Peter Julius M. Estacio

Grouped with Nino de Mesa and Osh Ong

ENGG 27.01

October 13, 2025

Project 2: Hybrid Root-Finding Approach

Code Locator:

Code Items	File name	Line numbers
Routine/Function implementing the hybrid root-finding approach	brent_root.cpp	48 – 198
Routine for the open method used	brent_root.cpp	Inverse Quadratic Interpolation: 107 - 115 Secant: 116 – 125
Routine for the bracketing method used	brent_root.cpp	133 – 152
Code used to decide between open and bracketing methods	brent_root.cpp	126 – 131

Project Evaluation – Hybrid Root-Finding Approach		
Item	Points	Rubrics
early work	8/8	(all or nothing) 8: at most 10% of the code in the final implementation differs from that in early work submission
required documentation and screenshots of at least one successful hybrid computation	50/50	0 - no documentation or screenshots or documentation or screenshots inconsistent with actual program run 50 - documentation and screenshots have all required details
trace details	10/10	0 - instructions not followed 10 - all details are correctly followed
modularity and generality of the hybrid root-finding routine	16/16	16 - routine can easily be used to find roots of other functions, with different parameters, and settings
C++ function implementation of the hybrid root-finding routine	8/8	8 - routine can easily be used to find roots of other functions, with different parameters, and settings and was encoded as a C++ function with appropriate parameters
code locator table	4/4	4 - all details in code locator table are accurate
self-evaluation	4/4	0 - no self-evaluation 2 - self-evaluation score differs from project score by more than 10 points 4 - self-evaluation accurate (or evaluating this item leads to an error)
total	100/100	

A screenshot of the program's output trace showcasing both open and bracketing methods is shown below:

```
=== Hybrid Root Finder (Brent) ===
  1) f(x) = x^3 - x - 2
  2) f(x) = cos(x) - x
  3) f(x) = e^(-x) - x
  4) f(x) = x*sin(x) - 1
  5) f(x) = (x-1)*(x-1)*(x-1)
  0) Exit
Choose a function [0-5]: 3
You chose: f(x) = e^(-x) - x
Enter a (left endpoint): -10
Enter b (right endpoint): 15
Enable step-by-step TRACE output? (y/n): y

Solving with Brent's method...
(secant method) 1.50e+01, f(1.49830e+01) = -1.49830e+01
(bisection method) 2.491497164046891e+00, f(2.49150e+00) = -2.40871e+00
(inverse quadratic interpolation) 1.002691329167518e-01, f(1.00269e-01) = 8.04325e-01
(bisection method) 1.295883148481822e+00, f(1.29588e+00) = -1.02223e+00
(bisection method) 6.980761406992867e-01, f(6.98076e-01) = -2.00535e-01
(secant method) 5.787749040511748e-01, f(5.78775e-01) = -1.81902e-02
(inverse quadratic interpolation) 5.7e-01, f(5.67137e-01) = 9.96892e-06
(bisection method) 5.73e-01, f(5.72956e-01) = -9.09966e-03
(bisection method) 5.70e-01, f(5.70046e-01) = -4.54724e-03
(bisection method) 5.69e-01, f(5.68592e-01) = -2.26923e-03
(bisection method) 5.679e-01, f(5.67864e-01) = -1.12978e-03
(bisection method) 5.675e-01, f(5.67501e-01) = -5.59944e-04
(bisection method) 5.673e-01, f(5.67319e-01) = -2.74997e-04
(bisection method) 5.6723e-01, f(5.67228e-01) = -1.32516e-04
(bisection method) 5.6718e-01, f(5.67182e-01) = -6.12743e-05
(bisection method) 5.6716e-01, f(5.67160e-01) = -2.56528e-05
(bisection method) 5.6715e-01, f(5.67148e-01) = -7.84200e-06
(secant method) 5.67143e-01, f(5.67143e-01) = -9.02645e-12
(inverse quadratic interpolation) 5.67143290410e-01, f(5.67143e-01) = 0.00000e+00

=== Result ===
Converged: yes
Root     : 5.671432904097838e-01
f(root)  : 0.000000e+00
Iterations: 20
Would you like to choose again? (y/n): n
```

