Informe Laboratorio: Análisis Numérico
Práctica No. 3

Daniel Delgado
Código: 2182066
Grupo: B2
Escuela de Ingeniería de Sistemas e Informática
Universidad Industrial de Santander

November 25, 2020

1 Introducción

La solución de muchos problemas matemáticos de manera manual, en términos generales, requieren de la aplicación de procesos los cuales, en términos computacionales, no podrían realizarse de la misma manera en lo que se harían de manera manual.
Como ejemplo de esto, tenemos el cálculo de las raíces de algunas funciones. Dentro de las "limitaciones" de la computación, la determinación de estos valores debe ser efectuada con el apoyo de algoritmia iterativa. El método de bisección, es uno de estos pero sufre de ser considerablemente lento en su implementación. En pos de mejorar los tiempos de computación en el cálculo de las raíces, se emplea el método de Newton-Raphson.
La compresión de este concepto, al igual que el desarrollo de la algoritmia relacionada, son los principales temas a a tratar durante el desarrollo del presente informe, así como la resolución de los problemas propuestos a manera de pregunta orientadora durante el desarrollo del componente práctico del mismo.

2 Desarrollo

1. Implementación básica

   Una de las partes más importantes respecto al desarrollo del trabajo de laboratorio en cuanto a su componente práctico se requiere a la implementación de algoritmia con el fin de cumplir con un objeto o dar solución a un problema propuesto de manera satisfactoria.

   De esto, se desarrolló la función `newtonRoot(fun, der, ini, ite)`. Esta función, en términos simples, realiza de manera iterativa el cálculo de una raíz a partir de un punto inicial para una función cualquiera.

```
function output = newtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;
        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
            return;
        else
            %% Skip %%
        end

        for index = 1:ite
```

```
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after
                    ', num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
                %% Skip %%
            end
        end

        if abs(fun(nextVal)) < 10^(-5)
            output = nextVal;
            disp(['Approximate root found! ', num2str(nextVal)]);
        else
            disp('No roots found...')
        end

    else
        disp('Passed params are invalid!')
    end
end
```

`newtonRoot`, en términos simples, realiza el cálculo de la raíz. Tras la verificación de los parámetros pasados a la función y asignar las variables a trabajar, se iniciará el ciclo iterativo en el cual se realizará el cálculo de una de las raíces de la función.

A partir de esto, se aplicará la función (1) con la cual se podrá aproximar la raíz de la función dada. Tras una comprobación del valor recientemente calculado en la función para determinar si es una raíz, se repetirá este proceso hasta encontrar una raíz o que se cumplan la cantidad dada de iteraciones. Finalmente, se dará el output del resultado dado.

$$k_n = k_{n-1} - \frac{f(k_{n-1})}{f' k_{n-1}} \tag{1}$$

Para probar el funcionamiento de `newtonRoot`, se ejecutó la función con los siguientes parámetros.

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 900;
ini = 12;
newtonRoot(f, der, ini, ite)
```

Tras la ejecución, la función dió como salida 11.9310, que, efectivamente para la función $x^3+13x^2-297.5x+0.00000375e^x$, da como resultado un valor bastante cercano a 0 como era de esperarse.

Algo a reconocer son las limitaciones presentes en la implementación de la función. La limitación más evidente es la necesidad de pedir la derivada de la función como un parámetro, debido a que no es posible calcular la derivad a partir de un `function_handle`, existe la posibilidad de que no sea posible calcular el valor de una raíz, o se calcule un valor incorrecto, de pasar de manera errónea la derivada de la función.

2. Modificación simple

   Como segunda parte de la implementación, se necesitaba el poder ver los valores de la iteración actual, el valor de la raíz calculada para la iteración actual, el valor de la función evaluada para el último punto calculado al igual que el valor de su derivada en ese mismo punto y el error absoluto entre el nuevo valor y el anterior.

Con el fin de dar cabida a las especificaciones requeridas, se desarrollo una versión modificada de `newtonRoot` llamada `modNewtonRoot`.

