-package: Program structure (subroutines)

routine	purpose
SLAPInt	
FCN	Problem function interface subroutine to the GIANT/Good Broyden package.
JAC	Jacobian interface subroutine to the GIANT/Good Broyden package.
SLPREI	Workspace splitting subroutine for preconditioner setup routines.
SLPRE1	Preconditioner setup selection subroutine.
MULJAC	"Matrix times vector" interface subroutine to the GIANT/Good Broyden package.
PRECON	Preconditioner interface subroutine to the GIANT/Good Broyden package. Does the setup of the references to the user workspace for the selected preconditioner.
SLPRE2	Preconditioner selection subroutine.
Preconditioners and their setup routines	
DSDS	SLAP — Diagonal scaling preconditioner setup routine. Needs the input matrix to be given in SLAP column format.
DSDI	SLAP — Diagonal scaling preconditioner
DS2LT	SLAP — Lower triangle preconditioner setup routine. Needs the input matrix to be given in SLAP column format.
DSLI2	SLAP — Lower triangle preconditioner
DSILUS	SLAP — Incomplete LU (ILU) decomposition preconditioner setup routine. Needs the input matrix to be given in SLAP column format.
DSLUI2	SLAP — Back-substitution for incomplete LU preconditioning
DSBDSC	Block diagonal scaling preconditioner setup routine
DSBDIC	Block diagonal scaling preconditioner
Further routines from the SLAP package	
DSMV	SLAP — Does the matrix times vector multiplication for a matrix supplied in SLAP column format.
DS2Y	SLAP — Converts SLAP matrix Triad format to column format.
QS2I1D	SLAP — Sort subroutine
XERROR	SLAP — Error Handling Package Top Level Subroutine
XERRWV	SLAP — Internal error handling subroutine, calls further error handling subroutines not extra listed in this Table: J4SAVE, XERABT, XERCTL, XERPRT, XERSAV, XGETUA
Other public domain subroutines	
DGEFA	LINPACK — LU -decomposition of a matrix stored in full mode.
DGESL	LINPACK — Forward -backward substitution using a LU-decomposition generated by DGEFA.
I1MACH	Machine dependent integer constants
Routines to be supplied by the user	
FCN1	User problem function subroutine
JAC1	Jacobian subroutine

