IMM
DEPARTMENT OF MATHEMATICAL MODELLING

Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. MINF
21.12.2000
HBN/ms

UCMINF – AN ALGORITHM FOR UNCONSTRAINED, NONLINEAR OPTIMIZATION

Hans Bruun Nielsen

TECHNICAL REPORT

IMM-REP-2000-19

IMM

UCMINF – AN ALGORITHM FOR UNCONSTRAINED, NONLINEAR OPTIMIZATION

Hans Bruun Nielsen

Contents

1. Introduction

This report documents the implementation in Matlab and Fortran77 of an algorithm for unconstrained, non-linear optimization. More specifically, for finding a (local) minimizer

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathcal{D}} \{F(\mathbf{x})\} \ , \tag{1.1}$$

where $F : \mathbb{R}^n \mapsto \mathbb{R}$ is a given, continuously differentiable function, and $\mathcal{D}$ is some region around $\mathbf{x}^*$. For reference we introduce the *gradient*

$$\mathbf{g}(\mathbf{x}) = \mathbf{F}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{bmatrix} \tag{1.2}$$

and the *Hessian*

$$\mathbf{H}(\mathbf{x}) = \mathbf{F}''(\mathbf{x}) \quad \text{with} \quad (\mathbf{H})_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j} \ . \tag{1.3}$$

The algorithm is of quasi-Newton type with BFGS updating of the inverse Hessian and soft line search with a trust region type monitoring of the input to the line search algorithm, see e.g. Chapters 3, 4 and 8 in [7]. The algorithm is outlined below. ε_1, ε_2 and $\nu_{\max}$ are given by the user.

$$\begin{array}{l}
\mathbf{x} := \mathbf{x}_0; \quad \mathbf{D} := \mathbf{D}_0; \quad \Delta := \Delta_0; \quad \nu := 0 \\
\textbf{repeat} \\
\quad \mathbf{h} := \mathbf{D} * (-\mathbf{F}'(\mathbf{x})); \\
\quad \textbf{if } \|\mathbf{h}\| > \Delta \textbf{ then } \mathbf{h} := (\Delta/\|\mathbf{h}\|) * \mathbf{h} \\
\quad [\alpha, \nu] := \mathit{linesearch}(\mathbf{x}, \mathbf{h}, \nu) \\
\quad \mathbf{x}_\mathrm{p} := \mathbf{x}; \quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{h} \\
\quad \text{Update } \Delta \text{ and } \mathbf{D} \\
\textbf{until} \ \ \|\mathbf{F}'(\mathbf{x})\|_\infty \le \varepsilon_1 \\
\qquad\quad \textbf{or } \|\mathbf{x}-\mathbf{x}_\mathrm{p}\|_2 \le \varepsilon_2(\|\mathbf{x}\|_2+\varepsilon_2) \\
\qquad\quad \textbf{or } \nu \ge \nu_{\max}
\end{array} \tag{1.4}$$

The vector $\mathbf{x}$ and matrix $\mathbf{D}$ are the current approximations to $\mathbf{x}^*$ and the inverse Hessian, $\mathbf{D} \simeq [\mathbf{H}(\mathbf{x})]^{-1}$, and Δ is the current "trust region" radius. In Section 2 we give some details about the steps involved in the algorithm and Section 3 reports experiments performed to fix some heuristic parameters. In Section 4 we compare with competing programs, and Sections 5 and 6 are User's guides to respectively the MATLAB and Fortran77 implementation.

2. Algorithmic Details

Any robust algorithm for nonlinear optimization involves a number of heuristic parameters, involved e.g. in updating certain parameters. In the development of `ucminf` we experimented with nine parameters, and in the presentation we name them $\mathbf{p} = [p_1, \ldots, p_9]$.

2.1. BFGS Updating

Let $\mathbf{x}$ and $\widetilde{\mathbf{x}}$ denote two consecutive iterates, and $\mathbf{D} \simeq [\mathbf{H}(\mathbf{x})]^{-1}$. Further, let

$$\mathbf{h} = \widetilde{\mathbf{x}} - \mathbf{x}, \quad \mathbf{y} = \mathbf{F}'(\widetilde{\mathbf{x}}) - \mathbf{F}'(\mathbf{x}), \quad \rho = \frac{1}{\mathbf{h}^\top \mathbf{y}} \ . \tag{2.1}$$

The BFGS update can be written in the form [7, (8.16)]

$$\widetilde{\mathbf{D}} = \left(\mathbf{I} - \rho \mathbf{h}\mathbf{y}^\top\right) \mathbf{D} \left(\mathbf{I} - \rho \mathbf{y}\mathbf{h}^\top\right) + \rho \mathbf{h}\mathbf{h}^\top \ . \tag{2.2}$$

This formulation shows that if $\mathbf{D}$ is symmetric, then also $\widetilde{\mathbf{D}}$ is symmetric, and for any $\mathbf{z} \in \mathbb{R}^n$ it holds that

$$\mathbf{z}^\top \widetilde{\mathbf{D}}\, \mathbf{z} = \mathbf{w}^\top \mathbf{D}\, \mathbf{w} + \rho\alpha^2 \ ,$$

where $\alpha = \mathbf{h}^\top \mathbf{z}$ and $\mathbf{w} = \mathbf{z} - \rho\alpha\mathbf{y}$. Therefore, if $\mathbf{D}$ is positive definite, $\rho > 0$ and $\mathbf{z} \neq \mathbf{0}$, then both terms in the above right-hand side are nonnegative and at least one of them is strictly positive. This shows

that also $\widetilde{\mathbf{D}}$ is positive definite. Therefore, if $\mathbf{D}_0$ is symmetric and positive definite, and the update is performed only if the *curvature condition* $\mathbf{h}^\top \mathbf{y} > 0$ is satisfied, then these properties are preserved throughout the iteration, implying that

$$\mathbf{h}^\top \mathbf{F}' = -\mathbf{F}'^\top \mathbf{D}\, \mathbf{F}' < 0 \ . \tag{2.3}$$

In other words, the direction given by $\mathbf{h}$ is downhill, cf. [7, Section 2.2].

