**Lecture Note-Numerical Analysis (4): Roots of Nonlinear Algebraic Equations**

1. **Background information on the numerical approximation of the derivative of a function**

**FDM: Finite Difference Method (유한차분법)을 이용한 도함수 계산**

* **Taylor series expansion of f(x+h) for a small value h around a given point x**



* **The 1st order approximation of the derivative of f(x) using one of the above equation**



**Therefore, the first order numerical approximation becomes**

 or 

**,which is called by the forward/backward difference formula**

* **The 2nd order approximation of the derivative of f(x) by subtracting the above equation**



**Therefore, the 2nd order numerical approximation becomes**



**,which is called by the central difference formula**

**(Example) calculate**  for  with varying h : True value is 

|  |  |  |
| --- | --- | --- |
| h | with 1st order | with 2nd order |
| 1 | 31 | 16 |
| 0.5 | 13.1875 | 7.5625 |
| 0.25 | 8.20703125 | 5.62890625 |
| 0.125 | 6.416259766 | 5.156494141 |
| 0.0625 | 5.665298462 | 5.039077759 |
| 0.03125 | 5.322419167 | 5.009766579 |
| 0.015625 | 5.158710539 | 5.002441466 |
| 0.0078125 | 5.078737739 | 5.000610355 |
| 0.00390625 | 5.039215386 | 5.000152588 |
| 0.001953125 | 5.019569434 | 5.000038147 |

1. **Definition of Jacobian and numerical approximation of Jacobian**

Multi-variable function: 

 Example: 

Definition of the Jacobian of the multi-variable function:

 Example: 

Numerical approximation of the Jacobian using the finite difference formula



Remark: The formula has the perturbed values only for the  with 

**(Example) Jacobian computing using the central difference formula**

 at  with 

1. Perturbation in x

Positive perturbation with 



Negative perturbation with 



Jacobian component due to variable x



1. Perturbation in y with the fixed values of 

Jacobian component due to variable y



1. Perturbation in z with the fixed values of 

Jacobian component due to variable y



1. Approximated Jacobian



1. Exact Jacobian



**(Example) Jacobian computing using the forward difference formula**

 at  with 



**(Example):**  at  with 

1. Perturbation in x

Jacobian component due to variable x



1. Perturbation in y with the fixed values of 



1. Perturbation in z with the fixed values of 

Jacobian component due to variable y



1. Approximated Jacobian



1. Exact Jacobian



**Tips**

1. **Use a small value for the perturbations **
2. **Use the 2nd order central difference formula to enhance accuracy**
3. **Use the 1st order difference formula to save computing time with small values for the perturbations**
4. **Taylor series expansion of the multi-variable functions**

**(3-1) Two-variable scalar function** 



**(3-2) Two-variable vector function** 



**(3-3) Three-variable scalar function** 



**(3-4) Three-variable vector function** 



1. **Newton-Raphson Method: One of the most popular iterative method**

* The 1st order Taylor series approximation of a function can be written as



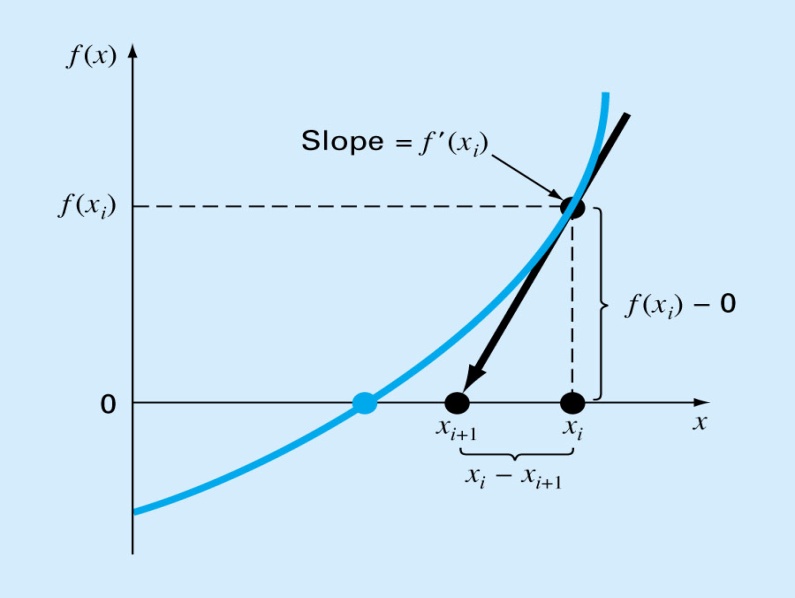
The Newton-Raphson method approximate the root with  satisfying 