Table A.2 purpose of EASYPACK subroutines

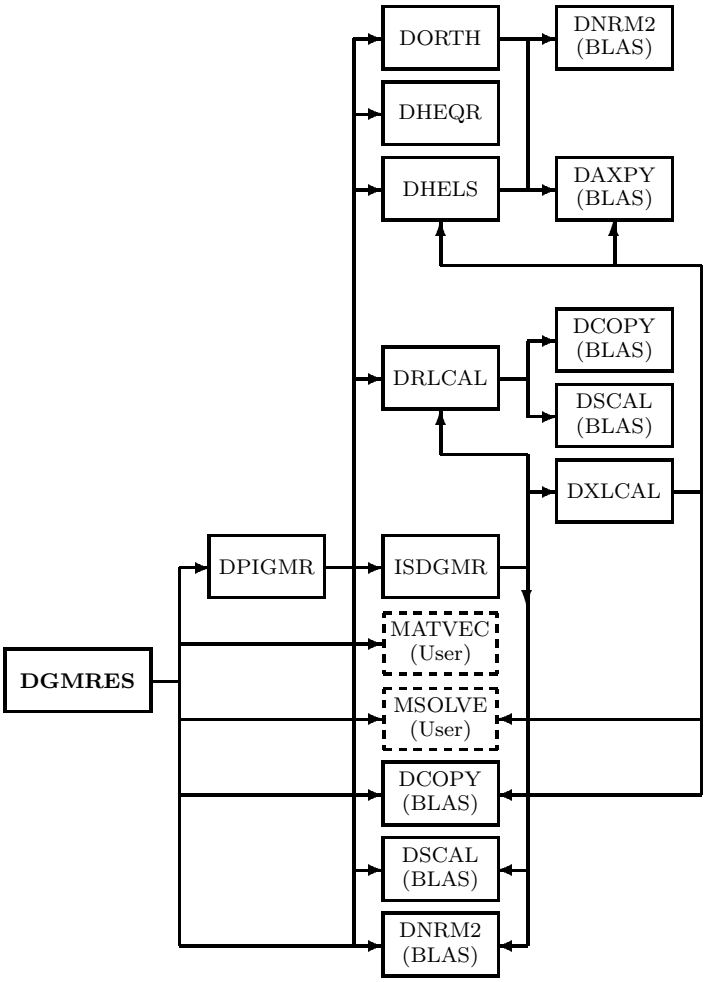


Figure A.4 GMRES: Program structure (subroutines)

routine	purpose
DGMRES	Preconditioned GMRES iterative sparse Ax=b solver. This routine uses the generalized minimum residual (GMRES) method with preconditioning to solve non-symmetric linear systems of the form: A*x = b.
DHELS	Solves the least squares problem: min $< b - A * x, b - A * x >$ using the factors computed by DHEQR. This routine is extraced from the LINPACK routine SGESL with changes due to the fact that A is an upper Hessenberg matrix.
DHEQR	This routine performs a QR decomposition of an upper Hessenberg matrix A using Givens rotations. There are two options available: 1) Performing a fresh decomposition 2) updating the QR factors by adding a row and a column to the matrix A.
DORTH	This routine orthogonalizes the vector VNEW against the previous KMP vectors in the V array. It uses a modified gram-schmidt orthogonalization procedure with conditional reorthogonalization.
DPIGMR	This routine solves the linear system A * Z = R0 using a scaled preconditioned version of the generalized minimum residual method. An initial guess of Z = 0 is assumed.
DRLCAL	This routine calculates the scaled residual RL from the V(I)'s.
DXLCAL	This routine computes the solution XL, the current DGMRES iterate, given the V(I)'s and the QR factorization of the Hessenberg matrix HES. This routine is only called when ITOL=11.
ISDGMR	Generalized Minimum Residual Stop Test. This routine calculates the stop test for the Generalized Minimum RESidual (GMRES) iteration scheme. It returns a nonzero if the error estimate (the type of which is determined by ITOL) is less than the user specified tolerence TOL.
DCOPY	BLAS — Copy a double precision array (vector) of length N.
DNRM2	BLAS — Euclidean norm of the vector v of length N.
DAXPY	BLAS — Compute the linear combination of vectors v,w scalar*v+w and store result to w.
DSCAL	BLAS — compute the vector scalar*v and store result to v.

