Assignment 5: Secant Method

By: Abrar Imon

Part 1 : First, we will create a function to perform the Secant Method on the different equations and different error tolerance from past assignments. Then, we will create a grid to display the relevant informations from performing the Secant Method.

```
secantMethod[eq_, ϵ_] := (
  count = 1;
  a = 1;
  b = 3;
  computerRoot = r /. FindRoot[{eq[r]}, {r, 2}];
  While[count ≤ 1000, p = N[b - eq[b] * (b - a) / (eq[b] - eq[a]), 64];
   (*For stopping the secantMethod, I am using the method we were taught in class,
   which is the same as the one on textbook.*)
   If[Abs[p - b] < ϵ, Break[]];
   count++;
   a = b;
   b = p;
   b = p];
  Return[{"y =" eq[x], ϵ, N[p, 5], computerRoot, Abs[p - computerRoot], count}]
 )
```

```
(*These three lines will translate the equations into
 functions. We want to rewrite the equations as functions so that
 the secantMethod function can take the equations as inputs.*)
equation1[x_] := x^5 - 2 x^4 - 1
equation2[x_] := x^3 - x - 1
equation3[x_] := x^2 + 2 x - 5
```

```
(*This cell makes a grid for the three
 equations for the four different tolerance levels.*)
Grid[{secantMethod[equation1, 0.1], secantMethod[equation1, 0.05],
  secantMethod[equation1, 0.01], secantMethod[equation1, 0.001],
  secantMethod[equation2, 0.1], secantMethod[equation2, 0.05],
  secantMethod[equation2, 0.01], secantMethod[equation2, 0.001],
  secantMethod[equation3, 0.1], secantMethod[equation3, 0.05],
  secantMethod[equation3, 0.01], secantMethod[equation3, 0.001]}, Frame → All]
```

```
(*This cell aligns everything in the grid.*)
Insert[%34, Alignment → Left, 2]
```

```
(*This cell titles the columns.*)
ReplacePart[%36,
 1 → Prepend[First[%36], {"Polynomial", "Error Tolerance (∈)", "Root from Secant Method",
    "Actual Root", "Actual Error", "Number of Steps  Secant Method Used"}]]
```

Out[•]=

Polynomial	Error Tolerance (∈)	Root from Secant Method	Actual Root	Actual Error	Number of Steps Secant Method Used
$y = (-1 - 2x^4 + x^5)$	0.1	1.0999	2.05597	0.956101	2
$y = (-1 - 2x^4 + x^5)$	0.05	-0.18749	2.05597	2.24346	7
$y = (-1 - 2x^4 + x^5)$	0.01	-0.18749	2.05597	2.24346	7
$y = (-1 - 2x^4 + x^5)$	0.001	-0.18749	2.05597	2.24346	7
$y = (-1 - x + x^3)$	0.1	1.1487	1.32472	0.176032	2
$y = (-1 - x + x^3)$	0.05	1.3242	1.32472	0.000490557	5
$y = (-1 - x + x^3)$	0.01	1.3242	1.32472	0.000490557	5
$y = (-1 - x + x^3)$	0.001	1.3247	1.32472	4.52636×10^{-6}	6
$y = (-5 + 2x + x^2)$	0.1	1.4211	1.44949	0.0284371	2
$y = (-5 + 2x + x^2)$	0.05	1.4502	1.44949	0.000694759	3
$y = (-5 + 2x + x^2)$	0.01	1.4495	1.44949	4.05584×10^{-6}	4
$y = (-5 + 2x + x^2)$	0.001	1.4495	1.44949	4.05584×10^{-6}	4

