MATH 140: Mathematical Methods for Optimization

Assignment 3—Spring 2024

Due February 06, 2024

By:Ronald Nap

1. **(5 Points)** (Optional, extra credit) Prove that if $f(x)$ is convex, then

$$(\nabla f(y) - \nabla f(x))^T (y-x) \geq 0.$$

In the one-dimensional case, represent this theorem pictorially.

Solution: To prove that if $f(x)$ is convex, then

$$(\nabla f(y) - \nabla f(x))^T (y-x) \geq 0,$$

we start with the definition of convexity. A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for any $x, y \in \mathbb{R}^n$ and $\lambda \in [0,1]$, the following holds[1]:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

Using the gradient, we express a property of convex functions:

$$f(y) \geq f(x) + \nabla f(x)^T (y-x).$$

Consider $g(t) = f(x + t(y-x))$ for $t \in [0,1]$. Since f is convex, so is $g(t)$, and thus:

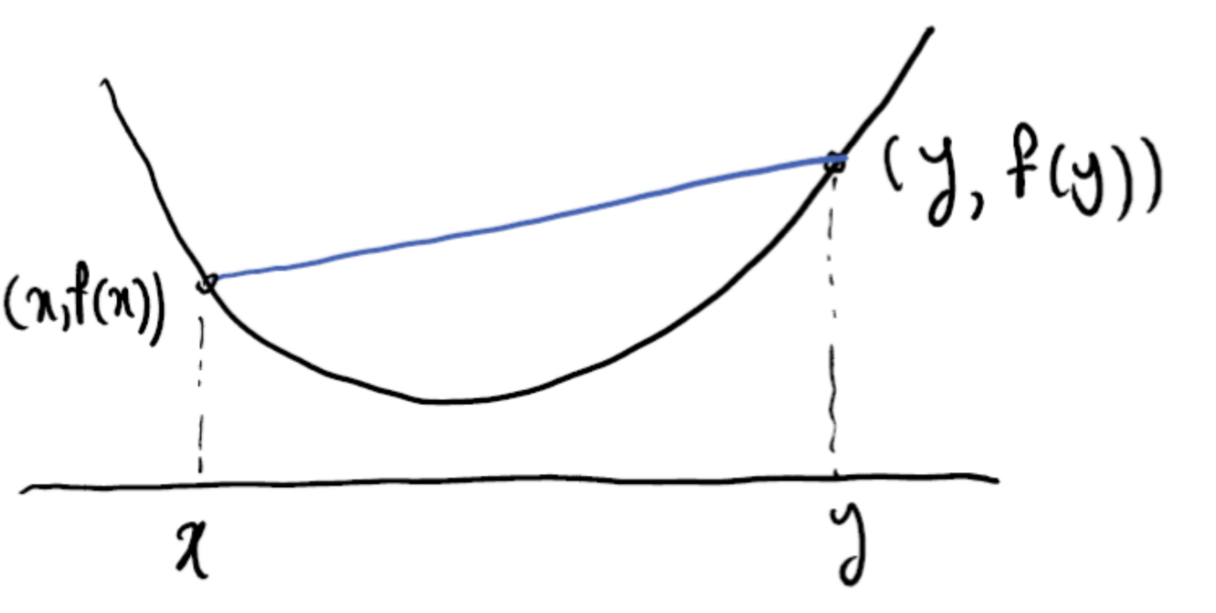
$$g'(t) = \nabla f(x + t(y-x))^T (y-x)$$

is non-decreasing. This implies $g'(1) \geq g'(0)$, or:

$$\nabla f(y)^T (y-x) \geq \nabla f(x)^T (y-x),$$

leading to:

$$(\nabla f(y) - \nabla f(x))^T (y-x) \geq 0,$$



2. **(5 Points)** Due to the finite precision in floating point number representation, there are gaps between consecutive numbers. The size of these gaps depends on the size of the number and on the precision (e.g., double or single precision). MATLAB provides the function `eps(·)` (the analogue in NumPy is the function `spacing`), which returns, for a number, the distance to the next floating point number in the same precision. Using the form of double and single floating point number representation, explain the values you find for

`eps(1)`,[2]
`eps(single(1))`,[3]
`eps(2^40)`,
`eps(single(2^40))`.