The function used to implement Brent's method is shown below, as written in brent_root.cpp:

```cpp
//Brent's Root-Finding Implementation
RootResult brent_root_find(function<double(double)> f,
  double a, double b,
  double xtol,
  double ftol,
  int max_iter,
  bool trace)
{
   RootResult R;

  if (!(xtol > 0)) xtol = 1e-12;
  if (!(ftol > 0)) ftol = 1e-12;
  if (max_iter <= 0) max_iter = 200;
  if (a == b) return R;
  if (a > b) swap(a, b);

  double fa = f(a);
  double fb = f(b);
  if (!isfinite(fa) || !isfinite(fb)) return R;
  if (fa * fb > 0.0) return R; // root not bracketed

  if (fabs(fa) < fabs(fb))
  {
    swap(a, b);
    swap(fa, fb);
  }

  double c = a;
  double fc = fa;
  double s = b;
  double fs = fb;
  double prev_b = b;
  bool mflag = true;
  double d = 0.0;

  for (int iter = 1; iter <= max_iter; ++iter)
  {
    R.iterations = iter;

    if (fabs(fb) <= ftol)
    {
      R.converged = true;
      R.root = b;
      R.froot = fb;
      return R;
    }

    if (fabs(b - a) <= xtol)
    {
      R.converged = true;
      R.root = b;
      R.froot = fb;
      return R;
    }
```

```
102
103     double s_candidate = b;
104     bool used_interp = false;
105     string method_used = "bisection method";
106
107     if (fa != fc && fb != fc)
108     {
109       s_candidate =
110         (a * fb * fc) / ((fa - fb) * (fa - fc)) +
111         (b * fa * fc) / ((fb - fa) * (fb - fc)) +
112         (c * fa * fb) / ((fc - fa) * (fc - fb));
113       used_interp = true;
114       method_used = "inverse quadratic interpolation";
115     }
116     else
117     {
118       double denom = fb - fa;
119       if (fabs(denom) > 1e-30)
120       {
121         s_candidate = b - fb * (b - a) / denom;
122         used_interp = true;
123         method_used = "secant method";
124       }
125     }
126     bool cond1 = (s_candidate < (3.0 * a + b) / 4.0) || (s_candidate > b);
127     bool cond2 = (mflag && (fabs(s_candidate - b) >= fabs(b - c) / 2.0));
128     bool cond3 = (!mflag && (fabs(s_candidate - b) >= fabs(c - d) / 2.0));
129     bool cond4 = (mflag && (fabs(b - c) < xtol));
130     bool cond5 = (!mflag && (fabs(c - d) < xtol));
131     bool cond6 = !used_interp;
132
133     if (cond1 || cond2 || cond3 || cond4 || cond5 || cond6)
134     {
135       s = 0.5 * (a + b);
136       method_used = "bisection method";
137       mflag = true;
138     }
139     else
140     {
141       s = s_candidate;
142       mflag = false;
143     }
144
145     fs = f(s);
146     if (!isfinite(fs))
147     {
148       s = 0.5 * (a + b);
149       fs = f(s);
150       method_used = "bisection method";
151       mflag = true;
152     }
```

```
153
154     if (trace)
155     {
156       int agree = sigfigs_agreement(s, prev_b);
157       if (s == prev_b)
158       {
159         cout << "(" << method_used << ") "
160              << scientific << setprecision(numeric_limits<double>::digits10) << s
161              << ", f(" << scientific << setprecision(7) << s
162              << ") = " << scientific << setprecision(7) << fs << endl;
163       }
164       else
165       {
166         print_trace_line(method_used, s, fs, agree);
167       }
168     }
169
170     d = c;
171     c = b;
172     fc = fb;
173
174     if (fa * fs < 0.0)
175     {
176       b = s;
177       fb = fs;
178     }
179     else
180     {
181       a = s;
182       fa = fs;
183     }
184
185     if (fabs(fa) < fabs(fb))
186     {
187       swap(a, b);
188       swap(fa, fb);
189     }
190
191     prev_b = b;
192   }
193
194   R.root = b;
195   R.froot = fb;
196   R.converged = false;
197   return R;
198 }
```