Numerical Analysis
Coding Examination

Balana, Renmar F. BSCS 4-A
Gerona, Miles
Oropesa, Xania Shane P.
Sabas, Jessa Lorenza M.

I. Bisection Method

1. Create an Octave program that follows the given algorithm for the bisection method. The output must display, for each iteration, the intervals being used, the computed midpoints, the function values at the endpoints of the intervals, and the function values at the obtained midpoints.

   function name: **fn_bisection_method.m**

Bisection

To find a solution to $f(x) = 0$ given the continuous function f on the interval $[a, b]$, where $f(a)$ and $f(b)$ have opposite signs:

INPUT endpoints a, b; tolerance TOL; maximum number of iterations N_0.

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 1$;
$FA = f(a)$.

Step 2 While $i \le N_0$ do Steps 3–6.

Step 3 Set $p = a + (b - a)/2$; (*Compute* p_i.)
$FP = f(p)$.

```
## STEP 1 & 2 & 5
for i = 1:N
  ## STEP 3
  n_str = num2str(i);
  p = (a + (b - a) / 2);
  Fp = f(p);
```

Step 4 If $FP = 0$ or $(b - a)/2 < TOL$ then
OUTPUT (p); (*Procedure completed successfully.*)
STOP.

```
## STOPPING CONDITION
## STEP 4
if (Fp==0) || (abs_error < TOL)
  iteration = i;
  output = p;
  p = p;
  fprintf('\nROOT REACHED: %.8f\n',p);
  plot(p, Fp, 'g*');
  text(p, Fp, [' p_{' n_str '}'],"fontsize",20);
```

Step 5 Set $i = i + 1$.

Step 6 If $FA \cdot FP > 0$ then set $a = p$; (*Compute* a_i, b_i.)
$FA = FP$
else set $b = p$. (*FA is unchanged.*)

```
## STEP 6
##solve for a[i], b[i];
if ((f(a)*Fp)>0) #p replaces a; same sign
  a = p;
  Fa = Fp;
else
  b = p;
  Fb = Fp;
endif
```

Step 7 OUTPUT ('Method failed after N_0 iterations, N_0 =', N_0);
(*The procedure was unsuccessful.*)
STOP.

```
## STEP 7
fprintf('error. Method failed after %d iterations, TOL:%f,\t last p: %f\n',N, TOL, p);
end
```

2. Construct separate codes that use your program for the bisection method to solve the following problems for the given intervals.
    a. Find the largest root of $f(x) = x^6 - x - 1 = 0$ accurate to 10^{-5} for the interval [1,2]. Discuss your observations regarding the convergence of the approximation.

As the bisection method takes place, the values of A and B get closer to each other. So over longer iterations, the convergence results in a consistent decrease in both relative and absolute error.

Code to run: **Ia.m**

Ia.m

```
#{
  BISECTION METHOD

 2.a Find the largest root of f(x) = x^6 - x - 1 = 0  accurate to within 10^-5 for the interval [1, 2].
     Discuss your observations regarding the convergence of the approximations.
#}

f = @(x) (x.^[6]) - x - 1;
fn_bisection_method(1, 2, f, 1e-5, 100);
```

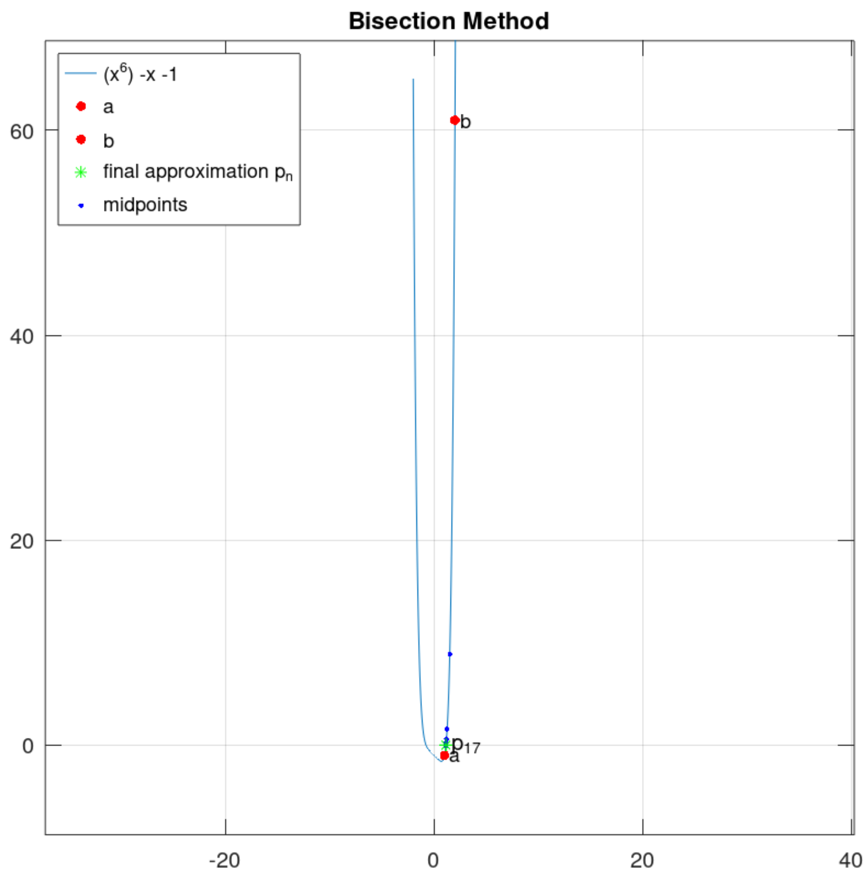
The code to be executed or run, for each question is enclosed in a **box**, like the one above. The function programs, like **fn_bisection_method.m** should NOT be run alone, because these will not work as they need arguments to be passed first, like the TOLerance, initial approximations, and max iteration..

Our written programs for each of the questions follows this general structure. A function handle represents the equation, usually denoted by variable f or g, then it is passed on to a function program in a naming convention "fn_function_name", alongside the other arguments such as the tolerance, initial approximation, and max iterations. Detailed explanation of the function inputs or parameters is briefly included in the actual function code.

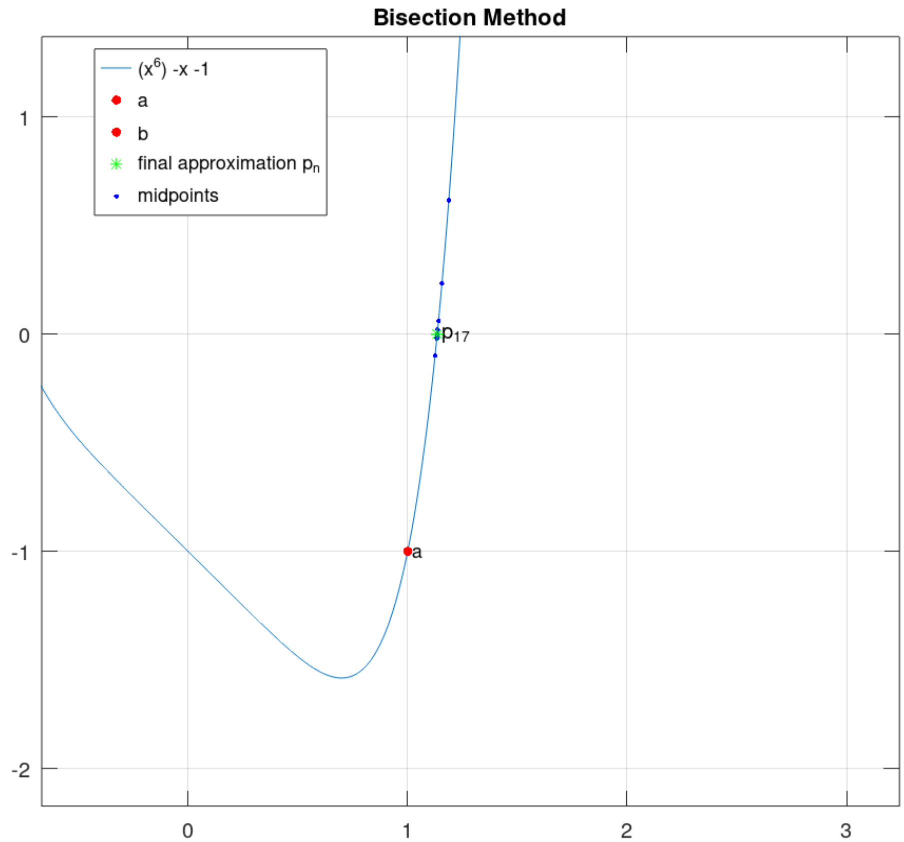
```
>> Ia
```

n	a	f(a)	b	f(b)	p - midpoint	f(p)	a_error	r_error
1	1.000000	-1.000000	2.000000	61.000000	1.500000	8.890625	0.500000	0.333333
2	1.000000	-1.000000	1.500000	8.890625	1.250000	1.564697	0.250000	0.200000
3	1.000000	-1.000000	1.250000	1.564697	1.125000	-0.097713	0.125000	0.111111
4	1.125000	-0.097713	1.250000	1.564697	1.187500	0.616653	0.062500	0.052632
5	1.125000	-0.097713	1.187500	0.616653	1.156250	0.233269	0.031250	0.027027
6	1.125000	-0.097713	1.156250	0.233269	1.140625	0.061578	0.015625	0.013699
7	1.125000	-0.097713	1.140625	0.061578	1.132812	-0.019576	0.007812	0.006897
8	1.132812	-0.019576	1.140625	0.061578	1.136719	0.020619	0.003906	0.003436
9	1.132812	-0.019576	1.136719	0.020619	1.134766	0.000427	0.001953	0.001721
10	1.132812	-0.019576	1.134766	0.000427	1.133789	-0.009598	0.000977	0.000861
11	1.133789	-0.009598	1.134766	0.000427	1.134277	-0.004591	0.000488	0.000430
12	1.134277	-0.004591	1.134766	0.000427	1.134521	-0.002084	0.000244	0.000215
13	1.134521	-0.002084	1.134766	0.000427	1.134644	-0.000829	0.000122	0.000108
14	1.134644	-0.000829	1.134766	0.000427	1.134705	-0.000201	0.000061	0.000054
15	1.134705	-0.000201	1.134766	0.000427	1.134735	0.000113	0.000031	0.000027
16	1.134705	-0.000201	1.134735	0.000113	1.134720	-0.000044	0.000015	0.000013
17	1.134720	-0.000044	1.134735	0.000113	1.134727	0.000034	0.000008	0.000007

```
ROOT REACHED: 1.13472748
```