Table A.3 purpose of DGMRES subroutines

B. GIANT performance for dcp5000

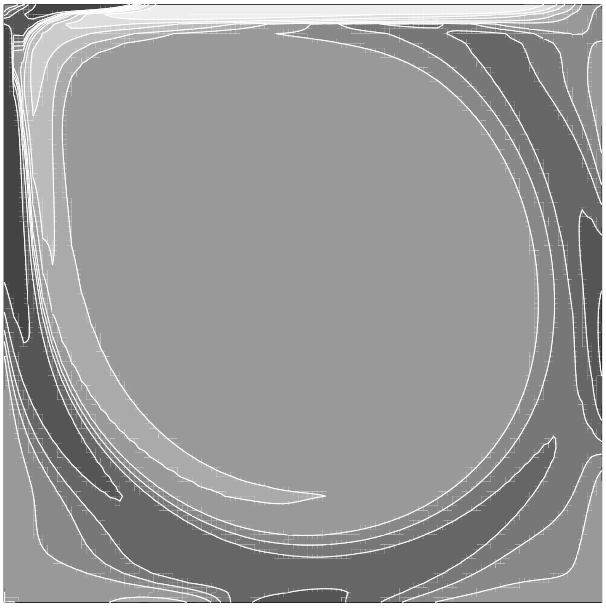


Figure B.1 dcp5000: vorticity ω

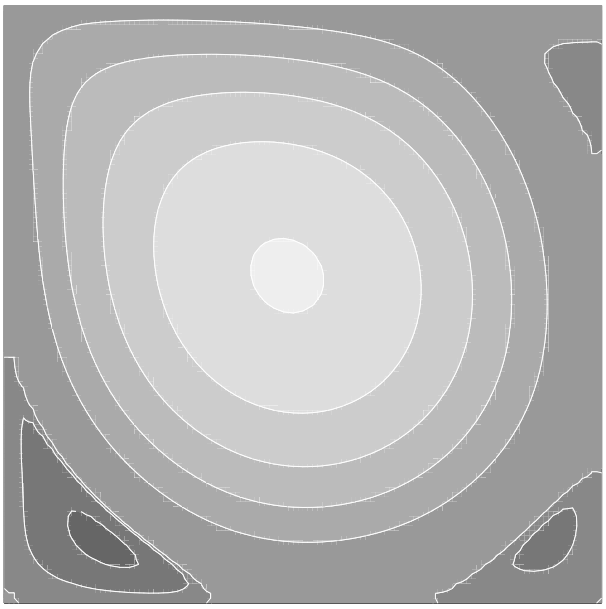


Figure B.2 dcp5000: stream function ψ

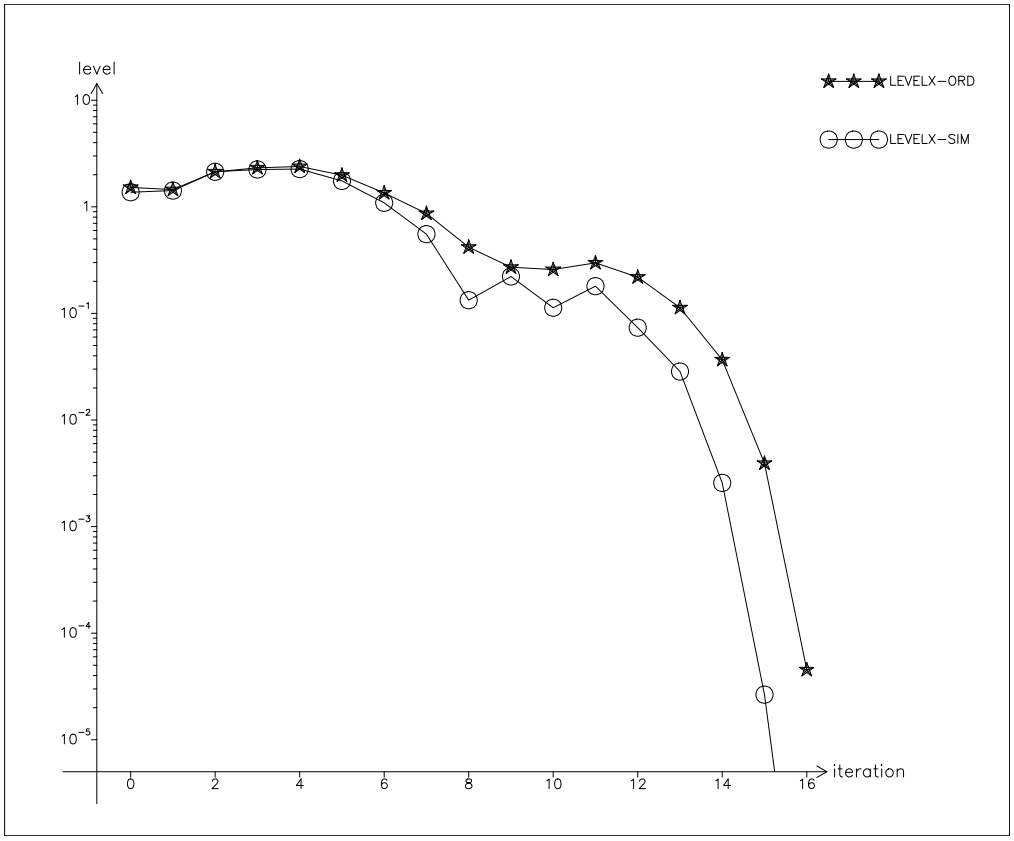


Figure B.3 dcp5000 ($\overline{\rho} = 4$): level functions $\|\Delta x_k\|, \|\overline{\Delta x}_{k+1}\|$