Therefore,

🡪 

🡪 🡪 

* Graphical depiction of the Newton-Raphson method



**Function NEWTON(x, ITmax, h,epsilon)**

**!Pseudo code for Newton\_Raphson method to find x satisfying f(x)=0 with a given initial point x**

**!Given variables: x(initial-value), h(small increment to calculate derivative)**

**! ITmax (number of maximum iteration allowed)**

**! Given tolerance: epsilon <<1**

**! Given external function: f(x)**

**! Using numerical calculation of derivative information of the function f(x)**

**Do j=1,2,3, …….,ITmax**

**! estimation of derivative using the central difference formula**

**!**

**fzero = f(x) !function value at zero perturbation**

**xplus = x+h ! positive perturvation**

**fplus = f(xplus) ! function value at xplus**

**!**

**xminus = x-h ! negative perturvation**

**fminus = f(xminus) ! function value at xminus**

**!**

**gradf = 0.5\*(fplus-fminus)/h ! derivative(gradient) estimation**

**!Newton-Raphson method**

**x0=x**

**x 🡨 x – fzero/gradf ! for the nonlinear system: gradf is a matrix**

**!convergence test**

**If abs(fzero) < epsilon, exit !converged solution**

**If abs(x-x0)<epsilon, exit !converged solution**

**!**

**End do**

**NEWTON = x !similar to return value in C/C++**

**!**

**End NEWTON**

1. **The Secant Method**

* In the Secant method the derivative is approximated using the 1st order finite difference formula with the following increment condition



🡪 

Then the Newton-Raphson formula can be represented by the formula for the Secant Method



* Modified Secant method by directly using the backward difference formula (1st order)

🡪 



* **Definition of open method**

- It needs functional information such as function value and its gradient at one points to find a root:Find x satisfying the nonlinear equation 

-Bracketing methods are always convergent. However, the convergence of open methods highly depend on the initial estimation of the root, where function value and its gradient are calculated.

