Numerical Analysis I Final Project

Corbin T. Rochelle

November 21, 2022

1 Thoughts

Although my script file only needed to have 3 outputs, one for each function, mine has ten: one for the actual value of the root and three for each function. Each function has three outputs to analyze the usefulness of each of these functions, namely the output, the time to completion, and the absolute error. The performance of each method will be looked at using two different metrics, the time to completion and the error.

1.1 Iterated Inverse

The iterated inverse method is the new method introduced in this project. It is starkly different from the other methods that we are comparing it to, namely that this uses a codified method that has a finite endpoint (the bottom right index) while the others have a tolerance they try to achieve. This is the advantage and disadvantage of this method. If we turn our eyes to the output of the script (section 2), we see that this method had the fastest time to completion by one or two orders of magnitude; so, this is 10x or 100x faster than the other methods. However, if we look at the relative error of this function, we see that it is the highest of all three and because this method has a finite endpoint, these values cannot be changed, unlike changing the tolerance values of the other two functions. Therefore, it is of my opinion, that this method should be used for evaluations that must be preformed quickly and at low accuracy.

1.2 Newton's Method

Newton's Method was heavily discussed in class and was one of the main focuses of the semester; so I will spend less time discussing the function and go directly into the analysis. If we look at the time to completion of this method, we see it was in the middle of the pack; although this method was 10x slower than IIA, if we look at the relative error (which was forced down to machine epsilon by the tolerance) it is almost prefect, down to 10^{-16} which is nine orders of magnitude better than IIA. It and the bisection method have the same accuracy but bisection has a 10x slower run time. So we should always use Newton's Method over bisection method? No! This is because Newtons method also requires an initial guess. From this analysis, I conclude that Newton's method is best used when high accuracy is needed about a well known function (for the initial guess).

1.3 Bisection Method

This method is the slowest method out of the three, but is just as accurate as Newton's Method. From this analysis, this method should only be used when high accuracy is needed about an unknown function.

2 Output of My Script

```
actual =

        0.567143290409784


IIA =

                     0.3                       0                       0                       0
                     0.4        0.558547178934186                       0                       0
                     0.5         0.56504124724086        0.567110774981956                       0
                     0.6        0.567544584837301          0.5671460205948        0.567142353641323


IIA_time =

              6.39116e-07


IIA_diff =

        0.267143290409784        0.567143290409784        0.567143290409784        0.567143290409784
        0.167143290409784       0.00859611147559758        0.567143290409784        0.567143290409784
       0.0671432904097838       0.00210204316892393       3.2515427827895e-05        0.567143290409784
       0.0328567095902161      0.000401294427517573      2.73018501628641e-06      9.36768461246018e-07


NM =

        0.567143290409784


NM_time =

       3.3504815498155e-06


NM_diff =

     1.11022302462516e-16


B =

        0.567143290409784


B_time =

              9.80875e-05


B_diff =

     1.11022302462516e-16
```

3 project_opt1_IIA_Rochelle.m

```
function [Q] = project_opt1_IIA_Rochelle(x_vec,y_vec)
%IIAA_Corbin(x_vec,y_vec) Takes two input vectors and
%uses the interatied inverse approximation algorithm
%and outputs a matrix with the calculated values.
%
%x_vec: distinct numbers on the intervals
%y_vec: y_n = f(x_n)
%Q: output matrix

if (size(x_vec) ~= size(y_vec))
error('Invalid inputs. Size of x_vec does not equal y_vec.')
```

```
end

Q = zeros(size(y_vec,2));
Q(:,1) = x_vec;

for k = 2:size(y_vec,2)
for l = 2:k
Q(k,l)=(y_vec(k)*Q(k-1,l-1)-y_vec(k-l+1)*Q(k,l-1))/(y_vec(k)-y_vec(k-l+1));
end
end

end
```

4 project_opt1_script_Rochelle.m

```
% This is Corbin T Rochelle Script file for the final project.

format longg;
func = @(x) x-exp(-x);

% Actual Value
actual = fzero(func,0)

% Iterated Inverse
x_vec = [.3,.4,.5,.6];
y_vec = func(x_vec);
IIA = project_opt1_IIA_Rochelle(x_vec,y_vec)
ii = @() project_opt1_IIA_Rochelle(x_vec,y_vec);
IIA_time = timeit(ii)
IIA_diff = abs(actual - IIA)

% Newton's Method
dfunc = @(x) 1+exp(-x);
guess = .45;
tol = 10.^(-15);
NM = NM_Diegel(guess,func,dfunc,tol,100)
nm = @() NM_Diegel(guess,func,dfunc,tol,100);
NM_time = timeit(nm)
NM_diff = abs(actual - NM)

% Bisection Method
B = bisect(func,0,1,tol,100)
b = @() bisect(func,0,1,tol,100);
B_time = timeit(b)
B_diff = abs(actual - B)
```