EXCEPTION HANDLING IN GLOBAL NEWTON'S ALGORITHM*

S. Fujita, H. Ohkata, T. Ohno, and T. Fukao

Department of Computer Science
Tokyo Institute of Technology

EXTENDED ABSTRACT

As the fields of computer applications expand, software reliability has gained great importance. In the computer systems applied such as nuclear power plants, for example, system failure may have disastrous outcomes.

One of the goal of software reliability is the ability to deal with exceptions. An algorithm/program should make it possible to trap undesired events and to specify suitable response to such events. The behavior of the software with this facility will become predictable even in anomalous situations.

Ada has been designed to support numerical applications and embedded aplications with real-time and concurrent requirements [3]. The purpose of this paper is to assess the exception handling facility in Ada through an experience with MicroAda/SuperMicro(TM).**

GLOBAL NEWTON'S ALGORITHM

The problem chosen as the basis of the assessment is

Problem (*) "Given a mapping $f : R^n \to R^n$, find points $\xi \in R^n$ such that

$$f(\xi) = 0. \qquad (1)$$

Often f is smooth, i.e., $f \in C^r(R^n)$, as we shall henceforth assume."

The algorithm of (the classical) Newton's method to find solutions of (1) consists of

(1) to select an starting approximation: x_0;

(2) to solve the systems of linear equations:

$$J(x_k)\,\Delta x_k = -\,f(x_k); \quad k = 0, 1, \ldots \qquad (2)$$

(3) to improve the next approximation:

$$x_{k+1} = x_k + \Delta x_k. \qquad (3)$$

It is well-known that this algorithm (and its variants) converge to a solution provided that the staring approximation x_0 is sufficiently close to ξ such that $\det\{J(\xi)\} \neq 0$ (local convergence), e.g., [1].

It may happen in the Newton's algorithm that the Jacobian matrices become singular during iterations. An algorithm which will converge for "most" approximations (globally convergent) is more useful. A point of departure from (the classical) Newton's algorithm lies in

(3') to improve the next approximation according to

$$x_{k+1} = x_k + \gamma_k \Delta x_k \qquad (4)$$

i.e., to control γ_k using an available information obtained previous iteration. A method which has some mathematical foundation is to vary γ_k as a function of x_k:

$$\gamma_k = \lambda\ \mathrm{sgn}(\det\{J(x_k)\}), \quad \lambda > 0. \qquad (5)$$

A theoretical result of this algorithm has been discussed in Hirsch and Smale [2]. No (reliable and efficient) software, however, was presented in Hirsch and Smale [2].

ADA PROGRAM WITH EXCEPTION HANDLERS

We present in this paper a sample of Ada program for the global Newton's algorithm. These texts were compiled with MicroAda/SuperMicro(TM).

Flow chart for the global Newton's algorithm is given in Fig. 1. A program for exception handler to detect which matrix is singular is shown in Fig. 2. A preliminary program with the exception handler is demonstrated in Fig. 3.

ACKNOWLEDGEMENT

The authors wish to thank the members of research group: H. Ohno, A. Ohshima, T. Kyozuka, and O. Iwasaki for their discussions and cooperations.

---

* This work was supported in part by the Ministry of Education, Japanese Government, under Grant 56460104.
** MicroAda/SuperMicro(TM) are trade marks of Western Digital Corporation.

