Middle East Technical University
Department of Mechanical Engineering
ME 310 Numerical Methods
Spring 2023 (Dr. Cuneyt Sert)
Study Set 2

For Homework 2 submit the answers of questions 1 and 9, and one of questions 5, 6, 7, 8. In total submit answers for 3 questions. Their grade percentages are not known at this point. It will be decided later.

Assigned: 30/03/2023 – Due: 12/04/2023, 23:59

Homework Rules and Suggestions:

- This is an individual assignment. Everything in your report should be the result of your own work. You are allowed to discuss its solution with your classmates and teaching staff up to a certain detail on ODTUClass. You are not allowed to use a solution manual. Put the following honor pledge at the top of your homework report and behave accordingly.

  "I understand that this is an individual assignment. I affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

  If you have exchanged ideas with other students outside ODTUClass, you need to put their names and the extent of your discussion at the beginning of your report.

- Homework submission will be allowed until 5 minutes past the due time. Late submission is not allowed unless you have a valid excuse. In such a case, it is better if you let us know about this before the submission deadline.
- Upload your report as a PDF document (not a Word document) together with all other files (such as codes) to ODTUClass. Name your MATLAB files properly. Follow MATLAB file naming rules such as "File names cannot start with a number", "They cannot contain special characters or spaces", etc.
- Also put your MATLAB codes in your report, but in doing that make sure that you format them properly to avoid line wrapping. Codes with wrapped long lines become unreadable and we cannot understand them. If the codes are very long, you can shorten them by omitting noncritical parts.
- In writing your codes, follow good programming practices such as "use explanatory header lines", "explain inputs and outputs of functions", "use self-explanatory variable names", "use comments", "use empty lines and spaces for readability", "use indentation for code blocks", etc.
- Pay attention to the format of your report. Your report should look like a serious academic work, not like a high school student work. Font types and sizes, page margins, empty spaces on pages, equations, figures, tables, captions, colors, etc. are all important to give the desired "academic work feeling". Language used is also important. Reports with poor use of English cannot get a good grade.
- Do not provide an unnecessarily long report, with useless details or wasted spaces in pages. The shorter your report, the better it is, as long as it answers the questions properly. There are about 100 students, and we can spend only about 10 minutes to grade each report. For this, your report should be easy to read and understand. Also we should be able to find the results and judge their correctness easily. We should not get lost in your report. The more we struggle to understand your report, the lower your grade will be. Use figures and tables cleverly for this purpose.
- Reports with only figures, tables and codes, but no comments or discussions will not get a good grade. Even when a question does not specifically ask for a discussion, you better write some comments on its key points and your key learnings.
- Figures and tables should be numbered and should have captions (at the bottom for figures and at the top for tables). Their titles should be self-explanatory, i.e., we should be able understand everything about the table or figure just by reading its title. They should all be referred properly in the written text (such as "… as shown in Fig. 3" or "… (See Table 2)").
- Do not use Appendices in your report. Do not put your codes in Appendices.
- You can have a numbered reference list at the end of your report. In that case, you need to refer to the references in the text.
- If you are inexperienced in programming, converting an algorithm into a code and writing it in a bug-free way can be time consuming and frustrating. This is not something that can be done at the last minute. You are advised to start working on the assignments as soon as they are assigned.

Reading Assignments:

Self-learning is an important skill. Not everything can be discussed in lectures. You need to learn certain things by yourself.

For those of you who haven't got a chance to buy the textbook yet, I extracted the pages of the following reading assignments and uploaded them to ODTUClass.

R1) Read section 5.2.1 Termination Criteria and Error Estimates (page 130 of 8th edition) to learn the equation (Eqn. (5.5)) that can be used to calculate the number of iterations necessary to reach a certain error level with the bisection method.

R2) Read section 5.3.2 Modified False Position (page 141 of 8th edition) to learn about the "trick" that can be used to speed up the false position method when the interval gets smaller only from one side due to the high non-linearity of the problem.

R3) Read Box 6.1 Convergence of Fixed-Point Iteration (page 151 of 8th edition) to learn the derivation of the convergence criteria and the convergence rate of the fixed-point iteration method.

R4) Read Box 6.2 Derivation and Error Analysis of the Newton-Raphson Method (page 154 of 8th edition) to learn how Newton-Raphson method and its convergence rate can be derived using Taylor series.

