Root Finding Speed Analysis Report

The Project

When given a chance to choose a project, I thought back to a long-time question I have had in this class:

"So what is the best method?"

and while the actual answer is always "it depends", I thought that if I could time all the methods in a wide variety of scenarios, I could start deducting trends from the data.

To be more precise, my goal was to time 4 of the root-finding methods that we did in class: Steffensen, secants, newtons, and mullers with a large number of functions, and find trends in the data, especially looking into how they performed as the tolerance expectation became more and more demanding.

Introducing the Gladiators!

To begin, I took the functions that I had used from the class, and cleaned them up some, trying to make them as fast as I could, after which I used octaves tik and tok functions: tik allows you to start a timer, and tok stop it, storing the time passed within itself.

```
function [x,t,e] = steffensen(f,x0,TOL,N0)

  tic;
  i=1;
  A=x0;
  B=f(A);



  while (i<=N0)
      % The first bit here is just getting 3 initial values by plugging in
the equation a few times
    x0=A;
    x1=B;
    x2=f(x1);
    if (abs(x2-x1)<TOL)

      x=x2;
      t = toc;
      e = i-1;
      return
    end%if
    % Once u have some values u can now use Aitkens to get a good
approximation
    x=x0-(x1-x0)^2/(x2-2*x1+x0)
    if (abs(x-x2)<TOL)
      x=x2;
      t = toc;
      e = i-1;
      return
    end%if
    % This part now sets the new values for the next iteration, turning the
Aitkens into a Stephenson
    A=x;
    B=f(A);
    i=i+1;
  end%while
  x=NaN;
  t =NaN;
  e=NaN;
end%function
```

As we can see, Stephenson was pretty straightforward, receiving a function, starting value, tolerance, and maxits, and outputting the value of the root, the time it took to calculate, and a total number of g(x) function calculations.

```
function [c, t, e] = seededSecant(f, x0, x1, TOL, N0)
    tic;
    i = 2;
    f_x0 = f(x0);
    f_x1 = f(x1);

    while (i <= N0)
        if abs(f_x1 - f_x0) < eps
            % The difference between function values is too small
            c = NaN;
            t = toc;
            e = NaN;
            return;
        end

        c = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0);

        if abs(c - x1) < TOL
            t = toc;
            e = i;
            return;
        end

        x0 = x1;
        f_x0 = f_x1;
        x1 = c;
        f_x1 = f(c);
        i = i + 1;
    end

    % If the maximum number of iterations is reached without convergence
    c = NaN;
    t = toc;
    e = NaN;
end
```

Steffenson is the same, just having the modifications to add the timing and counting capabilities.

```

function [x,t,e] = mullers(f,p0,p1,p2,TOL,N)

    tic;
    i = 1;
    while (i<=N)

        x1=p0;
        x2=p1;
        x3=p2;
        y1=f(x1);
        y2=f(x2);
        y3=f(x3);

        A = [x1^2, x1, 1; x2^2, x2, 1; x3^2, x3, 1];

        % Solve the least squares problem
        coeffs = A \ [y1; y2; y3];

        % Grab the coefficients
        a = coeffs(1);
        b = coeffs(2);
        c = coeffs(3);

        if (b >= 0)
```

```
            r1 = (-b - sqrt(b*b - 4*a*c))/(2*a);
            r2 = (2*c)/(-b - sqrt(b*b - 4*a*c));

            if (abs(-b - sqrt(b*b - 4*a*c)) > abs(-b + sqrt(b*b - 4*a*c)))
                p3 = r2;
            else
                p3 = r1;
            end

        else
            r1 = (-b + sqrt(b*b - 4*a*c))/(2*a);
            r2 = (2*c)/(-b + sqrt(b*b - 4*a*c));

            if (abs(-b + sqrt(b*b - 4*a*c)) > abs(-b - sqrt(b*b - 4*a*c)))
                p3 = r2;
            else
                p3 = r1;
            end

        end

        p0 = p1;
        p1 = p2;
        p2 = p3;

        if (abs(p2 - p1) < TOL)
            x = p3;
            t = toc;
            e = 3*i;
            return;
        end

        i = i + 1;
    end

    x = NaN;
    t = NaN;
    e = NaN;
end
```

Mullers is similar, but it has the advantage of using the least squares method to interpolate the polynomial for a little bit of a speed boost, as we will see soon though it does not help it too much when it comes to efficiency.

```
function [x, t, e] = newtons(g, x0, TOL, N0)
    % Initial step size based on tolerance
    h = sqrt(eps); % Square root of machine epsilon

    i = 1;
    tic;
    while (i <= N0)
        g_val = feval(g, x0); % Evaluate g(x0)

        % Richardson extrapolation with central difference formula
        g_prime_val_h1 = (feval(g, x0 + h) - feval(g, x0 - h)) / (2 * h);
        g_prime_val_h2 = (feval(g, x0 + h/2) - feval(g, x0 - h/2)) / h;
        g_prime_val = (4*g_prime_val_h2 - g_prime_val_h1) / 3;

        x = x0 - g_val / g_prime_val; % Newton's method formula
        if (abs(x - x0) < TOL)
            t = toc;
            e = 5*(i - 1);
            return
        end




        i = i + 1;
        x0 = x;
    end
    x = NaN;
    t = NaN;
    e = NaN;
end
```

Newtons by far is the one that gave me the biggest difficulty and has the biggest changes to it. While I could have my special assistant do the calculations, I was still very reluctant to give it the derivative, as it is a huge unfair boost to its time efficiency, and would not properly show the speed and costs of the function.

After some research and deliberation, I settled for an extremely good approximation of the derivative using the central difference formula and Richardson's Extrapolation. Richardson's Extrapolation uses Long's law, and is conceptually similar to Simpsons! Instead of using approximations derived by different methods to get a third better average, it takes advantage of using two step sizes and knowing the rate at which the error decreases relative to step size to make a third better approximation from the first two.

Introducing the Beasts!