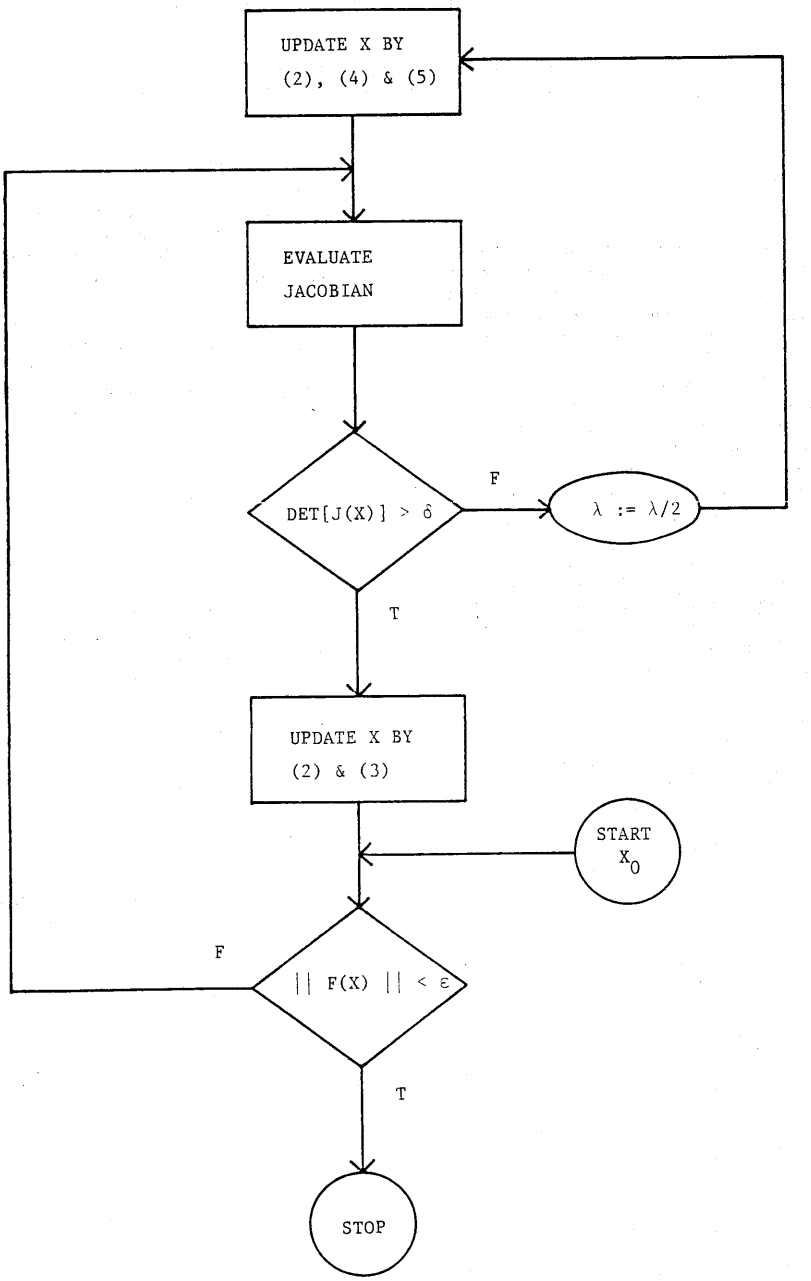

Fig. 1. Flow chart for the global Newton's algorithm

```
 0 > with inrut_outrut,console_io;use inrut_outrut,console_io;
 1 >
 2 > procedure D is
 3 >   N:inteser;
 4 > procedure TREAT_MATRIX(N:inteser) is
 5 >   SINGULAR : exception;
 6 >   type MATRIX is array (1..10,1..10) of float;
 7 >   EPSILON : constant float := 1.0E-5 ;
 8 >   SIZE : inteser;
 9 >   TEMP,DET : float;
10 >
11 > procedure DETERMINANT(SIZE:inteser;SYSTEM:in out matrix;ANS :in out float)  is
12 >     I,J:inteser;
13 >     E_FLAG:boolean;
14 >
15 >
16 >     procedure GAUSS(N:inteser) is
17 >       PIVOT:float;
18 >       I,J,K:inteser;
19 >
20 >       procedure CHANGE(XJ:inteser) is
21 >         II,JJ:inteser;
22 >         T     :float;
23 >         begin
24 >           II:=XJ+1;
25 >           while SYSTEM(II,XJ)=0.0 loop
26 >             II:=II+1;
27 >             if II>N then E_FLAG:=true;
28 >             end if;
29 >           end loop;
30 >           for JJ in 1..N loop
31 >             T:=SYSTEM(XJ,JJ);
32 >             SYSTEM(XJ,JJ):=SYSTEM(II,JJ);
33 >             SYSTEM(II,JJ):=T;
34 >           end loop;
35 >         end CHANGE;
36 >
37 >     begin
38 >       SUB:
39 >         for J in 1..N-1 loop
40 >           for  I in J+1..N loop
41 >             if SYSTEM(J,J)=0.0 then CHANGE(J);
42 >             end if;
43 >             exit SUB when E_FLAG;
44 >             PIVOT:=SYSTEM(I,J)/SYSTEM(J,J);
45 >             for K in 1..N loop
46 >               SYSTEM(I,K):=SYSTEM(I,K)-SYSTEM(J,K)*PIVOT;