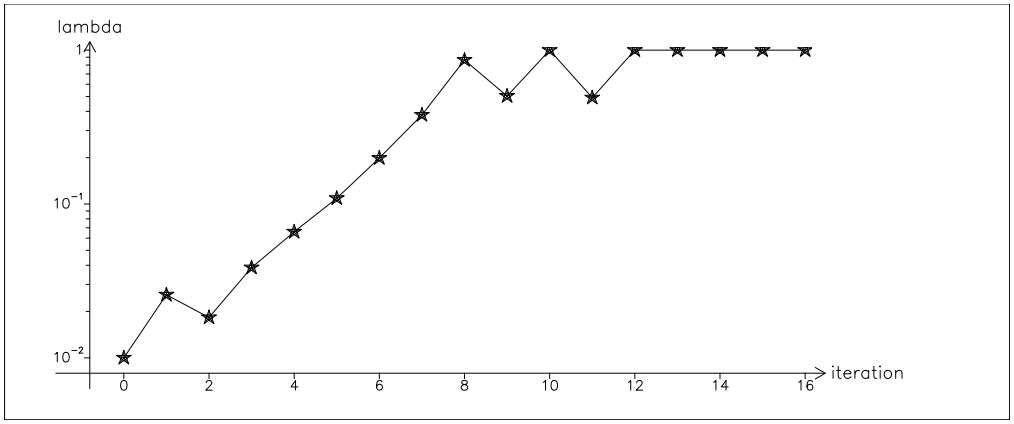


Figure B.4 dcp5000 ($\overline{\rho} = 4$): damping factors λ_k

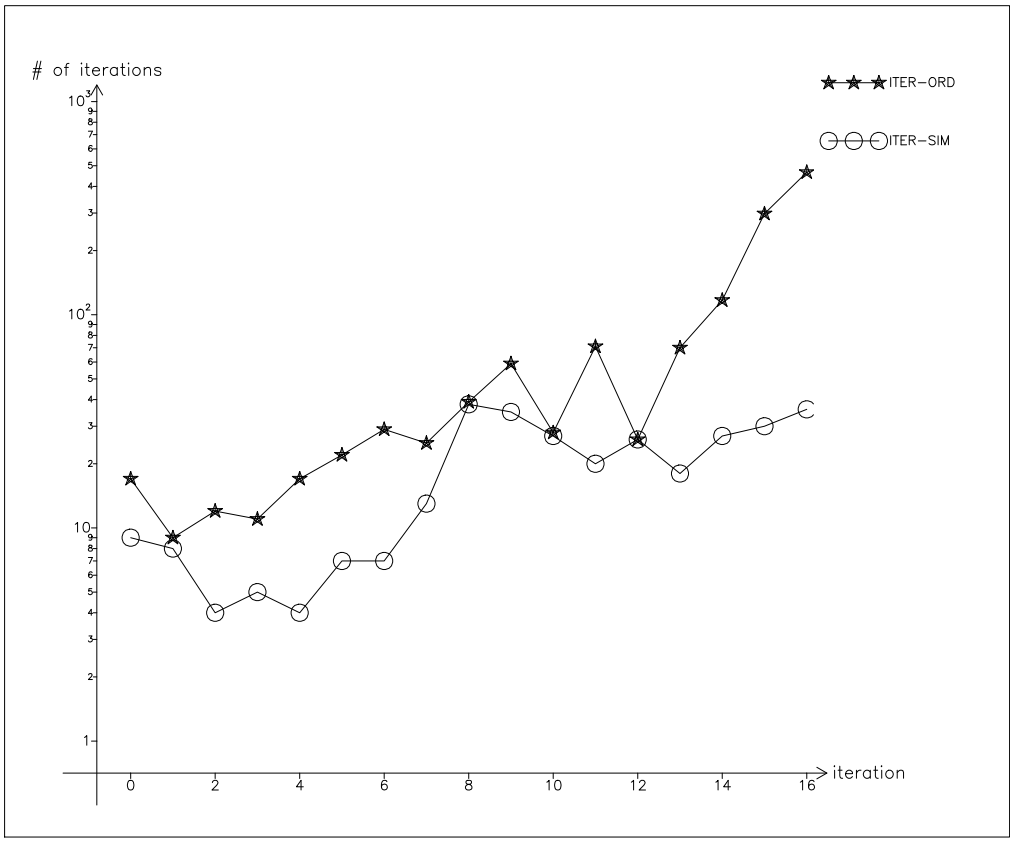


Figure B.5 dcp5000 ($\overline{\rho} = 4$): number of linear iterations

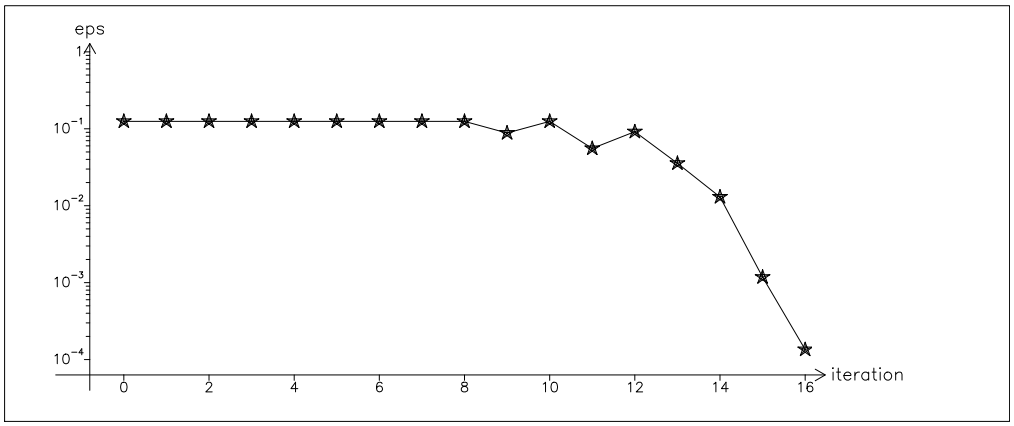