1. **Newton-Raphson Method for the System of Nonlinear Equations**

* Definition of the system of nonlinear equations

 , which has n unknowns and n nonlinear equations

Expanded form



* Jacobean of the system of nonlinear equations



* 1st Order approximation of function value around x



* Newton-Raphson Method for the system of Nonlinear Equations



The Newton-Raphson Method approximate the root with the vector satisfying

 🡪 

(Example) Newton-Raphson Method for the system of Nonlinear Equations



🡪  for 

By using 



**Function NEWTON\_SYS(n, x, ITmax, h, epsilon, error)**

**!--------------------------------------------------------------------------------------------**

**!Pseudo code for Newton\_Raphson method to find x satisfying f(x)=0 with a given initial point x**

**!Given variables:**

**! n : (input) number of equations**

**! x(1:n): (input/output) initial-value**

**! ITmax:(input) maximum allowed iteration**

**! h(1:n): (input) small increments to calculate derivative**

**! epsilon: (input) given tolerance (<<1 )**

**! error:(output) norm of function residual**

**!Be careful when we calculate the roots of nonlinear system of equations**

**! x(1:n), h(1:n), f(1:n), gradf(1:n, 1:n)**

**!--------------------------------------------------------------------------------------------**

**Do j=1,2,3, …….,ITmax**

**fzero(1:n) = f(x) !function value at zero perturbation**

**! estimation of derivative using the central difference formula**

**do k = 1, n**

**xplus(1:n) = x(1:n) ! positive perturvation**

**xplus(k) = xplus(k) + h(k)**

**fplus(1:n) = f(xplus) ! function value at xplus**

**!**

**xminus (1:n) = x(1:n) ! negative perturvation**

**xminus (k) = xminus (k) - h(k)**

**fminus (1:n) = f(xminus) ! function value at xminus**

**!**

**gradf (1:n,k)= 0.5\*(fplus(1:n)-fminus(1:n))/h(k) ! derivative(gradient) estimation**

**end do**

**!Newton-Raphson method**

**x0(1:n)=x(1:n)**

**x(1:n) 🡨 x(1:n) – (gradf)-1 \*fzero(1:n) ! for the nonlinear system: gradf is a matrix**

**!convergence test**

**If norm(fzero) < epsilon, exit !converged solution**

**If norm(x-x0)<epsilon, exit !converged solution**

**End do**

**!**

**error = norm(f(x)) ! residual in function value**

**!**

**End NEWTON\_SYS**

**Appendix: Problem set for the System of Nonlinear Equations**

**General Problem Statements**



1. **Problem #1**



1. **Problem #2**



1. **Problem #3**



1. **Problem #4**



1. **Problem #5**



1. **Problem #6 : Not converged**



1. **Problem #7**



1. **Problem #8**



1. **Problem #9**



1. **Problem #10: Not converged**



1. **Problem #11**



1. **Problem #12 : Poor convergence**



1. **Problem #13 : Poor convergence**



1. **Problem #14 : Poor convergence**



!-------------------------------------------------------------------------------------------------

! Problem Set for Nonlinear Algebraic Equations

!-------------------------------------------------------------------------------------------------

**SUBROUTINE** NAE\_Problems(IND\_PROBLEM,IND\_case,No\_variable,X,Fun,No\_fun\_call);

!--------------------------------------------------------------------------------------------------------------

**IMPLICIT** DOUBLE PRECISION (A-H,O-Z)

!--------------------------------------------------------------------------------------------------------------

! Input:

! IND\_PROBLEM: Problem Number

! IND\_Case

! = 1: initialization routine

! = 2: evaluation of function vectors

!--------------------------------------------------------------------------------------------------------------

**DIMENSION** X(\*),Fun(\*)

!-------------------------------------------------------------------------------------------------

**IF**(IND\_case==2) No\_fun\_call = No\_fun\_call + 1

!-------------------------------------------------------------------------------------------------

NSELECT = 0

**SELECT CASE** (IND\_PROBLEM)

!-------------------------------------------------------------------------------------------------

**CASE** (1) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = 1.0

X(2) = -2.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1) + 3.0\*LOG(X(1))-X(2)\*X(2)

Fun(2) = 2\*X(1)\*X(1) - X(1)\*X(2) - 5.0\*X(1) + 1.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (2) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = 1.2

X(2) = 2.5

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*X(1) + X(1)\*X(2)\*X(2) -9.0

Fun(2) = 3\*X(1)\*X(1)\*X(2) -X(2)\*\*3 - 4.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (3) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = 1.5

X(2) = 1.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1) + 2.0\*X(2) - 3.0

Fun(2) = 2\*X(1)\*X(1) + X(2)\*X(2) - 5.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (4) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = -0.5

X(2) = 0.25

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = 3.0\*X(1)\*X(1) + 4\*X(2)\*X(2) - 1.0

Fun(2) = X(2)\*\*3 - 8.0\*X(1)\*\*3 - 1.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (5) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = 1.0

X(2) = 0.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = 4.0\*X(1)\*X(1) + X(2)\*X(2) - 4.0

Fun(2) = X(1) + X(2) - SIN(X(1)-X(2))

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (6) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 3;

!

X(1) = -10.0

X(2) = -10.0

X(3) = -10.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*\*5 + (X(2)\*\*3)\*(X(3)\*\*4) + 1.0

Fun(2) = X(1)\*X(1)\*X(2)\*X(3)

Fun(3) = X(3)\*\*4 - 1.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (7) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 3;

!

X(1) = 5.0

X(2) = 0.0

X(3) = -2.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*\*2 + X(2) - 37.0

Fun(2) = X(1) - X(2)\*X(2) - 5.0

Fun(3) = X(1) + X(2) + X(3) - 3.0

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (8) ;

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 3;

!

X(1) = 3.0

X(2) = 0.0

X(3) = 1.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = 12.0\*X(1) - 3.0\*X(2)\*\*2 - 4.0\*X(3) - 7.17

Fun(2) = X(1)\*\*2 + 10.0\*X(2) - X(3) - 11.54

Fun(3) = X(2)\*\*3 + 7.0\*X(3) - 7.631

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (9) ; ! Ref Broyden A Class of Methods for Solving Nonlinear Simultaneous Equations

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = -1.2

X(2) = 1.0

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = 10.0\*(X(2) - X(1)\*\*2)

Fun(2) = 1.0 - X(1)

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (10) ; ! Ref Broyden A Class of Methods for Solving Nonlinear Simultaneous Equations

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) = 15.0

X(2) = 1.0

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = -13.0 + X(1) + ((-X(2) + 5.0)\*X(2) - 2.0)\*X(2)

Fun(2) = -29.0 + X(1) + (( X(2) + 1.0)\*X(2) - 14.0)\*X(2)

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (11) ; ! Ref Shanghai Multi-step Nonlinear ABS Methods and Their Efficiency Analysis 1991 (Extended Rosenbloack function)

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 4;

!