It is possible to give the algorithm an initial approximation $\mathbf{D}_0$, which must be symmetric and positive definite. The first of these conditions is straightforward to check and the seconding is checked by performing a symmetric LU-factorization of $\mathbf{D}_0$ and controlling that all diagonal elements in $\mathbf{U}$ are significantly positive.

In the case where $\mathbf{D}_0$ is not given explicitly, a simple starting guess is found by letting

$$\mathbf{D}_0 = \mathbf{I} \ . \tag{2.4a}$$

A more sophisticated initialization [7, pp 200 – 201] is obtained by letting

$$\mathbf{D}_0 = \frac{\mathbf{h}^\top \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \mathbf{I} \ , \tag{2.4b}$$

This scaling Wshould be performed between the first line search and the first BFGS updating. In the experiments of Section 3 the parameter p_1 decides the choice between these two initializations.

The formulation (2.2) is not suited for direct implementation, since it requires two matrix-matrix multiplications. We introduce

$$\mathbf{u} = \mathbf{D}\mathbf{y}, \qquad \mathbf{v} = \tfrac{1}{2}(1+\rho\mathbf{u}^\top\mathbf{y})\mathbf{h} - \mathbf{u} \ , \tag{2.5a}$$

and (exploiting that $\mathbf{D}$ is symmetric) we see that (2.2) can be rearranged as follows,

$$\begin{aligned}\widetilde{\mathbf{D}} &= \left(\mathbf{D} - \rho\mathbf{h}\mathbf{u}^\top\right)\left(\mathbf{I} - \rho\mathbf{y}\mathbf{h}^\top\right) + \rho\mathbf{h}\mathbf{h}^\top \\ &= \mathbf{D} - \rho\left(\mathbf{h}\mathbf{u}^\top + \mathbf{u}\mathbf{h}^\top\right) + \rho(1 + \rho\mathbf{u}^\top\mathbf{y})\mathbf{h}\mathbf{h}^\top \\ &= \mathbf{D} + \rho\left(\mathbf{h}\mathbf{v}^\top + \mathbf{v}\mathbf{h}^\top\right) \quad .\end{aligned} \tag{2.5b}$$

2.2. Line Search

A consequence of (2.3) is that there exists an $\overline{\alpha}$ such that

$$\varphi(\alpha) \equiv F(\mathbf{x} + \alpha\mathbf{h}) < F(\mathbf{x}) \quad \text{for} \quad 0 < \alpha < \overline{\alpha} \; .$$

We want to find an $\alpha \in]0, \overline{\alpha}]$ so that $F(\mathbf{x}+\alpha\mathbf{h})$ is "sufficiently" smaller than $F(\mathbf{x})$. To do that we use soft line search [7, Chapter 3]. We have experimented with two definitions (picked by the parameter p_2) of "sufficiently smaller", viz. the *Wolfe conditions,*

$$F(\mathbf{x} + \alpha\mathbf{h}) \le F(\mathbf{x}) + p_3\varphi'(0)\cdot\alpha \tag{2.6a}$$

combined with

$$\varphi'(\alpha) \ge p_4\varphi'(0) \; , \tag{2.6b}$$

and the *strong Wolfe conditions,* where (2.6a) is combined with

$$|\varphi'(\alpha)| \le |p_4\varphi'(0)|. \tag{2.6c}$$

The line search algorithm is formulated below. The variable ν counts the number of function evaluations with an upper bound given in p_5 and $\sigma = \varphi'(\alpha)/\varphi'(0)$ is used in the algorithm for updating the "trust region" radius, see Section 2.3.

$$[\alpha, F_n, \mathbf{g}_n, \nu, \sigma] = SLS(\mathbf{x}, \mathbf{h}, F, \mathbf{g}, \mathbf{p}) \tag{2.7a}$$

begin
  $\alpha := 0$; $F_n := F$; $\nu := 0$; $\sigma := 1$; $\varphi'_0 := \mathbf{h}^\top\mathbf{g}$
  if $\varphi'_0 \ge 0$ **then return**
  stop := **false**; ok := **false**; $\varphi_0 := F$; $\beta := 1$
  while not stop **do**
    $F_\beta := F(\mathbf{x} + \beta\mathbf{h})$; $\mathbf{g}_\beta := \mathbf{F}'(\mathbf{x} + \beta\mathbf{h})$; $\nu := \nu + 1$
    $\varphi'_\beta := \mathbf{h}^\top\mathbf{g}_\beta$
    if $F_\beta \le \varphi_0 + p_3\varphi'_0 \cdot \beta$ **then**
      if p_2 **and** $\varphi'_\beta > |p_4\varphi'_0|$ **then** stop := **true**
      else
        $\alpha := \beta$; $F_n := F_\beta$; $\mathbf{g}_n := \mathbf{g}_\beta$;
        ok := **true**; $\sigma := \varphi'_\beta/\varphi'_0$
        if $\nu < p_5$ **and** $\beta < p_6$
          and $\varphi'_\beta < p_4\varphi'_0$ **then** $\beta := 2 * \beta$
        else stop := **true**
      end
    else stop := **true**
  end
  stop := ok **or** $\nu \ge p_5$
  while not stop **do**
    $\gamma := INTP(\alpha, \beta, F_\alpha, \varphi'_\alpha, F_\beta)$
    $F_\gamma := F(\mathbf{x} + \gamma\mathbf{h})$; $\mathbf{g}_\gamma := \mathbf{F}'(\mathbf{x} + \gamma\mathbf{h})$; $\nu := \nu + 1$
    $\varphi'_\gamma := \mathbf{h}^\top\mathbf{g}_\gamma$
    if $F_\gamma \le \varphi_0 + p_3\varphi'_0 \cdot \gamma$ **then**
      $\alpha := \gamma$; $F_n := F_\gamma$; $\mathbf{g}_n := \mathbf{g}_\gamma$
      $\sigma := \varphi'_\gamma/\varphi'_0$; ok := **true**
    else
      $\beta := \gamma$; $F_\beta := F_\gamma$; ok := **false**
    end
    if p_2 **then** ok := *ok* **and** $|\mathbf{h}^\top\mathbf{g}_\gamma| \le |p_4\varphi'_0|$
      else ok := *ok* **and** $\mathbf{h}^\top\mathbf{g}_\gamma > p_4\varphi'_0$
    stop := ok **or** $\nu \ge p_5$ **or** $\beta \le \alpha$
  end
end