Now we will work on comparing the iteration numbers between our four root finding methods. First, we will create four functions (one for each of the root finding methods) that count the number of iterations each of our root finding methods use. Then, perform the functions on the different equations at different error tolerances. Then, organize the information needed to compare the different methods.

```
regulaFalsiCounter[eq_, ϵ_] := (
  count = 1;
  a = 1;
  b = 3;
  computerRoot = r /. FindRoot[{eq[r]}, {r, 2}];
  While[count ≤ 100, p = N[b - eq[b] * (b - a) / (eq[b] - eq[a]), 32];
   If[Abs[p - computerRoot] < ϵ, Break[]];
   count++;
   If[eq[p] * eq[b] < 0, a = b];
   b = p];
  Return[count]
 )
```

```
In[ ]:= bisectionTechniqueCounter[eq_, ϵ_] := (
  count = 0;
  a = 1;
  b = 3;
  While[count ≤ 1000, p = (a + (b - a) / 2);
   If[(eq[p] == 0) || (((b - a) / 2) < ϵ), myRoot = p;
    Break[]];
   count++ ;
   If[eq[p] * eq[a] > 0, a = p, b = p]];
  computerRoot = r /. FindRoot[{eq[r]}, {r, 2}];
  Return[count]
 )
```

```
newtonMethodCounter[eq_, ϵ_] := (
  count = 1;
  lastP = 1;
  While[count ≤ 250, p = N[lastP - (eq[lastP] / eq'[lastP]), 32];
   If[(eq[p] == 0) || ((Abs[lastP - p]) < ϵ), Break[]];
   count++; lastP = p];
  computerRoot = r /. FindRoot[{eq[r]}, {r, 2}];
  Return[count]
 )
```

```
secantMethodCounter[eq_, ϵ_] := (
  count = 1;
  a = 1;
  b = 3;
  computerRoot = r /. FindRoot[{eq[r]}, {r, 2}];
  While[count ≤ 1000, p = N[b - eq[b] * (b - a) / (eq[b] - eq[a]), 64];
   (*For stopping the secantMethod,
   I am using the method we were taught in class.*)If[Abs[p - b] < ϵ, Break[]];
   count++;
   a = b;
   b = p;
   b = p];
  Return[count]
 )

(*These next two cells will create a list of lists, called "listFinal",
which will have the information we need for the next grid,
in the format we need for the next grid.*)
list = Table[{"y =" eq[x], ϵ, regulaFalsiCounter[eq, ϵ], bisectionTechniqueCounter[eq, ϵ],
   newtonMethodCounter[eq, ϵ], secantMethodCounter[eq, ϵ]},
  {eq, {equation1, equation2, equation3}}, {ϵ, {0.1, 0.05, 0.01, 0.001}}]

In[ ]:= listFinal = Join[list⟦1⟧, list⟦2⟧, list⟦3⟧]

In[ ]:= (*This cell creates the grid.*)
Grid[listFinal, Frame → All]

(*This cell aligns the grid.*)
Insert[%52, Alignment → Left, 2]
```

```
(*This cell titles the columns of the grid.*)
ReplacePart[%53,
 1 → Prepend[First[%53], {"Polynomial", "Error Tolerance (ϵ)", "Regula Falsi count",
    "Bisection Method count", "Newton's Method count", "Secant Method count"}]]
```

Out[∘]=

Polynomial	Error Tolerance (ϵ)	Regula Falsi count	Bisection Method count	Newton's Method count	Secant Method count
$y = (-1 - 2x^4 + x^5)$	0.1	19	4	42	2
$y = (-1 - 2x^4 + x^5)$	0.05	22	5	43	7
$y = (-1 - 2x^4 + x^5)$	0.01	28	7	43	7
$y = (-1 - 2x^4 + x^5)$	0.001	37	10	43	7
$y = (-1 - x + x^3)$	0.1	4	4	3	2
$y = (-1 - x + x^3)$	0.05	6	5	3	5
$y = (-1 - x + x^3)$	0.01	11	7	4	5
$y = (-1 - x + x^3)$	0.001	17	10	4	6
$y = (-5 + 2x + x^2)$	0.1	2	4	2	2
$y = (-5 + 2x + x^2)$	0.05	2	5	3	3
$y = (-5 + 2x + x^2)$	0.01	3	7	3	4
$y = (-5 + 2x + x^2)$	0.001	5	10	3	4

This table suggests that the Secant method finds the root of a polynomial (for a given ϵ) with the least amount of iterations, on average. That is not necessarily true. It is just that, the stopping method we had to use for the Secant Method, will often stop the iterations even when the difference between its root and the actual root is greater than the ϵ.

Part 2: Now we will work on creating a graph to show the True Percent Error for the different root finding methods at each iteration number. These functions will be for the new equation: f(x) = e^x - 5x.

```
(*This cell sets up the equation for part 2.*)
equation4[x_] := (E^x) - 5 x
```

We will create four functions (one for each of the root finding methods) that takes an iteration number as an input and returns the absolute true percent error after that many iterations of the method has been performed.

```
In[∘]:= bisectionTechIterator[iteration_] := (
          computerRoot = r /. FindRoot[{equation4[r]}, {r, 2.5}];
          count = 0;
          a = 2;
          b = 3;
          While[count ≤ 1000, p = (a + (b - a) / 2);
           If[count == iteration, Break[]];
           myRoot = p;
           count ++ ;
           If[equation4[p] * equation4[a] > 0, a = p, b = p]];
          Return[N[(Abs[myRoot - computerRoot] / computerRoot) * 100, 5]]
         )

In[∘]:= newtonIterator[iteration_] := (
          count = 1;
          lastP = 2.5;
          While[count ≤ 250, p = N[lastP - (equation4[lastP] / equation4'[lastP]), 32];
           If[count == iteration, Break[]];
           count ++; lastP = p];
          computerRoot = r /. FindRoot[{equation4[r]}, {r, 2.5}];
          Return[N[(Abs[p - computerRoot] / computerRoot) * 100, 5]]
         )

        regulaFalsiIterator[iteration_] := (
          count = 1;
          a = 2;
          b = 3;
          computerRoot = r /. FindRoot[{equation4[r]}, {r, 2.5}];
          While[count ≤ 100, p = N[b - equation4[b] * (b - a) / (equation4[b] - equation4[a]), 32];
           If[count == iteration, Break[]];
           count ++;
           If[equation4[p] * equation4[b] < 0, a = b]; b = p];
          Return[N[(Abs[p - computerRoot] / computerRoot) * 100, 5]]
         )
```

```
In[•]:= secantIterator[iteration_] := (
          computerRoot = r /. FindRoot[{equation4[r]}, {r, 2.5}];
          count = 1;
          a = 2;
          b = 3;
          While[count ≤ 1000, p = N[b - equation4[b] * (b - a) / (equation4[b] - equation4[a]), 64];
           If[count == iteration, Break[]];
           count++;
           a = b; b = p; b = p];
          Return[N[(Abs[p - computerRoot] / computerRoot) * 100, 5]]
         )
```

