Lab 2

Table of Contents

Problem 1

```
print('Problem 1.')
disp(sprintf('\n'));
ten_digits_of_accuracy =.5E-10;
```

```
Problem 1.
```

a. log(1.9) = log(1-(-0.9)) To calculate log(1.9) x needs to be -0.9

```
reqx = -0.9;
```

b.

```
xtrue = 0.641853886172394 69;
k = 1;
while abs(eq1(reqx, k) - xtrue) > ten_digits_of_accuracy
    k = k + 1;
end
xcalc = eq1(reqx, k);
print('log(1.9) = log(1-(-0.9))');
disp(sprintf('True value: %0.11f', xtrue));
disp(sprintf('Calculated at %d (terms) iterations: %0.11f', k, xcalc));
```

```
log(1.9) = log(1-(-0.9))
True value: 0.64185388617
Calculated at 170 (terms) iterations: 0.64185388613
```

c. (x^(2*k - 1)/(2*k - 1)) To calculate log(1.9) x needs to be 0.9/2.9

```
reqx = 0.9/2.9;
```

d.

```
xtrue = 0.64185388617239469;
k = 1;
while abs(eq2(reqx, k) - xtrue) > ten_digits_of_accuracy;
    k = k + 1;
end
xcalc = eq2(reqx, k);
print(xcalc);
```

```
print('(x^(2*k - 1)/(2*k - 1))');
disp(sprintf('True value: %0.11f', xtrue));
disp(sprintf('Calculated at %d (terms) iterations: %0.11f', k, xcalc));
```

```
        0.6419

    (x^(2*k - 1)/(2*k - 1))
    True value: 0.64185388617
    Calculated at 9 (terms) iterations: 0.64185388615
```

e. The second Taylor series is better for approximating log(1.9) because it takes less iterations to get a value within 10 significant digits of accuracy. The second equations is faster because the exponent in the numerator is larger than the exponent in the first equation. The larger exponent means the second equation has a faster rate of convergence.

Problem 2.

```
disp(sprintf('\n'));
disp(sprintf('\n'));
```

a. ((4 + x)^(-1/2) - 2) / x

```
print('f(x) = ((4 + x)^(-1/2) - 2) / x')
print('f_fixed(x) = 1/(sqrt(4 + x) + 2);')
do_table(@eq2a, @eq2a_fixed)
disp(sprintf('\n'));
```

```
    f(x) = ((4 + x)^(-1/2) - 2) / x
    f_fixed(x) = 1/(sqrt(4 + x) + 2);
    Table:
    x   f(x)     (fixed) f(x)
    0.1   0.2484567313   0.2484567313
    0.01   0.2498439450   0.2498439450
    0.001   0.2499843770   0.2499843770
    0.0001   0.2499984375   0.2499984375
    1e-05   0.2499998438   0.2499998438
    1e-06   0.2499999843   0.2499999844
    1e-07   0.2499999985   0.2499999984
    1e-08   0.2499999763   0.2499999998
    1e-09   0.2500000207   0.2500000000
    1e-10   0.2500000207   0.2500000000
    1e-11   0.2499778162   0.2500000000
    1e-12   0.2500222251   0.2500000000
    1e-13   0.2486899575   0.2500000000
    1e-14   0.2220446049   0.2500000000
    1e-15   0.0000000000   0.2500000000
    1e-16   0.0000000000   0.2500000000
    1e-17   0.0000000000   0.2500000000
```

```
1e-18   0.0000000000   0.2500000000
1e-19   0.0000000000   0.2500000000
1e-20   0.0000000000   0.2500000000
```

The first function adds a very small number to 4 and than takes the square root of the sum. When x because so small that 4 + x is rounded to 4 (because of the limitations associated with representing floating point numbers) 'sqrt(4+x) - 2' becomes equal to 0. The 'better' function removes the subtraction of the two nearly equal numbers and consequently removes the error.

b. (1 - e^(-x))/x

```
print('f(x) = (1 - e^(-x))/x')
do_table(@eq2b, @eq2b_fixed)
disp(sprintf('\n'));
```

```
f(x) = (1 - e^(-x))/x
Table:
x   f(x)     (fixed) f(x)
0.1   0.9516258196   0.9048374180
0.01   0.9950166251   0.9900498337
0.001   0.9995001666   0.9990004998
0.0001   0.9999500017   0.9999000050
1e-05   0.9999950000   0.9999900000
1e-06   0.9999995000   0.9999990000
1e-07   0.9999999495   0.9999999000
1e-08   0.9999999939   0.9999999900
1e-09   0.9999999717   0.9999999990
1e-10   1.0000000827   0.9999999999
1e-11   1.0000000827   1.0000000000
1e-12   0.9999778783   1.0000000000
1e-13   1.0003109452   1.0000000000
1e-14   0.9992007222   1.0000000000
1e-15   0.9992007222   1.0000000000
1e-16   1.1102230246   1.0000000000
1e-17   0.0000000000   1.0000000000
1e-18   0.0000000000   1.0000000000
1e-19   0.0000000000   1.0000000000
1e-20   0.0000000000   1.0000000000
```

The first function was subtracting something very close to 1 from 1 itself. This caused a zero value to be in the numerator which caused the function to evaluate to zero. By using a Taylor series we can use the expanded terms to make the numerator non-zero after subtraction.

The code.

```
%function y = do_table(func, func_fixed)
%    disp(sprintf('Table:'));
%    disp(sprintf('x \t f(x) \t \t (fixed) f(x)'));
%    iterations = 20;
%    for i=1:iterations,
```

```
%       x=10^(-i);
%       disp(sprintf('%g \t %0.10f \t %0.10f', x, func(x), func_fixed(x)));
%     end
%
%function y = eq1(x, n)
%     k = 0;
%     y = 0;
%     while k < n
%         k = k + 1;
%         y = y + power(x, k)/k;
%     end
%     y = y * -1.0;
%
%function y = eq2a_fixed (x)
%     y = 1/(sqrt(4 + x) + 2);
%
%function y = eq2a (x)
%     y = (sqrt(4 + x) - 2) / x;
%
%function y = inside_function(x)
%     y = exp(-x);
%
%function y = eq2b_fixed (x)
%     y = ((1 - taylor(inside_function, 10))/x);
%
%function y = eq2b (x)
%     y = (1 - exp(-x))/x;
%     j
%function y = eq2(x, n)
%     y = double(0);
%     for k=1:n,;
%         % (x^(2*k - 1)/(2*k - 1))
%         y = y + (x ^ ((2 * k) - 1))/((2 * k) - 1);
%     end
%     y = y * (2);
%
%function text = print(s)
%     disp(s);
```

Published with MATLAB® 7.13