I decided to get my little assistant chatgpt for this, and had it define 50 polynomials of degrees two through five, 50 of degrees five through twenty, 50 trigonometric functions, 50 logarithmic functions, 50 exponential functions, and 50 functions that combined all of the above. Below is the list:

```
polys = {
    @(x) x.^2 - x,                                 % Degree 2, f(x) = x^2 - x
    @(x) x.^3 + x.^2 - 2,                          % Degree 3, f(x) = x^3 + x^2 - 2
    @(x) -x.^4 + x.^2 - 3,                         % Degree 4, f(x) = -x^4 + x^2 - 3
    @(x) 2*x.^5 - 3*x.^4 + x.^3 + 1,               % Degree 5, f(x) = 2x^5 - 3x^4 + x^3 + 1
    @(x) -2*x.^2 + 5*x - 1,                        % Degree 2, f(x) = -2x^2 + 5x - 1
    @(x) 3*x.^3 - 4*x.^2 + 3*x + 2,                % Degree 3, f(x) = 3x^3 - 4x^2 + 3x + 2
    @(x) -4*x.^4 + 5*x.^3 + 4*x.^2 - 3*x + 1,      % Degree 4, f(x) = -4x^4 + 5x^3 + 4x^2 - 3x + 1
    @(x) 5*x.^5 - 2*x.^4 + x.^3 - x.^2 + x,        % Degree 5, f(x) = 5x^5 - 2x^4 + x^3 - x^2 + x
    @(x) x.^2 - 3*x + 2,                           % Degree 2, f(x) = x^2 - 3x + 2
    @(x) x.^3 + 2*x.^2 - 4*x - 1,                  % Degree 3, f(x) = x^3 + 2x^2 - 4x - 1
    @(x) -x.^4 - 5*x.^3 + x.^2 + 4*x - 3,          % Degree 4, f(x) = -x^4 - 5x^3 + x^2 + 4x - 3
    @(x) 2*x.^5 + 3*x.^4 - 4*x.^3 - 5*x.^2 + 6*x,  % Degree 5, f(x) = 2x^5 + 3x^4 - 4x^3 - 5x^2 + 6x
    @(x) -3*x.^2 + 2*x + 5,                       % Degree 2, f(x) = -3x^2 + 2x + 5
    @(x) 4*x.^3 - x.^2 - 3*x + 2,                  % Degree 3, f(x) = 4x^3 - x^2 - 3x + 2
    @(x) -5*x.^4 + 4*x.^3 + 2*x.^2 - x + 3,        % Degree 4, f(x) = -5x^4 + 4x^3 + 2x^2 - x + 3
    @(x) 6*x.^5 - 3*x.^4 + 2*x.^3 - x.^2 + x + 1,  % Degree 5, f(x) = 6x^5 - 3x^4 + 2x^3 - x^2 + x + 1
    @(x) x.^2 + x + 1,                             % Degree 2, f(x) = x^2 + x + 1
    @(x) x.^3 - 2*x.^2 - x + 2,                    % Degree 3, f(x) = x^3 - 2x^2 - x + 2
    @(x) -x.^4 + 3*x.^3 - 2*x.^2 - 3*x + 1,         % Degree 4, f(x) = -x^4 +
3x^3 - 2x^2 - 3x + 1
    @(x) 2*x.^5 - x.^4 - 3*x.^3 + x.^2 + 2*x - 1,  % Degree 5, f(x) = 2x^5 -
x^4 - 3x^3 + x^2 + 2x - 1
    @(x) -x.^2 + 3*x + 2,                           % Degree 2, f(x) = -x^2 +
3x + 2
    @(x) x.^3 + x.^2 + x - 1,                       % Degree 3, f(x) = x^3 +
x^2 + x - 1
    @(x) -x.^4 + x.^2 - 2*x + 1,                    % Degree 4, f(x) = -x^4 +
x^2 - 2x + 1
    @(x) 2*x.^5 + 3*x.^4 + 4*x.^3 - 5*x.^2 - x + 1, % Degree 5, f(x) = 2x^5
+ 3x^4 + 4x^3 - 5x^2 - x + 1
    @(x) -x.^2 - 2*x - 1,                           % Degree 2, f(x) = -x^2 -
2x - 1
    @(x) x.^3 + 3*x.^2 + 2*x + 3,                   % Degree 3, f(x) = x^3 +
3x^2 + 2x + 3
    @(x) -x.^4 + 2*x.^3 + x + 2,                    % Degree 4, f(x) = -x^4 +
2x^3 + x + 2
    @(x) 2*x.^5 + x.^4 + x.^3 - 3*x.^2 - 4*x - 1,   % Degree 5, f(x) = 2x^5
+ x^4 + x^3 - 3x^2 - 4x - 1
    @(x) x.^2 + 2*x + 3,                            % Degree 2, f(x) = x^2 +
2x + 3
    @(x) x.^3 + 2*x.^2 + 3*x + 4,                   % Degree 3, f(x) = x^3 +
2x^2 + 3x + 4
    @(x) -x.^4 - 2*x.^3 + 3*x.^2 + 4*x + 5,         % Degree 4, f(x) = -x^4 -
2x^3 + 3x^2 + 4x + 5
    @(x) 2*x.^5 + x.^4 - 2*x.^3 + 3*x.^2 - 4*x + 5, % Degree 5, f(x) = 2x^5
+ x^4 - 2x^3 + 3x^2 - 4x + 5
    @(x) -x.^2 + 2*x + 4,                           % Degree 2, f(x) = -x^2 +
2x + 4
    @(x) x.^3 + x.^2 - x + 5,                       % Degree 3, f(x) = x^3 +
x^2 - x + 5
    @(x) -x.^4 + 3*x.^3 - 2*x.^2 + 3*x - 1,         % Degree 4, f(x) = -x^4 +
3x^3 - 2x^2 + 3x - 1
    @(x) 2*x.^5 - 3*x.^4 + x.^3 + 2*x.^2 + x - 3,  % Degree 5, f(x) = 2x^5 -
3x^4 + x^3 + 2x^2 + x - 3
    @(x) -x.^2 - x + 2,                             % Degree 2, f(x) = -x^2 -
x + 2
    @(x) x.^3 + 2*x.^2 - 3*x + 4,                   % Degree 3, f(x) = x^3 +
2x^2 - 3x + 4
    @(x) -x.^4 + x.^3 + 3*x + 1,                    % Degree 4, f(x) = -x^4 +
x^3 + 3x + 1
    @(x) 2*x.^5 - 3*x.^4 - 4*x + 5,                 % Degree 5, f(x) = 2x^5 -
```

```
3x^4 - 4x + 5
    @(x) x.^2 - x - 1,                          % Degree 2, f(x) = x^2 -
x - 1
    @(x) x.^3 - x.^2 + 2*x - 3,                 % Degree 3, f(x) = x^3 -
x^2 + 2x - 3
    @(x) -x.^4 + x.^3 + x - 4,                  % Degree 4, f(x) = -x^4 +
x^3 + x - 4
    @(x) 2*x.^5 - 3*x.^4 + x.^2 - x + 1,        % Degree 5, f(x) = 2x^5 -
3x^4 + x^2 - x + 1
    @(x) -x.^2 + x + 3,                         % Degree 2, f(x) = -x^2 +
x + 3
    @(x) x.^3 + 2*x + 1,                        % Degree 3, f(x) = x^3 +
2x + 1
    @(x) -x.^4 + x.^3 + 3*x + 2,                % Degree 4, f(x) = -x^4 +
x^3 + 3x + 2
    @(x) x.^4 + 2*x + 1,                        % Degree 4, f(x) = x^4 +
2x + 1
    @(x) -x.^5 + x.^3 + 3*x + 2,                % Degree 5, f(x) = -x^5 +
x^3 + 3x + 2
    @(x) 2*x.^5 - 3*x.^4 + 4*x.^3 - 5*x.^2 + 6*x   % Degree 5, f(x) = 2x^5 -
3x^4 + 4x^3 - 5x^2 + 6x
};

polynomialsLarge = {
    @(x) 2*x^8 - 3*x^9 + 5*x^10 - 7*x^11 + 4*x^12 - 6*x^13 + 8*x^14 - 9*x^15
+ 10*x^16 - 11*x^17 + 12*x^18 - 13*x^19 + 14*x^20, % Degree 20
    @(x) -4*x^6 + 3*x^7 - 2*x^8 + 5*x^9 - 7*x^10 + 8*x^11 - 9*x^12 + 6*x^13
- 11*x^14 + 12*x^15 - 13*x^16 + 14*x^17 - 15*x^18 + 16*x^19 + 17*x^20, %
Degree 20
    @(x) 5*x^5 + 6*x^6 - 3*x^7 + 8*x^8 - 9*x^9 + 10*x^10 + 4*x^11 - 12*x^12
+ 13*x^13 - 14*x^14 + 15*x^15 + 16*x^16 + 17*x^17 - 18*x^18 - 19*x^19 +
20*x^20, % Degree 20
    @(x) -6*x^9 + 7*x^10 - 8*x^11 + 9*x^12 - 10*x^13 + 11*x^14 - 12*x^15 +
13*x^16 - 14*x^17 + 15*x^18 - 16*x^19 + 17*x^20, % Degree 20
    @(x) 8*x^7 - 9*x^8 + 10*x^9 - 11*x^10 + 12*x^11 - 13*x^12 + 14*x^13 -
15*x^14 + 16*x^15 - 17*x^16 + 18*x^17 - 19*x^18 + 20*x^19, % Degree 19
    @(x) 3*x^8 - 4*x^9 + 5*x^10 - 6*x^11 + 7*x^12 - 8*x^13 + 9*x^14 -
10*x^15 + 11*x^16 - 12*x^17 + 13*x^18 - 14*x^19 + 15*x^20, % Degree 20
    @(x) 9*x^5 - 10*x^6 + 11*x^7 - 12*x^8 + 13*x^9 - 14*x^10 + 15*x^11 -
16*x^12 + 17*x^13 - 18*x^14 + 19*x^15 - 20*x^16, % Degree 16
    @(x) -5*x^6 + 6*x^7 - 7*x^8 + 8*x^9 - 9*x^10 + 10*x^11 - 11*x^12 +
12*x^13 - 13*x^14 + 14*x^15 - 15*x^16 + 16*x^17, % Degree 17
    @(x) 6*x^7 - 7*x^8 + 8*x^9 - 9*x^10 + 10*x^11 - 11*x^12 + 12*x^13 -
```

```
13*x^14 + 14*x^15 - 15*x^16 + 16*x^17 - 17*x^18 + 18*x^19 + 19*x^20, %
Degree 20
    @(x) -7*x^9 + 8*x^10 - 9*x^11 + 10*x^12 - 11*x^13 + 12*x^14 - 13*x^15 +
14*x^16 - 15*x^17 + 16*x^18 - 17*x^19 + 18*x^20, % Degree 20
    @(x) -3*x^6 + 4*x^7 - 5*x^8 + 6*x^9 - 7*x^10 + 8*x^11 - 9*x^12 + 10*x^13
- 11*x^14 + 12*x^15 - 13*x^16 + 14*x^17 - 15*x^18 + 16*x^19 - 17*x^20, %
Degree 20
    @(x) 4*x^5 - 5*x^6 + 6*x^7 - 7*x^8 + 8*x^9 - 9*x^10 + 10*x^11 - 11*x^12
+ 12*x^13 - 13*x^14 + 14*x^15 - 15*x^16 + 16*x^17 - 17*x^18 + 18*x^19, %
Degree 19
    @(x) 5*x^8 - 6*x^9 + 7*x^10 - 8*x^11 + 9*x^12 - 10*x^13 + 11*x^14 -
12*x^15 + 13*x^16 - 14*x^17 + 15*x^18 - 16*x^19 + 17*x^20, % Degree 20
    @(x) -6*x^7 + 7*x^8 - 8*x^9 + 9*x^10 - 10*x^11 + 11*x^12 - 12*x^13 +
13*x^14 - 14*x^15 + 15*x^16 - 16*x^17 + 17*x^18 - 18*x^19 + 19*x^20, %
Degree 20
    @(x) 7*x^6 - 8*x^7 + 9*x^8 - 10*x^9 + 11*x^10 - 12*x^11 + 13*x^12 -
14*x^13 + 15*x^14 - 16*x^15 + 17*x^16 - 18*x^17 + 19*x^18 - 20*x^19, %
Degree 19
    @(x) -8*x^5 + 9*x^6 - 10*x^7 + 11*x^8 - 12*x^9 + 13*x^10 - 14*x^11 +
15*x^12 - 16*x^13 + 17*x^14 - 18*x^15 + 19*x^16 - 20*x^17 + 21*x^18, %
Degree 18
    @(x) 9*x^6 - 10*x^7 + 11*x^8 - 12*x^9 + 13*x^10 - 14*x^11 + 15*x^12 -
16*x^13 + 17*x^14 - 18*x^15 + 19*x^16 - 20*x^17 + 21*x^18 - 22*x^19, %
Degree 19
    @(x) -10*x^5 + 11*x^6 - 12*x^7 + 13*x^8 - 14*x^9 + 15*x^10 - 16*x^11 +
17*x^12 - 18*x^13 + 19*x^14 - 20*x^15 + 21*x^16 - 22*x^17 + 23*x^18, %
Degree 18
    @(x) 11*x^6 - 12*x^7 + 13*x^8 - 14*x^9 + 15*x^10 - 16*x^11 + 17*x^12 -
18*x^13 + 19*x^14 - 20*x^15 + 21*x^16 - 22*x^17 + 23*x^18 - 24*x^19, %
Degree 19
    @(x) -12*x^5 + 13*x^6 - 14*x^7 + 15*x^8 - 16*x^9 + 17*x^10 - 18*x^11 +
19*x^12 - 20*x^13 + 21*x^14 - 22*x^15 + 23*x^16 - 24*x^17 + 25*x^18, %
Degree 18
    @(x) 13*x^4 - 14*x^5 + 15*x^6 - 16*x^7 + 17*x^8 - 18*x^9 + 19*x^10 -
20*x^11 + 21*x^12 - 22*x^13 + 23*x^14 - 24*x^15 + 25*x^16 - 26*x^17 +
27*x^18 - 28*x^19 + 29*x^20, % Degree 20
    @(x) -14*x^3 + 15*x^4 - 16*x^5 + 17*x^6 - 18*x^7 + 19*x^8 - 20*x^9 +
21*x^10 - 22*x^11 + 23*x^12 - 24*x^13 + 25*x^14 - 26*x^15 + 27*x^16 -
28*x^17 + 29*x^18 - 30*x^19 + 31*x^20, % Degree 20
    @(x) 15*x^4 - 16*x^5 + 17*x^6 - 18*x^7 + 19*x^8 - 20*x^9 + 21*x^10 -
22*x^11 + 23*x^12 - 24*x^13 + 25*x^14 - 26*x^15 + 27*x^16 - 28*x^17 +
29*x^18 - 30*x^19 + 31*x^20, % Degree 20
    @(x) -16*x^3 + 17*x^4 - 18*x^5 + 19*x^6 - 20*x^7 + 21*x^8 - 22*x^9 +
23*x^10 - 24*x^11 + 25*x^12 - 26*x^13 + 27*x^14 - 28*x^15 + 29*x^16 -
30*x^17 + 31*x^18 - 32*x^19, % Degree 19
    @(x) 17*x^4 - 18*x^5 + 19*x^6 - 20*x^7 + 21*x^8 - 22*x^9 + 23*x^10 -
24*x^11 + 25*x^12 - 26*x^13 + 27*x^14 - 28*x^15 + 29*x^16 - 30*x^17 +
31*x^18 - 32*x^19 + 33*x^20, % Degree 20
    @(x) -18*x^3 + 19*x^4 - 20*x^5 + 21*x^6 - 22*x^7 + 23*x^8 - 24*x^9 +
25*x^10 - 26*x^11 + 27*x^12 - 28*x^13 + 29*x^14 - 30*x^15 + 31*x^16 -
32*x^17 + 33*x^18 - 34*x^19, % Degree 19
    @(x) 19*x^4 - 20*x^5 + 21*x^6 - 22*x^7 + 23*x^8 - 24*x^9 + 25*x^10 -
26*x^11 + 27*x^12 - 28*x^13 + 29*x^14 - 30*x^15 + 31*x^16 - 32*x^17 +
33*x^18 - 34*x^19 + 35*x^20, % Degree 20
    @(x) -20*x^3 + 21*x^4 - 22*x^5 + 23*x^6 - 24*x^7 + 25*x^8 - 26*x^9 +
27*x^10 - 28*x^11 + 29*x^12 - 30*x^13 + 31*x^14 - 32*x^15 + 33*x^16 -
34*x^17 + 35*x^18 - 36*x^19, % Degree 19
    @(x) 21*x^4 - 22*x^5 + 23*x^6 - 24*x^7 + 25*x^8 - 26*x^9 + 27*x^10 -
28*x^11 + 29*x^12 - 30*x^13 + 31*x^14 - 32*x^15 + 33*x^16 - 34*x^17 +
35*x^18 - 36*x^19 + 37*x^20, % Degree 20
    @(x) -22*x^3 + 23*x^4 - 24*x^5 + 25*x^6 - 26*x^7 + 27*x^8 - 28*x^9 +
29*x^10 - 30*x^11 + 31*x^12 - 32*x^13 + 33*x^14 - 34*x^15 + 35*x^16 -
36*x^17 + 37*x^18 - 38*x^19, % Degree 19
    @(x) -23*x^3 + 24*x^4 - 25*x^5 + 26*x^6 - 27*x^7 + 28*x^8 - 29*x^9 +
30*x^10 - 31*x^11 + 32*x^12 - 33*x^13 + 34*x^14 - 35*x^15 + 36*x^16 -
37*x^17 + 38*x^18 - 39*x^19, % Degree 19
    @(x) 24*x^4 - 25*x^5 + 26*x^6 - 27*x^7 + 28*x^8 - 29*x^9 + 30*x^10 -
31*x^11 + 32*x^12 - 33*x^13 + 34*x^14 - 35*x^15 + 36*x^16 - 37*x^17 +
38*x^18 - 39*x^19 + 40*x^20, % Degree 20
    @(x) -25*x^3 + 26*x^4 - 27*x^5 + 28*x^6 - 29*x^7 + 30*x^8 - 31*x^9 +
32*x^10 - 33*x^11 + 34*x^12 - 35*x^13 + 36*x^14 - 37*x^15 + 38*x^16 -
39*x^17 + 40*x^18 - 41*x^19, % Degree 19
    @(x) 26*x^4 - 27*x^5 + 28*x^6 - 29*x^7 + 30*x^8 - 31*x^9 + 32*x^10 -