X(1:4) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = 10.0\*(X(2) - X(1)\*\*2)

Fun(2) = 1.0 - X(1)

Fun(3) = 10.0\*(X(4) - X(3)\*\*2)

Fun(4) = 1.0 - X(3)

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (12) ; ! ! Ref Shanghai Multi-step Nonlinear ABS Methods and Their Efficiency Analysis 1991 (Extended Powell singular function)

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 4;

!

X(1:4) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1) + 10.0\*X(2)

Fun(2) = SQRT(5.0)\*(X(3) - X(4))

Fun(3) = (X(2) - 2.0\*X(3))\*\*2

Fun(4) = SQRT(10.0)\*(X(1) - X(4))\*\*2

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (13) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #1

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1:2) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*\*2 - X(2)-1.0

Fun(2) = X(2)\*\*2 - X(1)-1.0

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (14) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #2

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1:2) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1) - X(2)\*\*2

Fun(2) = ((X(2)-1.0)\*\*2)\*((X(2)-2.0)\*\*2) + (X(1)-X(2)\*\*2)\*\*2

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (15) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #3

!-------------------------------------------------------------------------------------------------

a1 = 25.0; b1 = 1.0; c1 = 2.0;

a2 = 3.0 ; b2 = 4.0; c2 = 5.0;

**IF**(IND\_Case==1) **THEN**

No\_variable = 2;

!

X(1) =-10.0

X(2) = -1.0

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*\*3 -3.0\*X(1)\*X(2)\*\*2 +a1\*(2.0\*X(1)\*\*2 + X(1)\*X(2)) + b1\*X(2)\*\*2 + c1\*X(1) + a2\*X(2)

Fun(2) =-X(2)\*\*3 +3.0\*X(2)\*X(1)\*\*2 +a1\*( X(2)\*\*2 -4.0\*X(1)\*X(2)) + b2\*X(1)\*\*2 + c2

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (16) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #4

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 3;

!

X(1:3) = 0.1

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1) + X(2) +X(3) - 3.0

Fun(2) = X(1)\*X(2) + 2.0\*X(2)\*\*2 + 4.0\*X(3)\*\*2 - 7.0

Fun(3) = X(1)\*\*8 + X(2)\*\*4 + X(3)\*\*9 - 3.0

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

! Large Scale Problem by Adjusting N the number of equations

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (101) ; ! Ref An Autoadatative limited memory Broyden's Method to Solve Systems of NEs :: Broyden banded function

!-------------------------------------------------------------------------------------------------

N = 10

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

X(1:N) = 0.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = X(1)\*(2.0+5.0\*X(1)\*\*2)+1.0 - X(2)\*(1.0+X(2))

Fun(2) = X(2)\*(2.0+5.0\*X(2)\*\*2)+1.0 - X(1)\*(1.0+X(1)) - X(3)\*(1.0+X(3))

Fun(3) = X(3)\*(2.0+5.0\*X(3)\*\*2)+1.0 - X(1)\*(1.0+X(1)) - X(2)\*(1.0+X(2)) - X(4)\*(1.0+X(4))

Fun(4) = X(4)\*(2.0+5.0\*X(4)\*\*2)+1.0 - X(1)\*(1.0+X(1)) - X(2)\*(1.0+X(2)) - X(3)\*(1.0+X(3)) - X(5)\*(1.0+X(5))

Fun(5) = X(5)\*(2.0+5.0\*X(5)\*\*2)+1.0 - X(1)\*(1.0+X(1)) - X(2)\*(1.0+X(2)) - X(3)\*(1.0+X(3)) - X(4)\*(1.0+X(4)) - X(6)\*(1.0+X(6))

!

**DO** J = 6, N-1

Fun(J) = X(J)\*(2.0+5.0\*X(J)\*\*2)+1.0

**DO** K = J-5, J-1

Fun(J) = Fun(J) - X(K)\*(1.0+X(K))

**END DO**

Fun(J) = Fun(J) - X(J+1)\*(1.0+X(J+1))

**END DO**

!