In the refinement part of the algorithm (the second **while** loop) we approximate the variation of φ by a parabola,

$$\varphi(\alpha+\tau) \simeq \psi(\tau) \equiv F_\alpha + \varphi'_\alpha\tau + A\left(\frac{\tau}{\beta-\alpha}\right)^2$$

with A determined so that $\psi(\beta-\alpha) = F_\beta$. The algorithm can be expressed in the form

$\gamma := INTP(\alpha, \beta, F_\alpha, \varphi'_\alpha, F_\beta)$ (2.7b)
begin
  $\delta := \beta - \alpha; \quad A := F_\beta - F_\alpha - \delta\varphi'_\alpha$
  if $A \geq 5n\varepsilon_{\mathrm{M}}\beta$ **then**
    $c := \alpha - \frac{1}{2}\varphi'_\alpha\delta^2/A$
    $\gamma := \min\{\max\{\alpha+0.1\delta,\ c\}, \beta-0.1\delta\}$
  else
    $\gamma := (\alpha+\beta)/2$
  end
end

If A is strictly positive (ε_{M} is the unit roundoff on the computer), then a minimizer for ψ exists, and if this is in the mid-80% of the interval, this value is returned. For a non positive value of A we use bisection. The restriction to the middle 80 % is made to ensure that the interval $[\alpha, \beta]$ is shortened.

2.3. Updating Δ

If $\mathbf{h}$ has the right length, then one evaluation of F and $\mathbf{F}'$ is sufficient to satisfy the stopping criteria in the line search algorithm. Normally, this is no problem in the final stage of the algorithm, where $\mathbf{h}$ is small and close to a Newton step. In the start of the process, however, where we are far from $\mathbf{x}^*$ and where $\mathbf{D}$ may be a poor approximation to $\mathbf{H}^{-1}$, a "helping hand" may be useful. If $\mathbf{h}$ is too small, we need a number of expansion steps (first **while** loop in (2.7a) before the slope condition (2.6b) or (2.6c) is satisfied. On the other hand, if $\mathbf{h}$ is too large, we need some refinement steps before (2.6a) is satisfied.

We use a special version of a trust-region approach, see (1.4),

$\mathbf{h} := \mathbf{D} * (-\mathbf{F}')$
if $\|\mathbf{h}\| > \Delta$ **then** $\mathbf{h} := (\Delta/\|\mathbf{h}\|) * \mathbf{h}$; redu := **true**
  else redu := **false**

The radius Δ is updated according to the success of the line search, as reported by the values α and $\sigma = \varphi'(\alpha)/\varphi'(0)$. There are the following cases to consider,

$\alpha < 1$: $\|\mathbf{h}\|$ was too large; reduce Δ.

$\alpha = 1$, $\varphi'(\alpha) < 0$ ($\sigma > 0$): The step was accepted, but maybe Δ should be increased.

$\alpha = 1$, $\varphi'(\alpha) > 0$ ($\sigma < 0$): The step was accepted, but maybe Δ should be reduced.

$\alpha > 1$: $\|\mathbf{h}\|$ was too small; increase Δ.

After a number of exploratory experiments we settled for the following strategy,

if $\alpha < 1$ **then** $\Delta := \max\{p_7, \alpha\} * \Delta$
elseif redu **and** $\sigma < p_8$ **then** $\Delta := p_9 * \Delta$ (2.8)

Note that when $\alpha \geq 1$, we update Δ only if the constraint on $\|\mathbf{h}\|$ was active before the line search producing (α, σ).

3. Experiments

We used the test set given in [6]. It consists of 22 different functions. The first 21 functions are basically least squares problems, i.e.

$$F(\mathbf{x}) = \tfrac{1}{2}\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) , \tag{3.1}$$

where $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$. For some of these the size (the values of n and/or m can be varied, and our test set consisted of 34 problems, see Tables 1 and 2 in the next section.

We performed a large number of tests, varying the values of the 9 parameters. With each choice we solved all 34 problems with the stopping criteria in algorithm (1.4) given by

$$\varepsilon_1 = 10^{-6}, \quad \varepsilon_2 = 10^{-12}, \quad \nu_{\max} = 1000 \ .$$

The objective was defined as

$$\Psi(\mathbf{p}) = \sum_{k=1}^{34} \psi_k(\mathbf{p}) \quad \text{with} \quad \psi_k(\mathbf{p}) \equiv \frac{\nu_k(\mathbf{p})}{\min_{\mathbf{q}\in\widehat{P}} \nu_k(\mathbf{q})} \ . \tag{3.2}$$

Here, $\nu_k(\mathbf{p})$ is the number of iteration steps needed to solve problem no. k with the parameter choice $\mathbf{p}$, and $\widehat{P}$ is the union of all the parameter sets tried.