33*x^11 + 34*x^12 - 35*x^13 + 36*x^14 - 37*x^15 + 38*x^16 - 39*x^17 +
40*x^18 - 41*x^19 + 42*x^20, % Degree 20
    @(x) -27*x^3 + 28*x^4 - 29*x^5 + 30*x^6 - 31*x^7 + 32*x^8 - 33*x^9 +
34*x^10 - 35*x^11 + 36*x^12 - 37*x^13 + 38*x^14 - 39*x^15 + 40*x^16 -
41*x^17 + 42*x^18 - 43*x^19, % Degree 19
    @(x) 28*x^4 - 29*x^5 + 30*x^6 - 31*x^7 + 32*x^8 - 33*x^9 + 34*x^10 -
35*x^11 + 36*x^12 - 37*x^13 + 38*x^14 - 39*x^15 + 40*x^16 - 41*x^17 +
42*x^18 - 43*x^19 + 44*x^20, % Degree 20
    @(x) -29*x^3 + 30*x^4 - 31*x^5 + 32*x^6 - 33*x^7 + 34*x^8 - 35*x^9 +
36*x^10 - 37*x^11 + 38*x^12 - 39*x^13 + 40*x^14 - 41*x^15 + 42*x^16 -
43*x^17 + 44*x^18 - 45*x^19, % Degree 19
    @(x) 30*x^4 - 31*x^5 + 32*x^6 - 33*x^7 + 34*x^8 - 35*x^9 + 36*x^10 -
37*x^11 + 38*x^12 - 39*x^13 + 40*x^14 - 41*x^15 + 42*x^16 - 43*x^17 +
44*x^18 - 45*x^19 + 46*x^20, % Degree 20
    @(x) -31*x^3 + 32*x^4 - 33*x^5 + 34*x^6 - 35*x^7 + 36*x^8 - 37*x^9 +
38*x^10 - 39*x^11 + 40*x^12 - 41*x^13 + 42*x^14 - 43*x^15 + 44*x^16 -
45*x^17 + 46*x^18 - 47*x^19, % Degree 19
    @(x) 32*x^4 - 33*x^5 + 34*x^6 - 35*x^7 + 36*x^8 - 37*x^9 + 38*x^10 -
39*x^11 + 40*x^12 - 41*x^13 + 42*x^14 - 43*x^15 + 44*x^16 - 45*x^17 +
46*x^18 - 47*x^19 + 48*x^20, % Degree 20
    @(x) -33*x^3 + 34*x^4 - 35*x^5 + 36*x^6 - 37*x^7 + 38*x^8 - 39*x^9 +
40*x^10 - 41*x^11 + 42*x^12 - 43*x^13 + 44*x^14 - 45*x^15 + 46*x^16 -
47*x^17 + 48*x^18 - 49*x^19 + 50*x^20, % Degree 20
    @(x) 34*x^4 - 35*x^5 + 36*x^6 - 37*x^7 + 38*x^8 - 39*x^9 + 40*x^10 -
41*x^11 + 42*x^12 - 43*x^13 + 44*x^14 - 45*x^15 + 46*x^16 - 47*x^17 +
48*x^18 - 49*x^19 + 50*x^20 - 51*x^21, % Degree 21
    @(x) -35*x^3 + 36*x^4 - 37*x^5 + 38*x^6 - 39*x^7 + 40*x^8 - 41*x^9 +
42*x^10 - 43*x^11 + 44*x^12 - 45*x^13 + 46*x^14 - 47*x^15 + 48*x^16 -
49*x^17 + 50*x^18 - 51*x^19 + 52*x^20, % Degree 20
    @(x) 36*x^4 - 37*x^5 + 38*x^6 - 39*x^7 + 40*x^8 - 41*x^9 + 42*x^10 -
43*x^11 + 44*x^12 - 45*x^13 + 46*x^14 - 47*x^15 + 48*x^16 - 49*x^17 +
50*x^18 - 51*x^19 + 52*x^20 - 53*x^21, % Degree 21
    @(x) -37*x^3 + 38*x^4 - 39*x^5 + 40*x^6 - 41*x^7 + 42*x^8 - 43*x^9 +
44*x^10 - 45*x^11 + 46*x^12 - 47*x^13 + 48*x^14 - 49*x^15 + 50*x^16 -
51*x^17 + 52*x^18 - 53*x^19 + 54*x^20, % Degree 20
    @(x) 38*x^4 - 39*x^5 + 40*x^6 - 41*x^7 + 42*x^8 - 43*x^9 + 44*x^10 -
45*x^11 + 46*x^12 - 47*x^13 + 48*x^14 - 49*x^15 + 50*x^16 - 51*x^17 +
52*x^18 - 53*x^19 + 54*x^20 - 55*x^21, % Degree 21
    @(x) -39*x^3 + 40*x^4 - 41*x^5 + 42*x^6 - 43*x^7 + 44*x^8 - 45*x^9 +
46*x^10 - 47*x^11 + 48*x^12 - 49*x^13 + 50*x^14 - 51*x^15 + 52*x^16 -
53*x^17 + 54*x^18 - 55*x^19 + 56*x^20, % Degree 20
    @(x) 40*x^4 - 41*x^5 + 42*x^6 - 43*x^7 + 44*x^8 - 45*x^9 + 46*x^10 -
47*x^11 + 48*x^12 - 49*x^13 + 50*x^14 - 51*x^15 + 52*x^16 - 53*x^17 +
54*x^18 - 55*x^19 + 56*x^20 - 57*x^21, % Degree 21
    @(x) -41*x^3 + 42*x^4 - 43*x^5 + 44*x^6 - 45*x^7 + 46*x^8 - 47*x^9 +
48*x^10 - 49*x^11 + 50*x^12 - 51*x^13 + 52*x^14 - 53*x^15 + 54*x^16 -
55*x^17 + 56*x^18 - 57*x^19 + 58*x^20, % Degree 20
    @(x) 42*x.^4 - 43*x.^5 + 44*x.^6 - 45*x.^7 + 46*x.^8 - 47*x.^9 +
48*x.^10 - 49*x.^11 + 50*x.^12 - 51*x.^13 + 52*x.^14 - 53*x.^15 + 54*x.^16 -
55*x.^17 + 56*x.^18 - 57*x.^19 + 58*x.^20 - 59*x.^21  % Degree 21
};

trig_polys = {
    @(x) sin(x),                            % Trig function, f(x) = sin(x)
    @(x) cos(x),                            % Trig function, f(x) = cos(x)
    @(x) tan(x),                            % Trig function, f(x) = tan(x)
    @(x) sec(x),                                    % Trig function, f(x) = sec(x)
    @(x) csc(x),                                    % Trig function, f(x) = csc(x)
    @(x) cot(x),                                    % Trig function, f(x) = cot(x)
    @(x) acos(x),                                   % Inverse trig function, f(x) =
acos(x)
    @(x) asin(x),                                   % Inverse trig function, f(x) =
asin(x)
    @(x) atan(x),                                   % Inverse trig function, f(x) =
atan(x)
    @(x) acsc(x),                                   % Inverse trig function, f(x) =
acsc(x)
    @(x) asec(x),                                   % Inverse trig function, f(x) =
asec(x)
    @(x) acot(x),                                   % Inverse trig function, f(x) =
acot(x)
    @(x) sin(x) - 0.5,                              % Trig function, f(x) = sin(x)
- 0.5
    @(x) cos(x) + 0.5,                              % Trig function, f(x) = cos(x)
+ 0.5
    @(x) tan(x) - 1,                                % Trig function, f(x) = tan(x)
- 1
    @(x) sec(x) - 2,                                % Trig function, f(x) = sec(x)
- 2
    @(x) csc(x) + 1,                                % Trig function, f(x) = csc(x)
+ 1
    @(x) cot(x) - 0.5,                              % Trig function, f(x) = cot(x)
- 0.5
    @(x) acos(x) - 0.3,                             % Inverse trig function, f(x) =
acos(x) - 0.3
    @(x) asin(x) + 0.4,                             % Inverse trig function, f(x) =
asin(x) + 0.4
    @(x) atan(x) - 0.2,                             % Inverse trig function, f(x) =
atan(x) - 0.2
    @(x) acsc(x) + 0.1,                             % Inverse trig function, f(x) =
acsc(x) + 0.1
    @(x) asec(x) - 0.4,                             % Inverse trig function, f(x) =
asec(x) - 0.4
    @(x) acot(x) + 0.3,                             % Inverse trig function, f(x) =
acot(x) + 0.3
    @(x) sin(x) * cos(x),                           % Trig function, f(x) = sin(x)
* cos(x)
    @(x) tan(x) / sec(x),                           % Trig function, f(x) = tan(x)
/ sec(x)
```

```
    @(x) cot(x) * csc(x),                   % Trig function, f(x) = cot(x)
* csc(x)
    @(x) acos(cos(x)),                      % Inverse trig function, f(x) =
acos(cos(x))
    @(x) asin(sin(x)),                      % Inverse trig function, f(x) =
asin(sin(x))
    @(x) atan(tan(x)),                      % Inverse trig function, f(x) =
atan(tan(x))
    @(x) acsc(csc(x)),                      % Inverse trig function, f(x) =
acsc(csc(x))
    @(x) asec(sec(x)),                      % Inverse trig function, f(x) =
asec(sec(x))
    @(x) acot(cot(x)),                      % Inverse trig function, f(x) =
acot(cot(x))
    @(x) sin(x)^2 + cos(x)^2 - 1,           % Trig identity, f(x) =
sin(x)^2 + cos(x)^2 - 1
    @(x) 1 + tan(x)^2 - sec(x)^2,           % Trig identity, f(x) = 1 +
tan(x)^2 - sec(x)^2
    @(x) cot(x)^2 - csc(x)^2 + 1,           % Trig identity, f(x) =
cot(x)^2 - csc(x)^2 + 1
    @(x) sin(x) * cos(x) - 0.5,             % Trig function, f(x) = sin(x)
* cos(x) - 0.5
    @(x) tan(x) / sec(x) + 1,               % Trig function, f(x) = tan(x)
/ sec(x) + 1
    @(x) cot(x) * csc(x) - 2,               % Trig function, f(x) = cot(x)
* csc(x) - 2
    @(x) acos(cos(x)) + 0.3,                % Inverse trig function, f(x) =
acos(cos(x)) + 0.3
    @(x) asin(sin(x)) - 0.4,                % Inverse trig function, f(x) =
asin(sin(x)) - 0.4
    @(x) atan(tan(x)) + 0.2,                % Inverse trig function, f(x) =
atan(tan(x)) + 0.2
    @(x) acsc(csc(x)) - 0.1,                % Inverse trig function, f(x) =
acsc(csc(x)) - 0.1
    @(x) asec(sec(x)) + 0.4,                % Inverse trig function, f(x) =
asec(sec(x)) + 0.4
    @(x) acot(cot(x)) - 0.3                 % Inverse trig function, f(x) =
acot(cot(x)) - 0.3
    @(x) sin(x) + cos(x),          % Trig function, f(x) = sin(x) + cos(x)
    @(x) tan(x) * csc(x) - sec(x)  % Trig function, f(x) = tan(x) * csc(x)
- sec(x)
};
```

```
log_polys = {
    @(x) log(x^2 + 1),                          % Logarithmic function, f(x)
= log(x^2 + 1)
    @(x) log(x^3 + 2),                          % Logarithmic function, f(x)
= log(x^3 + 2)
    @(x) log(x^4 + 3),                          % Logarithmic function, f(x)
= log(x^4 + 3)
    @(x) log(x^5 + 4),                          % Logarithmic function, f(x)
= log(x^5 + 4)
    @(x) log(2*x + 5),                          % Logarithmic function, f(x)
= log(2x + 5)
    @(x) log(3*x + 0.5),                        % Logarithmic function, f(x)
= log(3x + 0.5)
    @(x) log(x^2 + sqrt(2)),                    % Logarithmic function, f(x)
= log(x^2 + sqrt(2))
    @(x) log(x^3 + sqrt(3)),                    % Logarithmic function, f(x)
= log(x^3 + sqrt(3))
    @(x) log(x^4 + sqrt(5)),                    % Logarithmic function, f(x)
= log(x^4 + sqrt(5))
    @(x) log(x^5 + exp(1)),                     % Logarithmic function, f(x)
= log(x^5 + e)
    @(x) log(2*x^2 + 10),                       % Logarithmic function, f(x)
= log(2x^2 + 10)
    @(x) log(3*x^3 + 100),                      % Logarithmic function, f(x)
= log(3x^3 + 100)
    @(x) log(x^4 + 0.1),                        % Logarithmic function, f(x)
= log(x^4 + 0.1)
    @(x) log(x^5 + 0.01),                       % Logarithmic function, f(x)
= log(x^5 + 0.01)
    @(x) log(2*x^2 + 0.001),                    % Logarithmic function, f(x)
= log(2x^2 + 0.001)
    @(x) log(3*x^3 + 0.0001),                   % Logarithmic function, f(x)
= log(3x^3 + 0.0001)
    @(x) log(x^4 + 0.00001),                    % Logarithmic function, f(x)
= log(x^4 + 0.00001)
    @(x) log(x^5 + 0.000001),                   % Logarithmic function, f(x)
= log(x^5 + 0.000001)
    @(x) log(2*x^2 + 0.0000001),                % Logarithmic function, f(x)
= log(2x^2 + 0.0000001)
    @(x) log(3*x^3 + 0.00000001),               % Logarithmic function, f(x)
= log(3x^3 + 0.00000001)
    @(x) log(x^4 + 2*x + 1),                    % Logarithmic function, f(x)
= log(x^4 + 2x + 1)
```

```
    @(x) log(x^5 - x^3 + 3*x^2 + 4),          % Logarithmic function, f(x)
= log(x^5 - x^3 + 3x^2 + 4)
    @(x) log(x^2 + 2*x + 5),                  % Logarithmic function, f(x)
= log(x^2 + 2x + 5)
    @(x) log(x^3 - 3*x^2 + 4*x + 0.5),        % Logarithmic function, f(x)
= log(x^3 - 3x^2 + 4x + 0.5)
    @(x) log(x^4 + 5*x^3 + 2*x^2 - x + 1),    % Logarithmic function, f(x)
= log(x^4 + 5x^3 + 2x^2 - x + 1)
    @(x) log(x^5 + 2*x^4 - 3*x^3 + 4*x^2 - 5*x),% Logarithmic function, f(x)
= log(x^5 + 2x^4 - 3x^3 + 4x^2 - 5x)
    @(x) log(x^2 - 2*x + 3),                  % Logarithmic function, f(x)
= log(x^2 - 2x + 3)
    @(x) log(x^3 + x^2 + 2*x + 4),            % Logarithmic function, f(x)
= log(x^3 + x^2 + 2x + 4)
    @(x) log(x^4 - x^3 - x^2 + x + 5),        % Logarithmic function, f(x)
= log(x^4 - x^3 - x^2 + x + 5)
    @(x) log(x^5 - 2*x^4 + 3*x^3 + x^2 - 4*x),  % Logarithmic function, f(x)
= log(x^5 - 2x^4 + 3x^3 + x^2 - 4x)
