Zubin Mishra
UID: 604644805
CEE/MAE M20
May 30, 2017

Homework 8

1 The Ranked-Choice Vote

1.1 Introduction

The goal in this problem is to develop a program that performs a ranked-choice vote counting process on a large set of election data until a winner has been found. This is accomplished through creating a function to remove the losing candidate from consideration until a winner with > 50% of the votes is found. It will be discussed how different tie-breaking strategies affect the results and how the results differ when using a simple weighted sum voting system.

1.2 Model and Methods

The script first loads in the voting data using MATLAB's load function. It then calculates the weighted sum voting system counts for later comparison:

```
ws_counts = zeros(max(max(votes)),1);
for j = 1:size(votes, 2)
    for i = 1:size(votes, 1)
        ws_counts(votes(i, j)) = ws_counts(votes(i, j)) + (size(votes,2)-
j+1);
    end
end
```

The script then sets the initial conditions for the loop and creates the array for the counts. It then prints out the top line of the results table for the output. The script then enters a while loop until a winner is found. Within the while loop, the script first resets the counts array to zeros:

```
counts(:) = 0;
```

It then counts the 1st-ranked votes for the ranked-choice system:

```
for i = 1:size(votes, 1)
    counts(votes(i, 1)) = counts(votes(i, 1)) + 1;
end
```

If a 50% majority is not found, then the losing candidate is removed using the removeCandidate function. Otherwise, the loop condition is set to false:

```
if max(counts)/sum(counts) <= 0.5
    minimum = min(counts(counts>0));
    for j = 1:length(counts)
        if counts(j) == minimum
            losingCandidate = j;
            break; %First losing candidate found is removed
        end
```

```
    end
    votes = removeCandidate(votes, losingCandidate);
else
    loopCondition = false;
end
```

The results for that wound in the ranked-choice system is then printed out and the round number is incremented. Once a winner has been found, the winner printed out:

```
[maximum winner] = max(counts);
fprintf('Winning Candidate: %0.0f\n', winner);
```

The function, removeCandidate, starts by creating a 2D array with one less column than the input votes array:

```
newVotes = zeros(size(votes, 1), size(votes, 2)-1);
```

It will then fill in this new 2D array with candidates that are not losingCandidate by iterating through the input votes and keeping track of the next position to be filled in in the new 2D array:

```
for i = 1:size(votes, 1)
    pos = 1;
    for j = 1:size(votes, 2)
        if votes(i, j) ~= losingCandidate
            newVotes(i, pos) = votes(i, j);
            pos = pos + 1;
        end
    end
end
```

It then returns the filled in new 2D array.

1.3 Results and Calculations

With votes1, the following output is produced:

	1	2	3	4	5	6
Round 1 Totals:	235	624	650	4625	3065	801
Round 2 Totals:	0	673	694	4678	3104	851
Round 3 Totals:	0	0	847	4847	3279	1027
Round 4 Totals:	0	0	0	5137	3554	1309

Winning Candidate: 4

With votes2 and breaking the tie by eliminating the earliest found losing candidate, the following output is produced:

	1	2	3	4	5	6	7	8
Round 1 Totals:	313	344	1827	607	321	1077	1236	1775
Round 2 Totals:	0	379	1867	647	379	1121	1284	1823
Round 3 Totals:	0	0	1942	728	429	1179	1344	1878

Round 4 Totals:	0	0	2027	812	0	1268	1432	1961
Round 5 Totals:	0	0	2234	0	0	1469	1634	2163
Round 6 Totals:	0	0	2713	0	0	0	2116	2671
Round 7 Totals:	0	0	3746	0	0	0	0	3754

Winning Candidate: 8

With votes2 and breaking the tie by eliminating the latest found losing candidate, the following output is produced:

	1	2	3	4	5	6	7	8
Round 1 Totals:	313	344	1827	607	321	1077	1236	1775
Round 2 Totals:	0	379	1867	647	379	1121	1284	1823
Round 3 Totals:	0	450	1933	701	0	1181	1349	1886
Round 4 Totals:	0	0	2027	812	0	1268	1432	1961
Round 5 Totals:	0	0	2234	0	0	1469	1634	2163
Round 6 Totals:	0	0	2713	0	0	0	2116	2671
Round 7 Totals:	0	0	3746	0	0	0	0	3754

Winning Candidate: 8

With votes1, the weighted sum produces the following counts and winner:

1:	30706
2:	31818
3:	31765
4:	43974
5:	39197
6:	32540

Winning Candidate: 4

With votes2, the weighted sum produces the following counts and winner:

1:	31339
2:	31356
3:	37275
4:	32451
5:	31365
6:	34299
7:	34791
8:	37124

Winning Candidate: 3

1.4 Discussion

When tie-breaking for votes2, there were not any significant differences between favoring one candidate over the other. The two candidates that were tied were candidates 2 and 5. In one case, candidate 2 was eliminated, and then in the next round, candidate 5 was. In the other case, candidate 5 was eliminated first, and in the next round candidate 2 was eliminated. In both cases candidate 8 was the overall winner.