```
function output = modNewtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;
        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
            return;
        else
            %% Skip %%
        end

        for index = 1:ite
            lastVal = nextVal;
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            disp(['Realizando iteracion #', num2str(index)] )
            disp(['Raiz actual: ', num2str(nextVal)])
            disp(['f(', num2str(nextVal),') = ', num2str(fun(nextVal))])
            disp(['f''(', num2str(nextVal),') = ', num2str(der(nextVal))])
            disp(['Error absoluto = ', num2str(abs(nextVal-lastVal))])
            disp('-----------')

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after
                    ', num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
                %% Skip %%
            end
        end

        if abs(fun(nextVal)) < 10^(-5)
            output = nextVal;
            disp(['Approximate root found! ', num2str(nextVal)]);
        else
            disp('No roots found...')
        end

    else
        disp('Passed params are invalid!')
    end
end
```

Como es posible verlo, en esencia, es la misma función que `newtonRoot` pero con diferentes `disp` dentro del ciclo iterativo los cuales imprimen los valores requeridos a la consola en cada iteración.

Con el fin de comprobar su correcto funcionamiento, se ejecutó la función con los siguientes parámetros:

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 3;
ini = 1;
modNewtonRoot(f, der, ini, ite);
```

Tras la ejecución, tendremos como salida en la consola lo siguiente:

```
Realizando iteracion #1
Raiz actual: -0.055866
f(-0.055866) = 16.6605
f'(-0.055866) = -298.9431
Error absoluto = 1.0559
-----------
Realizando iteracion #2
Raiz actual: -0.00013454
f(-0.00013454) = 0.04003
f'(-0.00013454) = -297.5035
Error absoluto = 0.055731
-----------
Realizando iteracion #3
Raiz actual: 1.1814e-08
f(1.1814e-08) = 2.3536e-07
f'(1.1814e-08) = -297.5
Error absoluto = 0.00013455
-----------
Approximate root found! 1.1814e-08
```

Esto, de ser calculado de manera manual, en el momento de ser calculado, podrá ser observado como el output de la función refleja los valores esperados.

3. implementación visual

   De manera final, se plantea la necesidad de una implementación visual del método Newton-Raphson, de esto, nuevamente, se modifica la función `newtonRoot` para poder dar cabida a los requerimientos. La función modificada, `visualNewtonRoot`, tiene la capacidad de graficar cada uno de los puntos calculados al igual que la función como tal.

```
function output = visualNewtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;

        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
            return;
        else
            %% Skip %%
        end

        for index = 1:ite
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            plot(nextVal, fun(nextVal),'-*',...
            'LineWidth',1,...
            'MarkerSize',5,...
            'MarkerEdgeColor','#7EA28E')
            hold on

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after
                    ', num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
```

```
            %% Skip %%
        end
    end

    fplot(fun, [(nextVal - (fun(nextVal)/der(nextVal)))-1, (nextVal - (fun(nextVal)/der(nextVal)))+1])
    hold on

    line([nextVal-2 nextVal+2], [0 0])

    plot(nextVal, fun(nextVal),'-.ko',...
    'LineWidth',1,...
    'MarkerSize',10,...
    'MarkerEdgeColor','#A2142F')
    hold off

    if abs(fun(nextVal)) < 10^(-5)
        output = nextVal;
        disp(['Approximate root found! ', num2str(nextVal)]);
    else
        disp('No roots found...')
    end

else
    disp('Passed params are invalid!')
end
end
```

Con el fin de probar el funcionamiento correcto de la graficación, se ejecutó la función con lo siguientes parámetros:

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 4;
ini = -27;
visualNewtonRoot(f, der, ini, ite);
```

De esto, tras le ejecución, tenemos con resultado la figura 1 donde se puede observar el como se representan de manera visual algunos de los puntos calculados y la raíz de la función.

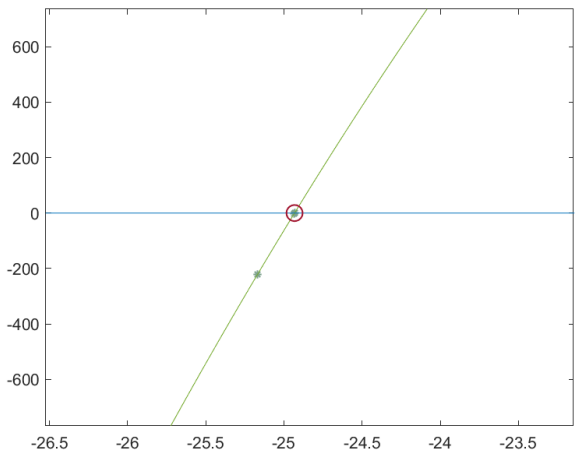

Figure 1: Gráfica resultante de `visualNewtonRoot`

3 Anexos

newtonRoot.m