    @(x) log(x^2 + 2*x - 1),                  % Logarithmic function, f(x)
= log(x^2 + 2x - 1)
    @(x) log(x^3 - 2*x^2 - x + 2),            % Logarithmic function, f(x)
= log(x^3 - 2x^2 - x + 2)
    @(x) log(x^4 + 3*x^3 + x^2 - 3*x + 1),    % Logarithmic function, f(x)
= log(x^4 + 3x^3 + x^2 - 3x + 1)
    @(x) log(x^5 - x^4 + 2*x^3 + 3*x^2 - 4*x), % Logarithmic function, f(x)
= log(x^5 - x^4 + 2x^3 + 3x^2 - 4x)
    @(x) log(x^2 - x + 5),                    % Logarithmic function, f(x)
= log(x^2 - x + 5)
    @(x) log(x^3 + 2*x^2 + x + 1),            % Logarithmic function, f(x)
= log(x^3 + 2x^2 + x + 1)
    @(x) log(x^4 - 3*x^3 + 2*x^2 + 3*x - 1),  % Logarithmic function, f(x)
= log(x^4 - 3x^3 + 2x^2 + 3x - 1)
    @(x) log(x^5 + 2*x^4 - 3*x^3 + 4*x^2 + 5*x),% Logarithmic function, f(x)
= log(x^5 + 2x^4 - 3x^3 + 4x^2 + 5x)
    @(x) log(x^2 + x + 1),                    % Logarithmic function, f(x)
= log(x^2 + x + 1)
    @(x) log(x^3 - x^2 + 2*x - 3),            % Logarithmic function, f(x)
= log(x^3 - x^2 + 2x - 3)
    @(x) log(x^4 + x^3 + x^2 - 3*x + 4),      % Logarithmic function, f(x)
= log(x^4 + x^3 + x^2 - 3x + 4)
    @(x) log(x^5 - 2*x^4 + x^3 + 2*x^2 - x + 5)% Logarithmic function, f(x)
= log(x^5 - 2x^4 + x^3 + 2x^2 - x + 5)
};
```

```
exp_polys = {
    @(x) exp(x) - 1,                              % Exponential function, f(x) =
exp(x) - 1
    @(x) exp(2*x) - 2,                            % Exponential function, f(x) =
exp(2x) - 2
    @(x) exp(3*x) - 3,                            % Exponential function, f(x) =
exp(3x) - 3
    @(x) exp(4*x) - 4,                            % Exponential function, f(x) =
exp(4x) - 4
    @(x) exp(5*x) - 5,                            % Exponential function, f(x) =
exp(5x) - 5
    @(x) exp(x/2) - 0.5,                          % Exponential function, f(x) =
exp(x/2) - 0.5
    @(x) exp(x/3) - 0.33,                         % Exponential function, f(x) =
exp(x/3) - 0.33
    @(x) exp(x/4) - 0.25,                         % Exponential function, f(x) =
exp(x/4) - 0.25
    @(x) exp(x/5) - 0.2,                          % Exponential function, f(x) =
exp(x/5) - 0.2
    @(x) exp(sqrt(2)*x) - sqrt(2),                % Exponential function, f(x) =
exp(sqrt(2)*x) - sqrt(2)
    @(x) exp(sqrt(3)*x) - sqrt(3),                % Exponential function, f(x) =
exp(sqrt(3)*x) - sqrt(3)
    @(x) exp(sqrt(5)*x) - sqrt(5),                % Exponential function, f(x) =
exp(sqrt(5)*x) - sqrt(5)
    @(x) exp(x^2) - x^2,                          % Exponential function, f(x) =
exp(x^2) - x^2
    @(x) exp(x^3) - x^3,                          % Exponential function, f(x) =
exp(x^3) - x^3
    @(x) exp(x^4) - x^4,                          % Exponential function, f(x) =
exp(x^4) - x^4
    @(x) exp(x^5) - x^5,                          % Exponential function, f(x) =
exp(x^5) - x^5
    @(x) exp(x) * exp(-x) - 1,                    % Exponential function, f(x) =
exp(x)*exp(-x) - 1
    @(x) exp(2*x) * exp(-2*x) - 1,                % Exponential function, f(x) =
exp(2x)*exp(-2x) - 1
    @(x) exp(3*x) * exp(-3*x) - 1,                % Exponential function, f(x) =
exp(3x)*exp(-3x) - 1
    @(x) exp(4*x) * exp(-4*x) - 1,                % Exponential function, f(x) =
exp(4x)*exp(-4x) - 1
    @(x) exp(5*x) * exp(-5*x) - 1,                % Exponential function, f(x) =
exp(5x)*exp(-5x) - 1
    @(x) exp(x) / exp(-x) - exp(2*x),          % Exponential function, f(x) =
exp(x)/exp(-x) - exp(2x)
    @(x) exp(2*x) / exp(-2*x) - exp(4*x),      % Exponential function, f(x) =
exp(2x)/exp(-2x) - exp(4x)
    @(x) exp(3*x) / exp(-3*x) - exp(6*x),      % Exponential function, f(x) =
exp(3x)/exp(-3x) - exp(6x)
    @(x) exp(4*x) / exp(-4*x) - exp(8*x),      % Exponential function, f(x) =
exp(4x)/exp(-4x) - exp(8x)
    @(x) exp(5*x) / exp(-5*x) - exp(10*x),     % Exponential function, f(x) =
exp(5x)/exp(-5x) - exp(10x)
    @(x) exp(2*x) - exp(x)^2,                  % Exponential function, f(x) =
exp(2x) - exp(x)^2
    @(x) exp(3*x) - exp(x)^3,                  % Exponential function, f(x) =
exp(3x) - exp(x)^3
    @(x) exp(4*x) - exp(x)^4,                  % Exponential function, f(x) =
exp(4x) - exp(x)^4
    @(x) exp(5*x) - exp(x)^5,                  % Exponential function, f(x) =
exp(5x) - exp(x)^5
    @(x) exp(2*x) / exp(x)^2 - exp(x),         % Exponential function, f(x) =
exp(2x)/exp(x)^2 - exp(x)
    @(x) exp(3*x) / exp(x)^3 - exp(x)^2,       % Exponential function, f(x) =
exp(3x)/exp(x)^3 - exp(x)^2
    @(x) exp(4*x) / exp(x)^4 - exp(x)^3,       % Exponential function, f(x) =
exp(4x)/exp(x)^4 - exp(x)^3
    @(x) exp(5*x) / exp(x)^5 - exp(x)^4,       % Exponential function, f(x) =
exp(5x)/exp(x)^5 - exp(x)^4
    @(x) exp(sqrt(2)*x) - exp(x*sqrt(2)),      % Exponential function, f(x) =
exp(sqrt(2)*x) - exp(x*sqrt(2))
    @(x) exp(sqrt(3)*x) - exp(x*sqrt(3)),      % Exponential function, f(x) =
exp(sqrt(3)*x) - exp(x*sqrt(3))
    @(x) exp(sqrt(5)*x) - exp(x*sqrt(5)),      % Exponential function, f(x) =
exp(sqrt(5)*x) - exp(x*sqrt(5))
    @(x) exp(x^2) - exp(2*x),                  % Exponential function, f(x) =
exp(x^2) - exp(2*x)
    @(x) exp(x^3) - exp(3*x),                  % Exponential function, f(x) =
exp(x^3) - exp(3*x)
    @(x) exp(x^4) - exp(4*x),                  % Exponential function, f(x) =
exp(x^4) - exp(4*x)
    @(x) exp(x^5) - exp(5*x),                  % Exponential function, f(x) =
exp(x^5) - exp(5*x)
    @(x) exp(2*x) - exp(x)^2,                  % Exponential function, f(x) =
exp(2*x) - exp(x)^2
```

```
    @(x) exp(3*x) - exp(x)^3,                   % Exponential function, f(x) =
exp(3*x) - exp(x)^3
    @(x) exp(4*x) - exp(x)^4,                   % Exponential function, f(x) =
exp(4*x) - exp(x)^4
    @(x) exp(5*x) - exp(x)^5,                   % Exponential function, f(x) =
exp(5*x) - exp(x)^5
    @(x) exp(sqrt(2)*x) - exp(x)^sqrt(2),       % Exponential function, f(x) =
exp(sqrt(2)*x) - exp(x)^sqrt(2)
    @(x) exp(sqrt(3)*x) - exp(x)^sqrt(3),       % Exponential function, f(x) =
exp(sqrt(3)*x) - exp(x)^sqrt(3)
    @(x) exp(sqrt(5)*x) - exp(x)^sqrt(5),       % Exponential function, f(x) =
exp(sqrt(5)*x) - exp(x)^sqrt(5)
    @(x) exp(x^2) - exp(x)^2,                   % Exponential function, f(x) =
exp(x^2) - exp(x)^2
    @(x) exp(x^3) - exp(x)^3,                   % Exponential function, f(x) =
exp(x^3) - exp(x)^3
    @(x) exp(x^4) - exp(x)^4,                   % Exponential function, f(x) =
exp(x^4) - exp(x)^4
    @(x) exp(x^5) - exp(x)^5                    % Exponential function, f(x) =
exp(x^5) - exp(x)^5
};

mixed_functions = {
    @(x) sin(x) + cos(x) + tan(x) - 1,                 % Mixed function 1
    @(x) log(x^2 + 1) + exp(x) - x - 2,                % Mixed function 2
    @(x) x^3 + log(x) + sqrt(x) - 4,                   % Mixed function 3
    @(x) exp(sin(x)) + log(cos(x)) - tan(x) + 1,       % Mixed function 4
    @(x) sin(x) * cos(x) + exp(x) - x^2 - 3,           % Mixed function 5
    @(x) sqrt(x) * log(x) + exp(x) - tan(x) + 2,       % Mixed function 6
    @(x) sin(x) * log(x) + exp(x) - x - 1,             % Mixed function 7
    @(x) x^2 + exp(x) + sin(x) - 5,                    % Mixed function 8
    @(x) exp(x) * sin(x) - cos(x) + tan(x) - 3,        % Mixed function 9
    @(x) exp(x) - log(x) + sin(x) - x^2 + 4,           % Mixed function 10
    @(x) exp(x) + sin(x) - cos(x) - tan(x) - 2,        % Mixed function 11
    @(x) sin(x) + cos(x) + log(x) + tan(x) - 2,        % Mixed function 12
    @(x) x^2 + log(x) + exp(x) - sin(x) - 3,           % Mixed function 13
    @(x) exp(x) + log(x) + sin(x) - cos(x) - 3,        % Mixed function 14
    @(x) exp(sin(x)) + log(cos(x)) + tan(x) - x - 1, % Mixed function 15
    @(x) x^3 + sin(x) + cos(x) - log(x) - 3,           % Mixed function 16
    @(x) exp(x) + sin(x) * cos(x) - tan(x) - x^2 - 1,% Mixed function 17
    @(x) exp(x) * log(x) - sin(x) + cos(x) - 3,        % Mixed function 18
    @(x) exp(sin(x)) - log(cos(x)) + tan(x) - x - 2, % Mixed function 19
    @(x) x^2 + sin(x) - cos(x) + log(x) - 4,           % Mixed function 20
    @(x) exp(x) + sqrt(x) + sin(x) - cos(x) - 4,     % Mixed function 21
    @(x) sin(x) + cos(x) + tan(x) + log(x) - 2,      % Mixed function 22
    @(x) exp(x) + sqrt(x) * log(x) + sin(x) - x - 4, % Mixed function 23
    @(x) exp(x) + log(x) * sin(x) - cos(x) - x^2 - 2,% Mixed function 24
    @(x) sin(x) * cos(x) + log(x) + exp(x) - tan(x), % Mixed function 25
    @(x) exp(x) * log(x) - sin(x) * cos(x) + tan(x), % Mixed function 26
    @(x) x^2 + sin(x) * cos(x) - exp(x) - log(x) + 2,% Mixed function 27
    @(x) exp(x) + sin(x) + cos(x) + tan(x) - log(x) - 3, % Mixed function 28
    @(x) exp(x) - sin(x) * cos(x) - log(x) + tan(x) + 2,% Mixed function 29
    @(x) x^3 + sin(x) * cos(x) + log(x) - exp(x) - 4,% Mixed function 30
    @(x) exp(x) + sin(x) * cos(x) + tan(x) - log(x) - 1, % Mixed function 31
    @(x) exp(x) + log(x) + sin(x) + cos(x) + tan(x) - 5, % Mixed function 32
    @(x) x^2 + exp(x) * log(x) + sin(x) * cos(x) - tan(x) - 4, % Mixed
function 33
    @(x) exp(x) + sin(x) * cos(x) + log(x) - tan(x) - 3, % Mixed function 34
    @(x) exp(x) - sin(x) + cos(x) + tan(x) + log(x) - 4, % Mixed function 35
    @(x) x^3 + sin(x) + cos(x) + tan(x) - exp(x) - log(x) - 2, % Mixed
function 36
    @(x) exp(x) + log(x) + sin(x) + cos(x) + tan(x) - x - 5, % Mixed
function 37
    @(x) exp(x) + sin(x) * cos(x) + log(x) + tan(x) - x - 4, % Mixed
function 38
    @(x) x^3 + exp(x) + sin(x) * cos(x) + tan(x) - log(x) - 4, % Mixed
function 39
    @(x) exp(x) + sin(x) + cos(x) + tan(x) - log(x) - 3, % Mixed function 40
    @(x) x^3 + sin(x) * cos(x) + exp(x) - tan(x) - log(x) - 1, % Mixed
function 41
    @(x) exp(x) + sin(x) * cos(x) + log(x) - tan(x) - x^2 - 2, % Mixed
function 42
    @(x) exp(x) + sin(x) + cos(x) + tan(x) - log(x) - x - 2, % Mixed
function 43
    @(x) x^3 + exp(x) + sin(x) * cos(x) + tan(x) - log(x) - 4, % Mixed
function 44
    @(x) exp(x) + sin(x) * cos(x) + tan(x) - log(x) - x^2 - 2, % Mixed
function 45
    @(x) exp(x) + sin(x) * cos(x) + tan(x) - log(x) - x - 2, % Mixed
function 46
    @(x) x^3 + exp(x) + sin(x) * cos(x) + tan(x) - log(x) - 4, % Mixed
function 47
    @(x) exp(x) + sin(x) * cos(x) + tan(x) - log(x) - x^2 - 2, % Mixed
function 48
    @(x) exp(x) + sin(x) * cos(x) + tan(x) - log(x) - x - 2, % Mixed
function 49
```

```
    @(x) x^3 + exp(x) + sin(x) * cos(x) + tan(x) - log(x) - 4 % Mixed function 50
};
```