Bisection Method
(x^6) -x -1
a
b
final approximation p_n
midpoints
b
p17
a
60
40
20
0
-20
0
20
40



Bisection Method
(x^6) -x -1
a
b
final approximation p_n
midpoints
p17
a
1
0
-1
-2
0
1
2
3

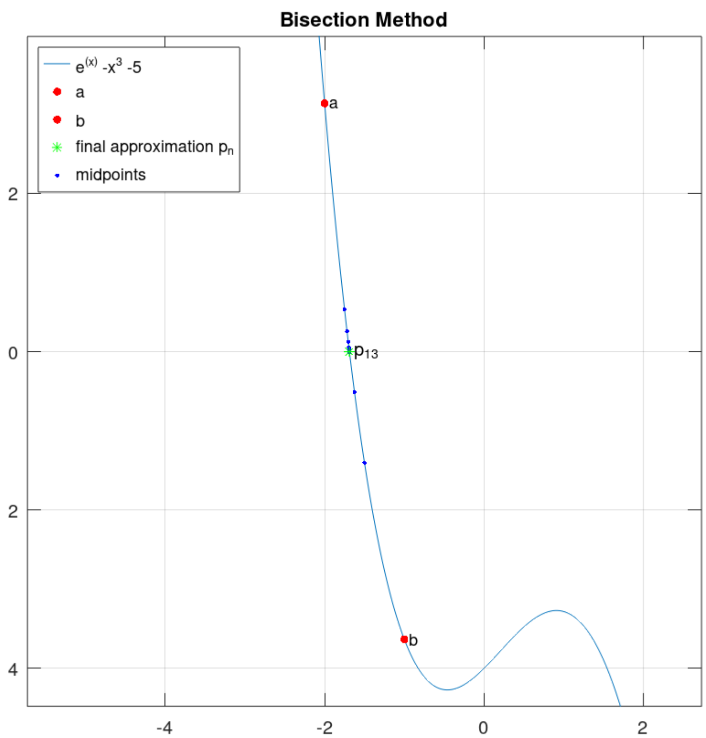
b. Solve the equation $f(x) = e^{x} - x^{3} - 5$ with the interval [-2, -1] accurate to at least within 10^{-4}.

code to run: **Ib.m**

```
>> Ib
```

n	a	f(a)	b	f(b)	p - midpoint	f(p)	a_error	r_error
1	-2.000000	3.135335	-1.000000	-3.632121	-1.500000	-1.401870	0.500000	0.333333
2	-2.000000	3.135335	-1.500000	-1.401870	-1.750000	0.533149	0.250000	0.142857
3	-1.750000	0.533149	-1.500000	-1.401870	-1.625000	-0.512073	0.125000	0.076923
4	-1.750000	0.533149	-1.625000	-0.512073	-1.687500	-0.009599	0.062500	0.037037
5	-1.750000	0.533149	-1.687500	-0.009599	-1.718750	0.256652	0.031250	0.018182
6	-1.718750	0.256652	-1.687500	-0.009599	-1.703125	0.122257	0.015625	0.009174
7	-1.703125	0.122257	-1.687500	-0.009599	-1.695312	0.056013	0.007812	0.004608
8	-1.695312	0.056013	-1.687500	-0.009599	-1.691406	0.023128	0.003906	0.002309
9	-1.691406	0.023128	-1.687500	-0.009599	-1.689453	0.006745	0.001953	0.001156
10	-1.689453	0.006745	-1.687500	-0.009599	-1.688477	-0.001432	0.000977	0.000578
11	-1.689453	0.006745	-1.688477	-0.001432	-1.688965	0.002656	0.000488	0.000289
12	-1.688965	0.002656	-1.688477	-0.001432	-1.688721	0.000612	0.000244	0.000145
13	-1.688721	0.000612	-1.688477	-0.001432	-1.688599	-0.000410	0.000122	0.000072

ROOT REACHED: -1.68859863



c. What will happen if the bisection method is used with the formula $f(x) = \frac{1}{x-2}$ and the interval is [3, 7]? You may try different values of tolerance if possible.

code to run: **Ic.m**

```
>> Ic
bisection method of 1/(x -2):

n       a               f(a)            b               f(b)
1       3.000000        1.000000        7.000000        0.200000
error: f(a) and f(b) are not opposite signs!
Would you like to continue bisection method despite f(a) and f(b) not being opposite signs?
        [Enter 'yes' or 'no']
        >> yes
```

```
## ERROR CATCHING. f(a) and f(b) must be opposite signs!
if (Fa*Fb)>0
  fprintf('n\ta\t\tf(a)\t\tb\t\tf(b) \n');
  fprintf('1\t%.6f\t%.6f\t%.6f\t%.6f \n',a,f(a),b,f(b));
  fprintf("error: f(a) and f(b) are not opposite signs!\n");

  prompt = sprintf('Would you like to continue bisection method despite f(a) and f(b) not being opposite signs?\n\t[Enter ''yes'' or ''no'']\n\t>> ');
  user_input = input(prompt, 's');

  # Check if the input is "yes" or "no" (case-insensitive)
  while ~strcmpi(user_input, 'yes') && ~strcmpi(user_input, 'no')
      fprintf('Invalid input. Please enter "yes" or "no".\n');
      user_input = input(prompt, 's');
  end

  if strcmpi(user_input, 'no')
      error('program terminated.');
  end

endif
```

This part of the code checks if f(a) and f(b) are opposite signs because it is a requirement in the bisection method. The user will be asked if the bisection method should still be continued. We used 10^{-10} tolerance, and max iterations N=20.

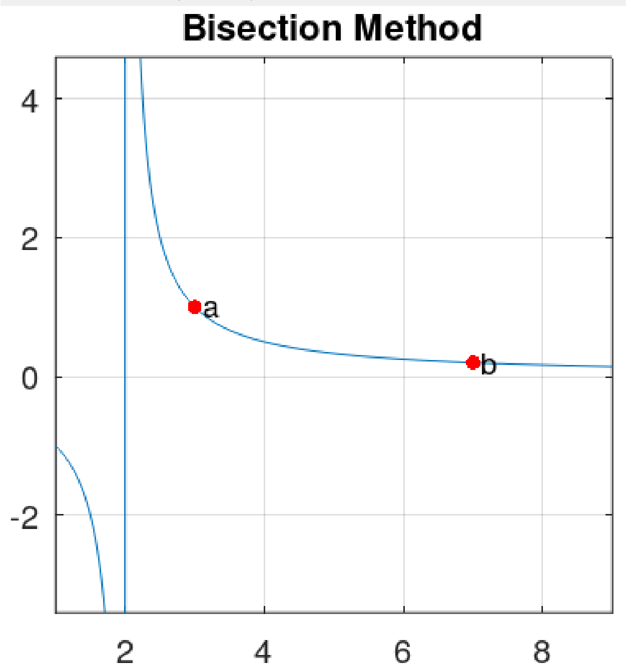
```
>> Ic

bisection method of 1/(x -2):
```

n	a	f(a)	b	f(b)	p - midpoint	f(p)	a_error	r_error
1	3.000000	1.000000	7.000000	0.200000	5.000000	0.333333	2.000000	0.400000
2	5.000000	0.333333	7.000000	0.200000	6.000000	0.250000	1.000000	0.166667
3	6.000000	0.250000	7.000000	0.200000	6.500000	0.222222	0.500000	0.076923
4	6.500000	0.222222	7.000000	0.200000	6.750000	0.210526	0.250000	0.037037
5	6.750000	0.210526	7.000000	0.200000	6.875000	0.205128	0.125000	0.018182
6	6.875000	0.205128	7.000000	0.200000	6.937500	0.202532	0.062500	0.009009
7	6.937500	0.202532	7.000000	0.200000	6.968750	0.201258	0.031250	0.004484
8	6.968750	0.201258	7.000000	0.200000	6.984375	0.200627	0.015625	0.002237
9	6.984375	0.200627	7.000000	0.200000	6.992188	0.200313	0.007812	0.001117
10	6.992188	0.200313	7.000000	0.200000	6.996094	0.200156	0.003906	0.000558
11	6.996094	0.200156	7.000000	0.200000	6.998047	0.200078	0.001953	0.000279
12	6.998047	0.200078	7.000000	0.200000	6.999023	0.200039	0.000977	0.000140
13	6.999023	0.200039	7.000000	0.200000	6.999512	0.200020	0.000488	0.000070
14	6.999512	0.200020	7.000000	0.200000	6.999756	0.200010	0.000244	0.000035
15	6.999756	0.200010	7.000000	0.200000	6.999878	0.200005	0.000122	0.000017
16	6.999878	0.200005	7.000000	0.200000	6.999939	0.200002	0.000061	0.000009
17	6.999939	0.200002	7.000000	0.200000	6.999969	0.200001	0.000031	0.000004
18	6.999969	0.200001	7.000000	0.200000	6.999985	0.200001	0.000015	0.000002
19	6.999985	0.200001	7.000000	0.200000	6.999992	0.200000	0.000008	0.000001
20	6.999992	0.200000	7.000000	0.200000	6.999996	0.200000	0.000004	0.000001

```
error. Method failed after 20 iterations, last value: 6.999996
>> |
```

Here, we used 10^-10 tolerance, and max iterations N=50. Our function for the bisection method thinks it has reached the root, because the absolute error has become smaller than the tolerance. However as you can see, the f(p) or evaluation of the midpoint in the 6th column does not equate to zero, its value is still at 0.2.

```
>> Ic

bisection method of 1/(x -2):

n       a               f(a)            b               f(b)            p - midpoint    f(p)            a_error         r_error
31      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000
32      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000
33      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000
34      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000
35      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000
36      7.000000        0.200000        7.000000        0.200000        7.000000        0.200000        0.000000        0.000000

ROOT REACHED: 7.00000000
```



Because the interval given is from x=2, until x=7, the bisection method can only reach 7 as the midpoint. The function 1/(x-2) as a whole, limits to zero as it approaches (right) positive and (left) negative infinity. This means y values approach zero as x becomes incredibly larger, or smaller, but y will never be exactly zero.

