Finding roots of equations

Newton-Raphson

The Newton-Raphson method iteratively calculates the roots of equations by fitting a tangent line to a given guess and using the intersection of this tangent line with the x-axis as the new guess. In other words, given a function $f(x)$ and an initial guess $x = x_0$ for the solution, we have that a better guess is given by

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Hence, we have that in general,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

For more information, see newtonRaphson.m.

An example of this method in action is shown below.

```
% function to find the zeros of
y = @(x) x.^3-3.*x-10;

% finding a single root with Newton-Raphson
sol1 = newtonRaphson(y, 1.1, eps);
disp("The root found by Newton-Raphson is x = " + ...
    sol1);
```

```
The root found by Newton-Raphson is x = 2.6129
```

```
disp("The value of y at this point is " + ...
    y(sol1));
```

```
The value of y at this point is 0
```

Muller's Method

Muller's method interpolates a quadratic through three guesses of the root for a function and recursively utilizes the roots of these polynomials as the next guesses. In other words, if we fit a quadratic through the evaluations of $f(x)$ for three guesses $x_{k-1}, x_{k-2}, x_{k-3}$, we have the following expression for this quadratic:

$$y_k(x) = a(x - x_{k-1})^2 + b(x - x_{k-1}) + c$$

and the next guess is given by

$$x_k = x_{k-1} - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}$$

where the sign of the denominator is chosen to maximize its magnitude. Since we are using a quadratic to interpolate, we can also calculate complex roots, which we cannot do with Newton-Raphson unless our initial guess/function is inherently complex.

For more information, see muller.m.

An example with the same function from before is shown below:

```
% finding a single root with Mullér's method
sol2 = muller(y, -1, -2, -3, 1e-5);
disp("The root found by Muller's method is x = " + ...
    sol2);
```

```
The root found by Muller's method is x = -1.3064+1.4562i
```

```
disp("The value of y at this point is " + ...
    y(sol2));
```

```
The value of y at this point is 3.9748e-09+3.2555e-09i
```

We can recursively apply these methods by repeatedly dividing this function by x minus each root found in order to find all the roots. See the functions below as an example:

```
% solving the equation from before
disp("The 3 roots of y are given by")
```

```
The 3 roots of y are given by
```

```
disp((findZeros(y, 3, 1e-5)))
```

```
  -1.3064 + 1.4562i
  -1.3064 - 1.4562i
   2.6129 - 0.0000i
```

```

% example of finding some of the roots for a transcendental expression
z = @(x) x .* exp(x) - 10 .* x.^2;
disp("5 roots of the function z are")
```

```
5 roots of the function z are
```

```
disp((findZeros(z, 5, 1e-5)))
```

```
   0.1118 + 0.0000i
   0.0000 + 0.0000i
   3.5772 + 0.0000i
   6.4712 -64.3024i
   5.6017 +26.4953i
```

```

% repeated roots
f = @(x) (x - 1).^4 .* (x - 5);
```

```
disp("5 roots of the function f are")
```

```
5 roots of the function f are
```

```
disp((findZeros(f, 5, 1e-7)))
```

```
   1.0070 - 0.0083i
   1.0029 - 0.0017i
   0.9999 - 0.0009i
   0.9999 - 0.0001i
   5.0000 - 0.0000i
```

```
function r = findZeros(y, n, threshold)
   r = findZerosRec(y, n, [], threshold);
end

function r = findZerosRec(y, n, l, threshold)
    % recursively find all roots
    if n == 0
        r = l;
    else
        root = muller(y, -1.2382394234, 1.23042234, 0.09234234, threshold);
        y1 = @(x) y(x)/(x - root);
        l = [l; root];
        r = findZerosRec(y1, n - 1, l, threshold);
    end
end
```