```
function output = newtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;
        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
            return;
        else
            %% Skip %%
        end

        for index = 1:ite
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after ',
                    num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
                %% Skip %%
            end
        end

        if abs(fun(nextVal)) < 10^(-5)
            output = nextVal;
            disp(['Approximate root found! ', num2str(nextVal)]);
        else
            disp('No roots found...')
        end

    else
        disp('Passed params are invalid!')
    end
end
```

newtonRunner.m

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 900;
ini = 12;
newtonRoot(f, der, ini, ite)
```

modNewtonRoot.m

```
function output = modNewtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;
        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
```

```
            return;
        else
            %% Skip %%
        end

        for index = 1:ite
            lastVal = nextVal;
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            disp(['Realizando iteracion #', num2str(index)] )
            disp(['Raiz actual: ', num2str(nextVal)])
            disp(['f(', num2str(nextVal),') = ', num2str(fun(nextVal))])
            disp(['f''(', num2str(nextVal),') = ', num2str(der(nextVal))])
            disp(['Error absoluto = ', num2str(abs(nextVal-lastVal))])
            disp('-----------')

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after ',
                    num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
                %% Skip %%
            end
        end

        if abs(fun(nextVal)) < 10^(-5)
            output = nextVal;
            disp(['Approximate root found! ', num2str(nextVal)]);
        else
            disp('No roots found...')
        end

    else
        disp('Passed params are invalid!')
    end
end
```

moddedRunner.m

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 3;
ini = 1;
modNewtonRoot(f, der, ini, ite);
```

visualNewtonRoot.m

```
function output = visualNewtonRoot(fun, der, ini, ite)
    if (isa(fun, 'function_handle') && isa(der, 'function_handle') && ite > 0)
        nextVal = ini;
        output = NaN;

        if(fun(nextVal) == 0)
            disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!'])
            output = nextVal;
            return;
        else
```

```
            %% Skip %%
        end

        for index = 1:ite
            nextVal = nextVal - (fun(nextVal)/der(nextVal));

            plot(nextVal, fun(nextVal),'-*',...
            'LineWidth',1,...
            'MarkerSize',5,...
            'MarkerEdgeColor','#7EA28E')
            hold on

            if(fun(nextVal) == 0)
                disp(['Root found! ', num2str(nextVal), ' is the root for ', func2str(fun), '!', ' Found after ',
                    num2str(index), 'iterations'])
                output = nextVal;
                return;
            else
                %% Skip %%
            end
        end

        fplot(fun, [(nextVal - (fun(nextVal)/der(nextVal)))-1, (nextVal - (fun(nextVal)/der(nextVal)))+1])
        hold on

        line([nextVal-2 nextVal+2], [0 0])

        plot(nextVal, fun(nextVal),'-.ko',...
        'LineWidth',1,...
        'MarkerSize',10,...
        'MarkerEdgeColor','#A2142F')
        hold off

        if abs(fun(nextVal)) < 10^(-5)
            output = nextVal;
            disp(['Approximate root found! ', num2str(nextVal)]);
        else
            disp('No roots found...')
        end

    else
        disp('Passed params are invalid!')
    end
end
```

visualRunner.m

```
f = @(x) (x^3)+(13*(x^2))-(297.5*x)+(0.00000375*(exp(x)));
der = @(x) (3*(x^2))+(26*x)-(297.5)+(0.00000375*(exp(x)));
ite = 4;
ini = -27;
visualNewtonRoot(f, der, ini, ite);
```