Solution: This output demonstrates the concept of machine epsilon and the differences in precision between single and double precision floating point representations. For the number 1.0, the double precision epsilon is approximately 2.22e-16, indicating a higher precision, while the single precision epsilon is approximately 1.19e-07, indicating a lower precision. Similarly, for the number 2^{40}, we see a larger gap in the representable numbers, with the double precision having a gap of about 0.000244 and the single precision having a much larger gap of 131072.0

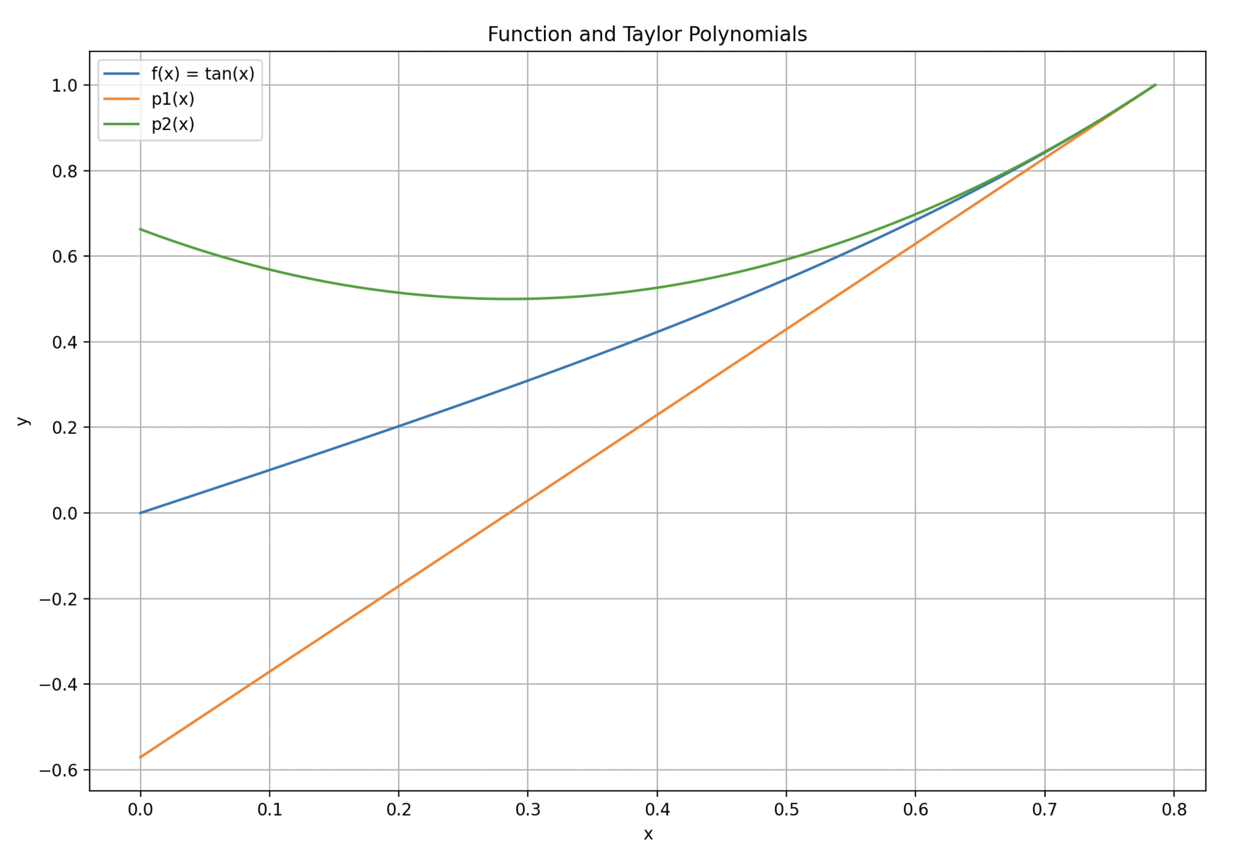
```
1.0 Double Precision:2.220446049250313e-16

1.0 Single Precision:1.1920928955078125e-07

2^40 Double Precision:0.000244140625

2^40 Single Precision:131072.0
```