47 >             end loop;
48 >           end loop;
49 >         end loop;
50 >       for I in 1..N loop
51 >         ANS:=ANS*SYSTEM(I,I);
52 >       end loop;
53 >     end GAUSS;
54 >
55 > begin
56 >     E_FLAG:=false; ANS:=1.0;
57 >     GAUSS(SIZE);
58 >     if abs(ANS) < EPSILON
59 >       then raise SINGULAR;
60 >     end if;
```

```
<    61 >   end DETERMINANT;
<    62 >
<    63 > procedure PRINT(SYSTEM:MATRIX) is
<    64 >     begin
<    65 >       for I in 1..SIZE loop
<    66 >         for J in 1..SIZE loop
<    67 >           TEMP:=SYSTEM(I,J);
<    68 >           put(TEMP);put(" ");
<    69 >         end loop;
<    70 >         put_line("");
<    71 >       end loop;
<    72 >     end PRINT;
<    73 >
<    74 >
<    75 > procedure TREAT_ONE_MATRIX is
<    76 >   M : MATRIX;
<    77 >
<    78 >   begin
<    79 >   set(SIZE);
<    80 >   for I in 1..SIZE loop
<    81 >     for J in 1..SIZE loop
<    82 >       set(TEMP);M(I,J):=TEMP;
<    83 >     end loop;
<    84 >   end loop;
<    85 >   DETERMINANT(SIZE,M,DET);
<    86 >   PRINT(M);
<    87 >   put_line("");
<    88 >   put(DET);
<    89 >   exception
<    90 >     when SINGULAR => put("MATRIX IS SINGULAR!!");
<    91 >
<    92 > end TREAT_ONE_MATRIX;
<    93 >
<    94 > begin
<    95 >   for I in 1..N loop
<    96 >     TREAT_ONE_MATRIX;
<    97 >   end loop;
<    98 > end TREAT_MATRIX;
<    99 >
<   100 > begin
<   101 >   set(N);
<   102 >   TREAT_MATRIX(N);
<   103 > end D;
<   104 >
```