When using a weighted sum system to count the votes, for votes1, the winning candidate stayed the same (candidate 4). However, for votes 2, the winning candidate ended up being candidate 3 instead of candidate 8. This discrepancy may mean that the “fairness” of ranked-choice voting is compromised in this case. The results from the weighted sum show that candidate 3 is more generally popular than candidate. However, in this case, candidate 8 comes in a close second place using the weighted sum system, which re-legitimizes the ranked-choice voting to some extent.

It is important to note that there are cases where ranked-choice voting has results that are completely at odds with the weighted sum system. For example, say only one person has candidate 1 as their first choice, but everyone else has candidate 1 as their second choice. With ranked-choice voting, candidate 1 would be immediately eliminated, but with the weighted sum system, candidate 1 would have a very high point total, reflecting his or her popularity as near-universal second choice. Ranked-choice voting cannot take this popularity into account and, in fact, entirely dismisses it.

2 Newton’s Method

2.1 Introduction

The goal in this problem is to develop a program that performs Newton's method on a particular function at several different initial guesses. This is accomplished through the use of a function handle. A function will be created to perform newton’s method that takes in this function handle as one of its arguments. It will be discussed how changing the initial guess and the tolerance affect the zeros found and the number of evaluations needed.

2.2 Model and Methods

The script begins by establishing the function handle and the x0 array:

```
f = @(x) 816*x.^3 - 3835*x.^2 + 6000*x - 3125;
x0 = linspace(1.43, 1.71, 29);
```

It then iterates through each x0 value and performs Newton’s method using the function Newton and prints out the results:

```
for i = 1:29
    [x n] = Newton(f, x0(i), 10^-3, 20);
    fprintf('x0 = %0.2f, evals = %2.0f, xc = %0.6f\n', x0(i), n, x);
end
```

The function Newton begins by establishing some values to use while conducting Newton's method, including the perturbation size for the central difference approximation:

```
h = 10^-5;
xc = x0;
fEvals = 0;
loopCondition = true;
```

It then enters a while loop to conduct the evaluations for Newton's method. Within the while loop, first the derivative at the current x-value is found using the central difference approximation:

$$f'(x_i) \approx \frac{f(x_i+h) - f(x_i-h)}{2h}$$

Then the next x-value to look at is found with the following formula:

$$x_+ = x_c - \frac{f(x_c)}{f'(x_c)}$$

The while loop then performs the accuracy check and increments the number of evaluations. When the while loop ends, the function is exited also. In code:

```
    while loopCondition && fEvals ~= fEvalMax
        % Calculate derivate with central difference approximation
        df = (f(xc+h)-f(xc-h))/(2*h);

        % Find next x-value to examine
        xc = xc - f(xc)/df;

        % Check accuracy condition
        if abs(f(xc)) <= delta
            loopCondition = false;
        end

        % Increment number of evals
        fEvals = fEvals + 1;
    end
```

2.3 Results and Calculations

With delta = 10^{-7}, the following output is produced:

x0 = 1.43, evals = 5, xc = 1.470588
x0 = 1.44, evals = 4, xc = 1.470588
x0 = 1.45, evals = 4, xc = 1.470588
x0 = 1.46, evals = 4, xc = 1.470588
x0 = 1.47, evals = 2, xc = 1.470588
x0 = 1.48, evals = 4, xc = 1.470588
x0 = 1.49, evals = 5, xc = 1.470588
x0 = 1.50, evals = 6, xc = 1.470588

x0 = 1.51, evals = 18, xc = 1.666667
x0 = 1.52, evals = 8, xc = 1.470588
x0 = 1.53, evals = 4, xc = 1.562500
x0 = 1.54, evals = 3, xc = 1.562500
x0 = 1.55, evals = 3, xc = 1.562500
x0 = 1.56, evals = 2, xc = 1.562500
x0 = 1.57, evals = 2, xc = 1.562500
x0 = 1.58, evals = 3, xc = 1.562500
x0 = 1.59, evals = 3, xc = 1.562500
x0 = 1.60, evals = 4, xc = 1.562500
x0 = 1.61, evals = 6, xc = 1.666667
x0 = 1.62, evals = 8, xc = 1.470588
x0 = 1.63, evals = 7, xc = 1.666667
x0 = 1.64, evals = 5, xc = 1.666667
x0 = 1.65, evals = 4, xc = 1.666667
x0 = 1.66, evals = 3, xc = 1.666667
x0 = 1.67, evals = 3, xc = 1.666667
x0 = 1.68, evals = 4, xc = 1.666667
x0 = 1.69, evals = 4, xc = 1.666667
x0 = 1.70, evals = 4, xc = 1.666667
x0 = 1.71, evals = 5, xc = 1.666667