There is no point in finding the minimizer of Ψ with great accuracy, and we settled for the values given by

$$\mathbf{p} = (1,\ 2,\ 0.05,\ 0.995,\ 5,\ 2,\ 0.35,\ 0.7,\ 3) \ . \tag{3.3}$$

Parameters $p_{2:6}$ show that for the line search we use the *strong Wolfe conditions* (2.6) in the form

$$F(\mathbf{x}+\alpha\mathbf{h}) \le F(\mathbf{x}) + 0.05\varphi'(0)\cdot\alpha \quad \text{and} \quad |\varphi'(\alpha)| \le 0.995|\varphi'(0)| .$$

Further, we allow only one doubling of β in (2.7) and at most 5 function evaluations in each line search. For the Δ-updating (2.8) takes the form

if $\alpha < 1$ **then** $\Delta := \max\{0.35, \alpha\} * \Delta$
elseif redu **and** $\sigma < 0.7$ **then** $\Delta := 3 * \Delta$

The experiments indicated that the scaling (2.4b) should not be used. This result came as a surprise, and with starting points closer to the respective solutions we would probably have arrived at a different optimal set of parameters. Below we give results for the behaviour on *Rosenbrock's problem* (given by $k = 7$ in Table 1 below). The parameters are given by (3.3) and $\mathbf{p} = (2,\ 2,\ 0.01,\ 0.990,\ 5,\ 2,\ 0.35,\ 0.7,\ 3)$.

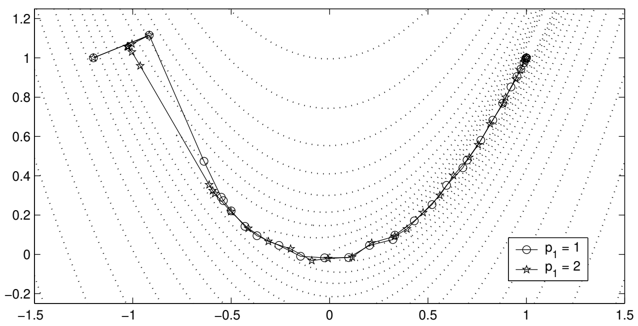

Figure 1. *Performance on Rosenbrock's function*

With $p_1 = 2$ the second step almost brings us down to the local bottom of the valley, and a number of steps is wasted on getting us started again. In the next figure we give the relative errors $\|\mathbf{D}(\mathbf{x}) - \mathbf{H}(\mathbf{x})^{-1}\|_2/\|\mathbf{H}(\mathbf{x})^{-1}\|_2$.

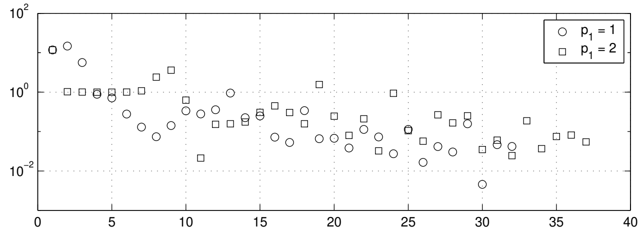

Figure 2. *Relative errors in* **D**

The scaling is seen to improve the accuracy of **D** (second and third set of values in Figure 2), but this does not help in the long run.

4. Comparisons

We have compared the performances of `ucminf` and two other implementations of the BFGS method with line search. We used the stopping criteria in algorithm (1.4) given by

$$\varepsilon_1 = 10^{-8}, \quad \varepsilon_2 = 10^{-10}, \quad \nu_{\max} = 1000$$

and the initial Δ as given in [6].

The first competitor is the Fortran subroutine `MINF` from the package in [3], which is based on the Harwell [2] subroutine `VA13CD`. We ran this with `MAXFUN = 1000` and `EPS = 1D-10`. The other is the MATLAB function `fminunc` from the optimization toolbox, Version 2.0 (R11) 09-Oct-1998. We used the standard options except for the values defined by the MATLAB command

```
optsfmin = optimset('Display','off', 'GradObj','on', ...
  'LargeScale','off', 'MaxFunEvals',900, 'MaxIter',900, ...
  'TolFun',1e-10, 'TolX',1e-10)
```

In Table 1 below we give ν for the first two programs. This is the number of evaluations of F and $\mathbf{F}'$ needed to meet the stopping criteria. The third program uses F-values only in the line search and here we give both ν_F and $\nu_{\mathbf{F}'}$. For all three programs we also give $\eta \equiv F(\mathbf{x}) - F(\mathbf{x}^*)$, where $\mathbf{x}$ is the computed solution and $\mathbf{x}^*$ is the exact minimizer. In some cases η comes out negative because $\mathbf{x}^*$ is stored with 6 decimal digits only.

Is is seen that `ucminf` compares favorably with the other two programs. This might be because it was tuned on this problem set, and as a further check we have performed another comparison on a subset of the 34 problems with other starting points. In Table 2a – 2b we present the new test set and the results are given in Table 3.

