Programming Assignment 02

Using Bisection and Secant Methods to Find A Distance Between Two Plates

Nick Handelman
1/31/2018

Project Description

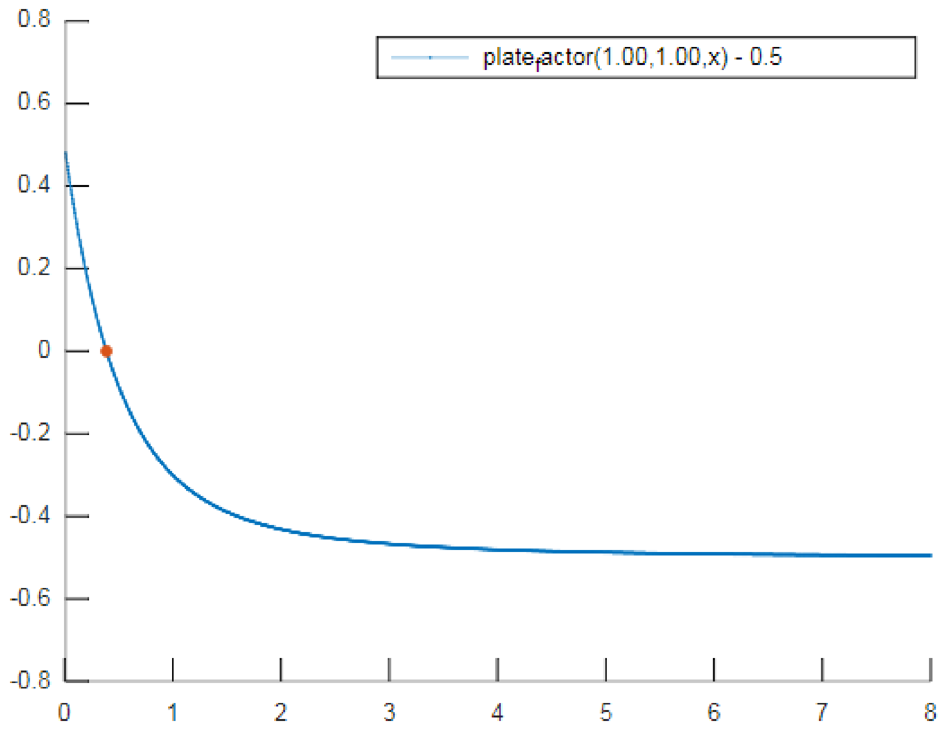
The goal of this project is to find the distance between two 1 meter^2 plates such that each plate fills 50% of the viewing area of the other plate. To find the distance, an equation describing the viewing area is rearranged to equal 0. Then, two bisection methods and a secant method are used to find a root of the equation, given 2 different intervals [a,b]. The root is the distance. The equation and root finding methods are coded in Octave (Appendix 1).

$$midpoint = a + \frac{b-a}{2} \qquad \text{(Bisection Midpoint Calculation 1)}$$

$$midpoint = \frac{a+b}{2} \qquad \text{(Bisection Midpoint Calculation 2)}$$

$$p_{n+1} = p_n - \frac{f(p_n) * (p_n - p_{n-1})}{f(p_n) - f(p_{n-1})} \qquad \text{(Secant Next Approximation Equation)}$$

Analysis



Graph of Plate Factor Equation

Interval 1 is [0.1, 4.0] and interval 2 is [0.3, 8.0]. The plate factor equation graph shows the interval [0.1, 8.0] and marks the root in red. The tolerance is set at 1e-12.
Bisection is guaranteed to converge to a root if given enough time. However, this method is slow so the

maximum number of iterations is set at 50, though at most 42 and 43 iterations are required to find the root in intervals 1 and 2, respectively.
The secant method converges faster than bisection, but there is no guarantee of convergence. Since there is no guarantee of convergence, no upper bound can be placed on the number of iterations required to find the root.

0.1 Upper Bounds for Number of Iterations on Bisection Algorithm

By choosing a smaller interval or lower tolerance, the upper bound decreases. However, an interval that is too small may not contain the root, and a tolerance too high may not approximate the root closely enough.

