NUMERICAL COMPUTING HW 1

RAPHAEL SOFAER

1. Exercise 1.1

1.1. **a.** I used a small ruby script to generate the values of x_k.

```
0.upto(10) { |n| puts 0.95**(2**n)}
```

Values of $x_k = 0.95^{2^k}$	
0	0.95
1	0.9025
2	0.81450625
3	0.663420431289062
4	0.440126668651765
5	0.193711484458501
6	0.037524139211116
7	0.00140806102353521
8	1.98263584599904e-06
9	3.93084489784032e-12
10	1.54515416108773e-23

.

For any value of c, $0 < c < 1$, $x_k = 0.95^{2^k}$ will converge to 0. It converges quadratically, with an asymptotic error constant of 1.

1.2. **b.** Once again, I used a ruby script to generate the values of x_k.

```
1.upto(12) { |n| puts 1.0/(n**n)}
```

Values of $x_k = 1/k^k$	
1	1.0
2	0.25
3	0.037037037037037
4	0.00390625
5	0.00032
6	2.14334705075446e-05
7	1.21426567890201e-06
8	5.96046447753906e-08
9	2.5811747917132e-09
10	1.0e-10
11	3.50493899481392e-12
12	1.12156654784615e-13

2. Exercise 1.2

2.1. **a.** I wrote my program in octave, the GNU clone of matlab.

```
function result = polynomial (x)
  result = x^3 - 2.5*x - 4.011;
endfunction

function [a, b] = bisect (maxit, a0, b0, f)
  a = a0;
  b = b0;
  if f(a)*f(b)>=0
    error("Function has the same sign at both sides of the given interval.");
    return;
  endif

  for i = 1:maxit
    new = a + (b-a)/2;
    value = f(new);
    if value == 0
      a = new;
      b = new;
      return;
    elseif value*f(a) < 0
      b = new;
    elseif value*f(b) < 0
      a = new;
    else
      error("Failure.");
    endif

    format = "After iteration %d, the interval of uncertainty is %f,
              from a = %f to b = %f, with values f(a) = %f and f(b) = %f.\n";
    printf(format, [i, a, b, f(a), f(b)]);
  endfor
endfunction

bisect(10, -2, 4, @polynomial);
```

2.2. **b.** For $f(x) = x^3 - 2.5x - 4.011$, we can tell whether there is a root in [-2,4] by checking whether it crosses the x-axis.

$f(-2) = -8 - (-5) - 4.011 \ = 3 - 4 = -1$, which is negative,

$f(4) \ = 4 * 4 * 4 - 2.5 * 4 - 1 * 4 = 16 * 4 - 3.5 * 4$, which is positive.

Since f(x) crosses the x axis, it must have a real root in [-2,4]. By the bisection program, this root is at $x = 2.1$.

We know $f'(x) = 3x^2 - 2.5$, which has roots at $\pm\sqrt{5/6}$. f(x) is negative at both of these roots of f'(x), and goes to negative infinity as x goes to negative infinity, so f(x) can only have 1 real root.

i. With a starting interval of [0.5,3.1], the interval of uncertainty after 12 iterations is: [2.099609, 2.100244], which contains x^*. These results are what I would expect.

ii. With a starting interval of [-2,3.1], the interval of uncertainty after 12 iterations is: [2.098926, 2.100171], which contains x^*. These results are what I would expect.

3. Exercise 1.3

3.1. **a.** $f(x) = (x-1)^7 = (x-1)(x-1)(x-1)(x-1)(x-1)(x-1)(x-1)$. Since the real numbers are an integral domain, a product of real numbers can only be 0 if one of the numbers being multiplied is 0. Therefore, if f(x) = 0, x-1 = 0, so x = 1.

3.2. **b.** With a starting interval of [0.95,1.01], the interval of uncertainty after 12 iterations is: [0.999248, 0.999263]. This interval is the right size, but it has moved fully under the root at x=1.

3.3. **c.** With a starting interval of [0.95,1.01], the interval of uncertainty after 12 iterations is: [0.999995, 1.000010]. This is what I would expect, because the interval is halving in size at each step and it contains the root. Since this result is different from the result using an expanded polynomial, there must be machine error (floating point error?) involved in the incorrect answer given by the expanded polynomial.

4. Exercise 1.4

4.1. **a.** I implemented Newton's method with an octave script.

```
function x = newton (maxit, f, fp, x0)
  x = x0;
  slope = fp(x);
  value = f(x);
  for i = 1:maxit
    x = (value-slope*x)/(-slope);
    slope = fp(x);
    value = f(x);
    printf("After iteration %d, x = %f, f(x) = %f, and f'(x) = %f.\n", [i, x, value, slope]);
    if abs(value) <= 10^(-15)
      disp("abs(f(x)) <= 10^(-15), stopping iteration.")
      return;
    endif
  endfor
endfunction
```

4.2. **b.** Using a maxit of 12 and a starting point of 3, Newton's method converged after 6 iterations. This is much faster than bisection, which took 12 iterations to reach an accuracy comparable to what Newton's method had after 3 iterations. Since Newton's method is supposed to converge quadratically, as opposed to bisection's linear convergence, this is what I would expect.

4.3. **c.** Using a maxit of 15 and a starting point of 2.1, Newton's method reached x=1.108941 after 15 iterations. I initially would have expected more rapid convergence, but looking at the values of $f'(x)$ makes it clear that f(x) is very flat near x=1. With very low values of f'(x), Newton's method converges slowly, so the slower convergence compared to part b (which had high values of f'(x)), makes sense.

5. Exercise 1.5

5.1. **a.** I implemented the Secant Method with an octave script.

```
function x = secant (maxit, f, x0, x1)
  for i = 1:maxit
    slope = (f(x0) - f(x1))/(x0-x1);
    x2 = (slope*x0 - f(x0))/slope;

    x0 = x1;
    x1 = x2;
    printf("After iteration %d, x = %f and f(x) = %f.\n", [i, x1, f(x1)]);

    if abs(f(x1)) <= 10^(-15)
      disp("abs(f(x)) <= 10^(-15), stopping iteration.")
      return;
    endif
  endfor
endfunction
```

5.2. **b.** The secant method converged to x=2.1 after 11 iterations, slower than Newton's method, but still much faster than bisection. The iterates oscillated between 1 and 3 before settling on 2.1, which I did not expect. At iteration 3, my program had reached x=3.085691, with f(x) = 17.655142, much farther from the exact root than iteration 2, x = 1.546445 and f(x) = -4.178802. The rate of convergence was what I expected, but the behavior of the iterates was not.

5.3. **c.** With x0 = 3.1 and x1 = -2, I expected the secant method to converge in 20 or so iterations, but it just oscillated wildly. At iteration 4, it gave x =21. At iteration 13, x=-26. It seemed to have calmed down slightly after 25 iterations, but it was still far off with x = -1.756616 and f(x) = -5.039848. This result was far worse than the bisection method, which reached the interval [2.098926, 2.100171] after 12 iterations with the same initial values. Perhaps the existence of local extrema in between the two starting points disrupted the secant method.