R5) Read section 6.6 Systems of Nonlinear Equations (page 170 of 8th edition) to learn how the fixed point iteration and Newton-Raphson methods can be used to solve more than one nonlinear equations together.

R6) Read section 8.4 Pipe Friction (page 214 of 8th edition) which is a Mechanical Engineering case study about friction factor calculation for flows inside pipes. It is related to Q12 of this study set.

R7) Read the Epilogue section of Part 2 (page 231 of 8th edition). Part 2 includes 4 chapters and its epilogue is at the end of Chapter 8. What we call "Chapter 2 Root Finding" in our lectures is this 2nd part of the textbook. We completely skipped the 7th chapter of the textbook, which is about finding the roots of polynomials. You can also skip it. You are NOT responsible for it.

Questions:

Q1. Develop a code that works in a hybrid way similar to MATLAB's own `fzero` function. You can read Handout 2 to learn more about `fzero`. The function you are going to write should be called as follows

```
findRoot(f, x0, tol, maxIter)
```

where `x0` can be a single starting value or an array of two values that bracket the root.

Locating the root will be done in two phases.

Phase 1 (searching for a bracketing interval): If a single value is provided as `x0`, the function will first search for an interval that will bracket the root. For this, it will check the sign of the function at a distance of $x_0/50$ towards left of x_0, and $x_0/50$ towards right of x_0 (use a distance of $1/50$ if x_0 is given as zero). If a sign change is detected, this is the interval that has the root, and the next phase of the calculation will start. If not, the search width will be doubled to 2 times of $x_0/50$ towards left and 2 times of $x_0/50$ towards right of x_0. And the search continues like this by doubling the search width until an interval with a sign change is found. This search continues at most 1000 steps (you can increase this value if necessary) and an error message will be given and the code will abort if no proper interval is found. To give an example, if $x_0 = 5$ is given, first the sign change will be checked for the interval $[4.9, 5.1]$. If this fails, the interval $[4.8, 5.2]$ will be checked next, and so on. If $x_0 = 0$ is given, the first check will be for the interval $[-0.02, 0.02]$. If necessary the second check will be for $[-0.04, 0.04]$.

If an array of two numbers is provided as `x0`, the search phase will be skipped. Instead sign change of the function will be checked at the ends of the provided interval. An error message will be given and the function will abort if no sign change is detected. Otherwise, next phase of the calculation will start.

Phase 2 (finding the root): This phase starts with an interval $[x_L, x_U]$ (either provided by the user or detected in the search phase) that has a root. We have two choices of methods to use, modified secant method (MSM) and bisection method (BM). We want to use MSM whenever possible due to its faster convergence property. BM will be used only when MSM predicts a root out of the interval $[x_L, x_U]$ in one of its iterations. Start with MSM with the initial guess being the mid-point of the interval $[x_L, x_U]$, using a perturbation of $\delta = (x_U - x_L)/100$. Perform iterations as long as the calculated roots are inside the $[x_L, x_U]$ range. If a root is found outside the range, stop the MSM iterations and perform one iteration with the BM method. The starting

interval of this BM iteration is either $[x_L, x_r]$ or $[x_r, x_U]$, where x_r is the last successful result of MSM. You should decide on the correct interval by doing a sign change check and update x_L or x_U accordingly. After only one BM iteration, the code will switch back to MSM to perform calculations in the new interval detected by the BM. Use the mid-point of the updated interval to start MSM iterations. Do NOT calculate a new perturbation value. Instead, use the original δ calculated at the beginning of phase 2. Switch to BM again when necessary, and continue like this. Stop the code if the absolute value of f calculated at the newest root estimate is less than the given tolerance. Do not use absolute or relative errors for this.

You can test your code with the "falling mass with air drag", "vibrating cantilever beam" and "Mars rover wheel design" problems (see the lecture videos and Bracket.m code for details). After these tests, select 2 of the applied problems given in Q10 – Q13 and work on them. It is important to try a number different initial guesses for each problem to see when the code works and when and why it fails. Provide code outputs and explain the behavior of the code. Use plots of the functions to understand and explain how the iterations progress.

Here are some of my sample runs and their outputs. Your code should print out the same information in a similar format, which is somewhat similar to that used by MATLAB's own `fzero` function (see Handout 2).