Building the Colosseum

Now with our Gladiators and Beasts, we needed to make a place for them to fight it out! This was by far the toughest part of the project.

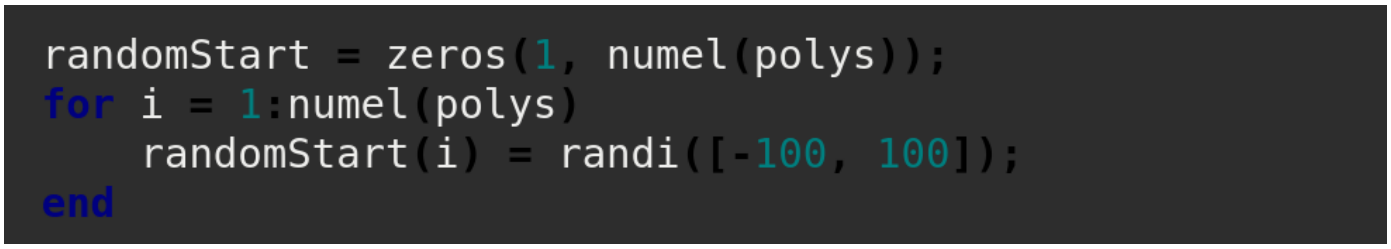
To begin, I wanted this to work with any list of functions, to any desired level of tolerance, as such I had to build two loops, one that would loop through all the functions in the list, and another loop above it that would change the tolerance to see what would happen as it became smaller and smaller.

```
for j = 1:iterations
    % Calculate tolerance based on the step size
    step_size = 10^(1/4); % 10/4 magnitude change
    tolerance = 1 / (10^(j/4) * step_size^(j-1));
    toleranceCount(j) = tolerance;
```

Once we have established the tolerance level, having it decrease by a magnitude of 10 every four steps, I then defined the inner loop, going through every function and having it add up the time and function calls to give an average of all of them at the end.

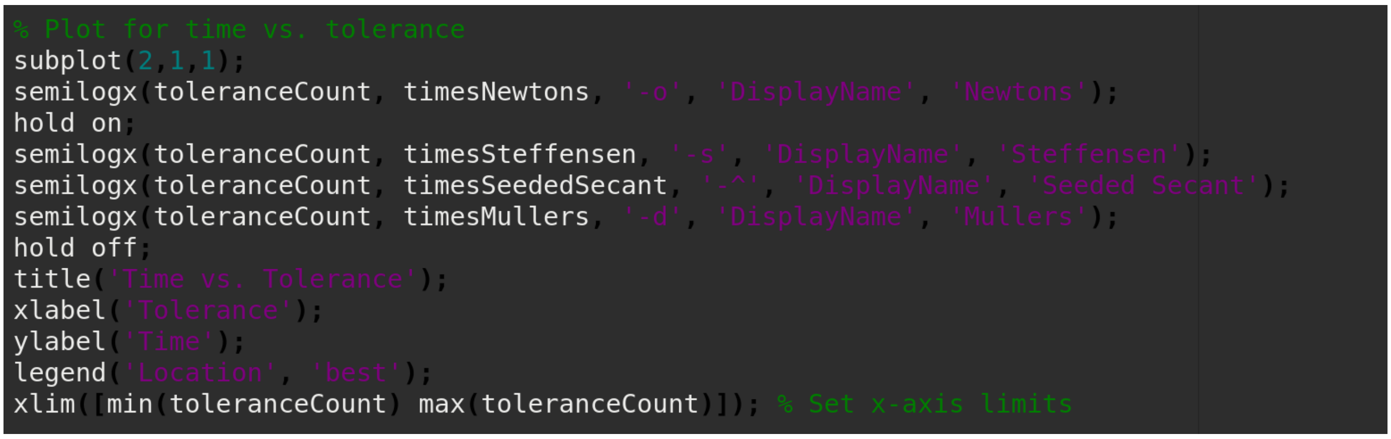
```
for i = 1:numel(polys)
    f = polys{i};
    randint = randomStart(i);

    % Newton's method
    [a, b, c] = newtons(f, randint, tolerance, n);
    if ~isnan(a)
        avgtimeNewtons = avgtimeNewtons + b;
        avgcalcsNewtons = avgcalcsNewtons + c;
    else
        failuresNewtons(j) = failuresNewtons(j) + 1;
    end

    % Steffensen's method
    [a, b, c] = steffensen(f, randint, tolerance, n);
    if ~isnan(a)
        avgtimeSteffensen = avgtimeSteffensen + b;
        avgcalcsSteffensen = avgcalcsSteffensen + c;
    else
        failuresSteffensen(j) = failuresSteffensen(j) + 1;
    end

    % Seeded Secant method
```

Before that however I had to tackle initial conditions, and my method was to loop through and assign each function a random initial condition between -100 and 100 through a loop at the beginning, which it would reuse for every tolerance shift to try and reduce the effect of lucky initial conditions from the analysis.



```
randomStart = zeros(1, numel(polys));
for i = 1:numel(polys)
    randomStart(i) = randi([-100, 100]);
end
```

Finally, I plotted the averages of both the time and number of calculations using the following code:



```
% Plot for time vs. tolerance
subplot(2,1,1);
semilogx(toleranceCount, timesNewtons, '-o', 'DisplayName', 'Newtons');
hold on;
semilogx(toleranceCount, timesSteffensen, '-s', 'DisplayName', 'Steffensen');
semilogx(toleranceCount, timesSeededSecant, '-^', 'DisplayName', 'Seeded Secant');
semilogx(toleranceCount, timesMullers, '-d', 'DisplayName', 'Mullers');
hold off;
title('Time vs. Tolerance');
xlabel('Tolerance');
ylabel('Time');
legend('Location', 'best');
xlim([min(toleranceCount) max(toleranceCount)]); % Set x-axis limits
```

Final result for the function is:

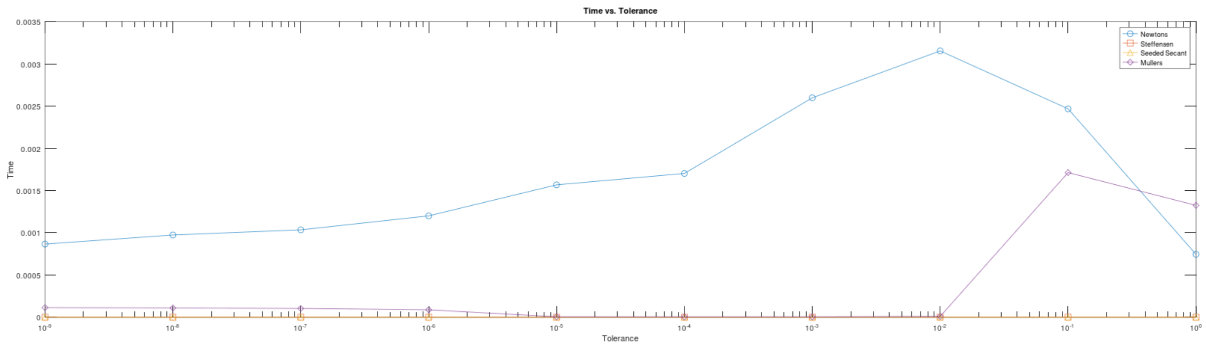
```
function DannyMethodTester(iterations, polys, n)
    timesNewtons = zeros(1, iterations);
    calcsNewtons = zeros(1, iterations);
    timesSteffensen = zeros(1, iterations);
    calcsSteffensen = zeros(1, iterations);
    timesSeededSecant = zeros(1, iterations);
    calcsSeededSecant = zeros(1, iterations);
    timesMullers = zeros(1, iterations);
    calcsMullers = zeros(1, iterations);
    toleranceCount = zeros(1, iterations);
    failuresNewtons = zeros(1, iterations);
    failuresSteffensen = zeros(1, iterations);
    failuresSeededSecant = zeros(1, iterations);
    failuresMullers = zeros(1, iterations);

    randomStart = zeros(1, numel(polys));
    for i = 1:numel(polys)
        randomStart(i) = randi([-100, 100]);
    end
```

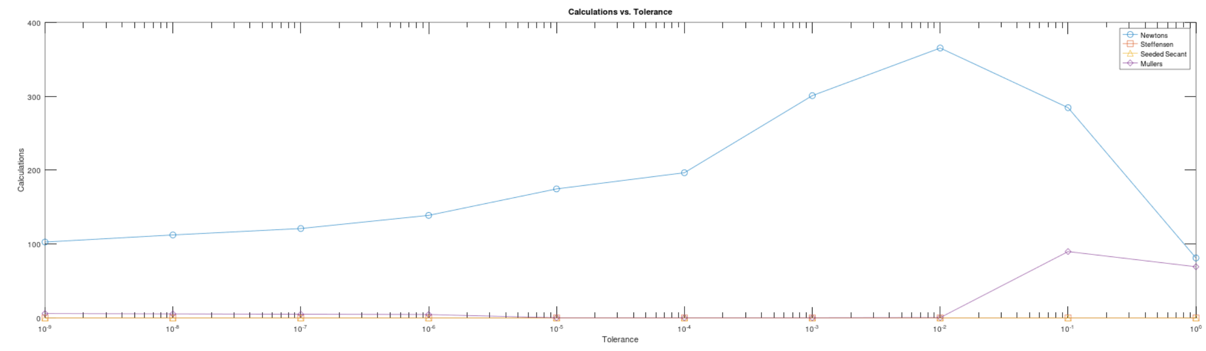
```
for j = 1:iterations
    % Calculate tolerance based on the step size
    step_size = 10^(1/4); % 10/4 magnitude change
    tolerance = 1 / (10^(j/4) * step_size^(j-1));
    toleranceCount(j) = tolerance;
    avgtimeNewtons = 0;
    avgcalcsNewtons = 0;
    avgtimeSteffensen = 0;
    avgcalcsSteffensen = 0;
    avgtimeSeededSecant = 0;
    avgcalcsSeededSecant = 0;
    avgtimeMullers = 0;
    avgcalcsMullers = 0;
    failuresNewtons(j) = 0;
    failuresSteffensen(j) = 0;
    failuresSeededSecant(j) = 0;
    failuresMullers(j) = 0;

    for i = 1:numel(polys)
        f = polys{i};
        randint = randomStart(i);

        % Newton's method
        [a, b, c] = newtons(f, randint, tolerance, n);
        if ~isnan(a)
            avgtimeNewtons = avgtimeNewtons + b;
            avgcalcsNewtons = avgcalcsNewtons + c;
        else
            failuresNewtons(j) = failuresNewtons(j) + 1;
        end

        % Steffensen's method
        [a, b, c] = steffensen(f, randint, tolerance, n);
        if ~isnan(a)
            avgtimeSteffensen = avgtimeSteffensen + b;
            avgcalcsSteffensen = avgcalcsSteffensen + c;
        else
            failuresSteffensen(j) = failuresSteffensen(j) + 1;
        end

        % Seeded Secant method
        [a, b, c] = seededSecant(f, randint, randint + 1, tolerance, n);
```

```
            if ~isnan(a)
                avgtimeSeededSecant = avgtimeSeededSecant + b;
                avgcalcsSeededSecant = avgcalcsSeededSecant + c;
            else
                failuresSeededSecant(j) = failuresSeededSecant(j) + 1;
            end

            % Muller's method
            [a, b, c] = mullers(f, randint, randint + 1, randint + 2,
tolerance, n);
            if ~isnan(a)
                avgtimeMullers = avgtimeMullers + b;
                avgcalcsMullers = avgcalcsMullers + c;
            else
                failuresMullers(j) = failuresMullers(j) + 1;
            end
        end

        % Compute averages, taking failures into account
        num_tests = numel(polys);
        timesNewtons(j) = avgtimeNewtons / (num_tests - failuresNewtons(j));
        calcsNewtons(j) = avgcalcsNewtons / (num_tests -
failuresNewtons(j));
        timesSteffensen(j) = avgtimeSteffensen / (num_tests -
failuresSteffensen(j));
        calcsSteffensen(j) = avgcalcsSteffensen / (num_tests -
failuresSteffensen(j));
        timesSeededSecant(j) = avgtimeSeededSecant / (num_tests -
failuresSeededSecant(j));
        calcsSeededSecant(j) = avgcalcsSeededSecant / (num_tests -
failuresSeededSecant(j));
        timesMullers(j) = avgtimeMullers / (num_tests - failuresMullers(j));
        calcsMullers(j) = avgcalcsMullers / (num_tests -
failuresMullers(j));
    end

    % Plotting
    figure;

    % Plot for time vs. tolerance
    subplot(2,1,1);
    semilogx(toleranceCount, timesNewtons, '-o', 'DisplayName', 'Newtons');
    hold on;
    semilogx(toleranceCount, timesSteffensen, '-s', 'DisplayName',
'Steffensen');
    semilogx(toleranceCount, timesSeededSecant, '-^', 'DisplayName', 'Seeded
Secant');
    semilogx(toleranceCount, timesMullers, '-d', 'DisplayName', 'Mullers');
    hold off;
    title('Time vs. Tolerance');
    xlabel('Tolerance');
    ylabel('Time');
    legend('Location', 'best');
    xlim([min(toleranceCount) max(toleranceCount)]); % Set x-axis limits

    % Plot for calculations vs. tolerance
    subplot(2,1,2);
    semilogx(toleranceCount, calcsNewtons, '-o', 'DisplayName', 'Newtons');
    hold on;
    semilogx(toleranceCount, calcsSteffensen, '-s', 'DisplayName',
'Steffensen');
    semilogx(toleranceCount, calcsSeededSecant, '-^', 'DisplayName', 'Seeded
Secant');
    semilogx(toleranceCount, calcsMullers, '-d', 'DisplayName', 'Mullers');
    hold off;
    title('Calculations vs. Tolerance');
    xlabel('Tolerance');
    ylabel('Calculations');
    legend('Location', 'best');
    xlim([min(toleranceCount) max(toleranceCount)]); % Set x-axis limits

    % Plot failure percentages
    figure;
    semilogx(toleranceCount, failuresNewtons ./ numel(polys) * 100, '-o',
'DisplayName', 'Newtons');
    hold on;
    semilogx(toleranceCount, failuresSteffensen ./ numel(polys) * 100, '-s',
'DisplayName', 'Steffensen');
    semilogx(toleranceCount, failuresSeededSecant ./ numel(polys) * 100, '-
^', 'DisplayName', 'Seeded Secant');
    semilogx(toleranceCount, failuresMullers ./ numel(polys) * 100, '-d',
'DisplayName', 'Mullers');
    hold off;
    title('Failure Percentage vs. Tolerance');
    xlabel('Tolerance');
    ylabel('Failure Percentage (%)');
    legend('Location', 'best');
    xlim([min(toleranceCount) max(toleranceCount)]); % Set x-axis limits
end
```

Handling when a Gladiator Dies

After I ran my code I got this output for small polynomials:





As we can clearly see, Newton is garbage and we should all use Steffensons! It became even more extreme with big polynomials, and was so strange that it made me think that maybe I should double-check what was going on!