At 10^-500 tolerance, and max iterations N=10,000, it completely fails. The absolute error can no longer catch up with the tolerance.

```
>> Ic

bisection method of 1/(x -2):

n       a               f(a)            b               f(b)            p - midpoint    f(p)

9999    7.000000        0.200000        7.000000        0.200000        7.000000        0.200000
10000   7.000000        0.200000        7.000000        0.200000        7.000000        0.200000
error. Method failed after 10000 iterations, TOL:0.000000,       last midpoint: 7.000000
```

II. Fixed Point Iteration

1. Construct an Octave function for the fixed-point iteration following the given algorithm in class.

   function name: **fn_fixed_point_iteration.m**

Fixed-Point Iteration

To find a solution to $p = g(p)$ given an initial approximation p_0:

INPUT initial approximation p_0; tolerance TOL; maximum number of iterations N_0.

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 1$.

Step 2 While $i \leq N_0$ do Steps 3–6.

Step 3 Set $p = g(p_0)$. (*Compute* p_i.)

```
## Fixed Point Iteration LOOP
## STEP 1, 2, & 5
for i = 1:N
  n_str = num2str(i);
  ## STEP 3
  p = f(p0);
```

Step 4 If $|p - p_0| < TOL$ then
OUTPUT (p); (*The procedure was successful.*)
STOP.

```
  if (abs_error < TOL)
    output = p;
    fprintf('\nFIXED POINT REACHED at x=%f at iteration %d, from p0=%f\n',p0,i,init_aprox);
```

Step 5 Set $i = i + 1$.

Step 6 Set $p_0 = p$. (*Update* p_0.)

```
  ## UPDATE VALUES
  ## STEP 6
  p0 = p;
```

Step 7 OUTPUT ('The method failed after N_0 iterations, N_0 =', N_0);
(*The procedure was unsuccessful.*)
STOP.

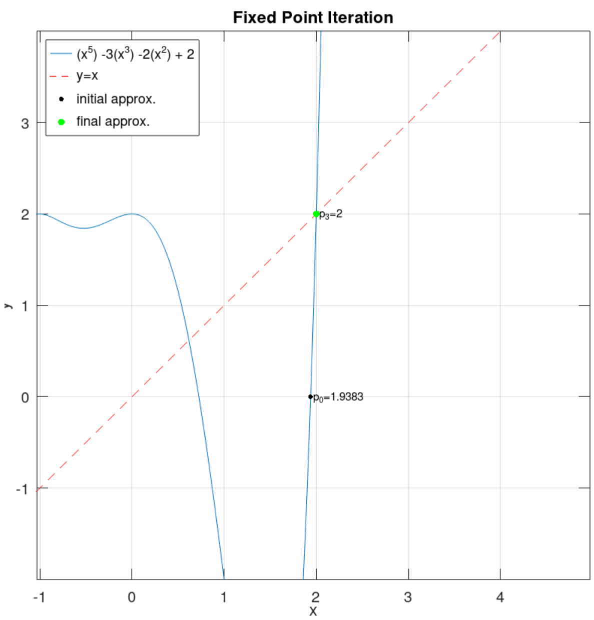
```
  ## STEP 7
  fprintf('error. Method failed after %d iterations, p0=%f. last value: %f\t\n',N,init_aprox, p);

end
```

2. Approximate the fixed points (if any) of each function below. You may try different values of initial approximations. Answers should be accurate to 10 decimal places. Discuss the convergence of the approximations. Produce a graph of each function and the line y = x that clearly shows any fixed points.

   a. $g(x) = x^5 - 3x^3 - 2x^2 + 2$

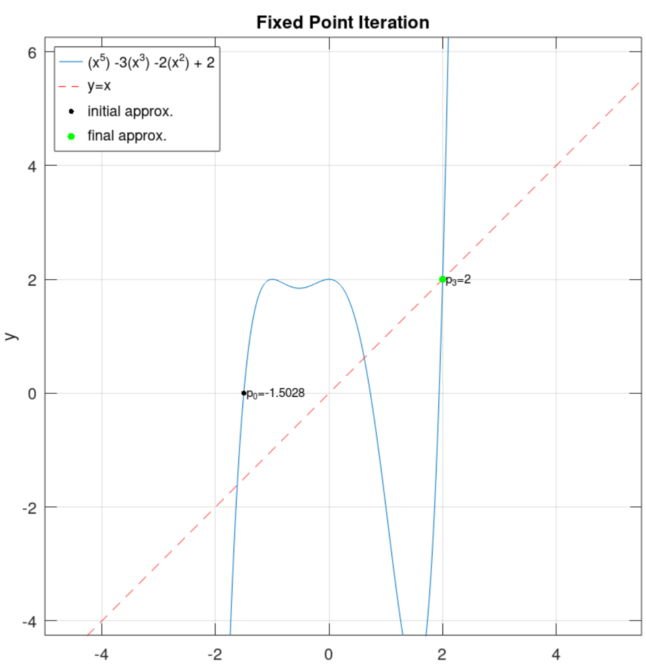
code to run: **IIa.m**

We found that if we start our initial approximation to be a root of this function, it quickly converges to the fixed point.



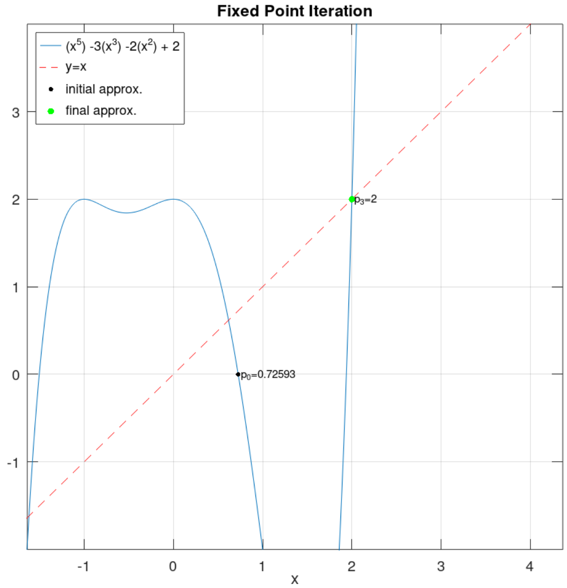
Trying p0 = 1.938346

```
Fixed point iteration of f(x) = (x^{5}) -3(x^{3}) -2(x^{2}) + 2
n        p              f(p)             a_error
1        1.9383459580   0.0000000000     1.93834595799
2        0.0000000000   2.0000000000     2.00000000000
3        2.0000000000   2.0000000000     0.00000000000

FIXED POINT REACHED at x=2.000000 at iteration 3, from p0=1.938346
>> |
```



Trying p0 = -1.502809

```
Fixed point iteration of f(x) = (x^{5}) -3(x^{3}) -2(x^{2}) + 2
n        p              f(p)             a_error
1        -1.5028089000  0.0000005604     1.50280946042
2        0.0000005604   2.0000000000     1.99999943958
3        2.0000000000   2.0000000000     0.00000000002

FIXED POINT REACHED at x=2.000000 at iteration 3, from p0=-1.502809
>> |
```



Trying p0 = 0.725931

```
>> IIa

Fixed point iteration of f(x) = (x^{5}) -3(x^{3}) -2(x^{2}) + 2
n        p               f(p)            a_error
1        0.7259306300    0.0000000248    0.72593060521
2        0.0000000248    2.0000000000    1.99999997521
3        2.0000000000    2.0000000000    0.00000000000

FIXED POINT REACHED at x=2.000000 at iteration 3, from p0=0.725931
>> |
```

Trying p0 = 0.7 did not converge to a fixed point. The consequent approximations p quickly grew to a very large negative number, until it was regarded as negative Infinity (-Inf). Even with p0, very close to 2, p0= 2.000001, still did not converge

```
>> IIa

Fixed point iteration of f(x) = (x^{5}) -3(x^{3}) -2(x^{2}) + 2
n        p               f(p)            a_error
1        0.7000000000    0.1590700000    0.54093000000
2        0.1590700000    1.9374203446    1.77835034455
3        1.9374203446    -0.0268101773   1.96423052183
4        -0.0268101773   1.9986202272    2.02543040452
5        1.9986202272    1.9504423101    0.04817791719
6        1.9504423101    0.3588377172    1.59160459287
7        0.3588377172    1.6098039363    1.25096621912
8        1.6098039363    -4.8872304918   6.49703442807
9        -4.8872304918   -2483.7122147068        2478.82498421501
10       -2483.7122147068        -9451617866459112O.0000000000   9451617866458864O.00000000000
11       -9451617866459112O.0000000000   -754276839885193323509604933618358032641654390197563840515258669029342704151941873664O.0
000000000        754276839885193323509604933618358032641654390197563840515258669029342704151941873664O.00000000000
12       -754276839885193323509604933618358032641654390197563840515258669029342704151941873664O.0000000000        -Inf    Inf
13       -Inf    NaN     NaN
14       NaN     NaN     NaN
15       NaN     NaN     NaN
error. Method failed after 15 iterations, p0=0.700000. last value: NaN
>> |
```

```
Fixed point iteration of f(x) = (x^{5}) -3(x^{3}) -2(x^{2}) + 2
n        p               f(p)            a_error
1        2.0000010000    2.0000360001    0.00003500006
2        2.0000360001    2.0012960799    0.00126007986
3        2.0012960799    2.0467597472    0.04546366726
4        2.0467597472    3.8183702079    1.77161046073
5        3.8183702079    617.5150876074  613.69671739953
6        617.5150876074  89791338382640.2968750000       89791338382022.78125000000
7        89791338382640.2968750000       583676522558649243264531101838051664854862386753089282567205490052300 8.0000000000      5
83676522558649243264531101838051664854862386753089282567205490052300 8.00000000000
8        583676522558649243264531101838051664854862386753089282567205490052300 8.0000000000       Inf     Inf
9        Inf     NaN     NaN
10       NaN     NaN     NaN
11       NaN     NaN     NaN
12       NaN     NaN     NaN
13       NaN     NaN     NaN
14       NaN     NaN     NaN
15       NaN     NaN     NaN
error. Method failed after 15 iterations, p0=2.000001. last value: NaN
>> |
```

fixed_point_attempts [1001x3 cell]