Run 1:

```
>> f = @(x) sqrt(9.81*x/0.25) * tanh(sqrt(9.81*0.25/x)*4) - 35;
```
← Falling object with air drag problem

```
>> findRoot(f, [80 400], 1e-10, 1000)
```
← Initial interval is [80, 400]

```
Iter       xL           xU           root          f(root)        Method
        80.000000   400.000000   -47.913283   -9.02027e+01   Modified secant (failed)
   1    80.000000   240.000000   160.000000    1.31841e+00   Bisection
   2    80.000000   160.000000   103.230684   -7.67756e-02   Modified secant
   3    80.000000   160.000000   105.401714    6.52869e-04   Modified secant
   4    80.000000   160.000000   105.382581   -1.73711e-05   Modified secant
   5    80.000000   160.000000   105.383090    4.59379e-07   Modified secant
   6    80.000000   160.000000   105.383076   -1.21502e-08   Modified secant
   7    80.000000   160.000000   105.383077    3.21364e-10   Modified secant
   8    80.000000   160.000000   105.383077   -8.49099e-12   Modified secant
```
← Convergence

Run 2:

```
>> findRoot(f, [0 300], 1e-10, 1000)
```
← Same problem as above. Initial interval is [0, 300]

```
Iter       xL           xU           root          f(root)        Method
   1     0.000000   300.000000    87.825997   -7.19541e-01   Modified secant
   2     0.000000   300.000000   103.270382   -7.53344e-02   Modified secant
   3     0.000000   300.000000   105.398571    5.42784e-04   Modified secant
   4     0.000000   300.000000   105.382690   -1.35302e-05   Modified secant
   5     0.000000   300.000000   105.383086    3.35452e-07   Modified secant
   6     0.000000   300.000000   105.383076   -8.31792e-09   Modified secant
   7     0.000000   300.000000   105.383077    2.06256e-10   Modified secant
   8     0.000000   300.000000   105.383077   -5.13012e-12   Modified secant
```
Bisection method is not needed

← Convergence

Run 3:

```
>> findRoot(f, [0 100], 1e-10, 1000)
```
← Same problem as above. Initial interval is [0, 100]

```
Error: The function does not change sign in the provided interval.
```

Run 4:

```
>> findRoot(f, 0, 1e-10, 1000)
```
← Same problem as above. Initial guess is 0 (not an interval)

Trial	xL	xU	f(xL)	f(xU)	Method
1	-0.020000	0.020000	-35.285898	-34.114111	Search
2	-0.060000	0.060000	-35.723648	-33.465595	Search
3	-0.140000	0.140000	-38.936643	-32.656157	Search
4	-0.300000	0.300000	-27.734825	-31.568965	Search
5	**-0.620000**	**0.620000**	**13.409157**	**-30.067578**	Search

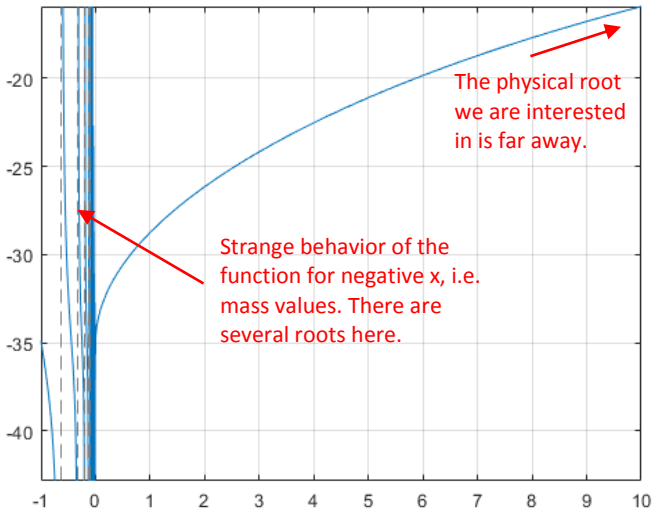
Interval search is done first

← Interval is detected. But not the interval we are interested in.

