# Certification of Algorithm 734: A Fortran 90 Code for Unconstrained Nonlinear Minimization

TIM HOPKINS
Computing Laboratory, University of Kent

Algorithm 734: A Fortran 90 Code for Unconstrained Nonlinear Minimization was ported to a number of compiler-platform combinations. The necessary changes to the code are given along with some comparative timings.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software—*certification and testing, efficiency, reliability and robustness*; G.1.6 [**Numerical Analysis**]: Optimization—*gradient methods*

Additional Key Words and Phrases: Fortran 90, nonlinear optimization

Algorithm 734 [Buckley 1994] was published at a time when there were very few Fortran 90 compilers available. We report the changes that we had to make to the code so that it would compile and execute on a number of compiler-platform combinations. We have also improved the readability and maintainability of the code by passing it through a tool that explicitly declares which information has been imported via each `USE` statement.

The compiler-platform combinations used were

(1) Edinburgh Portable Compilers (EPC) Fortran 90 (version 1.1.5.4) on an SGI Indy 4600 SC running Irix 5.3,
(2) Numerical Algorithms Group (NAG) f90 (version 2.1(591)) on an SGI Indy 4600 SC running Irix 5.3,
(3) Edinburgh Portable Compilers (EPC) Fortran 90 (version 1.1.6.1) on a Sun SparcCenter 1000 running Solaris 2.3,
(4) Numerical Algorithms Group (NAG) f90 (version 2.1(555)) on a Sun SparcCenter 1000 running Solaris 2.3,
(5) Craysoft Fortran 90 (version 1.0.4.0) on a Sun SparcCenter 1000 running Solaris 2.3,

Author's address: Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK; email: t.r.hopkins@ukc.ac.uk.

(6) Digital Fortran 90 (version 2.0-1) on a DEC Alpha WS200 running OSF1 version 3.0.

For each compiler we asked for the maximum amount of checking to take place during compilation and at run time. This included, at run time, tests to ensure that variables had all been assigned values before use, array element accesses were within the declared bounds, and pointer references were associated before they were dereferenced.

The following changes were made to the published code:

(1) The module `Precision` was renamed `Precisions` to avoid an illegal name clash with the Fortran 90 intrinsic function. This change affected both the definition of the module and all `USE`s of it.

(2) The interface block for `FName` and `F` in the routines `Evaluate_F` (line 2082) and `Minimize_F` (line 7029) respectively required the additional statement

```
USE Precisions, ONLY: stnd, short
```

(3) The name `Dynamic` was used both as a public parameter value in the module `Min_Codes` and as a module name. The module name (line 6048) and its one `USE` in the module `Minimize_F` (line 6961) were changed to `Dynamic_M`.

(4) The unused label 900 was removed from the routines `Evaluate_F` (line 2507), `Check_Point` (line 4104), `HOv_Multiply` (line 4977) and `Product_Multiply` (line 5979).

(5) In the interface block for the subroutine `FName` in the module `EvaluateF` (line 2082) the intent of the variable `X` was changed to `IN` to be consistent with the definition of `F` in `Minimize_F`.

(6) Entry to `Evaluate_F` with `First = .true.` was used to initialize the counts stored in the `Cts` structure. The value of `Cts%FEvals` was not assigned a value before it was used at line 2338. This `IF` statement needed to be moved after line 2355,

```
IF ( Ev%Derivatives == FirstTest ) dervs = Analytic
```

(7) The variable `Sh%Iteration_Time` was only initialized (line 7335) when `Method = Dynamic` although its value was used when `Method` took other values. The initialization was moved before the `SELECT CASE` statement at line 7284.

(8) In the assignment to `ForceRestart` at line 7790 the variables `Sh%DoForceRestart` and `Sh%RestartPt` might both be unassigned. The variable `Sh%RestartPt` was set to zero just before the `SELECT CASE` statement at line 7284 and `Sh%DoForceRestart` was set to `.TRUE.` just after label 20 (line 7474) in `Minimize_F`.

(9) The Edinburgh compiler required that the backslash character used in the module `Strings` be escaped, i.e.,

```
Backslash   = '\\'  ,   &
```

This change was allowed by all the compilers tested. In fact the variable `Backslash` was never used in the minimization code and could safely have been removed.

(10) In a number of modules, names have been declared `PUBLIC` while their user-defined types have been locally declared as `PRIVATE`. It would be better if both the name and its type were declared `PUBLIC`. This affects a large number of modules and we restrict ourselves to an example:
in module `Supp_Defs` the variable `InitPrVals` should also have its type, `PrintValues`, declared `PUBLIC`.

The following changes were needed to the extensive test program:

(1) All occurrences of `USE Precision` were replaced by `USE Precisions` (see 2 above).

(2) In the subroutine `Functions` in the module `Test_functions` the argument declarations (lines 150–153) were replaced by

```
Integer(short), INTENT(INOUT)     :: IFG
Real (stnd), INTENT(IN)           :: X(:)
Real (stnd)                       :: Work(2*size(x))
Real (stnd), INTENT(OUT)          :: F, G(size(X))
```

to ensure that the `INTENT` definitions agreed with the interface definitions.

(3) At line 692 the assignments to `W6` and `W7` should not take place when `K` is one. The two statements were replaced by

```
IF (K /= 1) THEN
  W6 = W8
  W7 = WORK(I2+K)
ENDIF
```

(4) The unused labels 1100, 1300, 2100, 2600, 3400, 3500, 3600, 3700, 5000 and 5100 were removed from subroutine `FUNCTIONS`. Each label was inserted as a comment just before the originally labelled statement for navigational purposes.

Finally the source code was processed by the NAG Fortran 90 tool *nag_ustan90* [Numerical Algorithms Group 1995]. This tool changes `USE` statements, wherever possible, to the form

```
USE module_name, ONLY: entity-list
```

This ensures that the source of all names used within a module is easily determined. The only drawback from using this tool is that there are no options available for preserving the mixed upper and lower case letters used for variable names; we decided to translate all variable names into lower case.

### Timings

With the above changes the code compiled and executed without any warning messages on all the compiler-platform combinations detailed above. (The NAG compiler generated a number of information messages reporting constant expressions in a number of `IF` statements in the test program; these are used to set tolerance values based on the precision of the arithmetic being used.)

Table 1 gives the time taken to compile the source code (excluding the driver program and associated modules) with all checks switched on and for maximum code optimization, along with the corresponding execution time for the main test program.

| Compiler | Platform | Checking | | Optimizing | |
|---|---|---|---|---|---|
| | | Compilation | Execution | Compilation | Execution |
| EPC | SGI Indy | 11.0 | 26.6 | 33.7 | 22.3 |
| NAG | SGI Indy | 16.5 | 54.7 | 27.7 | 26.8 |
| EPC | Sun SS 1000 | 21.0 | 147.0 | 59.0 | 56.0 |
| NAG | Sun SS 1000 | 99.0 | 204.0 | 92.0 | 56.0 |
| Craysoft | Sun SS 1000 | 208.0 | 50.0 | 252.2 | 43.0 |
| Digital | DEC Alpha | 22.7 | 49.9 | 37.3 | 10.5 |

Table 1. Compilation and Execution Timings

## REFERENCES

BUCKLEY, A. G. 1994. Algorithm 734: A Fortran 90 code for unconstrained nonlinear minimization. *ACM Transactions on Mathematical Software 20*, 3 (Sept.), 354–372.

Numerical Algorithms Group. 1995. *Nagware f90 Tools (Unix)* (First ed.). Oxford, UK: Numerical Algorithms Group.