Fun(N) = X(N)\*(2.0+5.0\*X(N)\*\*2)+1.0

**DO** K = N-5, N-1

Fun(N) = Fun(N) - X(K)\*(1.0+X(K))

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (102) ; ! Ref An Autoadatative limited memory Broyden's Method to Solve Systems of NEs :: Martinez function

!-------------------------------------------------------------------------------------------------

N = 10

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

X(1:N) = 0.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = (3.0 - 0.1\*X(1))\*X(1) + 1.0 - 2.0\*X(2) + X(1)

**DO** J = 2, N-1

Fun(J) = (3.0 - 0.1\*X(J))\*X(J) + 1.0 - X(J-1) - 2.0\*X(J+1) + X(J)

**END DO**

Fun(N) = (3.0 - 0.1\*X(N))\*X(N) + 1.0 - 2.0\*X(N-1) + X(N)

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (103) ; ! Ref An Autoadatative limited memory Broyden's Method to Solve Systems of NEs :: Broyden tridiagonal function

!-------------------------------------------------------------------------------------------------

N = 10

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

X(1:N) = 0.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = (3.0 - 2.0\*X(1))\*X(1) + 1.0 - 2.0\*X(2)

Fun(N) = (3.0 - 2.0\*X(N))\*X(N) + 1.0 - X(N-1)

**DO** J = 2, N-1

Fun(J) = (3.0 - 2.0\*X(J))\*X(J) + 1.0 - X(J-1) - 2.0\*X(J+1)

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (104) ; ! Ref An Autoadatative limited memory Broyden's Method to Solve Systems of NEs :: Spedicato function 4

!-------------------------------------------------------------------------------------------------

N = 10

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

X(1:N-1) = -1.2

X(N) = 1.0

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

K = 0

**DO** J = 1, N

K = K + 1

**IF**(K.EQ.1) **THEN** ; ! Odd case of J

Fun(J) = 1.0 - X(J)

**ELSE** ; ! Even case of J

Fun(J) = 100.0\*(X(J) - X(J-1)\*\*2)

K = 0

**END IF**

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (105) ; ! Ref An Autoadaptative limited memory Broyden's Method to Solve Systems of NEs :: Discrete integral equation function

!-------------------------------------------------------------------------------------------------

N = 10

H = 1.0/FLOAT(N+1)

HH= 0.5\*H

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

**DO** J = 1, N

TJ = H\*FLOAT(J)

X(J) = TJ\*(TJ-1.0)

**END DO**

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

**DO** J = 1, N

TJ = H\*FLOAT(J)

!

Sum1 = 0.0

**DO** K = 1, J

TK = H\*FLOAT(K)

Sum1 = Sum1 + TK\*(X(K)+TK+1.0)\*\*3

**END DO**

!

Sum2 = 0.0

**DO** K = J+1, N

TK = H\*FLOAT(K)

Sum2 = Sum2 + (1.0 - TK)\*(X(J)+TK+1.0)\*\*3

**END DO**

!

Fun(J) = X(J) + HH\*((1.0- TJ)\*Sum1 + TJ\*sum2)

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (106) ; ! Ref Shanghai Multi-step Nonlinear ABS Methods and Their Efficiency Analysis 1991 (Brown Problem)

!-------------------------------------------------------------------------------------------------

N = 10

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

!

X(1:N) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Pr1 = 1.0

**DO** K = 1, N

Pr1 = Pr1\*X(K)

**END DO**

Fun(N) = -1.0 + Pr1

!

**DO** J = 2, N-1

Fun(J) = -FLOAT(N+1)

**DO** K = J+1, N

Fun(J) = Fun(J) + X(K)

**END DO**

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (107) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #5

!-------------------------------------------------------------------------------------------------

N = 10

X0= 0.0

XN= 20.0

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

X(1:N) = 1.0

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

**DO** J = 1, N

**IF**(J==1) **THEN**

Xm = X0

Xx = X(J)

Xp = X(J+1)

**ELSE IF**(J==N) **THEN**

Xm = X(J-1)

Xx = X(J)

Xp = XN

**ELSE**

Xm = X(J-1)

Xx = X(J)

Xp = X(J+1)

**END IF**

!