Iter	xL	xU	root	f(root)	Method
	-0.620000	0.620000	0.622178	-3.00589e+01	Modified secant (failed)
1	-0.620000	0.000000	-0.310000	-2.16330e+01	Bisection
	-0.620000	-0.310000	-1.219533	-3.01571e+01	Modified secant (failed)
2	-0.620000	-0.465000	-0.542500	-2.89412e+01	Bisection
	-0.620000	-0.542500	-0.678680	-5.51971e+01	Modified secant (failed)
3	-0.620000	-0.581250	-0.600625	-1.41567e+01	Bisection
4	-0.620000	-0.600625	-0.616945	5.39358e+00	Modified secant
5	-0.620000	-0.600625	-0.612936	-1.93583e+00	Modified secant
6	-0.620000	-0.600625	-0.614891	1.29438e+00	Modified secant
7	-0.620000	-0.600625	-0.613758	-6.45845e-01	Modified secant
8	-0.620000	-0.600625	-0.614373	3.82447e-01	Modified secant
. . .					
. . .					
. . .					
41	-0.620000	-0.600625	-0.614148	-1.68540e-09	Modified secant
42	-0.620000	-0.600625	-0.614148	9.41242e-10	Modified secant
43	-0.620000	-0.600625	-0.614148	-5.25631e-10	Modified secant
44	-0.620000	-0.600625	-0.614148	2.93433e-10	Modified secant
45	-0.620000	-0.600625	-0.614148	-1.63809e-10	Modified secant
46	-0.620000	-0.600625	**-0.614148**	**9.13687e-11**	Modified secant

← Convergence to an unphysical root

This runs fails because of the physics of the problem and our initial guess. We are trying to find the mass of a falling object. The root should be a positive number. However, when we start with a single initial guess at zero, and search for an interval both towards right and left, we enter the "negative mass territory", which is meaningless. The function behaves very strange in that zone, as seen on the right. There are several negative roots of the function, one of which is detected by our code. But it is not what we are interested in.



Moral: It is good to know the physics of the problem and provide meaningful inputs.

Question: How will MATLAB's own `fzero` function behave in this case? It also finds a negative root. Even this extremely carefully written and well tested function cannot help us, if we do not provide meaningful inputs.

Run 5:

```
>> findRoot(f, 50, 1e-10, 1000)
```
← Same problem as above. Initial guess is 50 (not an interval)

Trial	xL	xU	f(xL)	f(xU)	Method
1	49.000000	51.000000	-3.700494	-3.461924	Search
2	47.000000	53.000000	-3.954339	-3.237289	Search
3	43.000000	57.000000	-4.514142	-2.825203	Search
4	35.000000	65.000000	-5.900223	-2.123870	Search
5	19.000000	81.000000	-10.613602	-1.069914	Search
6	-13.000000	113.000000	99.332617	0.251056	Search
7	**-77.000000**	**177.000000**	**-82.617687**	**1.576204**	Search

Interval search is done first

← Interval is detected. But again, xL is negative.

Iter	xL	xU	root	f(root)	Method
1	-77.000000	177.000000	81.178277	-1.06013e+00	Modified secant
2	-77.000000	177.000000	101.048058	-1.57523e-01	Modified secant
3	-77.000000	177.000000	105.318759	-2.25460e-03	Modified secant
4	-77.000000	177.000000	105.384393	4.61096e-05	Modified secant
5	-77.000000	177.000000	105.383049	-9.68540e-07	Modified secant
6	-77.000000	177.000000	105.383077	2.03334e-08	Modified secant
7	-77.000000	177.000000	105.383077	-4.26880e-10	Modified secant
8	-77.000000	177.000000	**105.383077**	**8.95284e-12**	Modified secant

← Luckily, it converged to the physical positive root

Run 6:

```
>> f = @(x) 0.4 - x + sin(x);
```
← Mars rover wheel design problem

```
>> findRoot(f, [0 2*pi], 1e-10, 1000)
```
← Initial interval is [0, 2π]

Iter	xL	xU	root	f(root)	Method
1	0.000000	6.283185	1.770345	-3.90189e-01	Modified secant
2	0.000000	6.283185	1.452829	-5.97791e-02	Modified secant
3	0.000000	6.283185	1.387394	-4.16550e-03	Modified secant
4	0.000000	6.283185	1.382486	-1.64000e-04	Modified secant
5	0.000000	6.283185	1.382292	-6.03873e-06	Modified secant
6	0.000000	6.283185	1.382284	-2.21740e-07	Modified secant
7	0.000000	6.283185	1.382284	-8.14139e-09	Modified secant
8	0.000000	6.283185	1.382284	-2.98918e-10	Modified secant
9	0.000000	6.283185	**1.382284**	**-1.09749e-11**	Modified secant