In the end, it became the most interesting little bug not in my code, but in my thought process, as it all had to do with how I handled the failure of a function.

At the time I had my code report NaN if the method reached max iteration limits, which was at the time hard coded to be 100, then I would have my code exclude it from the average, however, this created a vulnerability in my process.

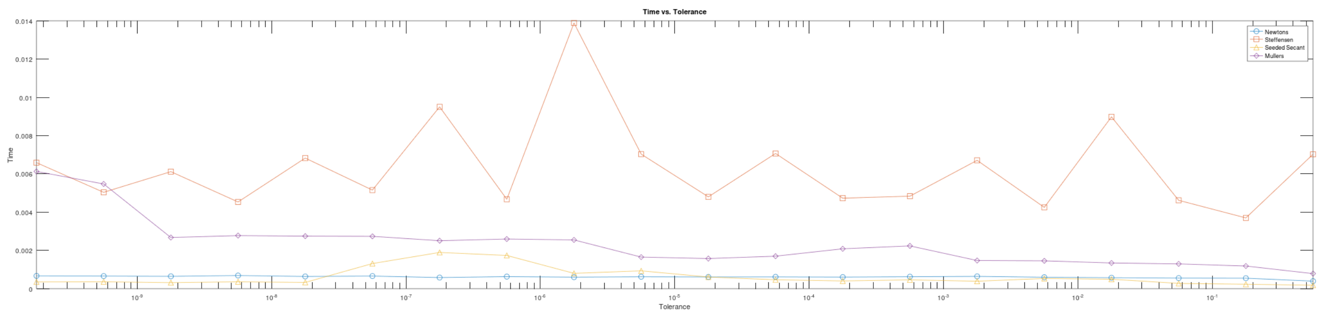
Take the hypothetical Danny method, which can either get lucky and calculate anything in one iteration, or fail, this method would be pretty bad in real life, but amazing in my analysis! All the times it fails would be ignored, and only the beautiful data would highlight it as the best.

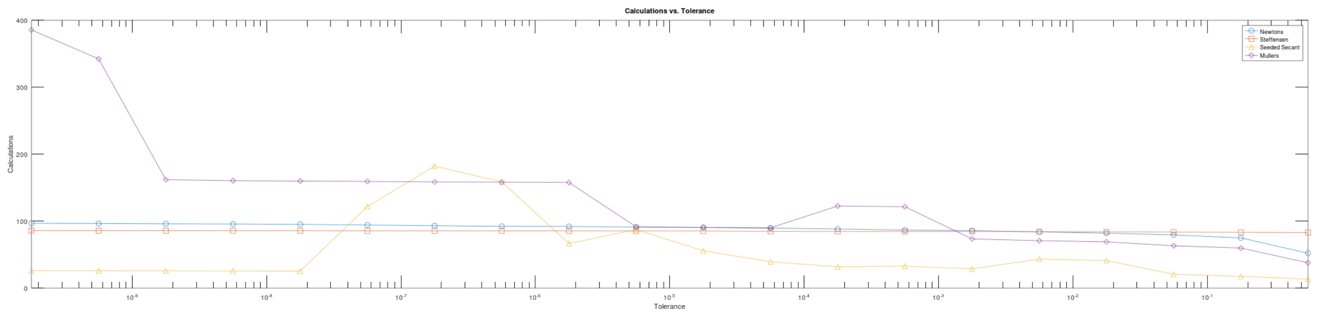
First, I changed the average countage to take into account the fewer points, then I changed the hard-coded number into an argument of my function, allowing the user to decide how many iterations to give, with my personal choice being 2000 per run of the function.

The most important change, however, was that I counted each function failure and coded up a second graph to display the failure rate for all the functions at different tolerances.

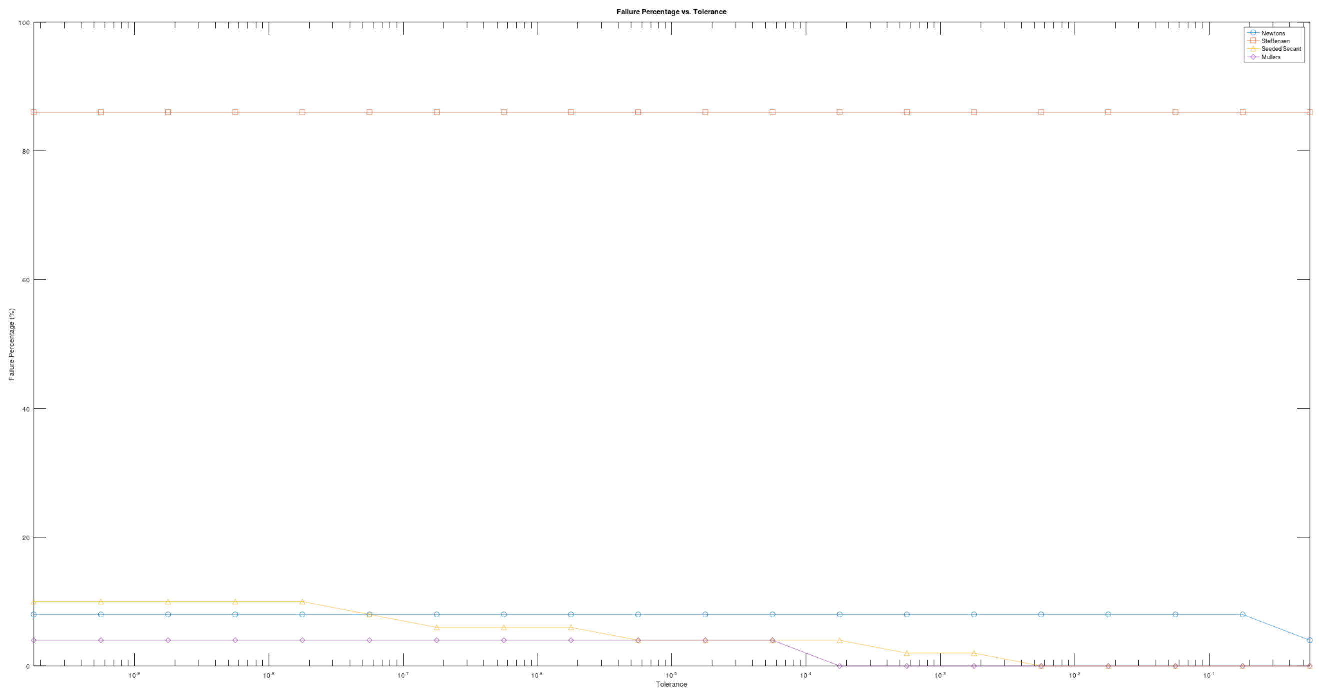
Analyzing the Results

Below are the first set of results for small polynomials of degree 5 or less:





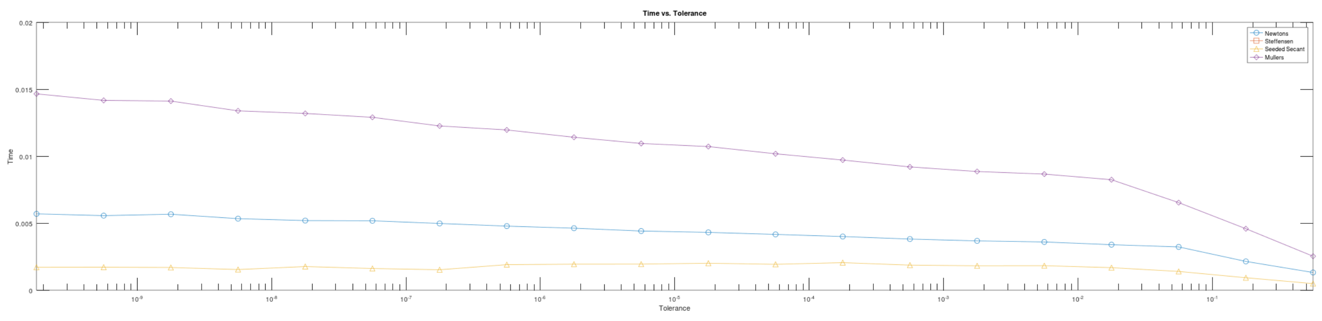
The first thing to note for this and all the following results is that a more consistent computer is needed, as well as many many more equation data points to smooth out the data. As we can see the time it takes for each method is just an average of how long it took my computer to calculate, and as such is subject to a lot of external error, even the calculations however also show that the calculations can decrease the lower the tolerance gets! Amazing Right! (Unfortunately, as we will see in a second this is not true)

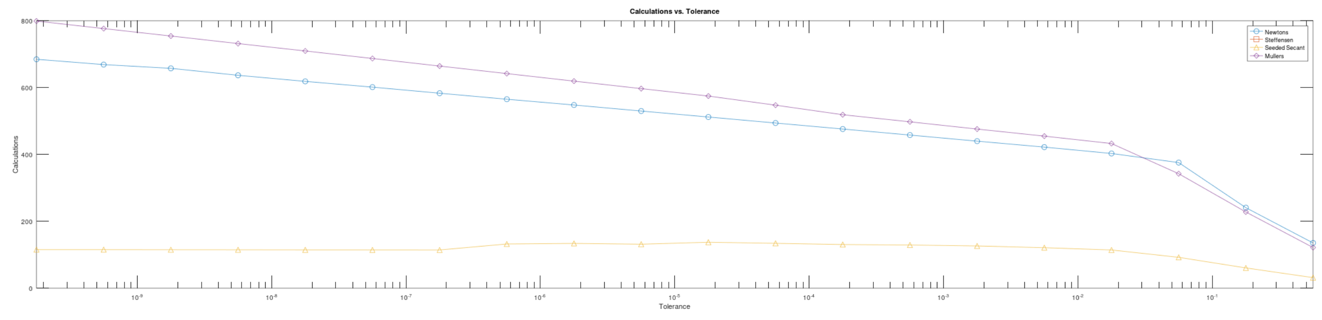


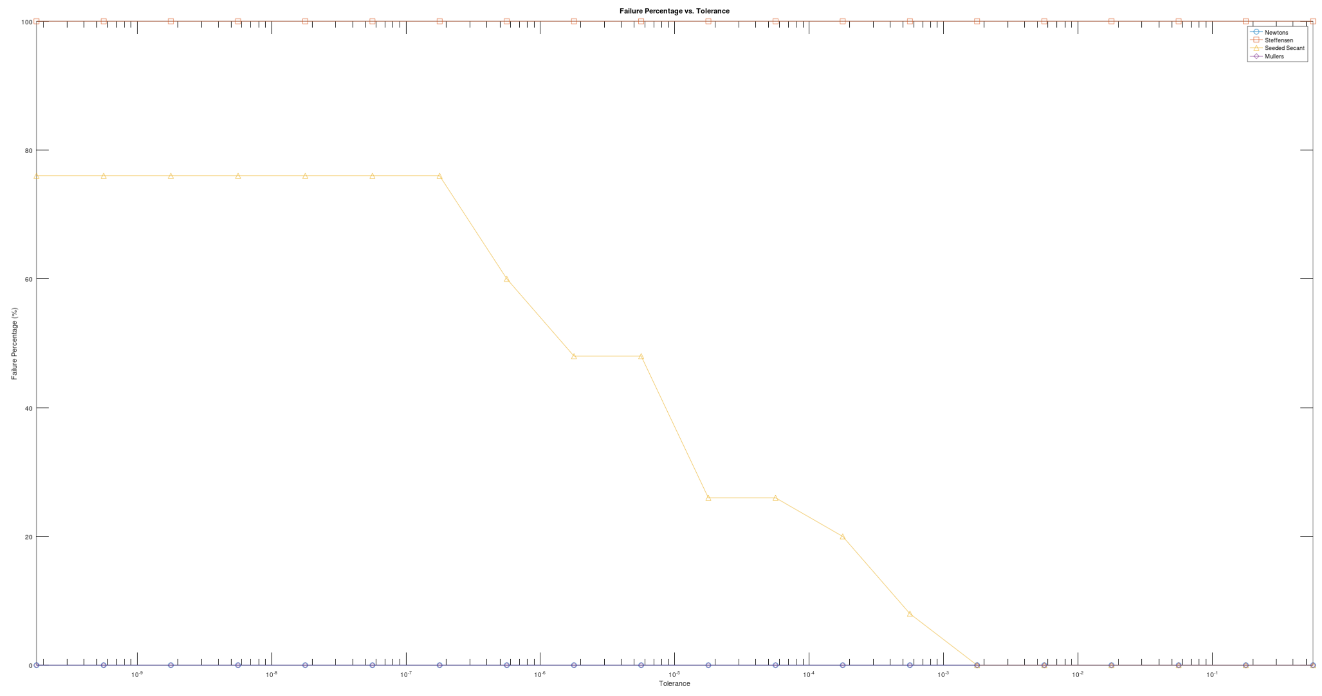
This graph tells so much of the story that I am amazed that I lucked into including it in the project. First on why Steffenson was so crazy, as we can see here, almost 85% of Stephenson ended up in failures, as most polynomials don't satisfy its derivative condition, and as such, the reduced data points allowed for small errors from my computer to become drastic.