Fun(J) = 3.0\*Xx\*(Xp -2.0\*Xx + Xm) + 0.25\*(Xp-Xm)\*\*2

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (108) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #6

!-------------------------------------------------------------------------------------------------

**IF**(IND\_Case==1) **THEN**

No\_variable = 10;

X(1:N) = -0.1

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

Fun(1) = (3.0-5.0\*X(1) )\*X(1) + 1.0 -2.0\*X(2)

Fun(10)= (3.0-5.0\*X(10))\*X(10) + 1.0 - X(9)

**DO** J = 2, 9

Fun(J)= (3.0-5.0\*X(J))\*X(J) - X(J-1) - 2.0\*X(J+1)

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**CASE** (109) ; ! ! Ref Atluri, A Modified Newton Method for Solving NAEs Example #7

!-------------------------------------------------------------------------------------------------

N = 10

H = 1.0/FLOAT(N+1)

H2= 1.0/H\*\*2

!

X0= 4.0

XN= 1.0

!

**IF**(IND\_Case==1) **THEN**

No\_variable = N;

X(1:N) = 0.5

!

NSELECT = 1;

!-------------------------------------------------------------------------------------------------

**ELSEIF**(IND\_Case==2) **THEN**

**DO** J = 1, N

**IF**(J==1) **THEN**

Xm = X0

Xx = X(J)

Xp = X(J+1)

**ELSE IF**(J==N) **THEN**

Xm = X(J-1)

Xx = X(J)

Xp = XN

**ELSE**

Xm = X(J-1)

Xx = X(J)

Xp = X(J+1)

**END IF**

!

Fun(J) = H2\*(Xp -2.0\*Xx + Xm) - 1.5\*Xx\*\*2

**END DO**

!

NSELECT = 1;

**END IF**

!-------------------------------------------------------------------------------------------------

!-------------------------------------------------------------------------------------------------

**END SELECT**

!

**IF**(NSELECT.EQ.0) **THEN**

**PRINT**\*,'NO PROBLEM IS SELECTED, SEE SUBROUTINE NLP\_Problems for IND\_PROBLEM = ', IND\_PROBLEM

**STOP**

**END IF**

!

**RETURN**

**END**

!================================================================================================

-----------------------------------------------------------------------------------------------------------------

Np It\_newt Nf\_newt Fn\_newt Dx\_newt It\_brdn Nf\_brdn Fn\_brdn Dx\_Brdn It\_Tmas Nf\_Tmas Fn\_Tmas Dx\_Tmas It\_mart Nf\_mart Fn\_mart Dx\_mart

-----------------------------------------------------------------------------------------------------------------

1 11 55 0.314018E-15 0.728109E-16 14 14 0.210486E-13 0.490187E-14 19 19 0.289978E-12 0.135531E-12 15 15 0.378128E-14 0.127834E-14 500 505 0.173576E-02 0.616325E-05

2 10 50 0.237783E-12 0.217161E-13 14 14 0.120334E-12 0.118316E-13 13 13 0.564844E-12 0.596211E-13 14 14 0.118498E-13 0.103636E-14 500 505 0.587316E-02 0.555262E-05

3 10 50 0.439626E-14 0.945924E-15 13 13 0.595989E-12 0.221141E-12 12 12 0.106766E-13 0.229617E-14 15 15 0.309272E-14 0.154266E-14 500 505 0.397494E-03 0.219393E-05

4 10 50 0.157009E-15 0.493563E-16 20 20 0.388806E-12 0.734353E-13 11 11 0.551088E-12 0.205790E-12 11 11 0.414695E-13 0.662863E-14 500 505 0.577830E-05 0.498691E-07

5 10 50 0.157009E-15 0.107252E-15 12 12 0.415029E-12 0.283751E-12 13 13 0.180706E-13 0.301705E-14 13 13 0.351984E-13 0.444482E-14 500 505 0.943471E-04 0.981775E-06

6 35 245 0.963470E-12 0.372914E-06 101 101 0.230735E+85 0.788668E+10 101 101 NaN NaN 101 101 0.194128E+27 0.102558E-13 500 505 NaN NaN

7 11 77 0.000000E+00 0.000000E+00 16 16 0.876837E-12 0.235614E-12 17 17 0.637615E-13 0.193977E-13 14 14 0.705456E-12 0.319848E-12 500 505 0.526664E-02 0.873318E-05

8 11 77 0.725195E-15 0.702075E-16 17 17 0.211677E-13 0.197035E-14 15 15 0.957971E-14 0.109693E-14 18 18 0.644159E-13 0.736436E-14 500 505 0.110273E-01 0.127884E-04