← Convergence

Run 7:

```
>> findRoot(f, [-10 5], 1e-10, 1000)
```
← Same problem. Initial interval is [-10, 5] (Not physical)

Iter	xL	xU	root	f(root)	Method
1	-10.000000	5.000000	-1.187348	6.59968e-01	Modified secant
2	-10.000000	5.000000	-0.004307	4.00000e-01	Modified secant
	-10.000000	5.000000	116.525372	-1.16408e+02	Modified secant (failed)
3	-0.004307	5.000000	2.497846	-1.49765e+00	Bisection
4	-0.004307	2.497846	1.381017	1.02915e-03	Modified secant
5	-0.004307	2.497846	1.382179	8.55914e-05	Modified secant
6	-0.004307	2.497846	1.382275	7.16083e-06	Modified secant
7	-0.004307	2.497846	1.382283	5.99391e-07	Modified secant
8	-0.004307	2.497846	1.382284	5.01735e-08	Modified secant
9	-0.004307	2.497846	1.382284	4.19991e-09	Modified secant
10	-0.004307	2.497846	1.382284	3.51565e-10	Modified secant
11	-0.004307	2.497846	1.382284	2.94288e-11	Modified secant

← The correct root is detected, because this function has only one root. This is a simpler problem than the previous one.

Note: For this problem the unknown is an angle in radians. It should be zero and 2π.

Run 8:

```
>> findRoot(f, 0, 1e-10, 1000)
```
← Same problem. Initial guess is zero (not an interval)

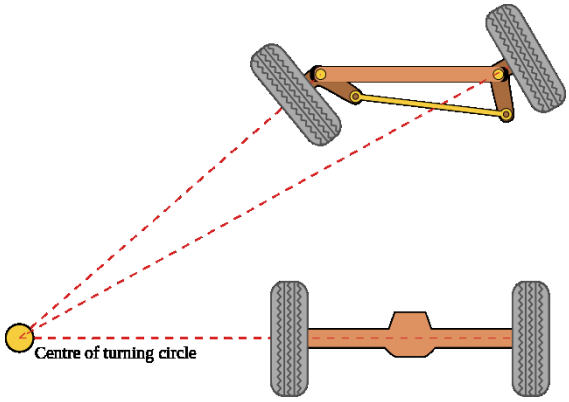
Trial	xL	xU	f(xL)	f(xU)	Method
1	-0.020000	0.020000	0.400001	0.399999	Search
2	-0.060000	0.060000	0.400036	0.399964	Search
3	-0.140000	0.140000	0.400457	0.399543	Search
4	-0.300000	0.300000	0.404480	0.395520	Search
5	-0.620000	0.620000	0.438965	0.361035	Search
6	-1.260000	1.260000	0.707910	0.092090	Search
7	-2.540000	2.540000	2.374044	-1.574044	Search

Interval search is done first

Iter	xL	xU	root	f(root)	Method
	-2.540000	2.540000	930.121868	-9.29513e+02	Modified secant (failed)
1	0.000000	2.540000	1.270000	8.51009e-02	Bisection
2	1.270000	2.540000	1.490515	-9.37357e-02	Modified secant
3	1.270000	2.540000	1.391340	-7.39879e-03	Modified secant
4	1.270000	2.540000	1.382600	-2.56607e-04	Modified secant
5	1.270000	2.540000	1.382294	-7.71010e-06	Modified secant
6	1.270000	2.540000	1.382284	-2.30389e-07	Modified secant
7	1.270000	2.540000	1.382284	-6.88321e-09	Modified secant
8	1.270000	2.540000	1.382284	-2.05645e-10	Modified secant
9	1.270000	2.540000	1.382284	-6.14386e-12	Modified secant

← Convergence

Q2. (This question is not directly related to root finding. It is for those who want to do some fun MATLAB coding practice) The Ackermann steering geometry is an arrangement of linkages used in the steering of a vehicle. It is designed to solve the problem of wheels on the inside and outside of a turn needing to trace out circles of different radii. It avoids the need for tires to slip sideways when following the path around a curve. It rotates the tires by different angles.