Next is the magic decrease in calculation and time, as we can see from this graph, seeded secants slowly raises in failure percentage as the tolerance decreases, and if we look at the humps for the increase, they perfectly line up with the weird effect we are seeing with the tolerance, leading me to make the hypothesis that if secants are dropping function calculations due to not having enough iterations, then those that it drops are likely to be the heaviest in terms of time and total calculations, and dropping even a few percent of those very heavy calculations has a drastic impact on average time and function calls.

With the basic trends out of the way, we can now check out large polynomials:

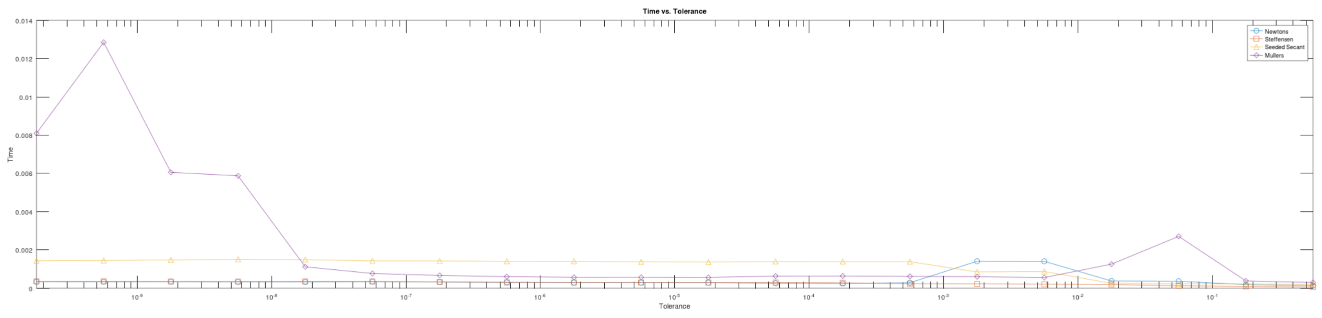


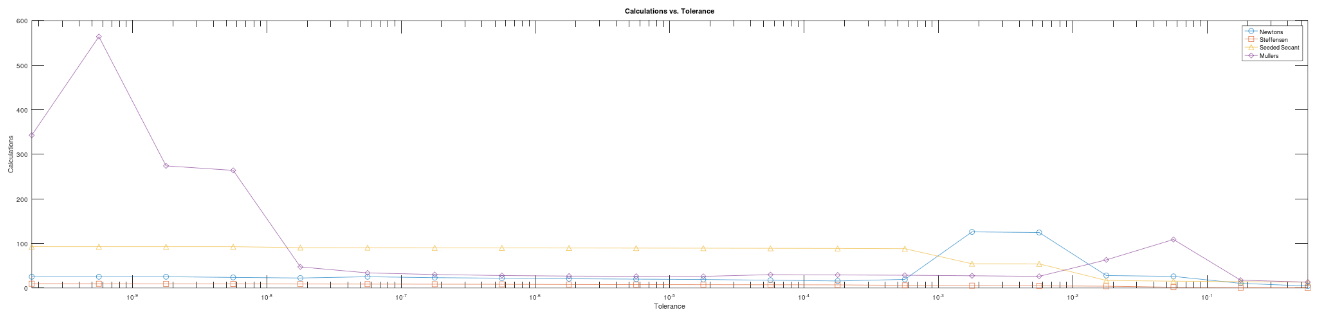


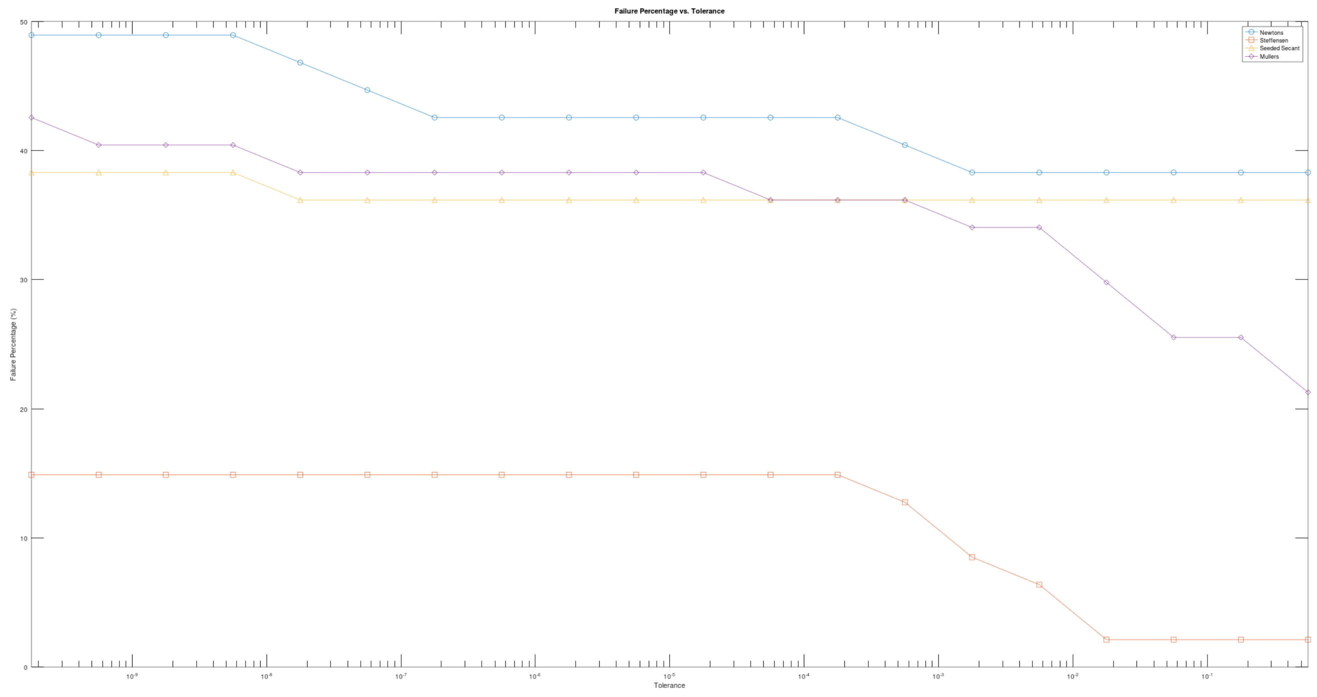


Once again, we can see that Steffensons is out of the question, it did not have a single equation satisfy it! While Secan seems to be the best option, it has a massive spike in failure rates as you decrease the tol, leaving either Mullers or Newtons as the best option, with both being very reliable and fast for both small and big polynomials.

Moving on to trig functions, we have:

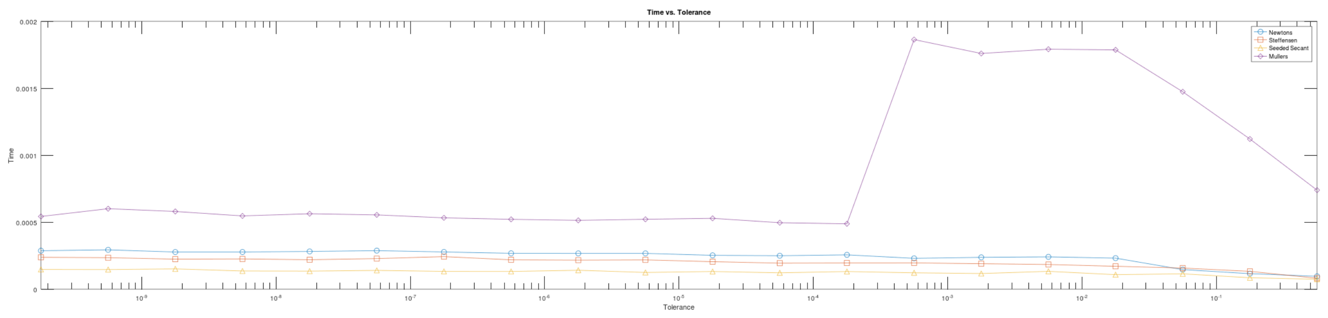


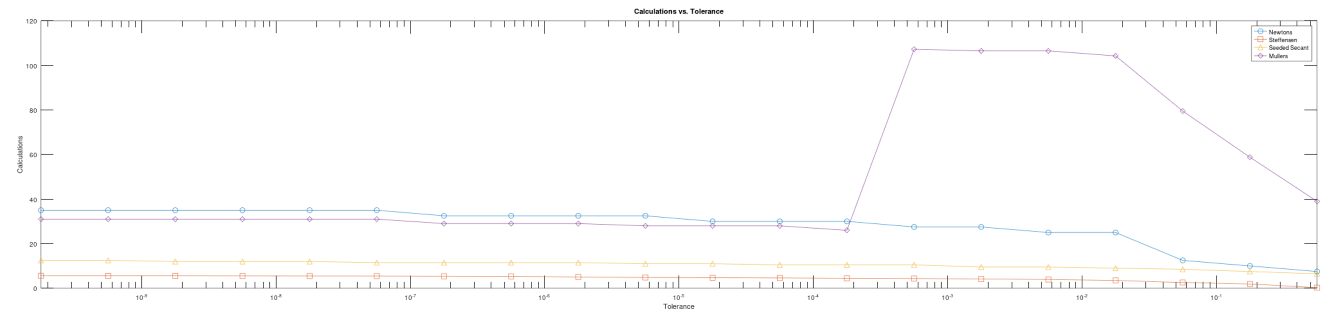


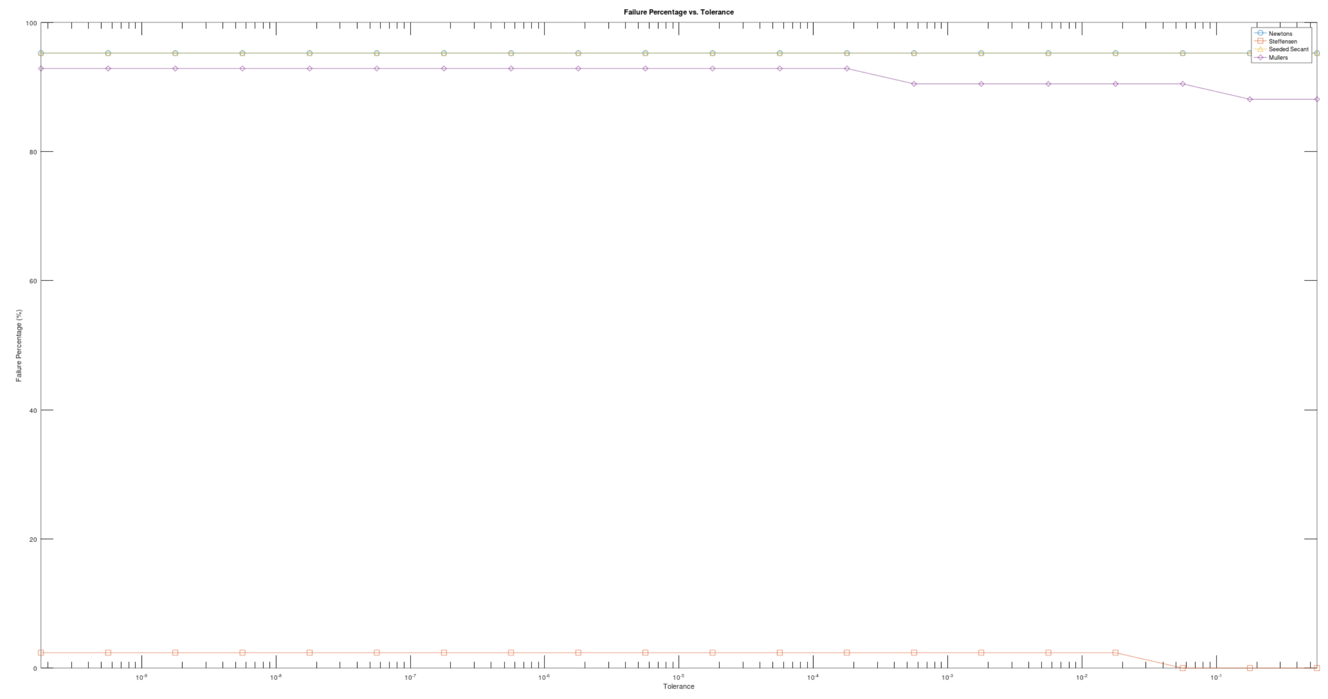


A complete reversal! We see that Stephenson has made a huge comeback, destroying the competition in both speed and failure rate. Mulers is reliable for low tolerances, but quickly loses the ability to compete, making Steffensons the clear choice.