	1	2	
1	p0	fixed point	itera
2	0.7000	NaN	100
3	0.7001	NaN	100
4	0.7002	NaN	100
5	0.7003	NaN	100
6	0.7004	NaN	100
7	0.7005	NaN	100
8	0.7006	NaN	100
9	0.7007	NaN	100
10	0.7008	NaN	100
	•	•	•
252	0.7250	NaN	100
253	0.7251	NaN	100
254	0.7252	NaN	100
255	0.7253	NaN	100
256	0.7254	NaN	100
257	0.7255	NaN	100
258	0.7256	NaN	100
259	0.7257	NaN	100
260	0.7258	NaN	100
261	0.7259	NaN	100
262	0.7260	NaN	100
263	0.7261	NaN	100
264	0.7262	NaN	100
265	0.7263	NaN	100
266	0.7264	NaN	100
267	0.7265	NaN	100
	•	•	•
996	0.7995	NaN	100
997	0.7996	NaN	100
998	0.7997	NaN	100
999	0.7998	NaN	100
1000	0.7999	NaN	100
1001	0.8000	NaN	100

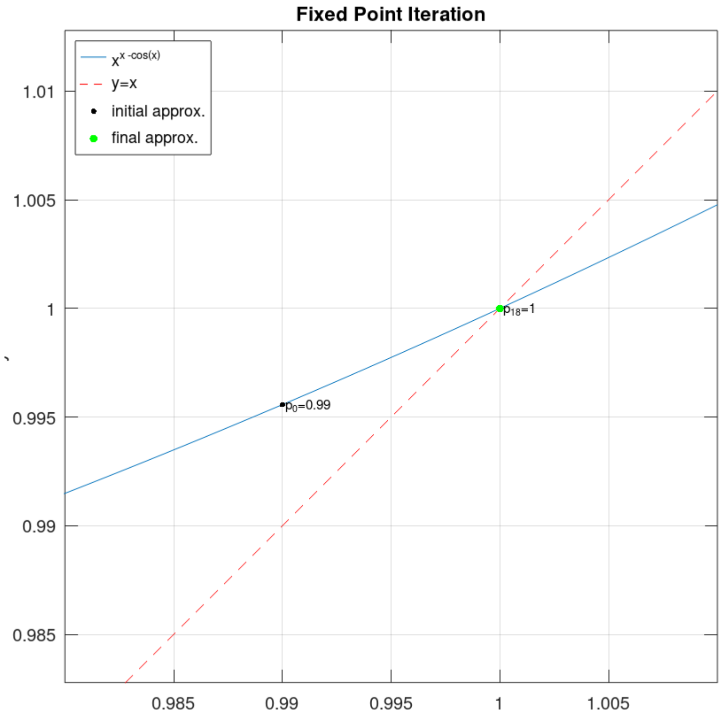
We tried to look for a p0, that is not a root, and not already a fixed point but still relatively close to a fixed point we know. We tested values from x= -0.7 until x=2.1, in an interval of 1,000 but did not find any. (extra code for trying many p0 values: **IIa2.m**)

Another observation we have is that of the few p0 that do converge to a fixed point, it always converges to x=2, even though there are two other fixed points that occur in this equation, (-0.618033 and positive 0.618033).

b. $g(x) = x^{x-cos(x)}$

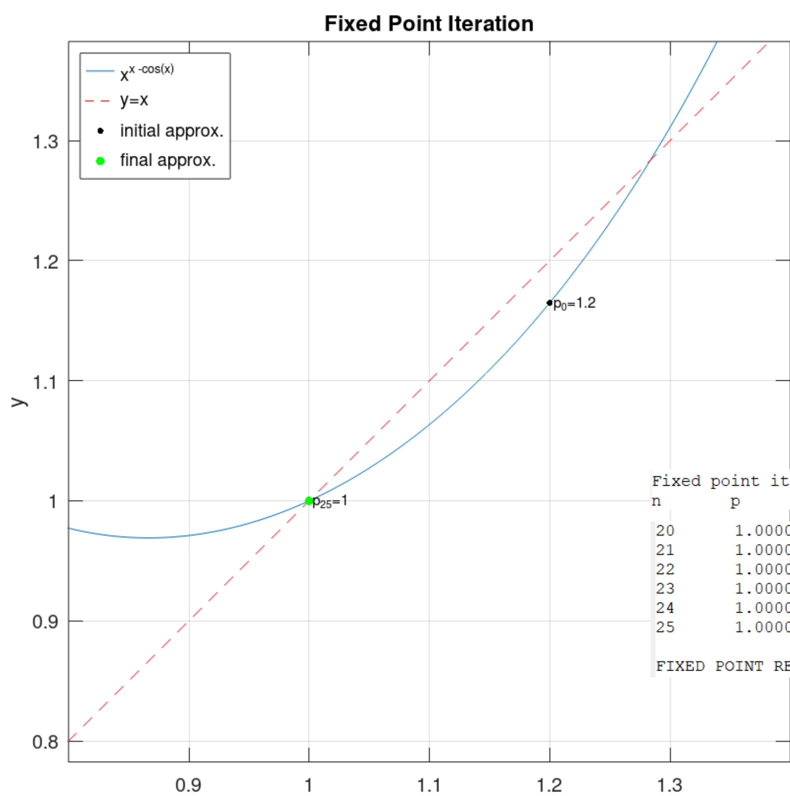
code to run: **IIb.m**

Trying: p0 = 0.99

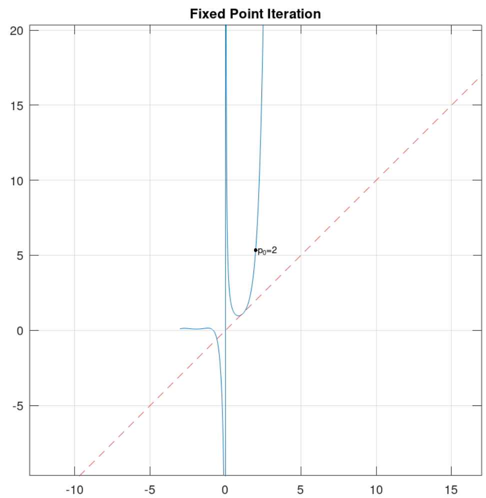



```
Fixed point iteration of f(x) = x^{x -cos(x)}
n       p               f(p)            a_error
1       0.9900000000    0.9955745064    0.00557450637
2       0.9955745064    0.9979992218    0.00242471545
3       0.9979992218    0.9990871193    0.00108789749
4       0.9990871193    0.9995817818    0.00049466245
5       0.9995817818    0.9998080464    0.00022626463
6       0.9998080464    0.9999118226    0.00010377626
7       0.9999118226    0.9999594784    0.00004765578
8       0.9999594784    0.9999813751    0.00002189672
9       0.9999813751    0.9999914388    0.00001006365
10      0.9999914388    0.9999960646    0.00000462577
11      0.9999960646    0.9999981909    0.00000212635
12      0.9999981909    0.9999991684    0.00000097746
13      0.9999991684    0.9999996177    0.00000044933
14      0.9999996177    0.9999998243    0.00000020656
15      0.9999998243    0.9999999192    0.00000009495
16      0.9999999192    0.9999999629    0.00000004365
17      0.9999999629    0.9999999829    0.00000002007
18      0.9999999829    0.9999999922    0.00000000922

FIXED POINT REACHED at x=1.000000 at iteration 18, from p0=0.990000
```

Trying: p0 = 1.2




```
Fixed point iteration of f(x) = x^{x -cos(x)}
n       p               f(p)            a_error
20      1.0000006550    1.0000003011    0.00000035387
21      1.0000003011    1.0000001384    0.00000016267
22      1.0000001384    1.0000000636    0.00000007478
23      1.0000000636    1.0000000292    0.00000003438
24      1.0000000292    1.0000000134    0.00000001580
25      1.0000000134    1.0000000062    0.00000000726

FIXED POINT REACHED at x=1.000000 at iteration 25, from p0=1.200000
```



Trying p0 = 2

```
95      NaN     NaN     NaN
96      NaN     NaN     NaN
97      NaN     NaN     NaN
98      NaN     NaN     NaN
99      NaN     NaN     NaN
100     NaN     NaN     NaN
error. Method failed after 100 iterations, p0=2.000000. last value: NaN
>> |
```

fixed_point_attempts [101x3 cell]

	1	2	3
1	p0	fixed point	iterat
2	0.5000	NaN	100
3	0.5152	1.0000	30
4	0.5303	1.0000	27
5	0.5455	1.0000	26
6	0.5606	1.0000	25
7	0.5758	1.0000	25
8	0.5909	1.0000	24
9	0.6061	1.0000	24
10	0.6212	1.0000	23
11	0.6364	1.0000	23
12	0.6515	1.0000	22
13	0.6667	1.0000	22
14	0.6818	1.0000	22
15	0.6970	1.0000	21
16	0.7121	1.0000	20
17	0.7273	1.0000	19
18	0.7424	1.0000	17
19	0.7576	1.0000	19
20	0.7727	1.0000	20
21	0.7879	1.0000	20
22	0.8030	1.0000	20
23	0.8182	1.0000	21
	•	•	•

23	0.8182	1.0000	21
	•	•	•
29	0.9091	1.0000	21
30	0.9242	1.0000	20
31	0.9394	1.0000	20
32	0.9545	1.0000	20
33	0.9697	1.0000	20
34	0.9848	1.0000	19
35	1	1	1
36	1.0152	1.0000	19
37	1.0303	1.0000	20
38	1.0455	1.0000	21
39	1.0606	1.0000	21
40	1.0758	1.0000	22
41	1.0909	1.0000	22
42	1.1061	1.0000	23
43	1.1212	1.0000	23
	•	•	•
46	1.1667	1.0000	24
47	1.1818	1.0000	25
48	1.1970	1.0000	25
49	1.2121	1.0000	25
50	1.2273	1.0000	26
51	1.2424	1.0000	27
52	1.2576	1.0000	28
53	1.2727	1.0000	30

46	1.1667	1.0000	24
47	1.1818	1.0000	25
48	1.1970	1.0000	25
49	1.2121	1.0000	25
50	1.2273	1.0000	26
51	1.2424	1.0000	27
52	1.2576	1.0000	28
53	1.2727	1.0000	30
54	1.2879	NaN	100
55	1.3030	NaN	100
56	1.3182	NaN	100
57	1.3333	NaN	100
58	1.3485	NaN	100
59	1.3636	NaN	100
60	1.3788	NaN	100
61	1.3939	NaN	100
	•	•	•
95	1.9091	NaN	100
96	1.9242	NaN	100
97	1.9394	NaN	100
98	1.9545	NaN	100
99	1.9697	NaN	100
100	1.9848	NaN	100
101	2	NaN	100

Testing different values of p0, from 0.5 to 2. We found out in our testing that using negative numbers, and numbers greater than 1.28 caused the fixed point iteration to no longer converge. So, we still need a close enough approximation to a fixed point. (extra code for trying many p0 values: **IIb2.m**)

Column 1 stands for the p0 used, and column 2 represents the location of the fixed point found. We used 10^{-8} tolerance and max iterations (N) of 100. When the initial approximation is already a fixed point, only 1 iteration is made. The closer the approximation is, to the fixed point, the fewer iterations are made.