Figure B.6 dcp5000 ($\overline{\rho} = 4$): required accuracy ε_k^{req}

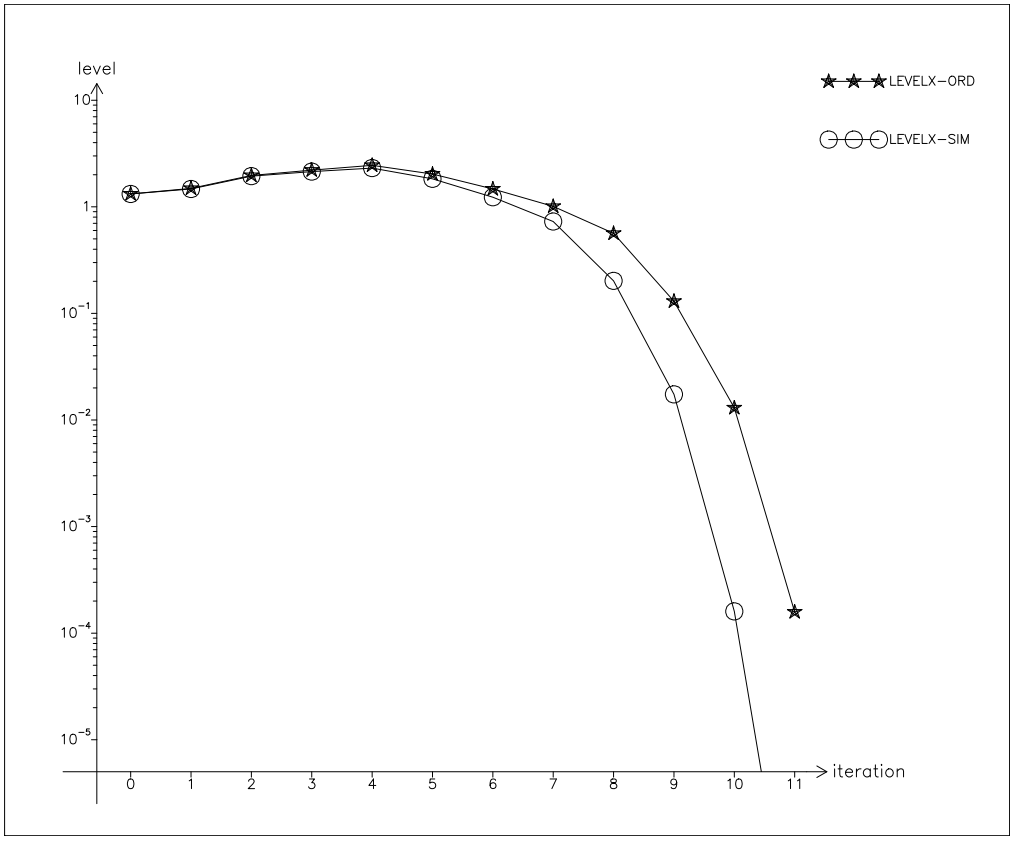


Figure B.7 dcp5000 ($\overline{\rho} = 40$): level functions $\|\Delta x_k\|, \|\overline{\Delta x}_{k+1}\|$