Fig. 2. Exception handler

```
<     0 > with console_io;use console_io;
<     1 > with text_io;use text_io;
<     2 > with input_output; use input_output;
<     3 >
<     4 > ------------------------------------------------------------------------------
<     5 >
<     6 > procedure newton1 is
<     7 >
<     8 >    type f_arrow is array (0..10) of float;
<     9 >    type matrix is array(0..10,1..11) of float;
<    10 >    Ja:matrix;
<    11 >    fail,buf: precis:float;
<    12 >    guess  :root:f_arrow;
<    13 >    n,i,J : integer;
<    14 >    found:boolean;
<    15 >    file : string (1..20);prin : out_file;
<    16 >    bell : character := character'val(7);
<    17 >    cr : character := character'val(13);
<    18 >    cl : character := character'val(24);
<    19 >    SIZE : INTEGER;
<    20 >    TEMP : FLOAT;
<    21 >
<    22 >      ------------------------------------------------------------------------
<    23 >
<    24 >    package problem is
<    25 >      function f(xif_arrau(i:integer) return float;
<    26 >      procedure Jac(x:f_arrau;Ja:out matrix);
<    27 >    end problem;
<    28 >
<    29 >    package body problem is
<    30 >     function f(xin f_arrau(i:in integer) return float is
<    31 >         begin
<    32 >            if i =1 then
<    33 >              return (x(1)**2-3.0*x(1)*x(2)**2+25.0*(2.0*x(1)**2+x(1)*x(2))
<    34 >                      +x(2)**2+2.0*x(1)+3.0*x(2));
<    35 >            else return(3.0*x(1)**2*x(2)-x(2)**3-25.0*(4.0*x(1)*x(2)-x(2)**2)
<    36 >                      -4.0*x(1)**2+5.0);
<    37 >             end if;
<    38 >         end f;
<    39 >
<    40 >
<    41 >    procedure Jac (x : f_arrau;Ja : out matrix)is
<    42 >       begin
<    43 >          Ja(1,1) := 3.0*x(1)**2-3.0*x(2)**2+25.0*(4.0*x(1)+x(2))+2.0;
<    44 >          Ja(1,2) := -6.0*x(1)*x(2)+25.0*x(1)+2.0*x(2)+3.0;
<    45 >          Ja(2,1) := 6.0*x(1)*x(2)-100.0*x(2)+8.0*x(1);
<    46 >          Ja(2,2) := 3.0*x(1)**2-3.0*x(2)**2-100.0*x(1)+50.0*x(2);
<    47 >       end Jac;
<    48 >     end problem;
<    49 >
<    50 >      ------------------------------------------------------------------------
<    51 >
<    52 >   package pro1 is
<    53 >     function f(xif_arrau(i:integer) return float;
<    54 >     procedure Jac(x:f_arrau;Ja:out matrix);
<    55 >   end pro1;
<    56 >
<    57 >   package body pro1 is
<    58 >     function f(xif_arrau(i:integer) return float is
<    59 >       begin
<    60 >         if i=1
```

```
<    61 >              then  return(x(1)**2+x(2)**2-1.0);
<    62 >              else  return (x(1)**2-x(2));
<    63 >           end if;
<    64 >         end f;
<    65 >
<    66 >         procedure Jac(x:f_array;Ja:out matrix) is
<    67 >           begin
<    68 >             Ja(1,1) := 2.0*x(1);
<    69 >             Ja(1,2) := 2.0*x(2);
<    70 >             Ja(2,1) := 2.0*x(1);
<    71 >             Ja(2,2) := -1.0;
<    72 >           end Jac;
<    73 >
<    74 >       end prol;
<    75 >
<    76 >       ------------------------------------------------------------------------
<    77 >
<    78 > procedure GAUSS(N : in INTEGER;Ja : in out matrix) is
<    79 >
<    80 >    PIVOT : FLOAT;
<    81 >    M: INTEGER;
<    82 >
<    83 >  begin
<    84 >     M:=N + 1;
<    85 >     for I in 1..N loop
<    86 >        PIVOT:=Ja(I,1);
<    87 >       text_io.put(prin,PIVOT);text_io.put_line(prin,"<= PIVOT");
<    88 >       for J in 1..M loop
<    89 >          Ja(I,J):=Ja(I,J)/PIVOT;
<    90 >       end loop;
<    91 >     for J in I+1..N loop
<    92 >        PIVOT:=Ja(J,1);
<    93 >        text_io.put(prin,PIVOT);text_io.put_line(prin,"<= PIVOT2");
<    94 >        for K in 1..N loop
<    95 >           Ja(J,K):=Ja(J,K)-PIVOT*Ja(I,K);
<    96 >        end loop;
<    97 >        Ja(J,I):=0.0;
<    98 >     end loop;
<    99 >   end loop;
<   100 >
<   101 >   for I in reverse 2..N loop
<   102 >     for J in 1..I-1 loop
<   103 >       Ja(J,M):=Ja(J,M)-Ja(J,I)*Ja(I,M);
