**Informe laboratorio 6**

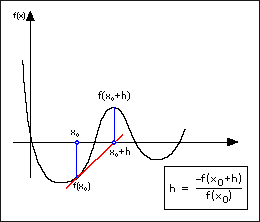
El método de Newton Raphson, trabaja evaluando la función en un punto inicial y calculando un plano tangente a la función en dicho punto, hallando un valor aproximado para la raíz de la función.

Figura 1. Ilustración gráfica Newton Raphson.

Para el caso de estudio se quiere dar solución al sistema de ecuaciones para el dimensionamiento de

Este método, resulta problemático porque las características de las funciones de estudio pueden hacer que el método diverja). Además, se encarga de evaluar la raíz más cercana al valor inicial.

|  |  |  |
| --- | --- | --- |
|  | **Ventajas** | **Desventajas** |
| Newton-Raphson | - Converge rápidamente dado que los saltos no son muy grandes.  - Convergencia cuadrática. | - Es necesario calcular el Jacobiano (dificulta el proceso). |

**Algoritmo**

**Código**

syms x y h s;

sol = [1 1]; % x0, y0

Z0ep = 51.607;

Z0op = 48.443;

error = 1;

% Condiciones Iniciales

Z0 = sqrt(Z0ep\*Z0op);

k0 = (Z0ep-Z0op)/(Z0ep+Z0op);

sol(1) = 4\*exp(-Z0/(59.952\*sqrt(0.987-0.171\*k0-1.723\*k0^3)))/pi;

r = (4/(pi\*sol(1)))^(0.001+1.117\*k0-0.683\*k0^2);

sol(2) = log((r+1)/(r-1)) - sol(1);

% Ecuaciones principales para aproximación

d = 0.11 - 0.83\*y + 1.64\*y^2 - y^3;

a = 1 + exp(16\*x - 18.272);

b = sqrt(5.905-x^4);

c = -0.8107\*y^3 + 1.3401\*y^2 - 0.6929\*y + 1.0892 + 0.014002/y - 0.000636/(y^2);

e = -0.15\*exp(-13\*x);

g = 2.23\*exp(-7.01\*y + 10.24\*y^2 - 27.58\*y^3);

k = 1 + 0.01\*(-0.0726 - 0.2145/y + 0.222573/(y^2) - 0.012823/(y^3));

l = 0.01\*(-0.26 + 0.6866/y + 0.0831/(y^2) - 0.0076/(y^3));

m = -0.1098 + 1.2138\*x - 2.2535\*x^2 + 1.1313\*x^3;

n = -0.019 - 0.016/y + 0.0362/(y^2) - 0.00234/(y^3);

f1 = x\*a/b;

f3 = tanh(pi\*(x+y)/2);

% Caso 1 – y < 0.9

f2\_1 = c - x\*d + e\*g

f4\_1 = k - x\*l + m\*n

Z0e\_1 = 59.952\*log(0.523962/f1\*f2\_1\*f3)

Z0o\_1 = 59.952\*log(0.523962\*f3/f1\*f4\_1)

V1\_1 = Z0e\_1 - Z0ep

V2\_1 = Z0o\_1 - Z0op

J\_1 = jacobian([V1\_1, V2\_1], [x,y])

eq1 = double(subs((J\_1^-1), {x,y}, sol))\*double(subs([V1\_1;V2\_1], {x,y}, sol));

% Caso 2 – y >= 0.9

f2\_2 = 1 + 0.004\*exp(0.9-y)

f4\_2 = 1

Z0e\_2 = 59.952\*log(0.523962/f1\*f2\_2\*f3)

Z0o\_2 = 59.952\*log(0.523962\*f3/f1\*f4\_2)

V1\_2 = Z0e\_2 - Z0ep

V2\_2 = Z0o\_2 - Z0op

J\_2 = jacobian([V1\_2, V2\_2], [x,y])

eq2 = double(subs((J\_2^-1), {x,y}, sol))\*double(subs([V1\_2;V2\_2], {x,y}, sol));

% Iteraciones con Newton-Raphson

while error > 0.7/100

if sol(2) < 0.9 % y0

new\_sol = sol' - eq1;

else

new\_sol = sol' - eq2;

end

double(subs(J, {x,y}, sol))

error = abs(sol-new\_sol);

sol = new\_sol;

end

Al parecer, las aproximaciones y la no-linealidad de las funciones hacen que la solución sea errada y que el método no converja.