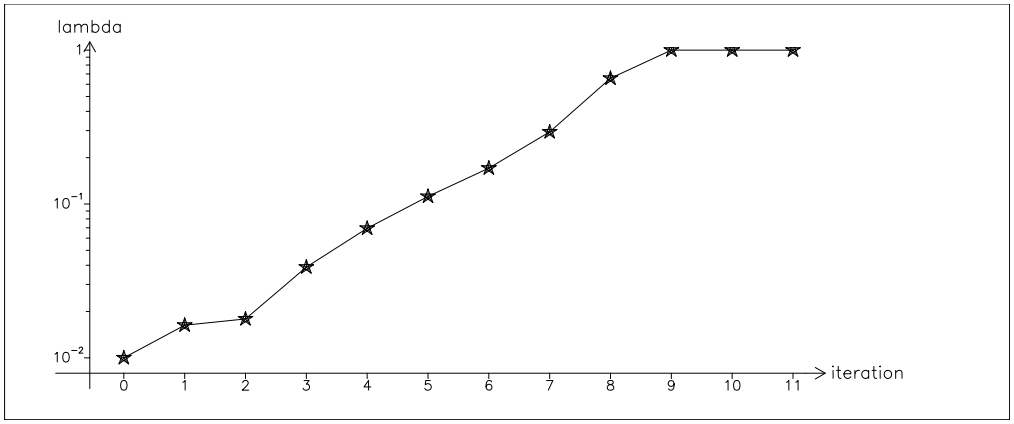


Figure B.8 dcp5000 ($\overline{\rho} = 40$): damping factors λ_k

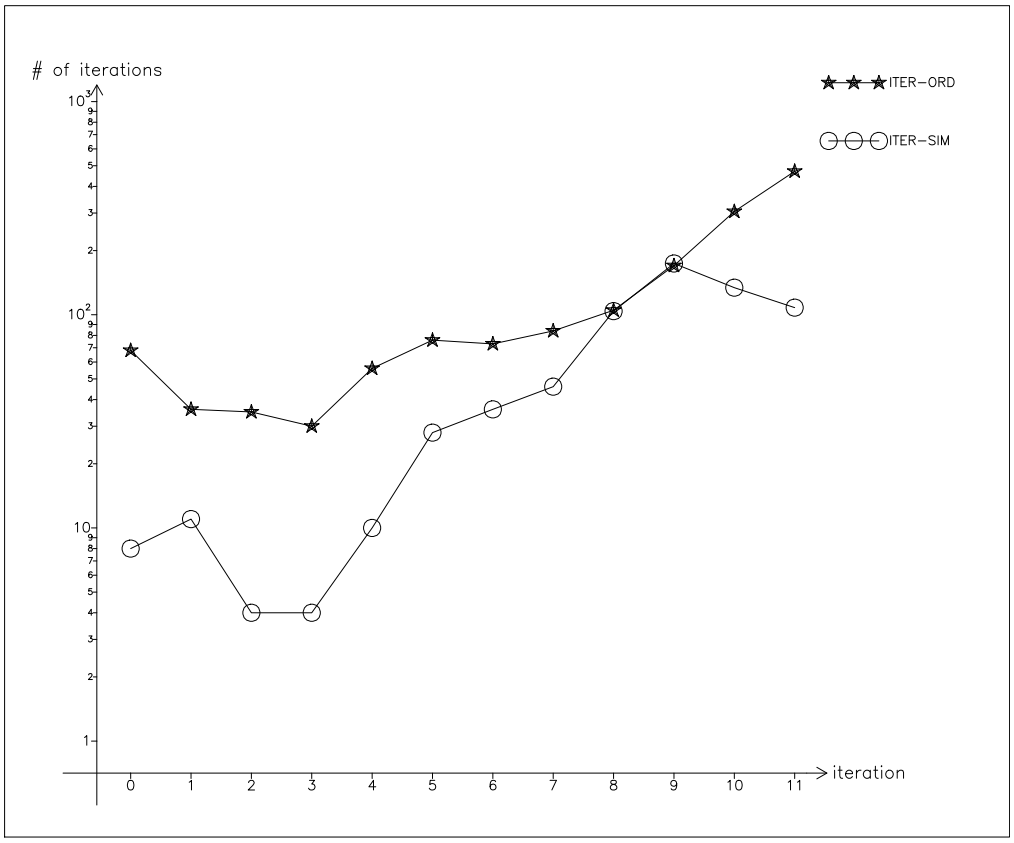


Figure B.9 dcp5000 ($\overline{\rho} = 