III. Newton's Method

Write the corresponding Newton's formula p_n for the given functions. Formulate codes that show the sequence of approximations for each function. Set the maximum tolerance to 10^{-8}. Try a different number of iterations. Tell something about the convergence of the approximations to the solutions for each given function.

1. Create an Octave program that utilizes the given algorithm for Newton's method to approximate the root of any function. Put an additional column in the output for the function evaluations using approximations.

   function name: **fn_newtons_method.m**

Newton's

To find a solution to $f(x) = 0$ given an initial approximation p_0:

INPUT initial approximation p_0; tolerance TOL; maximum number of iterations N_0.

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 1$.

Step 2 While $i \leq N_0$ do Steps 3–6.

Step 3 Set $p = p_0 - f(p_0)/f'(p_0)$. (*Compute* p_i.)

```
%Perform Newton-Raphson iteration
## STEP 1, 2, 5
for i = 1:N

    ## STEP 3
    p = p0 - f(p0) / df(p0);
```

Step 4 If $|p - p_0| < TOL$ then
OUTPUT (p); (*The procedure was successful.*)
STOP.

```
    abs_error = fn_abs_err(p, p0);

    ## STEP 4
    if (abs_error < TOL)   ## CHANGED to absolute error to accept 0 as initial guess
        fprintf('Converged to solution: p%d = %.9f using Newton-Raphson Method.\n\n',i, p);
```

Step 5 Set $i = i + 1$.

Step 6 Set $p_0 = p$. (*Update* p_0.)

```
    ## STEP 6
    p0 = p; % Update the approximation
```

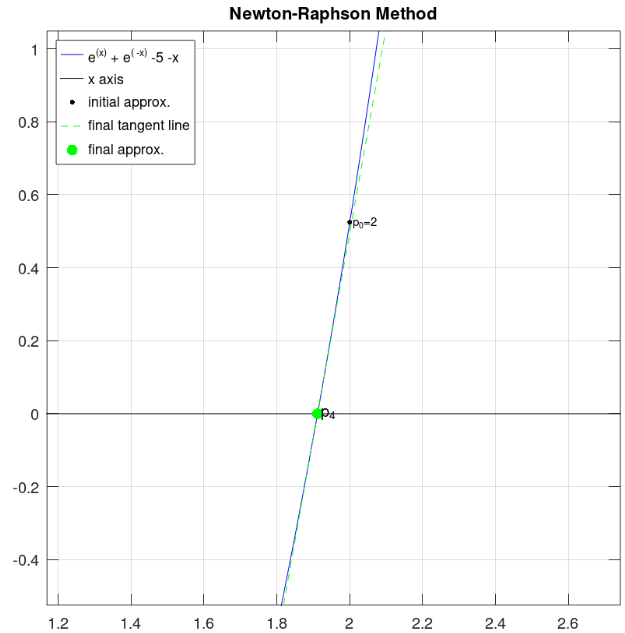
Step 7 OUTPUT ('The method failed after N_0 iterations, $N_0 =$', N_0);
(*The procedure was unsuccessful.*)
STOP.

```
## STEP 7
error('Method failed after %d iterations, last value: %f',N, p);
end
```

2. Consider the following functions and initial approximations.

a. $f(x) = e^{x} + e^{-x} - 5 - x$ with $p_0 = 2$.

code to run: **IIIa.m**



For this equation, the approximations given by Newton-raphson method quickly converge to the root within 10^{-8} tolerance, in only 4 iterations, so we didn't have to try larger iterations.

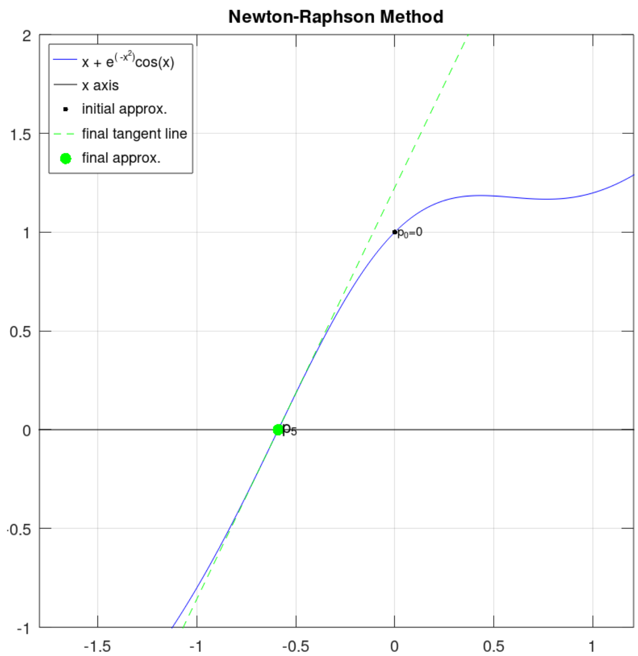
Newton's formula for p_n:

$$p_n = p_{n-1} - \left[\frac{(e^{\{(p_{n-1})\}} + e^{\{(-p_{n-1})\}} - 5 - p_{n-1})}{(e^{(p_{n-1})} - 1 - e^{(-p_{n-1})})} \right]$$

```
>> IIIa

f(x):   e^{(x)} + e^{( -x)} -5 -x
f'(x):  e^(x)-1-e^(-x)

i1: p0 = 2.000000000      f(p0) = 0.524391382      p1 = 1.916147299        f(p1) = 0.025755499
i2: p1 = 1.916147299      f(p1) = 0.025755499      p2 = 1.911586832        f(p2) = 0.000072083
i3: p2 = 1.911586832      f(p2) = 0.000072083      p3 = 1.911573996        f(p3) = 0.000000001
i4: p3 = 1.911573996      f(p3) = 0.000000001      p4 = 1.911573996        f(p4) = 0.000000000
Converged to solution: p4 = 1.911573996 using Newton-Raphson Method.

NEWTON's FORMULA:
pn= pn-1 - [(e^{(pn-1)} + e^{( -pn-1)} -5 -pn-1) / (e^(pn-1)-1-e^(-pn-1))]
>> |
```

b. $f(x) = x + e^{-x^2} cos(x)$ with $p_0 = 0$.

code to run: **IIIb.m**



Same with the first equation, this also quickly converges to the roots, even though the tolerance is 10^{-8}.

Newton's formula for p_n:

$$p_n = p_{n-1} - \left[\frac{(p_{n-1} + e^{\{(-p^2_{n-1})\}} cos\,(p_{n-1}))}{(-2p_{n-1} e^{(-p^2_{n-1})} cos\,(p_{n-1}) + 1 - e^{(-p^2_{n-1})} sin\,(p_{n-1}))} \right]$$

```
>> IIIb

f(x):   x + e^{( -x^{2})}cos(x)
f'(x):  -2xe^(-x^2)cos(x)+1-e^(-x^2)sin(x)

i1: p0 = 0.000000000    f(p0) = 1.000000000     p1 = -1.000000000      f(p1) = -0.801233890
i2: p1 = -1.000000000   f(p1) = -0.801233890    p2 = -0.530644017      f(p2) = 0.120174234
i3: p2 = -0.530644017   f(p2) = 0.120174234     p3 = -0.588626532      f(p3) = -0.000468625
i4: p3 = -0.588626532   f(p3) = -0.000468625    p4 = -0.588401777      f(p4) = -0.000000001
i5: p4 = -0.588401777   f(p4) = -0.000000001    p5 = -0.588401777      f(p5) = -0.000000000
Converged to solution: p5 = -0.588401777 using Newton-Raphson Method.

NEWTON's FORMULA:
pn= pn-1 - [(pn-1 + e^{( -pn-1^{2})}cos(pn-1)) / (-2pn-1e^(-pn-1^2)cos(pn-1)+1-e^(-pn-1^2)sin(pn-1))]
>> |
```

The function $f(x) = e^x + e^{-x} - 5 - x$ with $p_0 = 2$, the Newton-Raphson method achieved convergence to the x-axis in merely 4 iterations, reaching the root $f(x)$=0. Similarly, for the function $f(x) = x + e^{-x^2} cos(x))$ with $p_0 = 0$ convergence occurred in 5 iterations. The higher the value of p_0, the fewer iterations are required for convergence.

IV. Secant Method

1. Construct an Octave code that uses the given algorithm for the secant method to approximate the solution of any function. In the output, also indicate the function values at the obtained approximations.

   octave function name: **fn_secant_method.m**

Secant

To find a solution to $f(x) = 0$ given initial approximations p_0 and p_1:

INPUT initial approximations p_0, p_1; tolerance TOL; maximum number of iterations N_0.

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 2$;
$q_0 = f(p_0)$;
$q_1 = f(p_1)$.

```
## STEP 1,2,5
q0 = f(p0);
q1 = f(p1);
for i = 2:N

  ## STEP 3
  p = p1 - q1*(p1 - p0)/(q1 - q0);  # Compute p's of i
```

Step 2 While $i \le N_0$ do Steps 3–6.

Step 3 Set $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$. (*Compute* p_i.)

Step 4 If $|p - p_1| < TOL$ then
OUTPUT (p); (*The procedure was successful.*)
STOP.

```
  abs_error = fn_abs_err(p, p0);

  ## STEP 4
  if (abs_error < TOL)
      fprintf('\nConverged to solution: p%d = %.8f using Secant Method.\n\n',i, p);
      scatter(p, f(p), 20, 'g', 'filled');
      text(p, f(p), [' p_{' num2str(i) '}'],"fontsize",20);
```

Step 5 Set $i = i + 1$.