9 10 50 0.000000E+00 0.000000E+00 16 16 0.392523E-14 0.395443E-15 13 13 0.785046E-15 0.784943E-16 19 19 0.628086E-14 0.467219E-15 500 505 0.155750E-02 0.459616E-04

10 101 500 0.784772E+01 0.121673E+02 101 101 0.177408E+02 0.136895E+02 101 101 0.924939E+01 0.148465E+02 101 101 0.385372E+03 0.615735E+02 500 505 NaN NaN

11 9 81 0.000000E+00 0.000000E+00 77 77 0.209302E-12 0.162038E-13 20 20 0.858842E-12 0.864412E-13 41 41 0.111022E-14 0.117153E-15 500 505 0.159514E-02 0.942575E-05

12 101 900 NaN NaN 101 101 0.611603E-04 0.280025E-02 101 101 0.667863E-06 0.232192E-03 101 101 0.125840E-04 0.130658E-02 500 505 0.391605E-02 0.201710E-04

13 101 500 NaN NaN 12 12 0.239142E-12 0.106927E-12 12 12 0.239142E-12 0.106927E-12 12 12 0.155431E-14 0.695109E-15 500 505 0.426203E-02 0.289752E-04

14 101 500 0.826539E-11 0.366395E-05 78 78 0.982218E-12 0.300409E-07 101 101 0.997873E-10 0.211875E-06 35 35 0.405667E-12 0.386207E-06 500 505 0.416466E-03 0.258069E-03

15 101 500 0.447148E+01 0.294290E+00 101 101 0.409754E+01 0.150103E-02 101 101 0.198576E+02 0.191478E+00 101 101 0.334405E+04 0.548425E+01 500 505 0.116989E+09 0.193125E+01

16 30 210 0.904843E-12 0.271993E-12 89 89 0.173068E-12 0.612027E-13 23 23 0.213220E-12 0.675918E-13 41 41 0.317583E-12 0.677076E-13 500 505 0.419002E-02 0.251980E-04

101 17 357 0.276556E-12 0.376083E-13 101 101 0.919831E+13 0.132828E+04 101 101 0.165520E+00 0.317172E-01 101 101 0.835198E+01 0.296900E+00 500 505 0.728118E-01 0.130043E-03

102 10 210 0.289468E-14 0.212347E-14 34 34 0.900751E-12 0.346585E-12 22 22 0.340257E-13 0.133692E-13 30 30 0.314446E-12 0.866142E-13 500 505 NaN NaN

103 11 231 0.352834E-15 0.764035E-16 48 48 0.853439E-12 0.297119E-12 23 23 0.159601E-12 0.396714E-13 34 34 0.271327E-12 0.495800E-13 500 505 NaN NaN

104 9 189 0.000000E+00 0.000000E+00 101 101 0.192036E-03 0.289833E-05 39 39 0.306079E-13 0.380775E-15 30 30 0.213685E-12 0.414231E-14 500 505 0.594295E-01 0.480422E-04

105 10 210 0.202302E-16 0.173801E-16 19 19 0.482188E-12 0.479602E-12 18 18 0.315927E-12 0.312288E-12 16 16 0.303084E-12 0.307928E-12 500 505 0.700192E-04 0.813046E-06

106 101 2100 NaN NaN 101 101 0.316228E+00 0.123955E-13 101 101 0.316228E+00 0.471344E-07 101 101 0.316228E+00 0.931649E-12 500 505 0.316749E+00 0.361314E-02

107 22 462 0.315274E-12 0.426300E-10 101 101 0.553139E+12 0.145194E+06 101 101 0.136526E-03 0.260155E-01 101 101 0.207699E-01 0.181703E-01 500 505 0.114560E-01 0.146905E-03

108 11 231 0.464440E-16 0.938605E-17 33 33 0.724831E-12 0.216322E-12 22 22 0.650093E-12 0.150471E-12 31 31 0.169978E-12 0.382370E-13 500 505 0.298611E-03 0.918865E-06

109 11 231 0.333990E-13 0.202870E-15 101 101 0.245979E+04 0.117144E+02 34 34 0.330289E-12 0.113646E-13 101 101 0.637111E+05 0.157089E+03 500 505 0.171403E+01 0.871518E-04

-----------------------------------------------------------------------------------------------------------------