Again, in most cases `ucminf` comes out as the fastest, and it is the only one of the three programs that finds the solution to desired accuracy in all test cases. In the cases $k = 13$ and $k = 39$ `fminunc` fails miserably: it is stopped by $\nu_F > \nu_{\max}$, and a closer inspection shows that it got stuck at $\mathbf{x}_0$. Case $k = 41$ is not much better (except that it gives up after very few iterations). Finally, $k = 42$ shows that when $\mathbf{x}_0$ agrees on 2 digits with the solution to the modified Meyer problems, then `fminunc` does solve the problems – but it needs about 3 times as many function evaluations as the other two programs.

Table 1.				`ucminf`		`MINF`		`fminunc`		
k	pno	m	n	ν	η	ν	η	$\nu_{\mathbf{F}'}$	ν_{F}	η
1	1	8	8	3	4.6e-31	4	1.7e-31	3	6	0.0e+00
2		32	16	3	3.6e-15	4	-3.6e-15	3	6	3.6e-15
3	2	8	8	3	-1.1e-16	4	1.1e-15	3	6	4.4e-16
4		32	16	3	-2.4e-14	5	-6.4e-14	3	6	-1.1e-14
5	3	8	8	3	0.0e+00	4	-4.4e-16	3	6	-8.9e-16
6		32	16	3	-4.1e-14	4	-7.1e-15	3	6	1.7e-14
7	4	2	2	38	8.4e-21	46	6.1e-26	27	111	2.3e-12
8	5	3	3	29	2.2e-21	34	5.3e-28	23	91	4.0e-14
9	6	4	4	47	6.4e-13	126	1.3e-38	20	85	1.6e-10
10	7	2	2	12	-1.4e-14	16	2.7e-05	8	29	-3.6e-14
11	8	15	3	21	-8.1e-11	24	-1.4e-09	11	40	-8.1e-11
12	9	11	4	35	-5.4e-15	28	-2.0e-10	18	71	1.0e-11
13	10	16	3	373	-1.8e-05	512	2.8e-05	895	903	8.5e+08
14	11	31	6	48	-2.4e-10	46	2.7e-11	22	92	-2.4e-10
15		31	9	80	-3.9e-10	62	-3.1e-14	34	141	2.6e-06
16		31	12	87	-6.4e-08	104	-5.4e-16	28	117	7.5e-09
17	12	5	3	43	1.4e-16	31	8.5e-32	21	87	5.8e-12
18		10	3	41	6.4e-20	35	2.0e-30	21	83	9.7e-16
19	13	10	2	20	-3.1e-09	23	-8.8e-06	12	34	6.8e+01
20	14	20	4	28	-1.4e-10	33	8.1e-04	17	68	-1.5e-10
21	15	8	8	26	-1.2e-12	47	-3.1e-09	18	75	-9.6e-13
22		16	8	32	-8.3e-12	55	4.5e-08	18	73	-8.3e-12
23		9	9	32	-2.6e-12	53	2.4e-25	16	64	3.8e-11
24		18	9	24	-1.9e-11	61	-5.6e-03	10	36	-1.9e-11
25	16	5	5	14	1.6e-18	16	1.9e-29	7	25	2.8e-11
26		10	10	13	1.1e-19	20	1.5e-26	7	25	1.2e-13
27	17	33	5	67	-4.2e-11	80	-2.6e-11	217	903	2.4e-02
28	18	45	4	163	-5.4e-11	195	-2.3e-08	132	551	1.3e-10
29	19	45	2	26	-2.7e-16	36	-2.3e-08	12	48	1.4e-14
30	20	16	3	257	-3.2e-08	282	2.8e-11	196	816	-3.2e-08
31	21	16	2	23	-6.2e-04	35	2.8e-05	13	51	-6.2e-04
32	22	1	2	9	0.0e+00	12	0.0e+00	6	20	1.1e-16
33		1	5	13	1.1e-16	20	0.0e+00	9	31	1.0e-15
34		1	10	20	0.0e+00	20	0.0e+00	14	51	3.3e-15

Table 2a. *Problems*					
Name	pno	m	n	$\mathbf{x}^*$	$F(\mathbf{x}^*)$
Rosenbrock	4	2	2	$(1,1)$	0
Powellsingular	6	4	4	$\mathbf{0}$	0
Meyer	10	16	3	$(0.005610, 6181, 345.2)$	43.97
Multiexp. fit	18	45	4	$(-4,-5,4,-4)$	0.0050
ModifiedMeyer	20	16	3	$(2.482, 6.181, 3.452)$	$4.397 \cdot 10^{-5}$

Table 2b. *Tests*						
k	pno	m	n	$\mathbf{x}_0$	$F(\mathbf{x}_0)$	Δ_0
7	4	2	2	$(-1.2, 1)$	`1.21e+1`	1
35				$(1.5, 0.6)$	`1.36e+2`	1
36				$(-10,-10)$	`1.36e+2`	1
9	6	4	4	$(3,-1,0,1)$	`1.08e+2`	1
37				$(1,1,1,1)$	`6.10e+1`	1
38				$(5,-5,5,-5)$	`7.66e+4`	1
13	10	16	3	$(0.01, 4000, 250)$	`8.47e+8`	100
39				$(0.0056, 6200, 350)$	`3.04e+7`	10
28	18	45	4	$(-1,-2,1,-1)$	`3.64e-1`	1
40				$(-3,-6,3,-3)$	`2.04e+0`	1
30	20	16	3	$(8.85, 4.00, 2.50)$	`8.47e+2`	1
41				$(2,6,3)$	`1.06e+4`	1
42				$(2.5, 6.2, 3.5)$	`2.67e+1`	1