<   104 >       Ja(J,I):=0.0;
<   105 >     end loop;
<   106 >   end loop;
<   107 > end GAUSS;
<   108 >
<   109 >   ------------------------------------------------------------------------
<   110 >
<   111 > procedure newton(x:in out f_array;precis:float) is
<   112 >
<   113 >     use prol;
<   114 >
<   115 > procedure TREAT_MATRIX(n : integer) is
<   116 >  system:matrix;
<   117 >  singular : exception;
<   118 >  type matrix is array (1..10,1..10) of float;
<   119 >  epsilon : constant float := 1.0E-5 ;
<   120 >  size : integer;
<   121 >  temp,det : float;
<   122 >
```

```
<   122 > procedure determinant is
<   124 >   ans : float;
<   125 >   i,J:integer;
<   126 >   e_flag:boolean;
<   127 >
<   128 >
<   129 >   procedure gauss(n:integer) is
<   130 >     pivot:float;
<   131 >     i,J,k:integer;
<   132 >
<   133 >     procedure change(xJ:integer) is
<   134 >       ii,JJ:integer;
<   135 >       t     :float;
<   136 >       begin
<   137 >         ii:=xJ+1;
<   138 >         while system(ii,J)=0.0 loop
<   139 >           ii:=ii+1;
<   140 >           if ii>n then e_flag:=true;
<   141 >           end if;
<   142 >         end loop;
<   143 >         for JJ in 1..n loop
<   144 >           t:=system(xJ,JJ);
<   145 >           system(xJ,JJ):=system(ii,JJ);
<   146 >           system(ii,JJ):=t;
<   147 >         end loop;
<   148 >       end change;
<   149 >
<   150 >   begin
<   151 >     sub:
<   152 >       for J in 1..n-1 loop
<   153 >         for  i in J+1..n loop
<   154 >           if system(J,J)=0.0 then change(J);
<   155 >           end if;
<   156 >           exit sub when e_flag;
<   157 >           pivot:=system(i,J)/system(J,J);
<   158 >           for k in 1..n loop
<   159 >             system(i,k):=system(i,k)-system(J,k)*pivot;
<   160 >           end loop;
<   161 >         end loop;
<   162 >       end loop;
<   163 >     for I in 1..n loop
<   164 >       ans:=ans*system(i,i);
<   165 >     end loop;
<   166 >   end gauss;
<   167 >
<   168 > begin
<   169 >   system:=Ja;
<   170 >   e_flag:=false; ans:=1.0;
<   171 >   gauss(n);
<   172 >   text_io.put(prin,ans);text_io.put(prin,cr);
<   173 >   if abs(ans) < epsilon
<   174 >     then raise singular;
<   175 >   end if;
<   176 >   end determinant;
<   177 >
<   178 >
<   179 > begin
<   180 >   determinant;
<   181 >   exception
<   182 >     when singular =>
<   183 >                   text_io.put_line('MATRIX IS SINGULAR!!');
<   184 > end TREAT_MATRIX;
```

```
< 185 >
< 186 >   procedure J(x:f_array;Ja:out matrix) is
< 187 >     begin
< 188 >       Jac(x,Ja);
< 189 >       TREAT_MATRIX(n);
< 190 >     end J;
< 191 >
< 192 >
< 193 >   begin
< 194 >     found := false;
< 195 >     i:=1;
< 196 >     while not found and (i <= 100) loop
< 197 >       J(x,Ja);
< 198 >       for J in 1..n loop
< 199 >       Ja(J,n+1) := -f(x,J);
< 200 >       end loop;
< 201 >       for J in 1..n loop
< 202 >         for k in 1..n+1 loop
< 203 >           TEMP:=Ja(J,k);text_io.put(prin,TEMP);
< 204 >         end loop;
< 205 >         text_io.put_line(prin,'<=Ja");
< 206 >       end loop;
< 207 >       GAUSS(n,Ja);
< 208 >       for J in 1..n loop
< 209 >         root(J) := Ja(J,n+1) + x(J);
< 210 >         TEMP:=Ja(J,n+1);text_io.put(prin,TEMP);
< 211 >       end loop;
< 212 >       text_io.put_line(prin,'<=root");
< 213 >       for J in 1..n loop
< 214 >         x(J):=root(J);
< 215 >       end loop;
< 216 >       fall := 0.0;
< 217 >       for J in 1..n loop
< 218 >        fall := fall + abs(f(x,J));end loop;
< 219 >       if abs(fall) < precis
< 220 >         then found := true;end if;
< 221 >       i:=i+1;
< 222 >     end loop;
< 223 >     --create(prin,file);
< 224 >     if found
< 225 >     then for J in 1..n loop
< 226 >              text_io.put(prin,'    ');
< 227 >              text_io.put(prin,'root(');
< 228 >              text_io.put(prin,J);
< 229 >              text_io.put(prin,')= ');
< 230 >              buf := root(J);
< 231 >              text_io.put(prin,buf    );