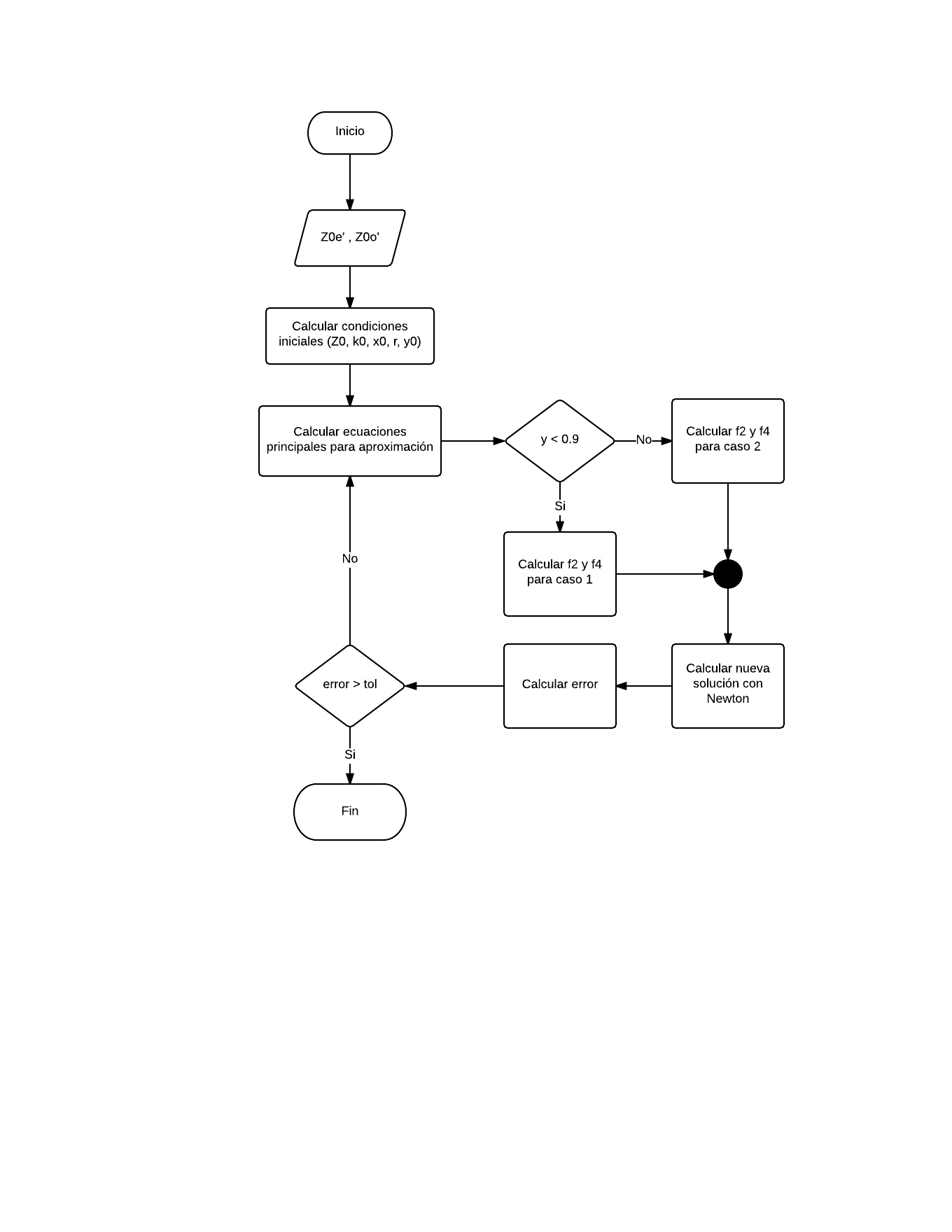


Figura . Diagrama de flujo algoritmo.

La tolerancia se define como en el libro, debe satisfacerse la siguiente desigualdad

Con lo que se garantiza que la precisión de la solución no sea peor al 0.1%