Zack Rimshnick
I pledge my honor that I have abided by the Stevens Honor System.
Project 1 - Report

First off, in my _start I initialized all of my registers storing data including my coefficient array, string, degree, lower and upper initial bounds, tolerance, and negative tolerance. These are loaded into usable registers and values are properly assigned. I also use SCVTF to convert integers in X registers to doubles in D registers. The procedure is called.

```
148 _start:
149     ADR  X0, msg
150     ADR  X1, coeff
151     ADR  X2, N
152     ADR  X3, a
153     ADR  X4, b
154     ADR  X6, t
155     ADR  X26, neg
156
157     LDR  X2, [X2]
158     LDR  D3, [X3]
159     LDR  D4, [X4]
160     LDR  D6, [X6]
161     LDR  D26,[X26]
162     FMUL D26, D26, D6
```

```
164     MOV   X5, 0
165     MOV   X7, 0
166     MOV   X8, 8
167     MOV   X13, 0
168     MOV   X14, 0
169     MOV   X15, 0
170     MOV   X16, X2
171     MOV   X17, 1
172     MOV   X18, 1
173     MOV   X19, 0
174     MOV   X20, 0
175     MOV   X22, 2
176
177     SCVTF D2, X2
178     SCVTF D5, X5
179     SCVTF D8, X8
180     SCVTF D13, X13
181     SCVTF D14, X14
182     SCVTF D15, X15
183     SCVTF D16, X16
184     SCVTF D17, X17
185     SCVTF D18, X18
186     SCVTF D19, X19
187     SCVTF D22, X22
188
189
190     BL  Proc
```

```
198 .data
199     coeff:  .double   0.2, 3.1, -0.3, 1.9, 0.2  // coefficients
200     N:      .dword    4    // degree/highest power of x
201     a:      .double   -1   // lower x bound
202     b:      .double   1    // upper x bound
203     t:      .double   .01  // tolerance must be double
204     neg:    .double   -1   // negative 1
205     msg:    .string  "root = %lf, function value = %lf \n"  // s
206
```

To complete this project, I started off by writing a few loops to get f(a), f(b), and f(c), all of which are essentially the same code, but with different registers for a, b, and c. The loop multiplies each coefficient by the x value, but also by x multiple times, depending on the degree of each element. At the end of the loop, it also resets registers involved in counters, indexes, etc. There's also a piece of code that gets midpoint c by adding (a+b)/2.

```
16  getfA:                  // procedure to get the LOWER BOUND a y value f(a)
17      LDR     D1, [X1, X7]    // loads current index (coefficient) into d1
18
19      FMUL    D1, D1, D17     // coefficient*x
20
21      FADD    D13, D13, D1    // puts product into the D13 f(a)
22
23      ADD     X7, X7, 8       // incremments index by 8
24      FMUL    D17, D17, D3    // increments the x by f(a)
25      ADD     X20, X20, 1     // incrememnt counter by 1
26
27      CMP     X20, X2         // counter and highest power
28      B.LE    getfA           // while counter less than, do getfA again
29
30      MOV     X7, 0           // decrement back to 0
31      FMOV    D17, D18        // decrement back to 1
32      MOV     X20, 0          // decrement back to 0
```

```
35  getfB:
36      LDR     D1, [X1, X7]
37
38      FMUL    D1, D1, D17
39
40      FADD    D14, D14, D1
41
42      ADD     X7, X7, 8
43      FMUL    D17, D17, D4
44      ADD     X20, X20, 1
45
46      CMP     X20, X2
47      B.LE    getfB
48
49      MOV     X7, 0
50      FMOV    D17, D18
51      MOV     X20, 0
52
```

```
53  getC:
54      FADD  D5, D3, D4
55      FDIV  D5, D5, D22
56
57
58  getfC:
59      LDR     D1, [X1, X7]
60
61      FMUL    D1, D1, D17
62
63      FADD    D15, D15, D1
64
65      ADD     X7, X7, 8
66      FMUL    D17, D17, D5
67      ADD     X20, X20, 1
68
69      CMP     X20, X2
70      B.LE    getfC
71
72      MOV     X7, 0
73      FMOV    D17, D18
74      MOV     X20, 0
```

Next, I wrote code that would compare the tolerance and midpoint y value after each previous loop, which checks if -t < f(c) AND f(c) < t. If so, then the code ends, but if not then it goes to a new loop that will choose the bounds for the next iteration through the program. It pretty much compares f(a), f(b), and f(c) to 0, and depending on which are positive or negative, it will re-assign new bounds for a and b (bounds: f(a) negative and f(c) positive, f(a) positive and f(c) negative, f(c) negative and f(b) positive, f(c) positive and f(b) negative).

```
76  /////// check if were done
77
78      FCMP   D26, D15       // compares negative tolerance and f(c)
79      B.LE   skip           // if -t < f(c), then gotta check next
80      B      chooseBounds   //  else, skip to chooseBounds
81
82  skip:                    // skip
83      FCMP   D15, D6        // compares f(c) and tolerance
84      B.LE   Done           // if f(c) < t, then done
85      B      chooseBounds   //  else, skip to chooseBounds
86
87
88  /////// NOW NEED TO CHECK WHICH TO USE EITHER f(a) and f(b) or f(b) and f(c)
89  chooseBounds:            // label to check/choose bounds
90      FCMP   D15, D19       // Compares midpoint f(c) and 0 to check if we are at
91      B.EQ   Done           // if equal, then go to Done
92
93      FCMP   D13, D19       // Compares f(a) and 0
94      B.EQ   Done           // if f(a) = 0, then Done
95
96      FCMP   D14, D19       // Compares f(b) and 0
97      B.EQ   Done           // if f(b) = 0, then Done
98
99      FCMP   D13, D19       // Compares f(a) and 0
100     B.GT   fApos          // if f(a) > 0, then check midpoint f(c)
101
102     B      fAneg          // if f(a) < 0, then go to fAneg
103
```

```
104 fApos:
105     FCMP   D15, D19
106     B.LT   gotAandC
107
108     FCMP   D14, D19
109     B.LT   gotBandC
110
111 fAneg:
112     FCMP   D15, D19
113     B.GT   gotAandC
114
115     B      gotBandC
```

```
117 gotAandC:
118     FMOV   D3, D3
119     FMOV   D4, D5
120
121     FMOV   D13, D19
122     FMOV   D14, D19
123     FMOV   D15, D19
124
125     B      getfA
126
127 gotBandC:
128     FMOV   D3, D5
129     FMOV   D4, D4
130
131     FMOV   D13, D19
132     FMOV   D14, D19
133     FMOV   D15, D19
134
135     B      getfA
```

The code runs itself from there on, the only thing left is the Done label which the program goes to when the tolerance requirements are met. Here, the x and y values for the midpoint (now the root) are moved into D0 and D1 so that they can be properly printed in the string. The stack size is also fixed up. RET is called to return back to the _start and finish the program.

```
137 Done:                   // Label for when done with program
138     FMOV   D0, D5       // copies c to d0 for printing root
139     FMOV   D1, D15      // copies f(c) to d1 for printing y value
140
141     BL     printf       // prints
142
143     LDR  LR, [SP]       // loads LR from stack pointer
144     ADD  SP, SP, 8      // sets stack pointer by size
145
146     RET                 // return end
```