0.1.1 On the Interval [0.1, 4.0]

$Maximum\_number\_of\_iterations = \log_2 \frac{4.0-0.1}{1.0e-12} = 41.8266$

0.1.2 On the Interval [0.3, 8.0]

$Maximum\_number\_of\_iterations = \log_2 \frac{8.0-0.3}{1.0e-12} = 42.808$

0.2 Single-Precision Round-Off Error Between the Two Midpoint Calculations

Conclusions

Bisection calculates a root of approximately 0.38695 in 42 iterations for interval 1 and 43 iterations for interval 2. The tolerance is very low, so it isn't surprising that the upper bound on the number of iterations was reached for both intervals.
The secant method fails to find a root for either interval. By the 7th iteration for the first interval and 6th iteration for the second interval, Octave Online's [1] interpreter provides two division by zero warnings and outputs infinity or NaN as the last approximation. The first warning is given due to the $(f\_p\_n - f\_p\_nminus1$ term in secant.m. The second warning is given due to the term $(pi. * X. * Y)$ in plate_factor.m. My guess is that the $f(p_n)$ and $f(p_{n-1})$ terms are so close that Octave considers their difference to be zero.

References

[1] https://octave-online.net/

Appendix 1 - Octave Code

```
clear all
close all

% setup the function
f = inline('plate_factor(1.00,1.00,x) - 0.5')

% solve using the bisection algorithm
%  using first interval of (0.1, 4.0)
x = bisection(f,0.1, 4.0, 1.0e-12, 50);

% solve using the bisection algorithm
%  using second interval of (0.3, 8.0)
bisection(f,0.3, 8.0, 1.0e-12, 50);

% solve using the bisection algorithm
%  using first interval of (0.1, 4.0)
secant(f,0.1, 4.0, 1.0e-12, 50);

% solve using the bisection algorithm
%  using second interval of (0.3, 8.0)
secant(f,0.3, 8.0, 1.0e-12, 50);

% plot the function
hold on
fplot( f , [0.01, 8.0] );
plot(x, 0);
hold off

function [ ret ] = secant ( func, begin_pt, end_pt, TOL, max_iter)

  total_iterations = 0;

  fprintf(1,'===================\n');
  fprintf(1,'Secant algorithm\n');
  fprintf(1,'  begin endpoint: %.3f\n', begin_pt);
  fprintf(1,'  end endpoint: %.3f\n', end_pt);
  fprintf(1,'  tolerance: %.2e\n', TOL);
  fprintf(1,'  max iterations: %d\n\n', max_iter);

  p_nminus1 = begin_pt;
  p_n = end_pt;
  p_nplus1 = 0;
  f_p_nplus1 = 0;

  do
    ++total_iterations;

    f_p_nminus1 = func(p_nminus1);
    f_p_n = func(p_n);
    p_nplus1 = p_n - (f_p_n*(p_n - p_nminus1))/(f_p_n - f_p_nminus1);
    f_p_nplus1 = func(p_nplus1);
    p_nminus1 = p_n;
    p_n = p_nplus1;
  until(abs(f_p_nplus1) < TOL || total_iterations == max_iter || isnan(p_nplus1) || isinf(

  if(isnan(p_nplus1) || isinf(p_nplus1))
    printf('  Diverged. Last estimate was infinite or NaN\n');
  elseif(f_p_nplus1 > TOL)
    fprintf('  Failed to converge within tolerance. Last estimate was x = %.20d\n', p_nplus
  else
    fprintf(1,'  f(x) = 0 at approximately x = %.20d\n', p_nplus1);
  endif

  fprintf(1,'  total iterations: %d\n\n', total_iterations);

endfunction

##
## bisection method
##

function [ ret ] = bisection ( func, begin_pt, end_pt, TOL, max_iter)

  total_iterations = 0;

  fprintf(1,'====================\n');
  fprintf(1,'Bisection algorithm\n');
  fprintf(1,'  begin endpoint: %.3f\n', begin_pt);
  fprintf(1,'  end endpoint: %.3f\n', end_pt);
  fprintf(1,'  tolerance: %.2e\n', TOL);
  fprintf(1,'  max iterations: %d\n\n', max_iter);

  #
  #  bisection algorithm goes here
  #
  do
    ++total_iterations;

    mid_pt = begin_pt + (end_pt-begin_pt)/2;

    f_begin_pt = func(begin_pt);
    f_mid_pt = func(mid_pt);
    f_end_pt = func(end_pt);

    if (f_begin_pt*f_mid_pt < 0)
        end_pt = mid_pt;
    else
        begin_pt = mid_pt;
    endif

  until(abs(end_pt-begin_pt) < TOL || total_iterations == max_iter)

  fprintf(1,'  f(x) = 0 at approximately x = %.20d\n', mid_pt);
  fprintf(1,'  total iterations: %d\n\n', total_iterations);
  ret = mid_pt;
endfunction
```