3. **(5 Points)** Consider $f(x) = \tan x$, and point $a = \frac{\pi}{4}$ in the interval $[\alpha, \beta] = [0, \frac{\pi}{4}]$.

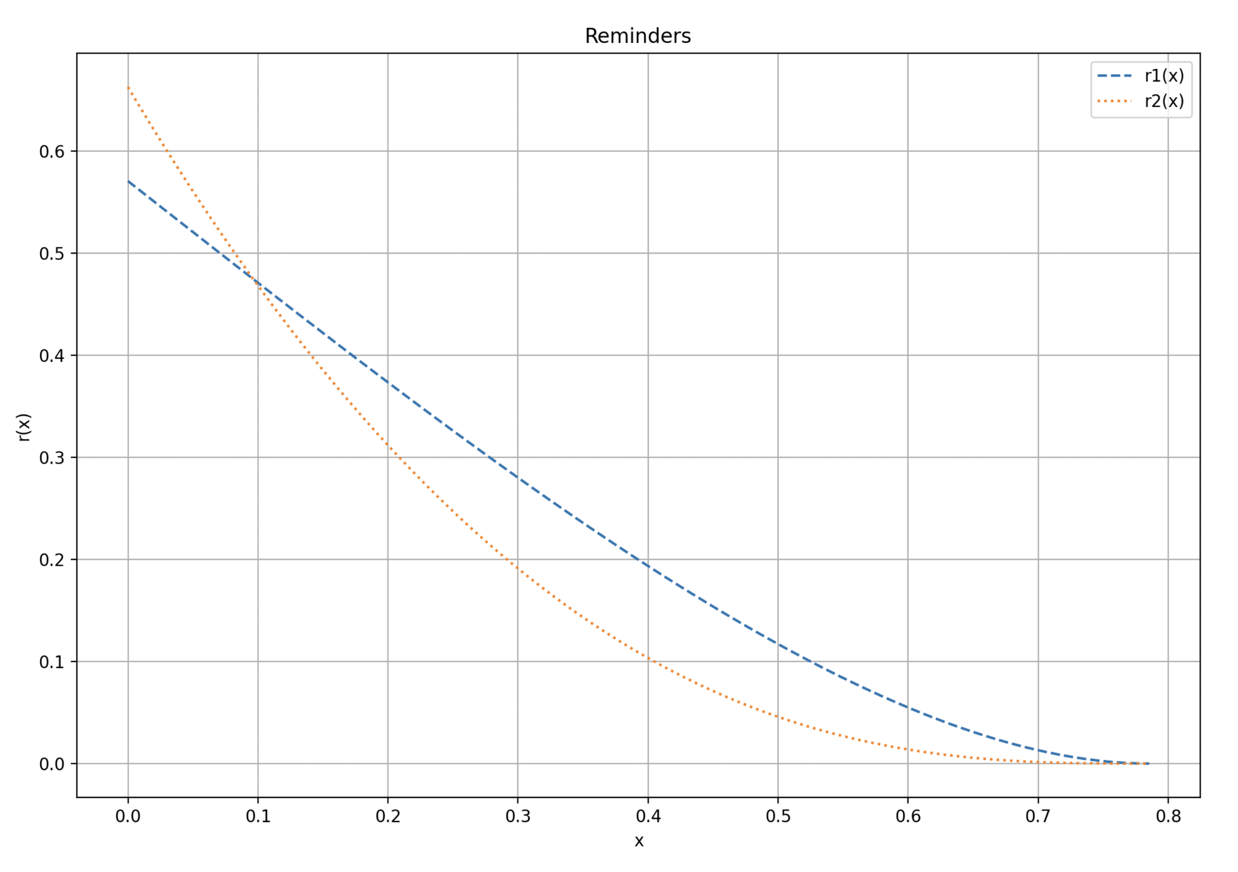
(a) Compute the Taylor polynomials $p_n(x)$, $n = 1, 2$ about a point a.