Table 3.				`ucminf`		`MINF`		`fminunc`		
k	pno	m	n	ν	η	ν	η	$\nu_{\mathbf{F}'}$	ν_{F}	η
35	4	2	2	28	`1.8e-19`	35	`0.0e+00`	9	34	`2.4e-12`
36				104	`2.6e-19`	50	`1.3e-29`	86	360	`2.3e-12`
37	6	4	4	43	`1.3e-12`	118	`4.5e-36`	20	85	`1.7e-10`
38				60	`1.6e-12`	102	`4.5e-35`	28	121	`3.0e-11`
39	10	16	3	114	`-1.8e-05`	85	`-3.3e-02`	896	903	`3.0e+07`
40	18	45	4	121	`-5.4e-11`	136	`-2.4e-13`	9	33	`2.8e-06`
41	20	16	3	462	`-3.2e-08`	116	`-3.2e-08`	3	6	`1.9e+03`
42				49	`-3.2e-08`	45	`-3.2e-08`	33	133	`-3.2e-08`

Except for the case $k = 39$ `MINF` finds the solution, and in some cases ($k = 12, 15, 17, 36, 41$) it is at least 25% faster than `ucminf`. On the other hand, `ucminf` is at least 25% faster than `MINF` in 21 out of the total of 42 cases.

5. Matlab User's Guide

A typical call is

```
[X,info {,perf {,D}}] = ucminf(fun,par,x0,opts {,D0})
```

Input Parameters

`fun` String with the name of the function. Must have the declaration

```
function  [F, g] = fun(x, par)
```

and should return $\mathtt{F} = F(\mathbf{x})$ and (when called with two output arguments) the vector $\mathtt{g} = \mathbf{F}'(\mathbf{x})$.

`par` Parameters of the function. Is transferred to `fun`. May be dummy.

`x0` Starting vector.

`opts` Vector with 4 elements:
`opts(1)` $= \Delta_0$. Default 1.
`opts(2)` $= \varepsilon_1$ in (1.4). Default $10^{-4}\|\mathbf{F}'(\mathbf{x}_0)\|_\infty$.
`opts(3)` $= \varepsilon_2$ in (1.4). Default 10^{-8}.
`opts(4)` $= \nu_{\max}$ in (1.4). Default 100.
Any non positive value is replaced by its default.

`D0` Optional. If present, then approximate inverse Hessian at at $\mathbf{x}_0$; must be symmetric and positive definite.
Otherwise, $\mathbf{D}_0 = \mathbf{I}$.

Output Parameters

`X` If `perf` is present, then array, holding the iterates columnwise. Otherwise, computed solution vector.

`info` Performance information, vector with 6 elements:
`info(1:3)`: Final values of $F(\mathbf{x})$, $\|\mathbf{F}'(\mathbf{x})\|_\infty$, $\|\mathbf{x}-\mathbf{x}_\mathrm{p}\|_2$.
`info(4:5)`: No. of iteration steps and evaluations of $(F, \mathbf{F}')$.
`opts(6)` = 1: Stopped by small gradient.
2: Stopped by small $\mathbf{x}-\mathbf{x}_\mathrm{p}$.
3: Too many function evaluations.
4: Stopped by zero step.

`perf` Optional. If present, then array, holding
`perf(1:2,:)` : Values of $F(\mathbf{x})$ and $\|\mathbf{F}'(\mathbf{x})\|_\infty$
`perf(3:5,:)` : Line search info: values of α, $\varphi'(\alpha)$ and no. of function evaluations.
`perf(6,:)` : "Trust region" radius Δ.

`D` Optional. If present, then array, holding the approximate inverse Hessian at $\mathbf{x} = \mathtt{X(:,end)}$.

Example 5.1. Consider the function given by

$$F(\mathbf{x}) = e^{-x_1-x_2-x_3} + p_1x_1^2 + p_2x_2^2 + p_3x_3^2 \ . \qquad (5.1)$$

This can be implemented in MATLAB as follows,

```
function  [F,g] = fun(x,par)
  e = exp(-sum(x));
  F = e + dot(par,x.^2);
  if  nargout > 1
    g = -e*ones(length(x),1) + 2*par(:).*x(:);
  end
```

We first take $p_j = \frac{1}{2}j^2$ and use the starting point $\mathbf{0}$.

```
opts = [1, 1e-8, 1e-10, 100];
par = .5*[1 4 9];   x0 = [0 0 0];
[x info pf D] = ucminf('fun',par,x0,opts);
x = x(:,end),  info,  D
```

We get the results

$$\mathtt{x} = \begin{matrix} 0.5037546 \\ 0.1259387 \\ 0.0559727 \end{matrix}$$

$$\mathtt{info} = [0.6764583,\ 5.06{\cdot}10^{-11},\ 7.94{\cdot}10^{-9},\ 9,\ 11,\ 1]$$

$$\mathtt{D} = \begin{bmatrix} 0.7009 & -0.0747 & -0.0333 \\ -0.0747 & 0.2313 & -0.0083 \\ -0.0333 & -0.0083 & 0.1074 \end{bmatrix}$$

Thus, the minimizer was found after 9 iteration steps involving a total of 11 function evaluations, and iterations were stopped by the gradient becoming sufficiently small. For comparison, the Hessian is

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} E{+}2p_1 & E & E \\ E & E{+}2p_2 & E \\ E & E & E{+}2p_3 \end{bmatrix} \quad \text{with} \quad E \equiv e^{-x_1-x_2-x_3} \ ,$$

and this leads to

$$[\mathbf{H}(\mathbf{x}^*)]^{-1} = \begin{bmatrix} 0.7012 & -0.0747 & -0.0332 \\ -0.0747 & 0.2313 & -0.0083 \\ -0.0332 & -0.0083 & 0.1074 \end{bmatrix} \ .$$

Thus, the computed $\mathbf{D}$ is a very good approximation to the inverse Hessian at the solution.