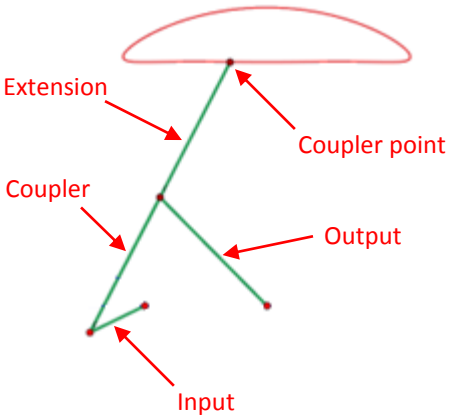
It uses a four-bar mechanism as shown in the following animation.
https://tr.m.wikipedia.org/wiki/Dosya:Ackerman_Steering_Linkage.gif

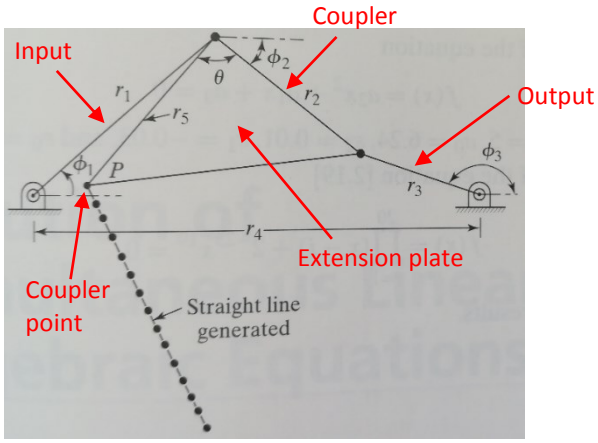
But this four-bar mechanism is in rocker-rocker configuration, i.e. neither the input nor the output link can make a full rotation. Change the FourBar.m code developed in the last MATLAB tutorial session to detect the minimum and maximum angles that these links can turn. Change the code such that the links swing back and forth between these limits. Use appropriate link lengths, such as roughly proportional to those shown in the figure. Also change the function that animates the mechanism such that the tires are also drawn as two rectangles, as well as the dashed lines that are perpendicular to the tires, intersecting at a common point.

Q3. (This question is not directly related to root finding. It is for those who want to do some fun MATLAB coding practice) The four-bar mechanism code we've developed in our last MATLAB tutorial session does not draw coupler curves. But coupler curves are important because sometimes we want special coupler points follow specially designed paths, such as the following straight line mechanisms. Change the code to draw coupler curves.

In the first mechanism shown below, known as the Chebyshev's lambda linkage, the coupler extension is just an extra bar along the coupler link. It is rigidly connected to the coupler link and moves with it. Dimensions are: input = 1, coupler = 2.5, output = 2.5, ground = 2, coupler extension = 2.5. Let the input link make a full rotation.

In the second one, the coupler extension is not just a straight bar, but a triangular plate. Its corner P is the coupler point. As the mechanism moves, the triangular coupler plate moves together with the coupler link. Now, both the angle θ and the distance r_5 are needed to locate the coupler point. Dimensions are: r_1 = 0.963, r_2 = 0.764, r_3 = 0.528, r_4 = 1.815, r_5 = 0.778, θ = 89.65°. Let the input angle vary between 0 and 90°.





Q4. a) Discuss the advantages and disadvantages of the following methods; a) bisection, b) false-position, c) fixed-point iteration, d) Newton-Raphson, e) Secant, f) modified secant, g) quadratic interpolation.

b) Fixed point iteration has a converge rate of 1 (linear convergence). Newton-Raphson has a convergence rate of 2 (quadratic convergence). If an iterative method squares the error every two iterations, what is its convergence rate?

c) Both the false position method and the secant method perform linear interpolation between function values at two previously calculated root estimates. Are they the same? What are their differences?

d) What is wrong in using relative error for convergence check when the root we are looking for is at zero or very close to zero? What should we do in such a case?

e) Why do we use logarithmic vertical axis for convergence plots that show Error vs. iterations?