Solution: The first Taylor polynomial $p_1(x)$ is $f(a) + f'(a)(x-a)$, where $f'(x) = \sec^2 x$. At a, $f'(a) = 2$. Thus, $p_1(x) = 1 + 2(x - \frac{\pi}{4})$. The second Taylor polynomial $p_2(x)$ includes the second derivative $f''(x) = 2\sec^2 x \tan x$, so $f''(a) = 4$. Therefore, $p_2(x) = 1 + 2(x - \frac{\pi}{4}) + 2(x - \frac{\pi}{4})^2$.

(b) Plot the function $f(x)$ and the Taylor polynomial $p_n(x)$, $n = 1, 2$ over the interval $[\alpha, \beta]$ (in the same plot).



(c) Plot the reminders $r_n(x) = f(x) - p_n(x)$, $n = 1, 2$ over the interval $[\alpha, \beta]$ (in the same plot).



(d) Bound the absolute value of the reminders $r_n(x) = f(x) - p_n(x)$, $n = 1, 2$. Are the reminders obtained in (c) less than your theoretica bounds for all x in $[\alpha, \beta]$?

Solution: To bound the absolute value of the remainders $r_n(x) = f(x) - p_n(x)$ for $n = 1, 2$, we use Taylor's theorem with the Lagrange form of the remainder. For $p_1(x)$, the bound is:

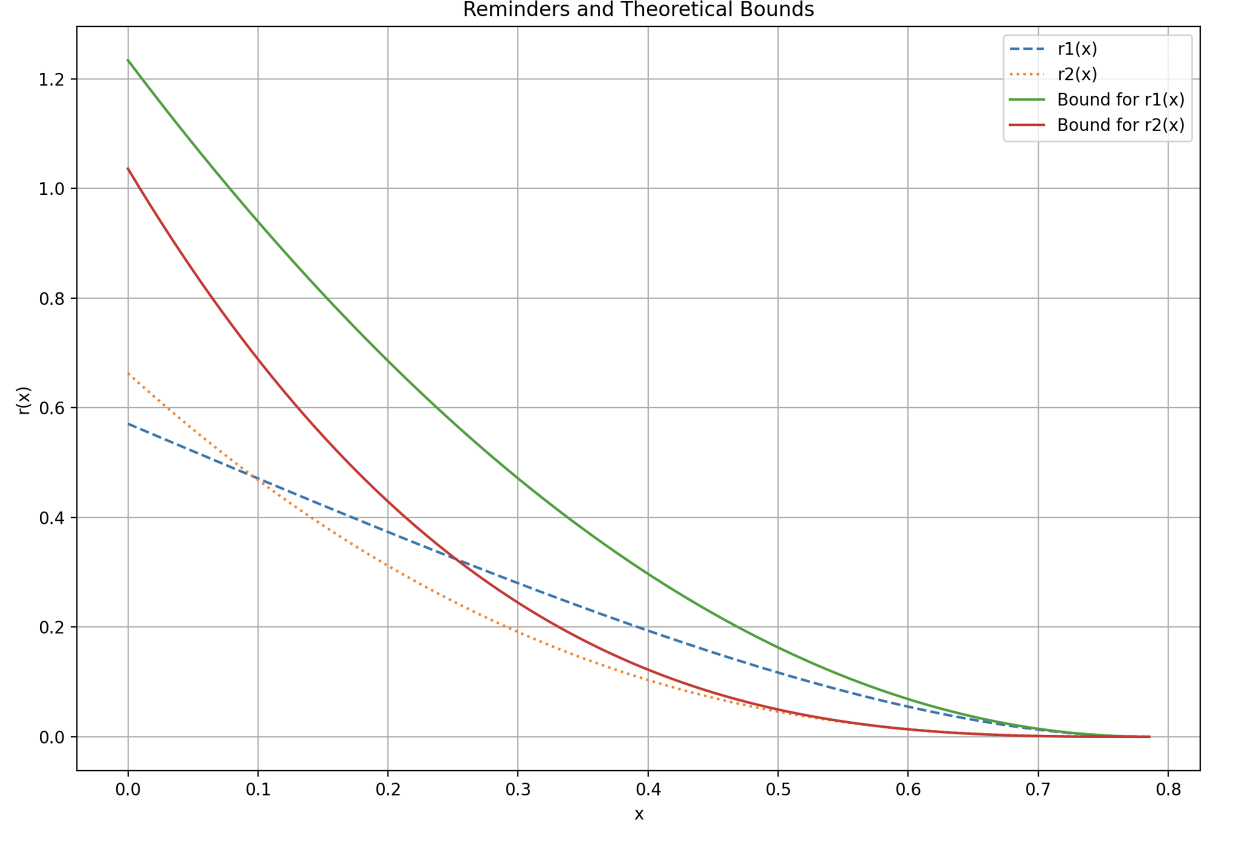
$$|r_1(x)| \leq \frac{|f''(\xi_1)|}{2!}(x-a)^2,$$