With delta = 10^{-5}, the following output is produced:

x0 = 1.43, evals = 4, xc = 1.470588
x0 = 1.44, evals = 4, xc = 1.470588
x0 = 1.45, evals = 4, xc = 1.470588
x0 = 1.46, evals = 3, xc = 1.470588
x0 = 1.47, evals = 2, xc = 1.470588
x0 = 1.48, evals = 3, xc = 1.470588
x0 = 1.49, evals = 4, xc = 1.470588
x0 = 1.50, evals = 6, xc = 1.470588
x0 = 1.51, evals = 17, xc = 1.666667
x0 = 1.52, evals = 8, xc = 1.470588
x0 = 1.53, evals = 3, xc = 1.562500
x0 = 1.54, evals = 3, xc = 1.562500
x0 = 1.55, evals = 2, xc = 1.562500
x0 = 1.56, evals = 2, xc = 1.562500
x0 = 1.57, evals = 2, xc = 1.562500
x0 = 1.58, evals = 2, xc = 1.562501
x0 = 1.59, evals = 3, xc = 1.562500
x0 = 1.60, evals = 3, xc = 1.562501
x0 = 1.61, evals = 6, xc = 1.666667
x0 = 1.62, evals = 8, xc = 1.470588
x0 = 1.63, evals = 7, xc = 1.666667
x0 = 1.64, evals = 5, xc = 1.666667

x0 = 1.65, evals = 4, xc = 1.666667
x0 = 1.66, evals = 3, xc = 1.666667
x0 = 1.67, evals = 2, xc = 1.666667
x0 = 1.68, evals = 3, xc = 1.666667
x0 = 1.69, evals = 4, xc = 1.666667
x0 = 1.70, evals = 4, xc = 1.666667
x0 = 1.71, evals = 4, xc = 1.666667

With delta = 10^{-3}, the following output is produced:

x0 = 1.43, evals = 3, xc = 1.470523
x0 = 1.44, evals = 3, xc = 1.470574
x0 = 1.45, evals = 3, xc = 1.470587
x0 = 1.46, evals = 2, xc = 1.470557
x0 = 1.47, evals = 1, xc = 1.470583
x0 = 1.48, evals = 2, xc = 1.470536
x0 = 1.49, evals = 3, xc = 1.470543
x0 = 1.50, evals = 5, xc = 1.470587
x0 = 1.51, evals = 17, xc = 1.666667
x0 = 1.52, evals = 7, xc = 1.470581
x0 = 1.53, evals = 3, xc = 1.562500
x0 = 1.54, evals = 2, xc = 1.562507
x0 = 1.55, evals = 2, xc = 1.562500
x0 = 1.56, evals = 1, xc = 1.562511
x0 = 1.57, evals = 1, xc = 1.562484
x0 = 1.58, evals = 2, xc = 1.562501
x0 = 1.59, evals = 2, xc = 1.562535
x0 = 1.60, evals = 3, xc = 1.562501
x0 = 1.61, evals = 5, xc = 1.666672
x0 = 1.62, evals = 7, xc = 1.470585
x0 = 1.63, evals = 6, xc = 1.666668
x0 = 1.64, evals = 4, xc = 1.666671
x0 = 1.65, evals = 3, xc = 1.666671
x0 = 1.66, evals = 2, xc = 1.666675
x0 = 1.67, evals = 2, xc = 1.666667
x0 = 1.68, evals = 2, xc = 1.666723
x0 = 1.69, evals = 3, xc = 1.666669
x0 = 1.70, evals = 3, xc = 1.666683
x0 = 1.71, evals = 4, xc = 1.666667

2.4 Discussion

Changing x0 changes which root Newton's method will find. Typically the root found is the one closest to x0, but this is not always the case. If the slope of the tangent line at the initial guess is close to 0, then the next x-value that Newton's method will look at will be far away from the initial guess, and as Newton's method progresses, it may converge on the root closer to this second x-value. This is what happens in the case of x0 = 1.62. The next x-value that is looked at is 1.2434, which is closer to the root at 1.47 than it is to the closest root to the initial guess at 1.67. As such, the root at 1.47 is found. Something similar happens with x0 = 1.51, for which the second x-value looked at is 18.9123. This additionally results in several more evaluations being needed to converge, due to how far off the value is from any roots.

Changing delta did not change results by much. Making delta larger reduced the accuracy of Newton's method and resulted in fewer required evaluations. This is especially apparent when comparing delta = 10^{-7} and delta = 10^{-3}. With delta = 10^{-7}, the roots found do not differ up to 6 decimal places, and every case requires at least 2 evaluations. With delta = 10^{-3}, the roots found start to differ at the fourth decimal place, and there are several cases where the number of evaluations is just 1.