40$): number of linear iterations

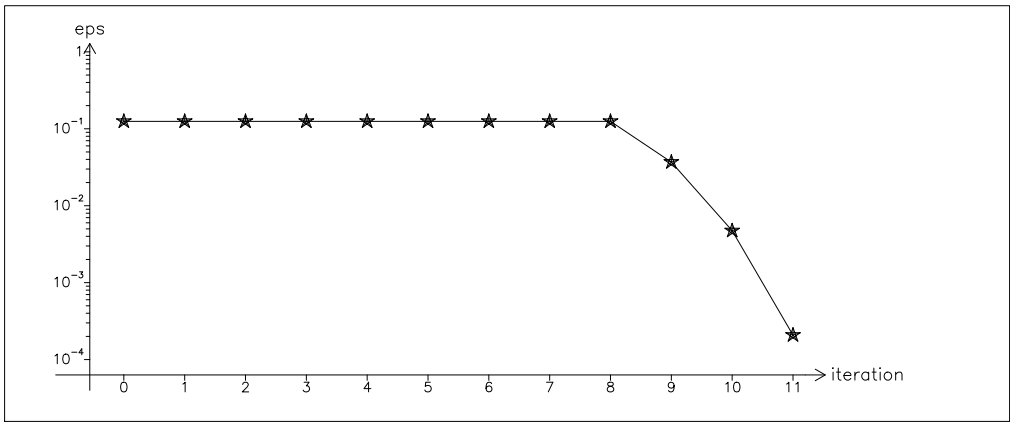


Figure B.10 dcp5000 ($\overline{\rho} = 40$): required accuracy ε_k^{req}

[This page is intentionally left blank]