where $\xi_1 \in [\alpha, \beta]$ and $|f''(\xi_1)|$ is the maximum value of the second derivative on the interval, which was computed to be 2 (Figure a). For $p_2(x)$, the bound is:

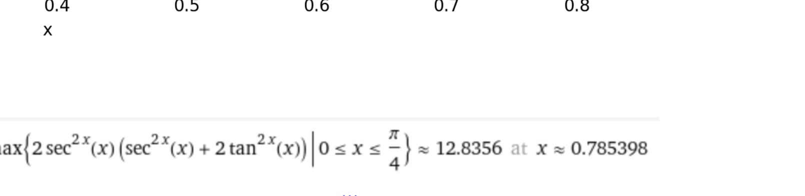
$$|r_2(x)| \leq \frac{|f'''(\xi_2)|}{3!}(x-a)^3,$$

where $\xi_2 \in [\alpha, \beta]$ and $|f'''(\xi_2)|$ is the maximum value of the third derivative on the interval, which was computed to be 12.83 (Figure b). Therefore, the actual bounds for the remainders are:

$$|r_1(x)| \leq \frac{2}{2}(x - \frac{\pi}{4})^2,$$
$$|r_2(x)| \leq \frac{12.83}{6}(x - \frac{\pi}{4})^3.$$