Step 6 Set $p_0 = p_1$; (*Update* p_0, q_0, p_1, q_1.)
$q_0 = q_1$;
$p_1 = p$;
$q_1 = f(p)$.

```
    ## STEP 6 - update values
    p0 = p1;
    q0 = q1;
    p1 = p;
    q1 = f(p);
```

Step 7 OUTPUT ('The method failed after N_0 iterations, N_0 =', N_0);
(*The procedure was unsuccessful.*)
STOP.

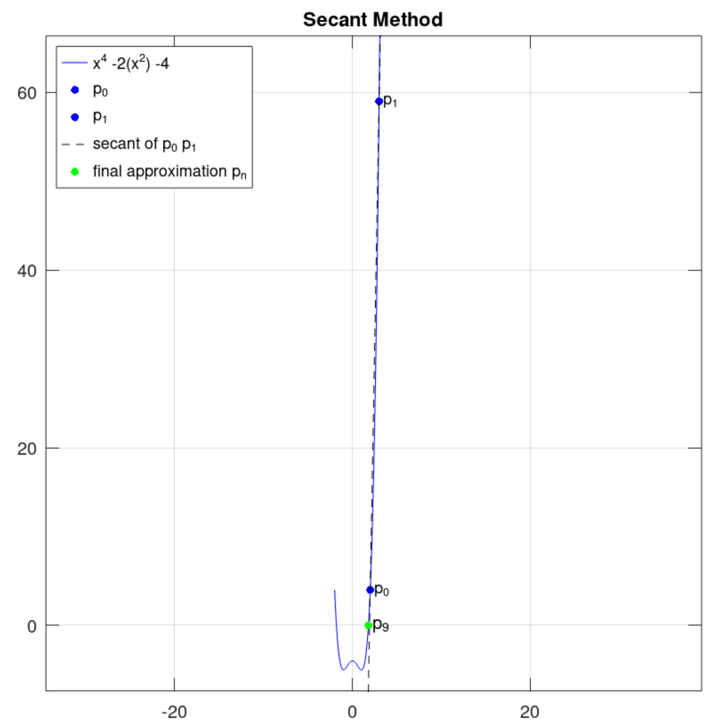
```
  ## STEP 7
  error('Method failed after %d iterations, last value: %f',N, p);
end
```

2. Suppose $f(x) - x^4 - 2x^2 - 4$. Start with $p_o = 2$ and $p_1 = 3$.

   a. Write the secant formula p_n for the given $f(x)$.

   b. With 10^{-6} tolerance value, approximate the solution of $f(x)$ using the indicated initial approximations. Use a separate m file for this. Try a different number of iterations and discuss the convergence of approximations to the solution.

Code to run: **IVa.m**




```
>> IVa

SECANT METHOD for f(x)= x^{4} -2(x^{2}) -4
Initial Approximations: p0=2.00000000    p1=3.00000000

 pn-2               f(pn-2)               pn-1               f(pn-1)               p(n)               f(pn)
 p0 = 2.0000000     f(p0)  = 4.0000000    p1 = 3.0000000     f(p1)  = 59.0000000   p2 = 1.9272727     f(p2)  = 2.36785968
 p1 = 3.0000000     f(p1)  = 59.0000000   p2 = 1.9272727     f(p2)  = 2.3678597    p3 = 1.8824207     f(p3)  = 1.46943092
 p2 = 1.9272727     f(p2)  = 2.3678597    p3 = 1.8824207     f(p3)  = 1.4694309    p4 = 1.8090626     f(p4)  = 0.16519961
 p3 = 1.8824207     f(p3)  = 1.4694309    p4 = 1.8090626     f(p4)  = 0.1651996    p5 = 1.7997708     f(p5)  = 0.01390382
 p4 = 1.8090626     f(p4)  = 0.1651996    p5 = 1.7997708     f(p5)  = 0.0139038    p6 = 1.7989169     f(p6)  = 0.00015158
 p5 = 1.7997708     f(p5)  = 0.0139038    p6 = 1.7989169     f(p6)  = 0.0001516    p7 = 1.7989074     f(p7)  = 0.00000014
 p6 = 1.7989169     f(p6)  = 0.0001516    p7 = 1.7989074     f(p7)  = 0.0000001    p8 = 1.7989074     f(p8)  = 0.00000000
 p7 = 1.7989074     f(p7)  = 0.0000001    p8 = 1.7989074     f(p8)  = 0.0000000    p9 = 1.7989074     f(p9)  = 0.00000000

Converged to solution: p9 = 1.79890744 using Secant Method.
```

V. Systems of Linear Equations

1. Construct a code that utilizes the given algorithm in class for Gaussian elimination with backward substitution to solve the solution of any linear system

   octave function name: **fn_gaussian_elimination.m**

```
## STEP 1
for i = 1:n-1
    En  = A(i, :);  ## row i
    col = A(:, i);  ## column i

    col(1:i-1) = 0;     ## ignore rows above i

    ## STEP 2
    p = find(col, 1);  ## find index of first non-zero value in col i
    if isempty(p)
      fprintf('No unique solution exists.\n');
      error('no unique solutions.');
      unique_solution = false;
      return;
    endif

    ## STEP 3
    if (p != i) #Swap
        temp_row = A(p, :);  ## temp row
        A(p, :) = A(i, :);
        A(i, :) = temp_row;
    endif

    ## STEP 4
    for j = i+1:n

      ## STEP 5
      mji = A(j,i)/A(i,i);

      ## STEP 6
      Ej = A(j, :) - (mji.*A(i, :));
      A(j, :) = Ej;
    endfor

endfor
```

Gaussian Elimination with Backward Substitution

To solve the $n \times n$ linear system

$$\begin{aligned} E_1: &\quad a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = a_{1,n+1} \\ E_2: &\quad a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = a_{2,n+1} \\ &\quad \vdots \qquad \vdots \qquad \vdots \qquad\qquad \vdots \qquad \vdots \\ E_n: &\quad a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = a_{n,n+1} \end{aligned}$$

INPUT number of unknowns and equations n; augmented matrix $A = [a_{ij}]$, where $1 \le i \le n$ and $1 \le j \le n+1$.

OUTPUT solution $x_1, x_2, \ldots, x_n$ or message that the linear system has no unique solution.

Step 1 For $i = 1, \ldots, n-1$ do Steps 2–4. (*Elimination process.*)

Step 2 Let p be the smallest integer with $i \le p \le n$ and $a_{pi} \ne 0$.
If no integer p can be found
then OUTPUT ('no unique solution exists');
STOP.

Step 3 If $p \ne i$ then perform $(E_p) \leftrightarrow (E_i)$.

Step 4 For $j = i+1, \ldots, n$ do Steps 5 and 6.

Step 5 Set $m_{ji} = a_{ji}/a_{ii}$.

Step 6 Perform $(E_j - m_{ji}E_i) \rightarrow (E_j)$;

Step 7 If $a_{nn} = 0$ then OUTPUT ('no unique solution exists');
STOP.

Step 8 Set $x_n = a_{n,n+1}/a_{nn}$. (*Start backward substitution.*)

Step 9 For $i = n-1, \ldots, 1$ set $x_i = \left[a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j\right] \Big/ a_{ii}$.

Step 10 OUTPUT $(x_1, \ldots, x_n)$; (*Procedure completed successfully.*)
STOP. ■

```
## STEP 7
if (A(n,n) == 0)
  fprintf('No unique solution exists.\n');
  error('no unique solutions.');
  unique_solution = false;
  return;
endif

## Backward Substitution
x = zeros(n,1);         #Store the answers. x is a row column containing solutions

## STEP 8
x(n,1) = A(n,n+1)./A(n,n);

## STEP 9
for i = n-1:-1:1
    summation = 0;
    for j = i+1:n
      summation = summation + (A(i,j) .* x(j,1));
    endfor
    ##debug fprintf('summation:%d\n',summation);
    x(i,1) = (A(i,n+1) - summation) ./ A(i,i);
endfor
fprintf("Augmented matrix in reduced echelon form:\n");
disp(A);

## STEP 10
fprintf("\nThe system of Linear equations A has been solved (Gaussian Elimination)! solution:\n");

for(i=1:n)
    fprintf('\tx%d: %f\n',i,x(i,1));
endfor
```

2. Consider the following system of linear equations:

$$5\alpha + \beta + \gamma = 5,$$
$$\alpha + 4\beta + \gamma = 4,$$
$$\alpha + 4\beta + 3\gamma = 3.$$

a. Apply the Gaussian elimination with backward substitution to solve the solution of the above manual computation.

$$5\alpha + \beta + \gamma = 5,$$
$$\alpha + 4\beta + \gamma = 4,$$
$$\alpha + \beta + 3\gamma = 3.$$

$E_1 \leftrightarrow E_2$

$$\alpha + 4\beta + \gamma = 4$$
$$5\alpha + \beta + \gamma = 5$$
$$\alpha + \beta + 3\gamma = 3$$

$E_2 - 5E_1 \rightarrow E_2$

$-5E_1$: $\quad -5\alpha - 20\beta - 5\gamma = -20$

E_2: $\quad 5\alpha + \beta + \gamma = 5$

$$-19\beta - 4\gamma = -15$$

$$\alpha + 4\beta + \gamma = 4$$
$$-19\beta - 4\gamma = -15$$
$$\alpha + \beta + 3\gamma = 3$$

$E_1 - E_3 \rightarrow E_3$

E_1: $\quad \alpha + 4\beta + \gamma = 4$

E_3: $\quad \alpha + \beta + 3\gamma = 3$

$$3\beta - 2\gamma = 1$$

$$\alpha + 4\beta + \gamma = 4$$
$$-19\beta - 4\gamma = -15$$
$$3\beta - 2\gamma = 1$$