On to logarithm, we have:

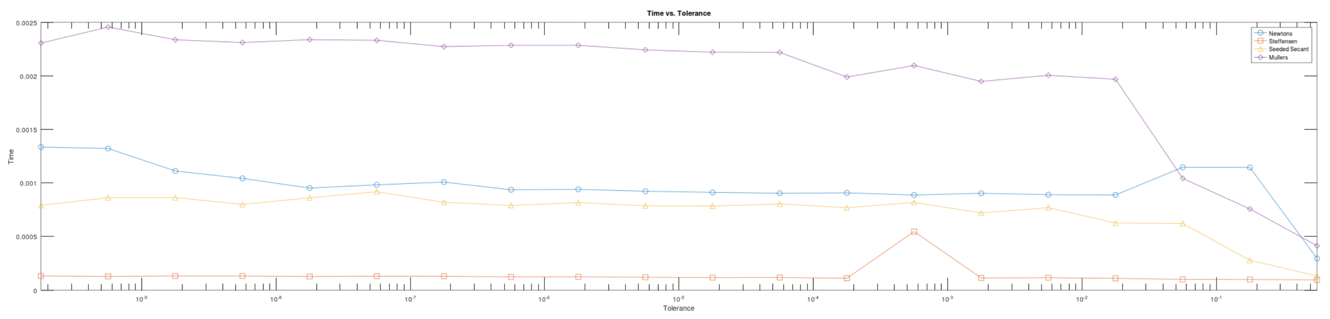


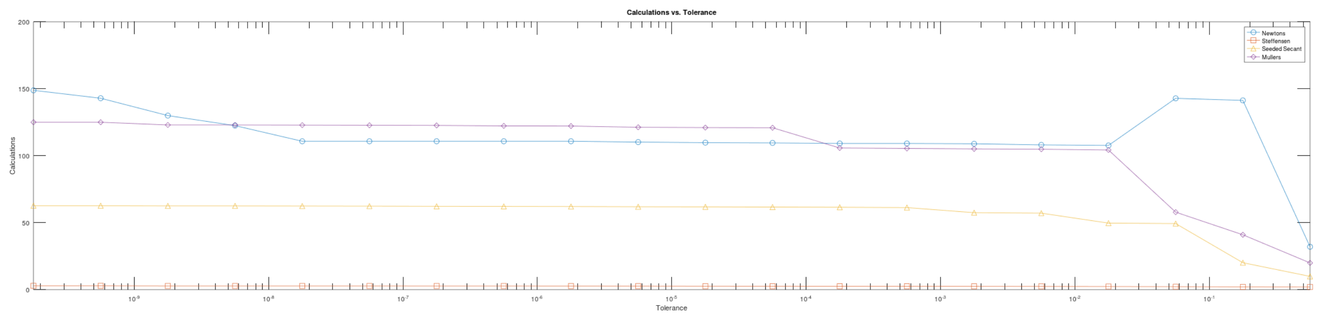


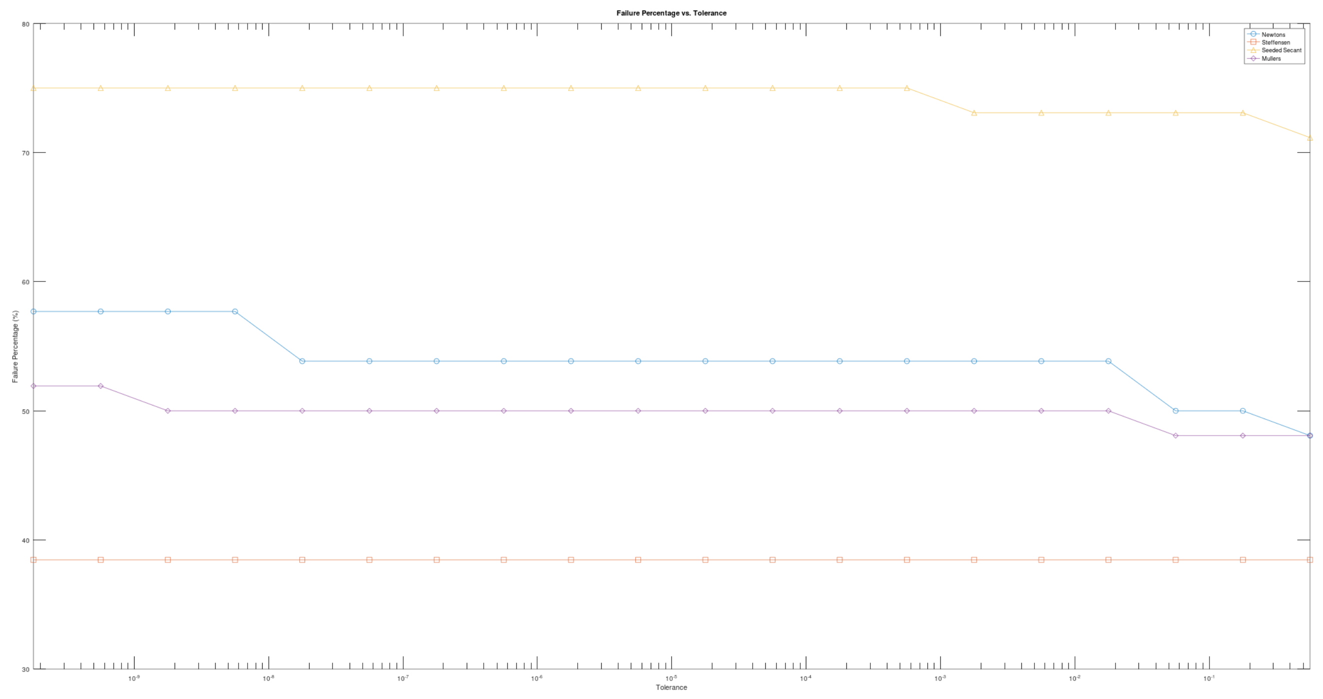


Both Seeded Secant and Newtons seem to fall for the trap down to Neverland on the left side of the logarithm, making them both very unreliable, Mullers seems to have similar issues, with a very high failure rate and unstable time and calc data, once again leaving Seffensons on top! makes sense as logarithms easily fulfill the derivative requirement.

Going next to exponentials, we have:

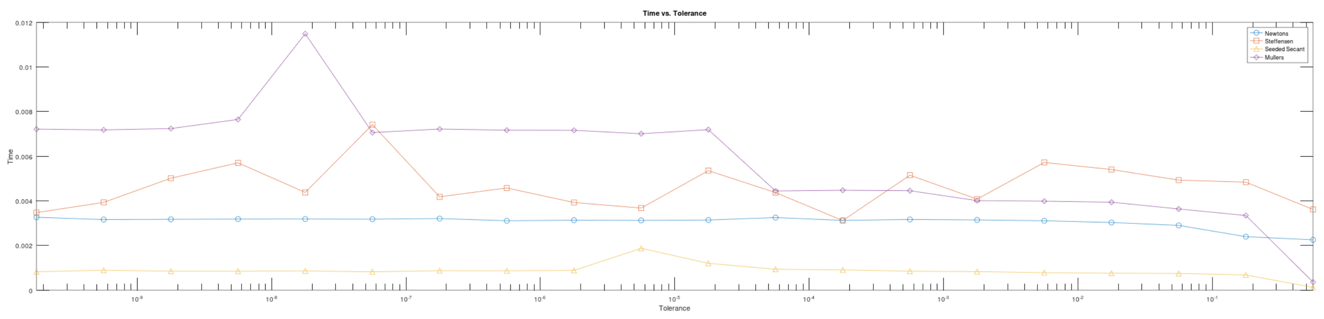


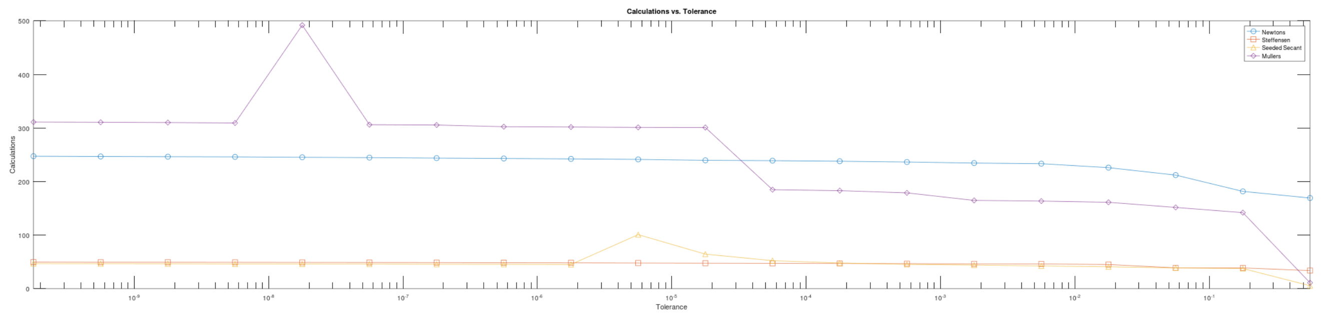


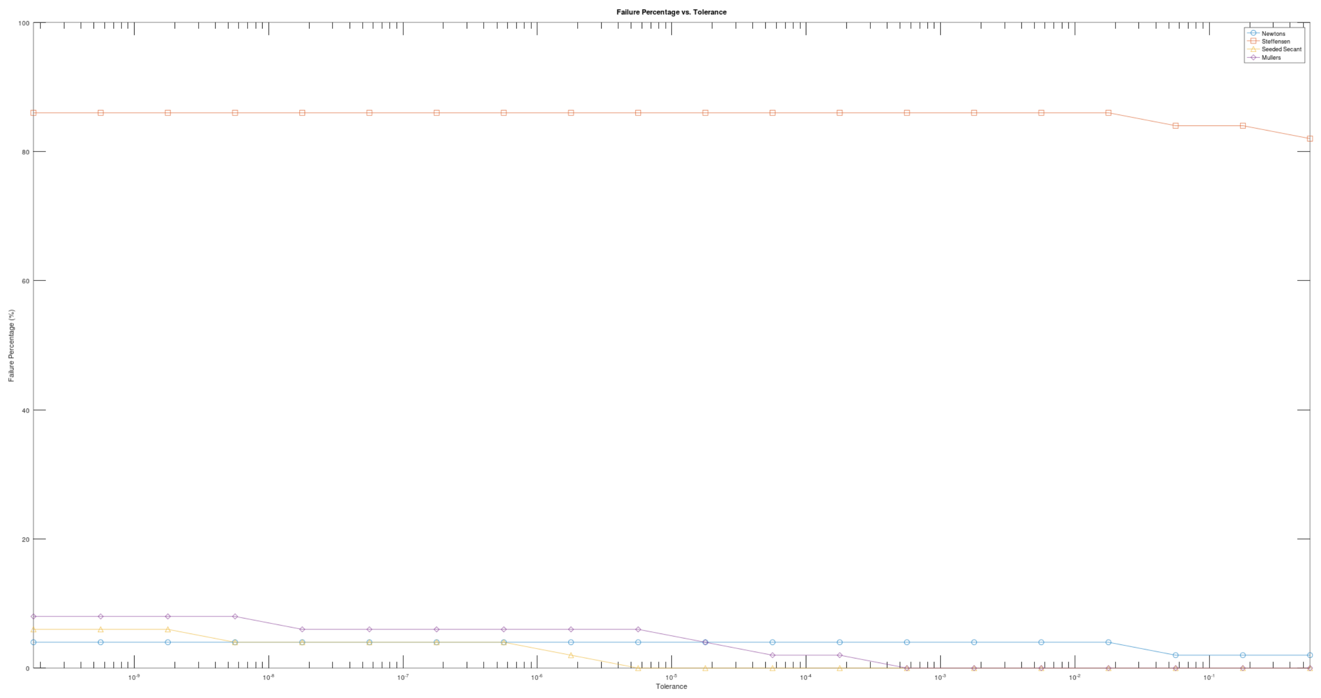


Steffenson's seems to shine here once again, however, it is important to note that the failure rate is very high for all functions, making the results pretty unreliable, but if it truly is not just luck then Steffenson has a lot going for it as a method!

Finnaly for the big bad mixup!







Here we see the unfortunate truth for Steffensons, while the other models are extremely flexible, and can handle most things thrown with under a 10% failure rate, as soon as you toss a big number or a polynomial, Steffens's start quickly failing. On another note, Secant's performance has been outstanding, being extremely reliable and fast for almost any function you throw at it. Makes me wonder how well the real newtons would have done if you gifted it the derivative, and if I were to do this experiment again I would add that as one of the methods.

Reflections

The biggest takeaway in my mind was how there was not a clear winner, the different methods and their effectiveness are hugely dependent on the function they are operating on, and as seen above just a small change can skyrocket the failure rate and time/function calls. Being familiar with a variety of methods that can tackle a lot of functions is key.

As a second takeaway, however, I think that Aitken's and Steffenson's methods are extremely powerful, and deserve a closer look, as they were often able to beat out the other methods when their conditions were met.

Finally, a chapter we skipped comes back to haunt us! I remember that there were modifications that could be done to newtons, secants, etc. that would bound them and prevent them from running off. These small changes would have changed everything, as the biggest problem with these methods was their tendency to wander off and fail when confronted with exponents and logs.