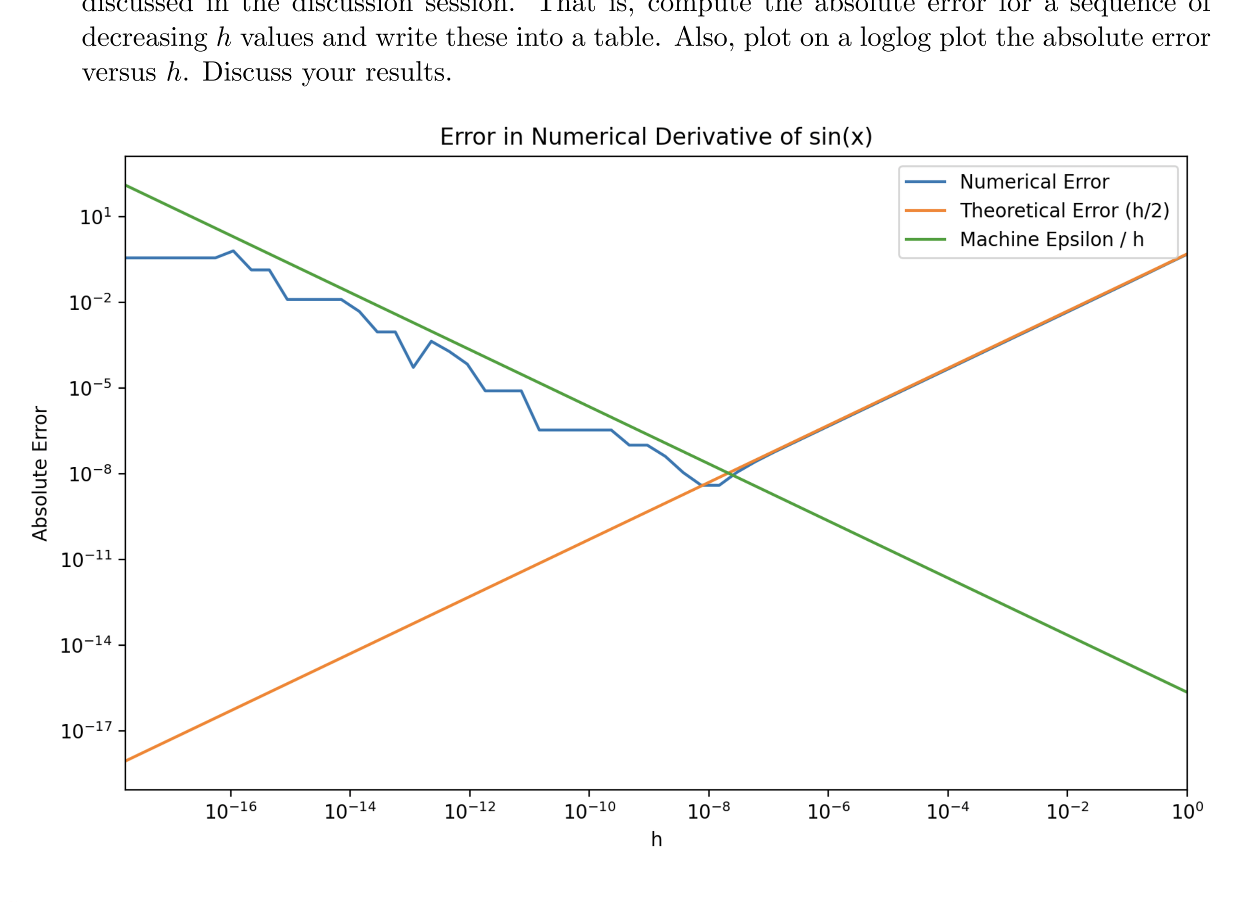


(a) For $|f''(\xi_1)|$



(b) For $|f'''(\xi_2)|$

4. **(5 Points)** Consider the problem of approximating the derivative $f'(x_0)$ of a given smooth function $f(x)$ at the point $x = x_0$. For instance let $f(x) = \sin(x)$ be defined on the real line $-\infty < x < \infty$, and set $x_0 = 1.2$ thus $f(x_0) = \sin(1.2) \approx 0.932\ldots$. Follow the example in the discussion session, where we used Taylor's series for the approximation, and adapt the a matlab code shared in catcourses to reproduce the loglog plot and the error table we discussed in the discussion session. That is compute the absolute error for a sequence of decreasing h values and write these in a table. Also, plot on a loglog plot the absolute error versus h. Discuss your results.