$E_3 - \left(-\frac{3}{19}\right) E_2 \rightarrow E_3$

$$\frac{3}{19}E_2: \quad 3\beta + \frac{12}{19}\gamma = \frac{45}{19}$$

$$E_3: \quad 3\beta - 2\gamma = 1$$

$$\frac{50}{19}\gamma = \frac{26}{19}$$

$$\alpha + 4\beta + \gamma = 4$$

$$-19\beta - 4\gamma = -15$$

$$\frac{50}{19}\gamma = \frac{26}{19}$$

$$E_3: \quad \frac{50}{19}x_3 = \frac{\frac{26}{19}}{\frac{50}{19}}$$

$$\boxed{x_3 = 13/25 \text{ or } 0.52}$$

$$E_2: \quad -19x_2 - 4x_3 = -15$$

$$-19x_2 - 4\left(\frac{13}{25}\right) = -15$$

$$-19x_2 = -15 + \frac{52}{25}$$

$$\frac{\cancel{-19}x_2}{\cancel{-19}} = \frac{-\frac{323}{25}}{-19}$$

$$\boxed{x_2 = 17/25 \text{ or } 0.68}$$

$$E_1: \quad x_1 + 4x_2 + x_3 = 4$$

$$x_1 + 4(7/25) + 13/25 = 4$$

$$x_1 = 4 - \frac{81}{25}$$

$$\boxed{x_1 = \frac{19}{25} \text{ or } 0.76}$$

b. Use your code in (1) to solve the solution of the given system and compare this to your answer in (a). Show that the obtained solution satisfies the given system of equations.

code to run: **V2.m**

V2.m

```
#{
  GAUSSIAN ELIMINATION

 Construct a code that utilizes the given algorithm in class for Gaussian elimination with backward
 substitution to solve the solution of any linear system.
#}

  A = [5 1 1;
       1 4 1;
       1 1 3;];
  b = [5;          # column vector of right side equation
       4;
       3;];


  augmented_A = [A, b]  #Augmented Matrix

  x = fn_gaussian_elimination(augmented_A);

```

```
>> V2

augmented_A =

   5   1   1   5
   1   4   1   4
   1   1   3   3

Augmented matrix in reduced echelon form:
   5.0000   1.0000   1.0000   5.0000
        0   3.8000   0.8000   3.0000
        0        0   2.6316   1.3684

The system of Linear equations A has been solved (Gaussian Elimination)! solution:
        x1: 0.760000
        x2: 0.680000
        x3: 0.520000
>> 
```

3. Suppose we have the following:

$$A = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 2 & -1 & 5 & 0 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ -33 \\ 20 \\ -15 \end{bmatrix}$$

LU substitution

$$A = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 2 & -1 & 5 & 0 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ -33 \\ 20 \\ -15 \end{bmatrix}$$

$E_2 - 2E_1 \rightarrow E_2$

$$\begin{array}{r|rrrr} -2E_1: & -2 & -2 & 0 & -8 \\ E_2: & 2 & -1 & 5 & 0 \\ \hline & 0 & -3 & 5 & -8 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ & & 1 & 0 \\ & & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 0 & -3 & 5 & -8 \\ 5 & 2 & 1 & 2 \\ -3 & 0 & 2 & 6 \end{bmatrix}$$

$E_3 - 5E_1 \rightarrow E_3$

$$\begin{array}{r|rrrr} -5E_1: & -5 & -5 & 0 & -20 \\ E_3: & 5 & 2 & 1 & 2 \\ \hline & & -3 & 1 & -18 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & & 1 & 0 \\ & & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 0 & -3 & 5 & -8 \\ 0 & -3 & 1 & -18 \\ 3 & 0 & 2 & 6 \end{bmatrix}$$

$E_4 - (-3E_1) \rightarrow E_4$

$$\begin{array}{r|rrrr} 3E_1: & 3 & 3 & 0 & 12 \\ E_4: & -3 & 0 & 2 & 6 \\ \hline & 0 & 3 & 2 & 18 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & & 1 & 0 \\ -3 & & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 0 & -3 & 5 & -8 \\ 0 & -3 & 1 & -18 \\ 0 & 3 & 2 & 18 \end{bmatrix}$$

$E_3 - E_2 \rightarrow E_3$

$$\begin{array}{r|rrr} -E_2: & 3 & -5 & 8 \\ E_3: & -3 & 1 & 18 \\ \hline & 0 & -4 & -10 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & 1 & 1 & 0 \\ -3 & & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ & -3 & 5 & -8 \\ & & -4 & -10 \\ & 3 & 2 & 18 \end{bmatrix}$$

$E_4 - (-E_2) \rightarrow E_4$

$$\begin{array}{r|rrr} E_2: & -3 & 5 & -8 \\ E_4: & 3 & 2 & 18 \\ \hline & 0 & 7 & 10 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & 1 & 1 & 0 \\ -3 & -1 & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ & -3 & 5 & -8 \\ & & -4 & -10 \\ & & 7 & 10 \end{bmatrix}$$

$E_4 - (-7/4)E_3 \rightarrow E_4$

$$\begin{array}{r|rr} 7/4\,E_3: & -7 & -35/2 \\ E_4: & 7 & 10 \\ \hline & 0 & -15/2 \end{array}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & 1 & 1 & 0 \\ -3 & -1 & -7/4 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 0 & -3 & 5 & -8 \\ 0 & 0 & -4 & -10 \\ 0 & 0 & 0 & -15/2 \end{bmatrix}$$

b. Use the LU factorization in (a) to solve the system Ax = b using the substitution Ux = y and Ly =b.

FORWARD SUBSTITUTION

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 5 & 1 & 1 & 0 \\ -3 & -1 & -7/4 & 1 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ -33 \\ 20 \\ -15 \end{bmatrix}$$

$$\boxed{y_1 = 1}$$

$$2y_1 + y_2 = -33$$
$$2(1) + y_2 = -33$$
$$y_2 = -33 - 2$$
$$\boxed{y_2 = -35}$$

$$5y_1 + y_2 + y_3 = 20$$
$$5(1) + (-35) + y_3 = 20$$
$$y_3 = 20 + 30$$
$$\boxed{y_3 = 50}$$

$$-3y_1 - y_2 - \tfrac{7}{4}y_3 + y_4 = -15$$
$$-3(1) - (-35) - \tfrac{7}{4}(50) + y_4 = -15$$
$$y_4 = -15 + \frac{111}{2}$$
$$\boxed{y_4 = \frac{81}{2} \text{ or } 40.5}$$

Backward Substitution

$$U = \begin{bmatrix} 1 & 1 & 0 & 4 \\ 0 & -3 & 5 & -8 \\ 0 & 0 & -4 & -10 \\ 0 & 0 & 0 & -\frac{15}{2} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ -35 \\ 50 \\ 81/2 \end{bmatrix}$$

$$-\tfrac{15}{4} x_4 = \frac{81/2}{-15/2}$$

$$\boxed{x_4 = -\frac{27}{5} \text{ or } -5.4}$$

$$-4x_3 - 10x_4 = 50$$
$$-4x_3 - 10(-27/5) = 50$$
$$-4x_3 = 50 - 54$$
$$\frac{\cancel{-4}x_3}{\cancel{-4}} = \frac{-4}{-4}$$
$$\boxed{x_3 = 1}$$

$$-3x_2 + 5x_3 - 8x_4 = -35$$
$$-3x_2 + 5(1) - 8(-27/5) = -35$$
$$-3x_2 = -35 - \frac{241}{5}$$
$$\frac{\cancel{-3}x_2}{\cancel{-3}} = \frac{-416/5}{-3}$$
$$\boxed{x_2 = \frac{416}{15} \text{ or } 27.7\overline{33}}$$

$$x_1 + x_2 + 4x_4 = 1$$
$$x_1 + \frac{416}{15} + 4\left(-\frac{27}{5}\right) = 1$$
$$x_1 = 1 - \frac{92}{15}$$
$$\boxed{x_1 = -\frac{77}{15} \text{ or } -5.1\overline{33}}$$

4. *(Bonus)* Write an Octave code that gives the LU factorization of any matrix.

Code to run: **Vbonus.m**

We made 2 programs for LU: 1) LU_decomposition which outputs the L and U matrix; and 2) LU_factorization which uses LU_decomposition and proceeds to forward and backward substitution with y and x respectively, to give the actual solutions.

This is the factorization:

LU Factorization

To factor the $n \times n$ matrix $A = [a_{ij}]$ into the product of the lower-triangular matrix $L = [l_{ij}]$ and the upper-triangular matrix $U = [u_{ij}]$; that is, $A = LU$, where the main diagonal of either L or U consists of all ones:

INPUT dimension n; the entries a_{ij}, $1 \le i, j \le n$ of A; the diagonal $l_{11} = \cdots = l_{nn} = 1$ of L or the diagonal $u_{11} = \cdots = u_{nn} = 1$ of U.

OUTPUT the entries l_{ij}, $1 \le j \le i$, $1 \le i \le n$ of L and the entries, u_{ij}, $i \le j \le n$, $1 \le i \le n$ of U.

Step 1 Select l_{11} and u_{11} satisfying $l_{11}u_{11} = a_{11}$.
If $l_{11}u_{11} = 0$ then OUTPUT ('Factorization impossible');
STOP.

Step 2 For $j = 2, \ldots, n$ set $u_{1j} = a_{1j}/l_{11}$; *(First row of U.)*
$l_{j1} = a_{j1}/u_{11}$. *(First column of L.)*

Step 3 For $i = 2, \ldots, n-1$ do Steps 4 and 5.

Step 4 Select l_{ii} and u_{ii} satisfying $l_{ii}u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}u_{ki}$.
If $l_{ii}u_{ii} = 0$ then OUTPUT ('Factorization impossible');
STOP.

Step 5 For $j = i+1, \ldots, n$

set $u_{ij} = \frac{1}{l_{ii}}\left[a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\right]$; *(ith row of U.)*

$l_{ji} = \frac{1}{u_{ii}}\left[a_{ji} - \sum_{k=1}^{i-1} l_{jk}u_{ki}\right]$. *(ith column of L.)*

Step 6 Select l_{nn} and u_{nn} satisfying $l_{nn}u_{nn} = a_{nn} - \sum_{k=1}^{n-1} l_{nk}u_{kn}$.
(Note: If $l_{nn}u_{nn} = 0$, then $A = LU$ but A is singular.)