Q5. For the equation $f(x) = x^2 - 3x + 2 = 0$, each of the following functions yield an equivalent fixed-point problem.

$$g_1(x) = \frac{x^2+2}{3}, \qquad g_2(x) = \sqrt{3x-2}, \qquad g_3(x) = 3 - \frac{2}{x}, \qquad g_4(x) = (x^2-2)/(2x-3)$$

a) Analyze the convergence properties of the corresponding fixed-point iteration schemes for the root $x = 2$ by considering $|g'(2)|$.

b) Confirm your analysis by performing a few iterations and finding the roots numerically. Show the details on paper. Start with an initial guess that is close to the root.

c) Also show the iterations graphically on scaled, accurate plots of the functions (not on rough hand sketches).

Q6. Heron's method (also known as Babylonian method) that we've studied in Chapter 1 finds the square root of number S iteratively using

$$x_{i+1} = \frac{x_i + S/x_i}{2}$$

a) Show that this is nothing but the Newton-Raphson method applied to the function $f(x) = x^2 - S$.

b) For finding square roots of positive numbers with positive starting initial guesses, will this method always converge or is there a risk of divergence? A plot of the function may help to answer this.

c) For $S = 1000$ and $x_0 = 500$, use Heron's method to perform a solution and determine whether the convergence rate is quadratic or not? Calculating $E_{t_{i+1}}/E_{t_i}^2$ ratio after each iteration may help.

Q7. a) Use the false-position method to estimate the root of $f(x) = \ln(x)$. Start the computation with of $x_L = 0.5$ and $x_U = 5.0$. Perform 4 iterations and show the calculation details. Also show the iterations graphically on a scaled, accurate plot of the function (not on a rough hand sketch).

b) Repeat part (a) using the secant method. Start the computation with $x_0 = 0.5$ and $x_1 = 5.0$.

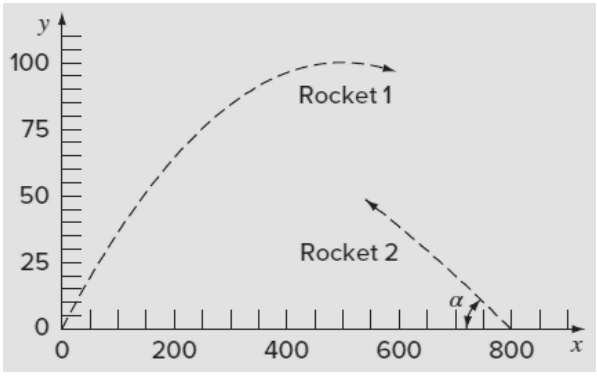
c) Repeat part (b) by starting the computation with $x_0 = 5$ and $x_1 = 0.5$.

d) Comment on the results.

Q8. a) Solve the "falling object with air drag" problem (see the lecture videos and Bracket.m code for details) using secant method starting with $x_0 = 10$ and $x_1 = 20$. Perform hand calculations for 5 iterations and show all the details. Also show the iterations graphically on a scaled, accurate plot of the function (not on a rough hand sketch). Comment on the results.

b) Repeat part (a) for the "vibrating cantilever beam" problem (see the lecture videos and Bracket.m code for details). Use $x_0 = 0.5$ and $x_1 = 1$.

Q9. (Read the reading assignment R5 first) The paths of two rockets in the x-y plane can be described by the following parametric equations



Rocket 1: $x_1 = 100t, \quad y_1 = 80t - 16t^2$

Rocket 2: $x_2 = 800 - 100\cos(\alpha)\,t, \quad y_2 = 80\sin(\alpha)\,t - 0.8t^2$

where t = time and α = launch angle of the second rocket. The two paths intersect but the rockets will only collide if they simultaneously arrive at the same point. Determine the value of t and α so that this occurs.

a) Solve the problem graphically by plotting the two functions that need to be solved together and locating their intersection points. Note that they are not the curves seen on the given figure.

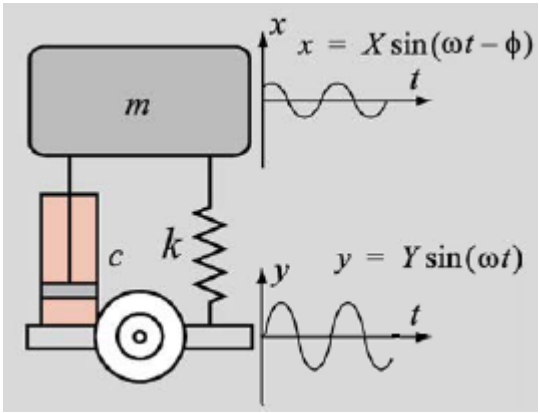
b) Solve the problem by writing a MATLAB code that makes use of either fixed-point iteration or Newton-Raphson. Use your code with at least two different starting guesses and demonstrate both converging and diverging solutions.