Summary: The log-log plot showcases the relationship between step size h and the absolute error in numerical derivative approximation of $\sin(x)$ at $x = 1.2$. As h decreases, the numerical error diminishes until it reaches a minimum threshold, beyond which the error starts to rise due to the limitations of machine precision. The plot reveals the nature of errors in numerical differentiation, the approximation error, which lessens with finer step size, and the round-off error, which escalates when h is too small. This graph underscores the importance of balancing step size to minimize the total error, exemplifying the compromise between approximation accuracy and computational limitations of floating-point arithmetic. It illustrates the optimum h region where the derivative estimation is most precise before the effects of finite precision become predominant.

```
h          f'(x)               f'(x) app            error     ratio
1.000e+00  0.3623577544766736  -0.1235426821476362  4.859e-01 N/A
5.000e-01  0.3623577544766736  0.1192514497964864   2.431e-01 2.00
2.500e-01  0.3623577544766736  0.2426956207814488   1.197e-01 2.03
1.250e-01  0.3623577544766736  0.3032382186036046   5.912e-02 2.02
6.250e-02  0.3623577544766736  0.3330051490731094   2.935e-02 2.01
3.125e-02  0.3623577544766736  0.3477368542494568   1.462e-02 2.01
1.562e-02  0.3623577544766736  0.3550616030651810   7.296e-03 2.00
7.812e-03  0.3623577544766736  0.3587138972325643   3.644e-03 2.00
3.906e-03  0.3623577544766736  0.3605364464289380   1.821e-03 2.00
1.953e-03  0.3623577544766736  0.3614473299653582   9.104e-04 2.00
9.766e-04  0.3623577544766736  0.3619025997877188   4.552e-04 2.00
4.883e-04  0.3623577544766736  0.3621301914774994   2.276e-04 2.00
2.441e-04  0.3623577544766736  0.3622439766764288   1.138e-04 2.00
1.221e-04  0.3623577544766736  0.3623008664262678   5.689e-05 2.00
6.104e-05  0.3623577544766736  0.3623293108760576   2.844e-05 2.00
3.052e-05  0.3623577544766736  0.3623435326335311   1.422e-05 2.00
1.526e-05  0.3623577544766736  0.3623506435687887   7.111e-06 2.00
7.629e-06  0.3623577544766736  0.3623541990382364   3.555e-06 2.00
3.815e-06  0.3623577544766736  0.3623559767779943   1.778e-06 2.00
1.907e-06  0.3623577544766736  0.3623568656621601   8.888e-07 2.00
9.537e-07  0.3623577544766736  0.3623573101358488   4.444e-07 2.00
4.768e-07  0.3623577544766736  0.3623575323726982   2.221e-07 2.00
2.384e-07  0.3623577544766736  0.3623576434329102   1.110e-07 2.00
1.192e-07  0.3623577544766736  0.3623576993122697   5.516e-08 2.01
5.960e-08  0.3623577544766736  0.3623577281832695   2.629e-08 2.10
2.980e-08  0.3623577544766736  0.3623577430864387   1.139e-08 2.31
1.490e-08  0.3623577544766736  0.3623577505350113   3.942e-09 2.89
7.451e-09  0.3623577544766736  0.3623577505350113   3.942e-09 1.00
3.725e-09  0.3623577544766736  0.3623577654361725   1.096e-08 0.36
1.863e-09  0.3623577544766736  0.3623577952384949   4.076e-08 0.27
9.313e-10  0.3623577544766736  0.3623578548431396   1.004e-07 0.41
4.657e-10  0.3623577544766736  0.3623575568504333   1.976e-07 0.51
2.328e-10  0.3623577544766736  0.3623580932617188   3.388e-07 0.58
1.164e-10  0.3623577544766736  0.3623580932617188   3.388e-07 1.00
5.821e-11  0.3623577544766736  0.3623580932617188   3.388e-07 1.00
2.910e-11  0.3623577544766736  0.3623580932617188   3.388e-07 1.00
1.455e-11  0.3623577544766736  0.3623580932617188   3.388e-07 1.00
7.276e-12  0.3623577544766736  0.3623657226562500   7.968e-06 0.04
3.638e-12  0.3623577544766736  0.3623657226562500   7.968e-06 1.00
1.819e-12  0.3623577544766736  0.3623352050781250   2.255e-05 0.35
9.095e-13  0.3623577544766736  0.3624267578125000   6.900e-05 0.33
4.547e-13  0.3623577544766736  0.3625488281250000   1.911e-04 0.36
2.274e-13  0.3623577544766736  0.3627929687500000   4.352e-04 0.44
1.137e-13  0.3623577544766736  0.3623046875000000   5.307e-05 8.20
5.684e-14  0.3623577544766736  0.3632812500000000   9.235e-04 0.06
2.842e-14  0.3623577544766736  0.3632812500000000   9.235e-04 1.00
1.421e-14  0.3623577544766736  0.3671875000000000   4.830e-03 0.19
7.105e-15  0.3623577544766736  0.3750000000000000   1.264e-02 0.38
3.553e-15  0.3623577544766736  0.3750000000000000   1.264e-02 1.00
1.776e-15  0.3623577544766736  0.3750000000000000   1.264e-02 1.00
8.882e-16  0.3623577544766736  0.5000000000000000   1.376e-01 0.09
4.441e-16  0.3623577544766736  0.5000000000000000   1.376e-01 1.00
2.220e-16  0.3623577544766736  0.0000000000000000   3.624e-01 0.38
1.110e-16  0.3623577544766736  1.0000000000000000   6.376e-01 0.57
5.551e-17  0.3623577544766736  0.0000000000000000   3.624e-01 1.76
2.776e-17  0.3623577544766736  0.0000000000000000   3.624e-01 1.00
1.388e-17  0.3623577544766736  0.0000000000000000   3.624e-01 1.00
6.939e-18  0.3623577544766736  0.0000000000000000   3.624e-01 1.00
3.469e-18  0.3623577544766736  0.0000000000000000   3.624e-01 1.00
1.735e-18  0.3623577544766736  0.0000000000000000   3.624e-01 1.00
```