Next, we change p_3 from 4.5 to 4.8. We expect this to give a small change in $\mathbf{x}^*$ and take $\Delta_0 = 0.1$ and the computed `x` and `D` to initialize the new minimization. To ensure that $\mathbf{D}$ is symmetric to working accuracy we use $\mathbf{D} := \frac{1}{2}(\mathbf{D} + \mathbf{D}^\top)$.

```
par(3) = 4.8;   opts(1) = .1;
[xp infp] = ucminf('fun',par,x,opts,(D+D')/2)
```

We get the results

$$\mathtt{xp} = \begin{matrix} 0.5048029 \\ 0.1262007 \\ 0.0525836 \end{matrix}$$

$$\mathtt{infp} = [0.6773413,\ 4.16{\cdot}10^{-11},\ 4.25{\cdot}10^{-8},\ 3,\ 4,\ 1]$$

With the command `ucminf('fun',par,x,opts)` (corresponding to the starting value $\mathbf{D}_0 = \mathbf{I}$) it takes 9 function evaluations to find the new minimizer.

6. Fortran User's Guide

The functionality is slightly different from the Matlab function. Partly because of differences between the two languages, partly because the Fortran version was developed to replace `MINF` of [3]. A typical call is

```
CALL UCMINF(FDF,N,X,DX,EPS,MAXFUN,W,IW,ICONTR)
```

The parameters are

FDF `SUBROUTINE` written by the user with the following declaration

```
SUBROUTINE FDF(N,X,G,F)
REAL*8 X(N),G(N),F
```

It must calculate the values of the function and its gradient at the point $\mathbf{x} = [\mathtt{X(1)}, \ldots, \mathtt{X(N)}]$ and store these numbers as follows,

$$\mathtt{F} = F(\mathbf{x}), \qquad \mathtt{G(J)} = \frac{\partial F}{\partial x_{\mathtt{J}}}(\mathbf{x}), \quad \mathtt{J} = 1, \ldots \mathtt{N}$$

The name of this subroutine (which can be chosen freely by the user) must appear in an `EXTERNAL` statement in the calling program.

N `INTEGER`. Number of unknowns, n. Must be positive. Is not changed.

X `REAL*8 ARRAY` with `N` elements. The use depends on the entry value of `ICONTR`.

$\mathtt{ICONTR} > 0$: On entry: Initial approximation to $\mathbf{x}^*$.
On exit: Computed solution.

$\mathtt{ICONTR} \le 0$: Point at which the Jacobian should be checked. Not changed.

DX `REAL*8`. The use depends on the entry value of `ICONTR`.

$\mathtt{ICONTR} > 0$: On entry: Δ_0
On exit: Final value of Δ.

$\mathtt{ICONTR} \le 0$: Gradient check with by finite differences with steplength `DX`, cf. [5]. Must be significantly nonzero. Is not changed.

EPS `REAL*8 ARRAY` with 2 elements. Used only when $\mathtt{ICONTR}_{\text{entry}} > 0$. Desired accuracy. The algorithm stops when $\|\mathbf{F}'(\mathbf{x})\|_\infty \le \mathtt{EPS(1)}$ or $\|\mathbf{x} - \mathbf{x}_{\text{p}}\|_2 \le \mathtt{EPS(2)}(\mathtt{EPS(2)} + \|\mathbf{x}\|_2)$. Must be positive. Is not changed.

MAXFUN `INTEGER`. Used only if $\mathtt{ICONTR}_{\text{entry}} > 0$.

On entry: Upper bound on the number of calls of `FDF` Must be positive.

On exit: Number of calls of `FDF`.

W `REAL*8 ARRAY` with `IW` elements. Work space.

On entry: If $\mathtt{ICONTR}_{\text{entry}} > 2$ then $\mathtt{W(4N+1,\ldots,4N+}\tfrac{1}{2}\mathtt{N(N+1))}$ should hold (an approximation to) the lower triangle of the inverse of $\mathbf{H}(\mathtt{X})$, stored columnwise. Otherwise entry values of `W` are not used.

On exit with $\mathtt{ICONTR}_{\text{entry}} \le 0$: Results of the gradient check are returned in the first 7 elements of `W` as follows, cf. [5]

`W(1)`	Maximum element in $\lvert\mathbf{F}'(\mathtt{X})\rvert$.
`W(2)`, `W(5)`	δ^F and j^F.
`W(3)`, `W(6)`	δ^B and j^B.
`W(4)`, `W(7)`	δ^E and j^E.

In case of an error the indices `W(5,6,7)` point out the erroneous gradient component.

On exit with $\mathtt{ICONTR}_{\text{entry}} > 0$:

$\mathtt{W(1)} = F(\mathtt{X})$, the objective value at the computed minimizer,

$\mathtt{W(2)} = \|\mathbf{F}'(\mathtt{X})\|_\infty$, the largest element in the absolute value of the gradient at `X`,

$\mathtt{W(3)}$ = length of the last step.

$\mathtt{W(4N+1,\ldots,4N+}\tfrac{1}{2}\mathtt{N(N+1))}$: The lower triangle of the final approximation to the inverse Hessian.

`IW` `INTEGER`. Length of work space `W`. Must be at least

$$\begin{array}{rl} \max\{\frac{1}{2}n(n+11), 7\} & \text{if } \texttt{ICONTR}_{\text{entry}} \le 2 \\ n \cdot \max\{n+1, \frac{1}{2}(n+11)\} & \text{if } \texttt{ICONTR}_{\text{entry}} > 2. \end{array}$$

Is not changed.

`ICONTR` `INTEGER`.