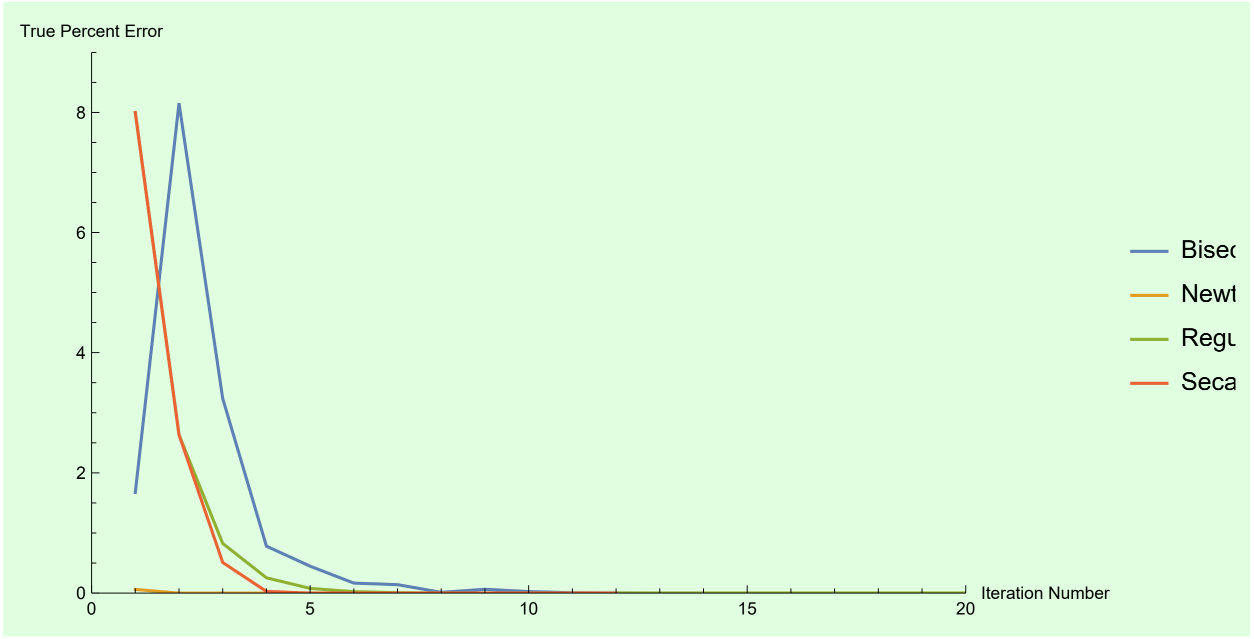
Now, we will create four datasets, one for each of the root finding methods. We will use these data to create graph, later.

```
In[•]:= bisectionError = Table[{i, bisectionTechIterator[i]}, {i, 1, 20}]

In[•]:= newtonError = Table[{i, newtonIterator[i]}, {i, 1, 20}]

In[•]:= regulaFalsiError = Table[{i, regulaFalsiIterator[i]}, {i, 1, 20}]

In[•]:= secantError = Table[{i, secantIterator[i]}, {i, 1, 12}]

Out[•]= {{1, 7.99966}, {2, 2.63523}, {3, 0.509696}, {4, 0.0289049}, {5, 0.000306872},
         {6, 1.85935 × 10^-7}, {7, 1.20513 × 10^-12}, {8, 0.}, {9, 0.}, {10, 0.}, {11, 0.}, {12, 0.}}
```

```
(*For the secant method,
the dataset is displayed above.The problem with doing high number of iterations of the
 secant method for this new equation is that we end up with a really small number
 in the denominator in the "p=N[b-equation4[b]*(b-a)/(equation4[b]-equation4[a])"
 part of the function. We have to estimate at places,
to make these iterations possible for computers to do in reasonable
 time.This leads to the low numbers in denominators being rounded to 0,
which leads to an error. This should not be a problem in comparing the
 Secant Method with the other methods, since by the fifth iteration,
we are already really close to a true percent error of 0. So,
I just stopped the iterations after 12.*)
```

```
(*This cell creates a graph to show the True Percent Error
 of the different root finding methods on the new equation.*)
ListLinePlot[{bisectionError, newtonError, regulaFalsiError, secantError},
 PlotRange → {{0, 20}, {0, 9}},
 AxesLabel → {"Iteration Number", "True Percent Error"}, PlotLegends →
  {"Bisection Method", "Newton's Method", "Regula Falsi Method", "Secant Method"}]
```

Out[∘]=



The Newton’s method seems to be the best, here. This is because the root in the interval (2,3) of the equation y = `(e^x) - 5 x` is pretty close to 2.5, which is very close to our first guess for the Newton’s Method.