5. **(5 Points)**

(a) Read Section 2.3 in the NewtonsMethod-BurdenFaires file attached. Summarize and report on questions if you have any, otherwise no need to write up a summary, unless you'd like to.

Summary: The Newton's Method section in the Burden and Faires document describes iterative techniques for finding function roots. Newton's Method uses tangents for fast convergence, requiring derivative calculations. The Secant Method omits derivatives, using secant lines instead, benefiting situations where derivatives are complex. It converges slower than Newton's but can be more practical. The Method of False Position also avoids derivatives and keeps the root within bounds, potentially increasing the number of steps needed.

(b) Implement the Newton and Secant methods and test your code by applying it to solve Example 3 on page 74 in the NewtonsMethod-BurdenFaires file attached. If your code works properly, you should be able to reproduce columns three and four of Table 2.6 for both methods.

```
n    Secant          Newton
0    0.5000000000    0.7853981634
1    0.7853981634    0.7395361335
2    0.7363841388    0.7390851781
3    0.7390581392    0.7390851332
4    0.7390851493    0.7390851332
5    0.7390851332
(ml) ronald@ucmerced-10-34-99-174 MATH140 %
```

(c) Given your results above explain what advantages does Newton method have over Secant method and what advantages Secant method has over Newton's method.

Pros and Cons: Newton's method offers rapid convergence with a precise initial guess and a differentiable function, making it highly efficient for well-behaved functions. However, its necessity for derivative calculation adds computational complexity and can be a significant limitation for complex functions. The Secant method, not requiring derivatives, is simpler and more versatile, especially for complicated functions. However, it generally converges slower than Newton's method and might fail with poorly chosen initial guesses.

[1] Resource utilized for: Problem #1.

[2] This value, `eps(1)` = `eps` $\approx 2.22 \times 10^{-16}$ is usually referred to as machine epsilon, and it gives an indication of the rounding error when dealing with double precision floating point numbers.

[3] Note that the MATLAB command `single` switches to single precision.