Q10. (Thermodynamics) The van der Waals equation of state,

$$\left(p + \frac{a}{v^2}\right)(v - b) = RT$$

relates the pressure p, specific volume v, and temperature T of a gas, where R is a universal constant and a and b are constants that depend on the particular gas. In appropriate units, $R = 0.082054$, and for carbon dioxide, $a = 3.592$ and $b = 0.04267$. Compute the specific volume v for a temperature of 300 K and for pressures of 1 atm, 10 atm, and 100 atm. Compare your results to those for the ideal gas law, $pv = RT$. The latter can be used as a starting guess for an iterative method to solve the van der Waals equation.

Q11. (Vibration) A simplified model of the suspension of a car consists of a mass m, a spring with stiffness k and a damper with damping coefficient c. A bumpy road is modeled by a sinusoidal up-and-down motion of the wheel $y = Y\sin(\omega t)$. From the solution of the equation of motion for this model, the up-and-down motion of the car (mass m) is given by $x = X\sin(\omega t - \phi)$. The ratio between the amplitude X and the amplitude Y is given by



$$\frac{X}{Y} = \sqrt{\frac{\omega^2 c + k^2}{(k - m\omega^2) + (\omega c)^2}}$$

For $m = 2500$ kg, $k = 3 \times 10^5$ N/m and $c = 3.6 \times 10^4$ Ns/m, find the frequency ω for which $X/Y = 0.4$.

Q12. (Fluid mechanics) (You may want to read the reading assignment R6 first). As a fluid flows inside a horizontal pipe, it experiences pressure drop due to its interaction with the pipe wall. The pressure drop in a section of pipe can be calculated as

$$\Delta p = f\frac{L\rho V^2}{2D}$$

where Δp = the pressure drop (Pa), f = the friction factor, L = the length of pipe (m), ρ = fluid density (kg/m^3), V = fluid average speed (m/s), and D = pipe diameter (m). For turbulent flow, the following non-linear Colebrook equation provides a means to calculate the friction factor,

$$\frac{1}{\sqrt{f}} = -2\log\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right)$$

where ε = the pipe wall roughness (m) and Re = the Reynolds number, $Re = \rho VD/\mu$, where μ = dynamic viscosity (N s/m^2).

Determine Δp for a 0.2 m long smooth pipe with ρ = 1.2 kg/m^3, $\mu = 1.8 \times 10^{-5}$ N s/m^2, D = 0.01 m, V = 45 m/s, and ε = 0.045 mm. Use a numerical method to determine the friction factor f. An initial guess can be obtained using the simpler Blasius formula, $f = 0.316 / \mathrm{Re}^{0.25}$, which is valid in a limited Re number range for smooth pipes.

Q13. (Heat transfer) A silicon chip measuring 5 × 5 × 1 mm is embedded in a substrate. At steady state, the chip generates 0.03 W of waste heat. Although the bottom and sides are insulated, the top surface is exposed to air flow and subject to both radiation and convective heat transfer. The radiation heat flux, J_{rad} (W/m^2), can be determined via the Stefan-Boltzmann law, $J_{rad} = \varepsilon\sigma\left(T_{a,\infty}^4 - T_{a,s}^4\right)$, and the convective heat flux, J_{conv} (W/m^2), by $J_{conv} = h\left(T_{a,\infty} - T_{a,s}\right)$, where ε = emissivity, σ = Stefan Boltzmann constant [= 5.67 × 10^{-8} W/(m^2 K^4)], $T_{a,\infty}$ = ambient temperature (K), $T_{a,s}$ = the chip's surface temperature (K), and h = convective heat transfer coefficient (W/(m^2 K)). Use a steady-state heat balance (equate the rate of waste heat removed from chip's top surface (0.03 W / 25 mm^2) to radiation plus convection heat transfer rates) to compute the chip's surface temperature using the following parameter values: ε = 0.9, $T_{a,\infty}$ = 300 K, and h = 60 (W/(m^2 K)).