**Utilización del algoritmo de *Newton* para la Determinación de las Raíces de un Sistema de Ecuaciones No Lineales**

**Determinación de las Raíces del Sistema de Ecuaciones del Ejemplo 04**

**Función a evaluar: *Ex\_04\_func***

function f = Ex\_04\_func(x,c0,K1,K2)

// Evaluación del sistema de ecuaciones del Ejemplo 04

// c0(1) = ca0 / c0(2) = cb0 / c0(3) = cc0 / c0(4) = cd0

ca = c0(1) - 2\*x(1)\*c0(2) - x(2)\*c0(4);

cb = (1 - x(1))\*c0(2);

cc = c0(3) + x(1)\*c0(2) + x(2)\*c0(4);

cd = (1 - x(2))\*c0(4);

f(1) = cc / ca^2 / cb - K1;

f(2) = cc / ca / cd - K2;

f = [f(1);f(2)]; // Vector columna

endfunction

**Jacobiano de la función a evaluar**

function J = Jacobian(x,c0)

// Jacobian for Example\_04

// c0(1) = ca0 / c0(2) = cb0 / c0(3) = cc0 / c0(4) = cd0

ca = c0(1) - 2\*x(1)\*c0(2) - x(2)\*c0(4);

cb = (1 - x(1))\*c0(2);

cc = c0(3) + x(1)\*c0(2) + x(2)\*c0(4);

cd = (1 - x(2))\*c0(4);

J(1,1) = c0(2)\*((ca^2)\*cb + cc\*(4\*ca\*cb + ca^2))/(ca^4)/(cb^2);

J(1,2) = c0(4)\*((ca^2)\*cb + 2\*cc\*ca\*cb)/(ca^4)/(cb^2);

J(2,1) = c0(2)\*(ca+2\*cc)/(ca^2)/cd;

J(2,2) = c0(4)\*(ca\*cd+cc\*(cd+ca))/(ca^2)/(cd^2);

endfunction

**Algoritmo de Newton Modificado para la Determinación de las Raíces de un Sistema de Ecuaciones No Lineales**

function [solution,iter] = Newton(c0,K1,K2,rho,x0,tol)

// Solves the non-linear vector equation f(x)=0 set using Newton Raphson iteration

// (but with a the ability to reduce the length of the step until the a decrease in error is

// detected)

// INPUTS:

// rho = Relaxation factor

// x0 = Initial Guess (a vector)

// tol = Tolerance

//

// OUTPUTS

// solution = The solution to f(x) = 0

// iter = Iterations

x = x0;

itemax = 100;

// set the error 2\*tol to make sure the loop runs at least once

error = 2\*tol;

iter = 1;

while error > tol

//calculate the function values at the current iteration

f = Ex\_04\_func(x,c0,K1,K2);

//calculate the error

error1 = max(abs(f));

//calculate the jacobian matrix

J = Jacobian(x,c0);

//calculate the update (solve the linear system)

dx = J\(-f);

//calculate the error

xnew = x + dx;

f = Ex\_04\_func(xnew,c0,K1,K2);

error2 = max(abs(f));

//update the x value

i=1;

imax=10;

while (error2>=error1||~isreal(f))

xnew = x+(rho^i)\*dx;

f = Ex\_04\_func(xnew,c0,K1,K2);

error2 = max(abs(f));

i = i+1;

if i > imax

disp(' No convergence: Change relaxation factor or initial guesses')

solution = xnew;

resume

end

end

x = xnew;

//calculate the error

f = Ex\_04\_func(x,c0,K1,K2);

error = max(abs(f))

iter = iter + 1;

if iter > itemax

disp(' No convergence: Change initial guesses or maximum number of iterations ')

solution = x;

resume

end

end //while loop

solution = x;

endfunction

**Ejecución del programa en Scilab**

1. **Se ingresan los datos del sistema de ecuaciones (eventualmente se lo puede hacer a través de la carga de un fichero de datos:load(‘datos.dat’)):**

--> c0 = [40 15 0 10] // Concentraciones iniciales

c0 =

40. 15. 0. 10.

--> K1=5e-04 // Constante de equilibrio de la 1er. reacción química

K1 =

0.0005

--> K2=4e-02 // Constante de equilibrio de la 2da. reacción química

K2 =

0.04

--> rho=0.5 // Factor de relajación

rho =

0.5

--> x0 = [0.1 0.4]' // Estimaciones iniciales de la conversión (vector columna)

x0 =

0.1

0.4

--> tol=1e-10 // Tolerancia del algoritmo

tol =

1.000D-10

1. **Se ejecutan las functions**

**2.a Ex\_04\_func**

**2.b Jacobian**

**2.c Newton**

**luego se escribe en la ventana de comandos de Scilab:**

--> [solution,iter] = Newton(c0,K1,K2,rho,x0,tol)

iter =

5.

solution =

0.1202667

0.4786707