Step 7 OUTPUT (l_{ij} for $j = 1, \ldots, i$ and $i = 1, \ldots, n$);
OUTPUT (u_{ij} for $j = i, \ldots, n$ and $i = 1, \ldots, n$);
STOP.

```
## STEP1:
U(1,1) = A(1,1) / L(1,1);
if (L(1,1) * U(1,1)) == 0
    fprintf('error. LU Factorization impossible.\n');
    error('impossible');
    return;
endif

## STEP2:
for (j=2:n)
    U(1,j) = A(1,j)/L(1,1);    # first row of U
    L(j,1) = A(j,1)/U(1,1);    # first col of L
endfor

## STEP3:
for(i=2:n-1)

    ## STEP4:
    summation = 0;
    for (k=1:i-1)
        summation = summation + (L(i,k) * U (k,i));
    endfor
    U(i,i) = (A(i,i) - summation)/ L(i,i);

    if (L(i,i)*U(i,i)) == 0
      fprintf('error. LU Factorization impossible.\n');
      error('L(i,i)*U(i,i) == 0');
      return;
    endif

    ## STEP5:
    for(j=i+1:n)
        summation1=0;
        summation2=0;
        ## summation:
        for(k=1:i-1)
            summation1 = summation1 + (L(i,k) .* U(k,j));
            summation2 = summation2 + (L(j,k) .* U(k,i));
        endfor

        U(i,j) = 1/(L(i,i)) * (A(i,j) - summation1);
        L(j,i) = 1/(U(i,i)) * (A(j,i) - summation2);
    endfor

endfor
```

```
## STEP6:
summation = 0;
for (k=1:n-1)
    summation = summation + (L(n,k) * U (k,n));
endfor
U(n,n) = (A(n,n) - summation)/ L(n,n);

if (L(n,n) .* U(n,n)) == 0
    fprintf('NOTE: matrix A is singular.\n');
endif

## STEP7:
## A is now factored into U and L triangular matrices.
```

Sample execution:

First, the user is prompted to input the desired size of matrix:

```
Command Window
                Numerical Analysis
                ---Coding Exam---

Enter the size of your square matrix: |
```

```
Command Window
                Numerical Analysis
                ---Coding Exam---

Enter the size of your square matrix: 4

matrix size: 4

Enter row 1 values (separated by spaces): |
```

Then, the user enters the matrix entries row by row, separated by spaces.

```
Command Window
                Numerical Analysis
                ---Coding Exam---

Enter the size of your square matrix: 4

matrix size: 4

Enter row 1 values (separated by spaces): 1 2 3 4
Enter row 2 values (separated by spaces): 2 3 4 5
Enter row 3 values (separated by spaces): 6 7 8 9
Enter row 4 values (separated by spaces): v 1 2
Invalid input. Please enter 4 numeric values.    or enter 'x' to quit
Enter row 4 values (separated by spaces): |
```

The program catches invalid input, and prompts the user to re-enter the values. (error catching)

Additionally, the program will tell the user if the matrix they entered is impossible to be factored using LU decomposition. This example is impossible to be factored without swapping any rows.

```
Command Window
                    Numerical Analysis
                    ---Coding Exam---


matrix size: 4

User Matrix:
   1   2   3   4
   2   3   4   5
   6   7   8   9
   9   8   7   6


error. LU Factorization impossible.
error: L(i,i)*U(i,i) == 0
error: called from
    fn_LU_factorization at line 47 column 11
    Vbonus at line 60 column 8
>> |
```

Samples of successful executions:

```
Command Window
                Numerical Analysis
                ---Coding Exam---


matrix size: 4

User Matrix:
   1   1   0   4
   2  -1   5   0
   5   2   1   2
  -3   0   2   6


Upper triangular matrix U:
    1.0000    1.0000         0    4.0000
         0   -3.0000    5.0000   -8.0000
         0         0   -4.0000  -10.0000
         0         0         0   -7.5000


Lower triangular matrix L:
   1.0000         0         0         0
   2.0000    1.0000         0         0
   5.0000    1.0000    1.0000         0
  -3.0000   -1.0000   -1.7500    1.0000


>> |
```

```
Command Window
                Numerical Analysis
                ---Coding Exam---


matrix size: 4

User Matrix:
    2   14    3    1
    1    5   -1    3
    1   -2    2   -3
    3   -4   -3   -4


Upper triangular matrix U:
    2.0000   14.0000    3.0000    1.0000
         0   -2.0000   -2.5000    2.5000
         0         0   11.7500  -14.7500
         0         0         0   -6.9362


Lower triangular matrix L:
    1.0000         0         0         0
    0.5000    1.0000         0         0
    0.5000    4.5000    1.0000         0
    1.5000   12.5000    2.0213    1.0000


>> |
```

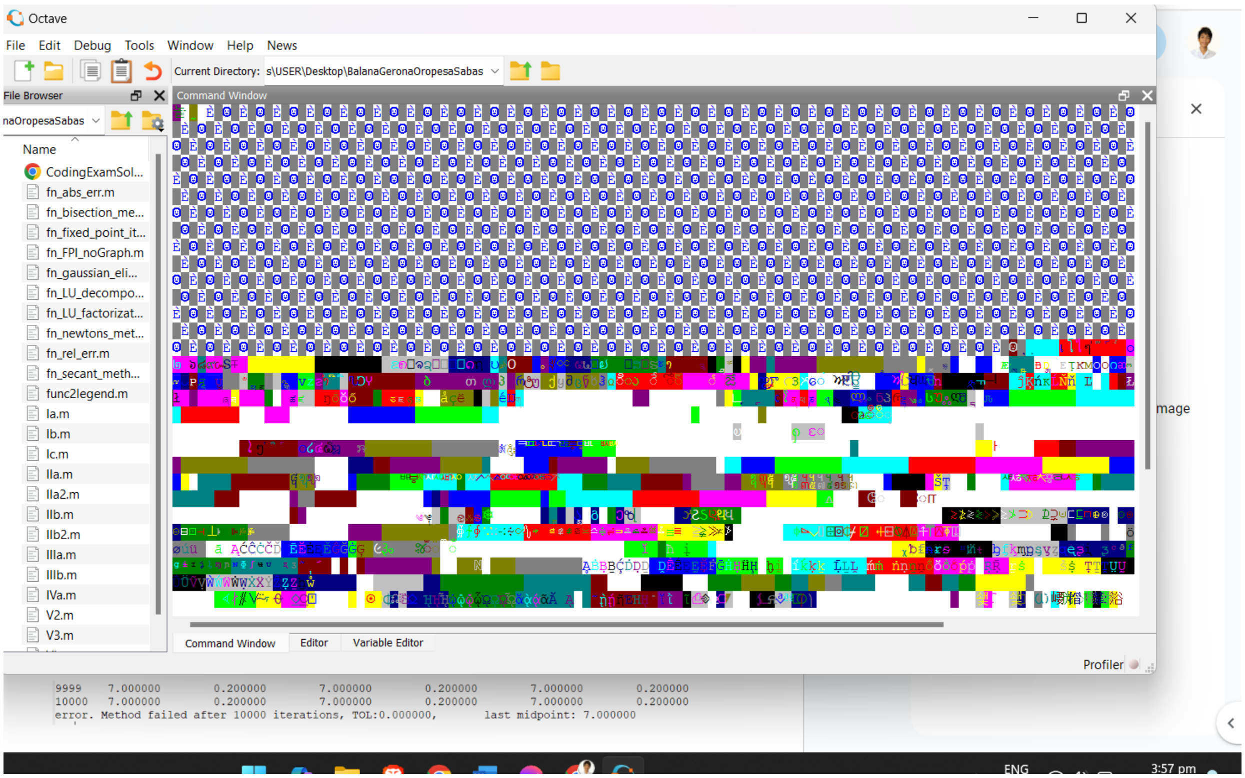
The user can also choose to continue to find solutions, given a set of b, or right side of the linear equations.:

```
Command Window
                   Numerical Analysis
                   ---Coding Exam---


matrix size: 3

User Matrix:
     1     2     3
     3     8    14
     2     6    13


Upper triangular matrix U:
   1   2   3
   0   2   5
   0   0   2


Lower triangular matrix L:
   1   0   0
   3   1   0
   2   1   1


Would you like to continue and find solutions?
         [1] Enter 1 for Yes.
         [0] Enter 0 for No.
>>1
         Enter b1: 1
         Enter b2: 2
         Enter b3: 3
```

```
Command Window
                   Numerical Analysis
                   ---Coding Exam---


matrix size: 3

User Matrix:
     1     2     3
     3     8    14
     2     6    13


Upper triangular matrix U:
   1   2   3
   0   2   5
   0   0   2


Lower triangular matrix L:
   1   0   0
   3   1   0
   2   1   1


right side of equations:
        b1: 1.000000
        b2: 2.000000
        b3: 3.000000

Unique solution found:
        x1: 4.000000
        x2: -3.000000
        x3: 1.000000
>> 
```

Problems encountered:

1. Sometimes, when trying to execute the program using Octave-8.4.0 - GUI, the command windows will freeze mid-execution and the program won't finish executing. Other times, the command window breaks (see below). One workaround is by using Octave CLI, this issue does not happen. However it is a hassle to use because we have to change directory first >>cd C:\location…\BalanaGeronaOropesaSabas, then run the program.



Sample execution in Octave CLI:

```
Select Octave-8.4.0 (CLI)

octave:1> cd C:\Users\USER\Desktop\BalanaGeronaOropesaSabas
octave:2> Ia
bisection method of (x^{6}) -x -1:

n       a           f(a)        b           f(b)        p - midpoint    f(p)
        r_error
1       1.000000    -1.000000   2.000000    61.000000   1.500000    8.890625
2       1.000000    -1.000000   1.500000    8.890625    1.250000    1.564697
3       1.000000    -1.000000   1.250000    1.564697    1.125000    -0.097713
4       1.125000    -0.097713   1.250000    1.564697    1.187500    0.616653
5       1.125000    -0.097713   1.187500    0.616653    1.156250    0.233269
6       1.125000    -0.097713   1.156250    0.233269    1.140625    0.061578
7       1.125000    -0.097713   1.140625    0.061578    1.132812    -0.019576
8       1.132812    -0.019576   1.140625    0.061578    1.136719    0.020619
9       1.132812    -0.019576   1.136719    0.020619    1.134766    0.000427
10      1.132812    -0.019576   1.134766    0.000427    1.133789    -0.009598
11      1.133789    -0.009598   1.134766    0.000427    1.134277    -0.004591
12      1.134277    -0.004591   1.134766    0.000427    1.134521    -0.002084
13      1.134521    -0.002084   1.134766    0.000427    1.134644    -0.000829
14      1.134644    -0.000829   1.134766    0.000427    1.134705    -0.000201
15      1.134705    -0.000201   1.134766    0.000427    1.134735    0.000113
16      1.134705    -0.000201   1.134735    0.000113    1.134720    -0.000044
17      1.134720    -0.000044   1.134735    0.000113    1.134727    0.000034

ROOT REACHED: 1.13472748
octave:3>
```