On entry: Controls the computation,

`ICONTR` ≤ 0 : Check gradient, cf. [5]. No iteration.

`ICONTR` > 0 : Start minimization with

`ICONTR` ≤ 2 : $\mathbf{D}_0 = \mathbf{I}$. Otherwise, $\mathbf{D}_0$ is given in `W(4N+1,...,4N+`$\frac{1}{2}$`N(N+1))`.

If `ICONTR` $= 2$ or `ICONTR` > 3, then information is printed during the iteration.

On exit: Information about performance,

`ICONTR` $= 1$: Stopped by small gradient.

`ICONTR` $= 2$: Stopped by small step.

`ICONTR` $= 3$: Stopped because too many iterations were needed, see `MAXFUN`.

`ICONTR` $= 4$: Stopped by zero step from line search.

`ICONTR` < 0 : Computation did not start for the following reason,

`ICONTR` $= -2$: `N` ≤ 0

`ICONTR` $= -4$: $|$`DX`$|$ is too small

`ICONTR` $= -5$: `EPS` ≤ 0

`ICONTR` $= -6$: `MAXFUN` ≤ 0

`ICONTR` $= -7$: Given $\mathbf{D}$ is not positive definite.

`ICONTR` $= -8$: `IW` is too small.

Example 6.1. The following program treats the problem from Example 5.1.

```
      IMPLICIT          NONE
      INTEGER           ICONTR,IW,J,MAXFUN,N,DNO
      DOUBLE PRECISION  DX,EPS(2),X(3),W(21),P(3)
      EXTERNAL          FUN
      INTRINSIC         ABS, MAX
```



```
      COMMON  /PROB/  P
      DATA  IW, N,       P,            EPS,      X
     +     /21, 3, .5D0,2D0,4.5D0, 1D-8,1D-10, 3*0D0 /
C
        ICONTR = 1
        MAXFUN = 100
        DX = 1D0
        CALL UCMINF(FUN,N,X,DX,EPS,MAXFUN,W,IW,ICONTR)
        WRITE(6,'(2(A,I3,2X))') 'ICONTR =',ICONTR, 'NEVAL =',MAXFUN
        WRITE(6,'(A,3F12.7)') 'x =', (X(J), J=1,3)
        WRITE(6,'(A,F10.7,3(2X,A,1P1D9.2))') 'F(x) =',W(1),
     +    'max|g| =',W(2), 'last dx =',W(3)
        DNO = 4*N
        WRITE(6,'(A,1P3D12.3/3X,1P3D12.3)') 'D =',(W(DNO+J), J=1,6)
        P(3) = 4.8D0
        DX = .1D0
        ICONTR = 3
        CALL UCMINF(FUN,N,X,DX,EPS,MAXFUN,W,IW,ICONTR)
        WRITE(6,'(2(A,I2,2X))') 'ICONTR =',ICONTR, 'NEVAL =',MAXFUN
        WRITE(6,'(A,3F12.7)') 'x =', (X(J), J=1,3)
        WRITE(6,'(A,F10.7,3(2X,A,1P1D10.2))') 'F(x) =',W(1),
     +    'max|g| =',W(2), 'last dx =',W(3)
        STOP
      END

      SUBROUTINE FUN(N,X,G,F)
      IMPLICIT          NONE
      INTEGER           N, J
      DOUBLE PRECISION  X(N),G(N),F, P(3), E,Y
      INTRINSIC         EXP
      COMMON  /PROB/  P
C
        F = 0D0
        Y = 0D0
        DO  10  J = 1, N
          Y = Y - X(J)
          F = F + P(J) * X(J)**2
   10   CONTINUE
        E = EXP(Y)
        F = F + E
        DO  20  J = 1, N
   20     G(J) = 2D0 * P(J) * X(J) - E
        RETURN
      END
```

From the first call of `UCMINF` we get

```
ICONTR =  1  NEVAL = 11
x =    0.5037546    0.1259387    0.0559727
F(x) = 0.6764583  max|g| = 5.06E-11  last dx = 7.94E-09
D =    7.009E-01  -7.472E-02  -3.328E-02
       2.313E-01  -8.308E-03   1.074E-01
```

By comparison with Example 5.1 we see how the elements in the lower triangle of $\mathbf{D}$ are stored columnwise. Also with the second call the results agree with the MATLAB version.

```
ICONTR = 1  NEVAL = 4
x =    0.5048029    0.1262007    0.0525836
F(x) = 0.6773413  max|g| =  4.16E-11  last dx =  4.25E-08
```

References

[1] J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart. (1988): *An Extended Set of Fortran Basic Linear Algebra Subprograms,* ACM Trans. Math. Soft. **14**, 1–17.

[2] *Harwell Subroutine Library.* (1984). Report R9185, Computer Science and Systems Division, Harwell Laboratory, Oxfordshire, OX11 ORA, England.

[3] K. Madsen, O. Tingleff, P.C. Hansen and W. Owczarz (1990): *Robust Subroutines for Non-Linear Optimization.* Report NI-90-06, Institute for Numerical Analysis (now part of IMM), Technical University of Denmark.

[4] H.B. Nielsen (1999): *Damping Parameter in Marquardt's Method.* Report IMM-REP-1999-05, IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/TR9905.ps.Z

[5] H.B. Nielsen (2000a): *Checking Gradients.* IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/Software/checkgrad.ps

[6] H.B. Nielsen (2000b): *UCTP – Test Problems for Unconstrained Optimization.* Report IMM-REP-2000-17, IMM, DTU. Available at http://www.imm.dtu.dk/~hbn/publ/TR0017.ps

[7] J. Nocedal and S.J. Wright (1999): *Numerical Optimization.* Springer, New York.