< 232 >          end loop;text_io.put(prin,cr);
< 233 >          for J in 1..n loop
< 234 >             text_io.put(prin,'  f');
< 235 >             text_io.put(prin,J);
< 236 >             text_io.put(prin,'(root)= ');
< 237 >             buf:=f(x,J);
< 238 >             text_io.put(prin,buf);
< 239 >          end loop;
< 240 >          text_io.put(prin,bell);
< 241 >          text_io.put(prin,cr);
< 242 >     else text_io.put(prin,'    ');
< 243 >          text_io.put(prin,'ROOT NOT FOUND IN 100 ITERATIONS');
< 244 >          text_io.put(prin,'root so far =');
< 245 >          for J in 1..n loop buf := root(J);
< 246 >             text_io.put(prin,buf    );
```

```
<   247 >                text_io.put(prin," ; f");
<   248 >                text_io.put(prin,j);
<   249 >                text_io.put(prin,"(root) = " );
<   250 >                buf:=f(root,j);
<   251 >                text_io.put(prin,buf);
<   252 >             end loop;
<   253 >             text_io.put(prin,cr);
<   254 >       end if;
<   255 >       --close(prin);
<   256 >       end newton;
<   257 >
<   258 >       ------------------------------------------------------------------
<   259 >
<   260 >       procedure pute(temp:float) is
<   261 >          begin
<   262 >            text_io.put(prin,temp);
<   263 >          end pute;
<   264 >       procedure gete(temp :out float) is
<   265 >         begin
<   266 >          text_io.get(temp);
<   267 >        end gete;
<   268 >
<   269 > procedure fset(f:out strings(1..20)) is
<   270 >     c : character;
<   271 >     begin
<   272 >     loop
<   273 >        text_io.put(cl);
<   274 >        text_io.put_line("ENTER VOL/FILE NAME TO PRINT OUT!");
<   275 >        text_io.put(bell);
<   276 >        text_io.get(f);
<   277 >        text_io.put(cr);
<   278 >        text_io.put("O.K.????  (Y/N)");
<   279 >        text_io.get(c);
<   280 >        text_io.put(cr);if (c = 'n') or (c = 'N') then exit;
<   281 >          end if;
<   282 >       end loop;
<   283 >    end fset;
<   284 >
<   285 >       ------------MAIN   ROUTINE-------------------------------------
<   286 >
<   287 >   begin
<   288 >   fset(file);
<   289 >   open(prin,file);
<   290 >   text_io.put("INPUT SIZE!!!!! 1+++^^^^ INTEGER");
<   291 >   text_io.put(bell);text_io.put(prin,cr);
<   292 >    text_io.get(n); text_io.put_line("");
<   293 > text_io.put_line("INPUT GUESS!!!!");text_io.put(bell);
<   294 > for J in 1..n loop
<   295 >   gete(guess(J));
<   296 > end loop;
<   297 > text_io.put_line("");
<   298 > text_io.put_line("INPUT PRECIS!!!!!!");
<   299 > Text_io.put(bell);
<   300 > gete(precis);
<   301 > for J in 1..n  loop
<   302 >    text_io.put(prin,"X(");text_io.put(prin,J);
<   303 >    text_io.put(prin,") = ");
<   304 >    buf := guess(J);
<   305 >    text_io.put(prin,buf);text_io.put(prin,cr);
<   306 > end loop;
<   307 >    for I in 1..2 loop
<   308 >      text_io.put(prin,cr);
```

```
<   209 >    end loop;
<   210 > text_io.put(prin,'precis = ');
<   211 > text_io.put(prin,precis);
<   212 > text_io.put(prin,cr);
<   213 > newton(guess,precis);
<   214 > close(prin);
<   215 > end newton1;
<   216 >
```

Fig. 3. A preliminary program with the exception handler

REFERENCES

[1] Hiebert, K. L. : "An evaluation of mathematical software that solves systems of nonlinear equations", ACM-TOMAS, vol.8, 5 - 20, (1982).

[2] Hirsch, M. W. and S. Smale, : "On algorithms for solving f(x) = 0", Comm. Pure and Appl. Math., vol. XXXII, 281 - 312, (1979).

[3] Ichbiah et al., : "Rationale for the design of the Ada programming language", ACM-SIGPLAN NOTICES, vol. 14, Part b, (1979).

[4] Fujita, S. et al., : "A special-purpose data driven multiprocessor for asynchronous parallel Newton's algorithms", Papers of IPS of Japan - SIGARCH, vol. 40, (1981).

[5] Fujita, S. : "Distributed MIMD multiprocessor system with MicroAda/SuperMicro (TM) for asynchronous concurrent Newton's algorithms", ACM-SIGSMALL Conf. Proc., (1982).